Verificação Formal de Blocos Complexos em Arquiteturas

Guilherme Henrique R. Jorge
IC / Unicamp RA: 096231
guijorge@gmail.com

Resumo

O crescimento exponencial de complexidade dos dispositivos de integração em escala muito grande (do inglês VLSI) vem acontecendo continuamente atraves das ultimas décadas e não mostra sinais de desaceleração. Esse crescimento tem levado ao limite nossa capacidade para projetar e em particular para verificar dispositivos VLSI. A industria de projeto VLSI está em constrante inovação, introduzindo novas metodologias e novas tecnologias para lidar com esse crescimento de complexidade[1]. Uma tecnologia promissora que vem sendo desenvolvida recentemente é a verificação baseada em asserções (Assertion-Based Verification ou ABV) com o uso de ferramentas de análise baseadas em simulação e métodos formais. A proposta desse trabalho é prover uma visão sobre ABV e analise formal e suas metodologias.

Palavras-Chave

Verificação Formal, VLSI, Asserções, Analise Formal, Complexidade.

1. Verificação Baseada em Asserções

Fundamentalmente, ABV envolve o uso de asserções como forma de validar parte ou toda a lógica comportamental funcional de um projeto. Asserções são afirmações formais totalmente não ambíguas sobre um comportamento requerido ou esperado. Asserções podem ser criadas e aplicadas em qualquer lugar na hierarquia de projeto. No nível mais alto, elas essencialmente provêem especificações de comportamento geral. Em níveis mais baixos, podem ser usadas para expressar o desejo do projetista. Um simulador pode checkar automaticamente as asserções durante o curso normal em que se roda um testbench enquanto uma ferramenta de analise formal pode checkar totalmente as asserções sem o uso de um testbench[2].

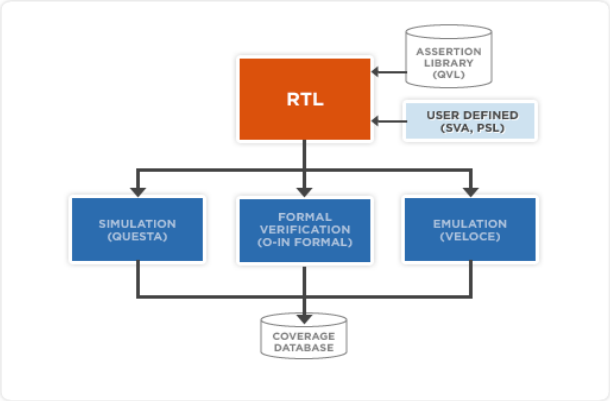

Figura 1. Fluxo de Verificação contendo ABV

O grande beneficio do uso de ABV é o aumento de produtividade no processo de design e especialmente na tarefa de verificação. Existem muitas razões para isso:

- Asserções permitem uma visibilidade melhor do design.

Com o uso de técnicas de verificação tradicionais erros comumente passam desapercebidos. Isso acontece porque normalmente os erros não se propagam até o ponto de serem visíveis ao ambiente de verificação. Uma vez que asserções normalmente monitoram sinais internos, elas provem uma visão melhor do design. O resultado, quando usado em ambientes baseados em simulação, é que elas aumentam a chance de se encontrar um bug no design.

- Asserções provem um mecanismo adicional de provar que o design esta correto.

Os tipos de erros feitos quando se escreve asserções são bem diferentes dos de quando se desenvolve código RTL (Register Transfer Level, ou Verilog/VHDL sintetizável) e de quando se escreve testbenchs.

Por causa disso, erros em RTL e testbenchs serão freqüentemente pegos por asserções, enquanto erros em asserções serão pegos pelo correto exercício funcional do RTL.

- Asserções ajudam na documentação do código.

Normalmente, um projetista escreverá uma asserção que indica uma presunção feita durante o projeto do bloco. Futuramente, o mesmo projetista, ou outra pessoa, pode ler essas mesmas presunções para se familiarizar com o design.

Alem do mais, presunções em um bloco podem ser comparadas com outros blocos afim de assegurar que esses outros blocos não estimulam ou controlam esse bloco de forma diferente da que foi assumida.

- Asserções permitem um debug mais rápido.

Devido ao fato de as asserções estarem associadas a partes bem específicas do hardware que elas verificam, a falha de uma assertiva normalmente indica uma falha num ponto bem específico do RTL. Isso facilita a identificação da causa raiz da falha. O uso de ABV em sistemas baseados em simulação e com o uso de analise formal são complementarias. Asserções escritas para uso em analise formal também podem ser usadas em simulações. Reciprocamente, asserções escritas para uso em sistemas baseados em simulação também podem ser usadas em analise formal. Outras ferramentas como emuladores também podem ser usados para provar as mesmas asserções. Essa

interoperabilidade a nível de asserções possibilita que os esforços feitos para uma forma de ABV sejam reutilizados em outras formas.

2.Analise Formal

Analise formal é o processo de verificação funcional (ao contrário de elétrica, de tempo, etc) de asserções, ou a procura de bugs funcionais em um design através do uso de ferramentas de análise formal[3]. Através do uso de algorítimos sofisticados, ferramentas de analise formal são capazes de checar completamente todos os possíveis estados operacionais e todas as combinações de entrada de um determinado design, como ilustrado na Figura 2.

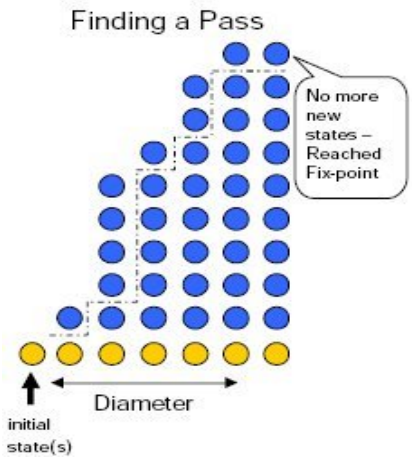

Figura 2. Exploração de asserções.

Por esse motivo, elas pode conclusivamente comprovar que uma asserção é verdadeira em todos os seus estados possíveis.

Analise formal tem ganhado atenção recentemente pelo seu potencial em aumentar a qualidade e a produtividade além do que pode ser alcançado por métodos de verificação tradicionais, ou através do uso de ABV por simulações somente.A qualidade aumenta porque a analise formal encontra bugs que a verificação baseada em simulação pode não encontrar. A produtividade aumenta porque:

- Os bugs são isolados o que torna o debug e correção de problemas mais fácil e rápido.
- Os bugs são identificados mais facilmente e mais cedo no ciclo de projeto.
- Quanto antes se acha um bug, menos pessoas são envolvidas.
- Eliminar os bugs mais cedo significa menos pessoas impactadas e o fluxo de projeto flui melhor.

A combinação de melhor qualidade e produtividade significa melhor time-to-market e reduz custos (pela necessidade de menos re-spins).

2.1 Modelos de Analise formal baseado em asserções

Como mostrado na Figura 3, asserções documentam a proposta do design para as entradas e saídas de um bloco assim como para o comportamento interno do bloco. Asserções que fazem referencia apenas a E/S do bloco são chamadas “black-box” uma vez que elas não dependem da implementação interna do bloco. Asserções “End-to-end” especificam o comportamento das entradas para as saídas “end to end”. Elas então relacionam o resultado esperado nas saídas com os valores das entradas. Asserções que fazem referencia as estruturas internas do design são chamadas “White-box”. Elas ficam “amarradas” a implementação do RTL[4].

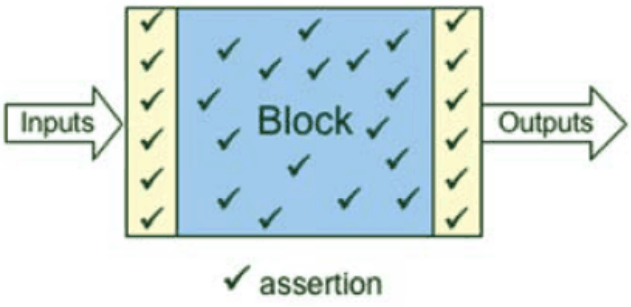

Figura 3. Asserções dentro do design

Asserções que especificam protocolos de entrada e saída refletem não apenas a um único bloco, mas a interação entre o bloco e seus vizinhos. Asserções de saída checam que os resultados do bloco estão de acordo com o que é esperado pelos outros blocos que recebem os seus sinais de saída. De modo inverso, asserções de entrada representam a gama de entradas para a qual o bloco foi projetado. Elas averiguam entradas ilegais vindas de outros blocos.

Asserções de entrada são especialmente importantes para a análise formal, pois podem ser aplicadas ao bloco antes de qualquer testbench ou teste serem escritos. Como mostrado na Figura 4, asserções de entrada são tratadas como constantes (constraints). Isso assegura que a análise formal considera apenas combinações válidas de entrada. Um problema que venha a ser detectado pelo uso de entradas ilegais geralmente não é de interesse do projetista. Quando se está preocupado em verificar o comportamento de um bloco, normalmente, isso não é um bug real uma vez que a especificação do sistema não deve permitir que os outros blocos tenham tal comportamento.

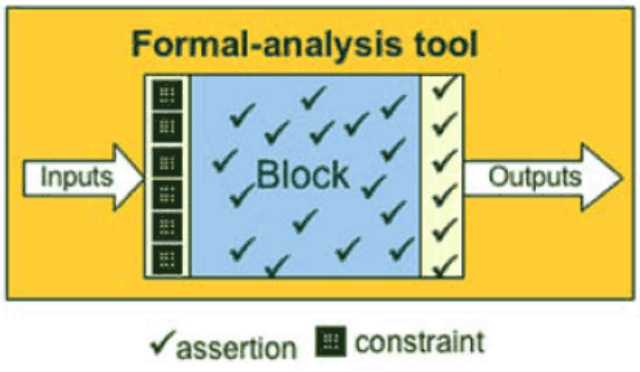

Figura 4. Asserções em analise formal

Para cada asserção que não é convertida em constante, a analise formal tenta provar que a assertiva não será nunca violada ou gerará um “contra-exemplo” mostrando como/quando uma violação pode ocorrer. Algumas ferramentas conseguem gerar o contra-exemplo na forma de caso de teste. Esse caso de teste pode ser rodado em uma simulação. O bug que está causando a falha da asserção pode ser então diagnosticado em um ambiente mais familiar.

A sintaxe de uma asserção simples parece muito com o Verilog tradicional. Uma asserção que garante que uma FIFO não deve sofrer um “underflow” deve ser escrita da seguinte forma:

```
assert_no_underflow:   assert   property
(@(posedge                          clk)
   !(fifo_read && fifo_empty && rst_n));
```

Essa asserção, nomeada de assert_no_underflow pelo projetista, reporta uma violação ou falha quando o sinal de escrita da FIFO e o sinal de full (cheio) da FIFO estão ambos ativos. Esse teste é feito apenas quando o sinal de reset não está ativo e apenas na borda de subida do clock. Sinais em transição não são considerados. Esse é um bom exemplo de uma asserção de grande valor em uma simulação chip-level.

3.Complexidade em Analise Formal

3.1O que define complexidade em Analise Formal?

Em geral, o que define a complexidade em Analise Formal é uma analise que demore um tempo longo para apresentar resultados ou que consuma uma quantidade muito grande memória.

3.2Impacto da complexidade nos resultados

Muitos fatores contribuem para o problema da complexidade em Analise Formal. Além dos fatores de primeira ordem (que incluem características como tamanho e diâmetro do design), existem vários componentes adicionais em um problema complexo (fatores de complexidade de segunda e terceira ordem como a estrutura do design e transições de maquinas de estado). Esses componentes adicionam várias dimensões adicionais ao problema. Para encontrar maneiras de superar a complexidade nós devemos nos concentrar em fatores que podem ser facilmente entendidos e que podem assim nos ajudar a superar a complexidade com relativa facilidade.

Muitas ferramentas de analise formal executam suas analises com base em um tempo limite de time-out. Esse tempo de time-out é geralmente controlado pelo usuário. Quando a complexidade de um problema é muito grande para ser superado em uma dada quantidade de tempo, as ferramentas não irão produzir um resultado de PASS ou FAIL. Nesse caso, a maior parte das ferramentas prove algum tipo de métrica que informa o quanto da análise foi feita até que o tempo limite fosse atingido. No IFV (Incisive Formal Verifier) da Cadence, esse tipo de resultado é chamado de “explored” e a métrica associada é chamada de “depth”. Portanto, um resultado “explored” é uma função de tempo em relação a complexidade.

Quando se fala em superar complexidade, quer-se dizer que é desejado diminuir o tempo de execução, aumentar o “depth” de um resultado “explored” ou, mais ainda, transformar um “explored” em um resultado PASS ou FAIL. No final, quer-se ser capaz de conseguir uma melhor performance da ferramenta através da redução da complexidade[5].

3.3Precisão das medidas de complexidade

Conforme os fatores de complexidade são encontrados, tenha em mente que esses fatores variam muito de caso para caso. Por exemplo, de um lado é possível ver casos aonde as medidas de complexidade são altas, mas as ferramentas ainda sim são capazes de convergir a um resultado em um tempo razoavelmente curto e usando uma quantidade de memória pequena. Isso é possível porque as ferramentas usam meios automáticos de superar as complexidades.[6]

De outro lado, é possível encontrar casos aonde as medidas de complexidade são baixas e ainda sim as ferramentas não chegam a resultados conclusivos.

A complexidade nesses casos é causada por fatores de segunda e terceira ordem cuja discussão e analise estão além do escopo desse trabalho.

Em outras palavras, não é sempre possível de se fazer uma estimativa direta do desempenho a ser esperado das ferramentas baseado apenas nos fatores aqui apresentados. Existe sempre a possibilidade de se encontrar os dois tipos de caso listados: altas medidas de complexidade com bons resultados e baixas medidas de complexidade com resultados “explored”.

4.Conclusão

Com todos os benefícios que o uso da análise formal prove, pode parecer que tudo o que precisa ser feito para verificar um projeto é:

- Escrever asserções que especifiquem como o design deve se comportar.
- Colocar o design e as asserções em uma ferramenta formal e analisar se as asserções provam o comportamento correto ou não do design.

Há se fosse tão fácil! Na realidade existem outras coisas que precisam ser levadas em consideração quando do uso de analise formal em designs na vida real.

As ferramentas de analise formal tem mais limitações que as simulações em termos de complexidade e tamanho do design com que elas podem trabalhar pelo simples motivo que o numero de estados que precisam ser analisados cresce de forma exponencial de acordo com o numero de elementos em cada design.

Isso significa que poucos chips hoje podem ser analisados por métodos formais em apenas uma grande rodada. E em outros casos, blocos de complexidade relativamente alta podem ser analisados separadamente.

Isso não significa que a analise formal não dê bons retornos ao seu investimento em designs reais, mas sim que para conseguir bons resultados você precisa de bons modelos, metodologias, e o treinamento necessário para usar correta e eficientemente as ferramentas.

5.REFERENCIAS

[1] Maniliac, David. 2002 Assertion-Based Verification Smoothes The Road to IP Reuse. Electronic Design Online ID #2748

[2] Lee, James M. Assertion Based Verification. http://www.jmlzone.com/

[3] Bjesse, Per. 2005 What is formal verification? ACM, New York, NY, USA.

[4] Anderson, Tom. 2007 SystemVerilog Assertions and Functional Coverage Support Advanced Verification. Chip Design Magazine

[5] Cadence web site. http://www.cadence.com

[6] Mentor web site. http://www.mentor.com