Técnicas e ferramentas de validação de Sistemas Digitais

Todo Sistema Digital (SD), inclusive Sistemas Embarcados (SE), são construídos a partir de um projeto. Esse projeto descreve quais são os principais requisitos do sistema. Esses requisitos são chamados de propriedades ou especificações. Eles fornecem os detalhes sobre implementação e funcionamento de um sistema como um todo.

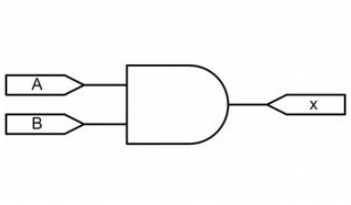
Os requisitos de um sistema devem ser claros e evitar ambiguidades. Estes são comuns em línguas naturais, como o inglês e o português. Uma forma de evitar esses problemas é descrever as propriedades matematicamente e de forma precisas. Essa especificação matemática é também conhecida como especificação formal. Uma das notações existentes é a lógica temporal.

Ferramentas de simulação lógica podem usar descrição de projetos a nível de RTL (*register-transfer level*) para realizar a verificação da corretude do sistema. O projeto no nível de transferência entre registradores (RTL) ocupa um espaço importante no contexto de ASICs (*Application Specific Integrated Circuits*), principalmente nos PLDs (dispositivos lógicos programáveis). RTL é necessária no contexto de projeto de sistemas digitais, uma vez que, mesmo os potentes computadores que estão disponíveis no mercado, não são capazes de dado uma descrição de um circuito lógico em linguagem comum, converter isso para um circuito lógico funcional (físico). A subfamília que mais se vale das linguagens de descrição de hardware são os PLDs. A RTL pode ser entendida como uma forma de se projetar ou descrever um circuito síncrono e seu fluxo de dados.

A simulação tem um papel importante no processo de validação de um sistema digital, isso porque ele ainda é um dos principais veículos para a verificação funcional. Entretanto, ela pode ser inadequada devido ao tamanho do projeto.

O projeto em nível de transferência entre registradores, ou projeto RTL, é composto por uma grande variedade de abordagens. Todas as abordagens, geralmente, possuem uma semelhança, onde um projetista especifica os registradores de um circuito, descreve as possíveis transferências e operações a serem realizadas com os dados de entrada, de saída e dos registradores, além de definir o controle que especifica quando transferir e operar com os dados.

Pode-se usar uma linguagem de descrição de hardware (HDL, do inglês, Hardware Description Languages) para capturar o comportamento de um circuito e implementar uma RTL. HDL é usada para descrever a estrutura de sistemas eletrônicos e seu comportamento ao longo do tempo utilizando expressões padrão baseadas em texto. Ela utiliza sintaxe e semântica para deixar explícito notações que expressam simultaneidade, além de noções explícitas de tempo, a qual é um atributo primário do hardware. Ele é usado para escrever especificações executáveis para o hardware. A Figura 1 mostra um exemplo da criação da porta AND usando HDL.




```
1  ENTITY portas_2_1 IS
2    PORT ( a, b : IN BIT;
3             x : OUT BIT);
4  END portas_2_1;
5
6  ARCHITECTURE pand OF portas_2_1 IS
7  BEGIN
8    x <= a and b;
9  END pand;
```

Figura 1: Exemplo do uso de HDL

A criação da HDL e RTL visam fornecer ao projetista de hardware a capacidade de modelar, simular e testar uma peça de hardware antes mesmo que ela seja criada fisicamente. Isso é feito ao implementar a semântica subjacente das instruções de linguagens e simular o progresso de execução com o passar do tempo. Há HDLs direcionadas para suportar modelagem de eventos discretos (digital) e de tempo contínuo (analógico). O projeto HDL tem em um dos seus passos, a construção de uma máquina de estados de alto nível.

A tradução das especificações de um projeto para um modelo RTL, seja por meio de HDL, ou outra abordagem, é um processo manual e está sujeito a erros. Isso porque as especificações são realizadas, normalmente, com o uso de linguagens informais. As linguagens informais são sujeitas a diferentes interpretações, uma vez que ambiguidade é comum. Considerando esse cenário, o processo de validação é de suma importância dentro do projeto de sistemas digitais, principalmente quando envolve HDL e RTL. Isso ocorre porque bugs em hardware são difíceis de se recuperar, muito caros, e bastante diferente do que ocorre em softwares.

Devido as vantagens das máquinas de estados de alto nível, o seu uso se sobressai sobre as máquinas de estados finitos. A FSM lida somente com entradas e saídas booleanas de somente um bit, enquanto as máquinas de estados de alto nível oferecem uma entrada com mais bits e operações (atribuição, soma). Outro ponto é que as FSMs não permitem o armazenamento local de dados (o único dado armazenado é o valor do próprio estado). Além disso, as ações nos estados e as condições para as transições das máquinas de estados de alto nível envolvem operações com dados, ao passo que uma FSM permite somente equações expressões booleanas. Pode-se entender as máquinas de estados de alto nível, ou FSM com dados (FSMD), como uma extensão das FSMs. Há outros tipos de máquinas de estados, como as máquinas algorítmicas de estados, ou ASM (*Algorithmic State Machine*) que se assemelham a fluxogramas com a inclusão da noção de tempo. Ela é mais restrita que as máquinas de estados comuns, uma vez que restringem as transições de tal forma que a computação se assemelhe um algoritmo (uma sequência ordenada de instruções). O uso das HDLs acabaram por substituir o de ASMs. As FSM ajudam no processo de simulação e validação dos hardwares. Diversas linguagens podem ser simuladas, contudo nem todas podem ser sintetizadas (convertidas para um sistema físico, por exemplo por meio de um CPLDs (Complex Programmable Logic Device) e FPGAs (Field Programmable Gate Array).

O processo de verificação ocupa grande parte do esforço e energia empregados no design de sistemas digitais. Conforme cresce a complexidade dos sistemas criados, cresce a complexidade do processo de verificação. Para facilitar esse processo, linguagens de

verificação de hardware foram criadas, como por exemplo SystemVerilog, SystemC e OpenVera. Os projetistas utilizam de diversas abordagens para verificar e depurar tanto o hardware e software para projetos de SoC (*System on Chip*). Essas abordagens envolvem a utilização de simuladores, emulação e/ou um protótipo de FPGA.

A validação de SD é um importante ao passo que é possível definir se um sistema possui os comportamentos esperados, ou se está agindo de forma inadequada, além de verificar se a implementação está de acordo com a especificação. Vale ressaltar que, levando-se em consideração o conceito de probabilidade, pode-se afirmar que um sistema com um número infinito de entradas ou comportamentos distintos pouco testado/validado, tem a mesma probabilidade de estar correto do que um que foi bastante validado.

Considerando esse cenário, um dos caminhos é validar os aspectos mais intrínsecos/essencial do sistema. Essa validação pode ocorrer por meio de verificadores de modelos e de teoremas. Eles atestam fielmente se o sistema implementado atende aos requisitos que foram descritos no início do projeto. Esse coloca em foco os mais diversos sistemas formais para verificação de corretude de sistemas digitais.

Antes de qualquer verificação de um projeto é necessário um planejamento. Este planejamento envolve a identificação de quais propriedades e partes do sistemas interessam ser validadas. As partes do sistemas que serão avaliadas partem dos projetistas que avaliam as especificações do sistema e podem sugerir outras verificações. É necessário definir até qual nível será testado, se será somente o modelo comportamental ou o modelo físico. Um ponto de confluência é a de verificar funcionalidades e interconexões entre os componentes. Uma vez que foi definido o que será testado, passa-se a definir o como será testado. Isso inclui agrupar testes com requisitos semelhantes para testar-se partes específicas do sistema, como a verificação da ULA, o funcionamento de uma CPU, a qual envolve o teste da ULA indiretamente. É desejável a criação de bancadas de teste para cada parte do sistema. O processo de verificação pode incluir novos recursos ao projeto, uma vez que pode-se ter projetistas responsáveis pelo design da verificação.

Um dos problemas envolvendo o ser humano no processo de verificação é a ambiguidade que sua interpretação pode trazer. Não necessariamente a verificação é feita de acordo com a especificação original. O processo de verificação pode ser viciado, uma vez que fica dependente da interpretação do ser humano. Pode acontecer que as especificações contenham ambiguidades, as quais influenciam nos passos seguintes. Uma das formas de contornar o fator humano na verificação é por meio de redundância e de automatização. O processo de redundância consiste em fazer com que o engenheiro que criou sistema, não se envolva no processo de verificação. Nesse caso, um outro engenheiro é usado para somente criar o testes para verificação e os dois projetos são criados simultaneamente, todavia separados. Já a verificação automatizada é feita de outras formas:

- Verificação formal: utiliza procedimentos formais para provar características de um modelo. Normalmente, essa verificação pode ocorrer de duas formas: Prova de Teorema e checagem de modelo (*model checking*). A verificação formal ocorre geralmente em sistemas fechados, ou seja, não há entradas. A verificação formal consiste em técnicas que aplicam modelos e linguagem matemática no projeto, especificação, refinamento, implementação, verificação e validação de um software.

A exatidão proporcionada pelo formalismo matemático reduz a ambiguidade, erros e inconsistência do projeto, além de viabilizar a automação de testes e até a geração de código.

- Verificação funcional: pretende provar que o modelo RTL satisfaz as especificações. Nesse caso pode-se mostrar que as funcionalidades especificadas são satisfeitas, entretanto, não é possível provar que ela é satisfeita. A verificação funcional utiliza de ambientes abertos. Os sistemas abertos são aqueles que mantém interações com o ambiente, por meio de entradas e, dependendo do sistema, gerando saídas para alterar o ambiente.
- Bancada de testes: Geram diversos estímulos para verificar certas propriedades do modelo. Isso pode ser feito tentando violar uma propriedade.

O *model checking* é um método algorítmico para determinar se um sistema satisfaz uma especificação formal descrita como uma lógica temporal (comportamento do circuito ao longo do tempo). O ponto central dessa abordagem é a noção do conjunto de estados acessíveis do modelo do sistema. Um dos desafios ao checar um modelo é encontrar a abstração mais simples, as quais irão fornecer as provas da segurança. Isso porque o espaço de avaliação é menor e pode ser checado mais eficientemente.

No contexto de *model checking*, existem diversas ferramentas disponíveis no mercado para determinar o conjunto de estados alcançáveis de um sistema e verificar se eles satisfazem as especificações na lógica temporal. Um dos exemplos é o SMV (*Symbolic model verifier*), da Carnegie Mellon University. Ele foi a primeira ferramenta de verificação de modelos a usar diagramas de decisão binária, a qual é uma estrutura de dados compacta para representar uma função booleana. Os verificadores de modelos atuais também dependem fortemente de solvers SATs (*Boolean satisfiability*), os quais são programas para decidir se uma fórmula lógica proposicional pode ser avaliada como verdadeira. Outro verificador de modelos é o SPIN, desenvolvido entre as décadas de 1980 e 1990 na Bell Labs. Ela usa uma linguagem de especificação chamada Promela (para processos e metalinguagem) ao invés de FSMs. A Promela permite especificações de códigos que se parecem com programas multithread. O SPIN incorpora técnicas de compressão de estados, como compactação de hash e a redução de ordem parcial (considera um subconjunto das intercalações de processos possíveis para reduzir o número de estados acessíveis a serem explorados).

Outra forma de verificação formal é a Prova de Teorema, a qual utiliza a inferência lógica. A inferência lógica se baseia na geração de provas matemáticas de restrições a partir do sistema e suas especificações utilizando provadores de teoremas como: HOL, ACL2, Isabelle, Coq ou PVS ou Satisfiability Modulo Theories (SMT). Nesse caso, é necessário um usuário com profundo conhecimento do software na formulação dos teoremas a serem provados e da especificação. O conjunto de ferramentas pode ser dividido em dois grupos: os Provadores de Teoremas Automáticos, que utilizam lógica de primeira ordem e relativamente pouca interferência humana, e os Assistentes de Provas, que utilizam lógicas de ordem superior e necessitam de maior interferência do usuário. A maior utilização da prova de teorema se dá na verificação de projetos de hardware, especialmente por grandes fabricantes como IBM, Intel, Motorola e AMD.

Ao contrário do Model Checking, no Theorem Proving o usuário fornece de forma manual as fórmulas (hipóteses e axiomas) e o código ou modelo executável para que o algoritmo realize a verificação. Se a prova do teorema não for bem sucedida, o usuário pode verificar as fórmulas ou tentar provar teoremas intermediários. Portanto, é um método de automação menor que necessita de usuários experientes para que sua aplicação seja bem sucedida.

Outras formas de abstração começaram a ser usadas para a verificação formal. Dentre elas abstrações automáticas, as quais vêm desempenhando um importante papel na aplicação da verificação de modelos diretamente ao software. O SLAM desenvolvido pela Microsoft é um exemplo de verificação de modelo baseado em abstração de software. O SLAm combina o CEGAR (refinamento de abstração guiado por contra-exemplo) com uma forma particular de abstração chamada de abstração de predicado. Nesse caso, os predicados em um programa são abstrações de variáveis booleanas. O CEGAR é uma eficiente e amplamente técnica usada para automaticamente computar simples abstrações. Um passo importante da técnica proposta no SLAM é a de verificar se um contra-exemplo gerado no modelo abstrato, é de fato, um contra-exemplo verdadeiro. Essa checagem é feita usando solvers de satisfabilidade para lógicas mais complexas que as lógicas proposicionais. Estes solvers são chamados de procedimento de decisão baseado em SAT ou solvers de teorias de módulo de satisfabilidade. Técnicas que vem ganhando espaço são as do tipo de aprendizagem indutiva, as quais realizam a generalização a partir de dados amostrais.

De forma geral, as condições de verificação se um modelo satisfaz condições especificadas cabe interpretação, isso porque, por exemplo, pode-se verificar se o modelo criado consegue chegar até todos os estados de uma FSM e não ocorrem deadlocks, ou se uma interface responde corretamente sob condições dadas, ou ambas, ou outras condições. O problema está em identificar dentro das especificações do projeto quais são as condições que devem ser verificadas.

Os métodos formais, por não serem técnicas clássicas no desenvolvimento de software, enfrentam muitos desafios na indústria de software. A aplicação de métodos formais em projetos de software significa uma mudança importante nos processos de desenvolvimento. Isso porque é necessário destinar uma parcela significativa de recursos, tempo, pessoas e dinheiro, necessária a cada fase. A abordagem formal torna muito mais custosas as fases iniciais do projeto como o desenvolvimento de requisitos. Entretanto, esse maior gasto no início do projeto, se comparado a métodos tradicionais, implica em fases finais menos onerosas, como é o caso da economia na verificação e validação.

A bancada de testes, ou *testbench*, visa gerar automaticamente de estímulos para simulação. Normalmente, essa geração é baseada em medidas de cobertura de código. Entretanto, algumas questões surgem, como quais são as partes do código RTL que devem ser testadas, ou quais estímulos são necessários para ativar e verificar cada parte do código. Cabe ao projetista definir esses pontos, além de verificar os resultados da simulação. Vale ressaltar que os estímulos produzidos não verificam a funcionalidade do projeto, são só gatilhos para ativar o sistema, o qual será verificado.

Normalmente, as formas pelas quais ocorrem o design de verificação funcional podem ser divididas em duas formas: hardware e simulação. A verificação usando hardware físico envolve a construção de um protótipo com um ASIC, especificamente um FPGA. O teste de

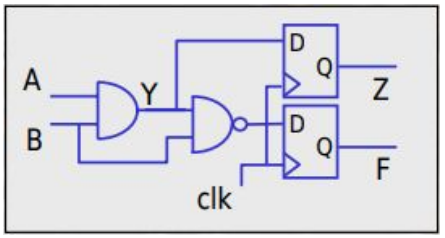
bancada tende a ser feito em hardware, o qual envolve testes com fontes de sinais e analisadores lógicos. Uma das ressalvas desse tipo de verificação é que as características elétricas e dinâmicas do ambiente de simulação podem ser diferentes da tecnologia alvo, todavia, o processo de validação tem de conseguir abstrair essas diferença. Um ponto positivo é que é possível verificar seu funcionamento praticamente em tempo real, além de permitir a verificação com estímulos complexos de sistemas físicos, uma vez que são difíceis de se modelar.

A verificação por meio de simulação utiliza dos modelos criados no projeto RTL apresenta uma flexibilidade maior do que a verificação por meio do hardware, contudo, ela é mais lenta. A verificação pode ocorrer em diversos estágios do projeto. Pode ocorrer desde a concepção o projeto RTL com HDL, a síntese (nível de portas), até o posicionamento dos componentes (verificação temporal) apresentados na *netlist*. Para o caso da verificação temporal, pode-se usar diversos modelos de atrasos, os quais são distribuídos nas bibliotecas que caracterizam a tecnologia alvo. Pode-se utilizar da construção de estímulos para detectar defeitos e simular falhar, além das consequências dos mesmos na saída do circuito. Dentre as suas vantagens, destaca-se a capacidade de se observar o estado de qualquer nó ou componente em qualquer momento, o que facilita a depuração, além de poder forçar sinais de estados desejados. Como é uma simulação, é possível parar e reiniciar a simulação em qualquer estado, além do controle preciso da temporização dos eventos assíncronos.

Envolto ao processo de simulação, há algumas questões. Não existe um procedimento formal para gerar os estímulos da simulação. Uma abordagem é a criação por processos heurísticos, os quais têm como base a intuição e conhecimento do projetista sobre o sistema. Deve-se usar processos heurísticos, pois é impraticável a verificação de todo o sistema (exaustiva). Por exemplo, um sistema com 70 entradas booleanas resultará em uma combinação no total de 2^{70} ($1.18e^{21}$) testes. Outro ponto importante é que os estímulos verificam a funcionalidade, contudo, o modelo simulado é uma aproximação, e não necessariamente, reflete o resultado do sistema implementado. Os estímulos em uma simulação podem ser divididos em duas naturezas: para verificação (detectar e diagnosticar erros de projeto, entretanto, é impossível enumerar todos os erros do projeto) e para teste (detectar e diagnosticar defeitos de fabricação - esses defeitos são modelados como erros lógicos).

Normalmente, o processo de simulação pode se basear em três abordagens: (i) simulação baseada em código compilado; (ii) simulação baseada em ciclo; e (iii) simulação orientada a objetos. A simulação baseada em código compilado consiste em traduzir o circuito em instruções de um processador, como é mostrado na Figura 2. Ela suporta apenas a verificação funcional, pois ela não suporta análise temporal. São usadas subrotinas para modelar o comportamento dos componentes. O seu uso é mais adequado para simular o comportamento de sistemas síncronos. A simulação baseada em ciclo pode fazer uso deste tipo de simulação, uma vez que há uma resposta para cada ciclo de clock. Já a simulação baseada em ciclo trabalha exclusivamente com sistemas síncronos. Para esse caso, todas as funções lógicas combinacionais são traduzidas em equações e só interessa conhecer o estado das entradas D dos Flip-Flops. Este tipo de simulação foca apenas na simulação das transferências entre registradores, os quais são sincronizados pelo sinal de clock. Para esse

caso, não são suportados qualquer informação temporal do circuito, por esse motivo, assume-se que todos os requisitos temporais dos Flip-Flops são satisfeitos, o que faz com que entradas assíncronas e circuitos com múltiplos domínios de relógio (clock) sejam desconsiderados. Para se realizar alguma análise temporal é necessário com ferramentas de análise estática, as quais só podem ser aplicadas a circuitos síncronos.



(a) Circuito a ser verificado

```
MOV AL,A
AND AL,B
MOV Y,AL
MOV AL,B
AND AL,Y
MOV AH,AL
NOT AL
MOV Z,AH
MOV F,AL
```

(b) Código gerado para cada transição de clk (código x86):

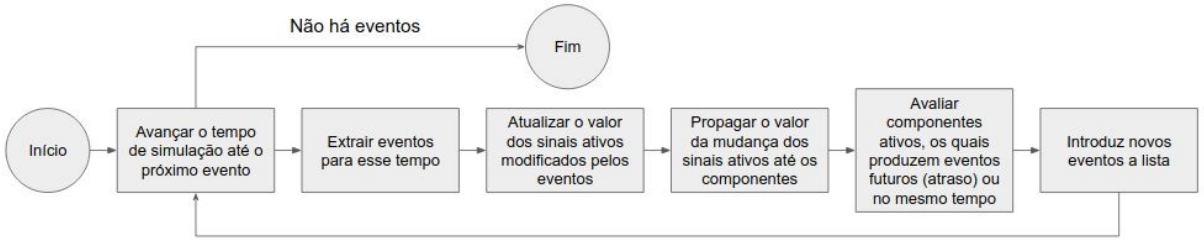
Figura 2: Exemplo de simulação baseado em código fonte

Os eventos ocorridos na simulação orientada a eventos são considerados como mudanças do estado de um sinal num tempo dado. Os componentes que usam esse sinal são então ativados e avaliados. Essa avaliação e troca de estado resultam em mudanças que geram novos eventos. Para que os eventos sejam gerados e propagados usa-se um modelo do circuito, este que pode ser um modelo RTL. Os estímulos para o sistema podem ser gerados por meio de eventos nas entradas primárias. Eventos nos nós restantes do modelo são gerados pela avaliação dos componentes do mesmo. Esse tipo de simulação permite processar estímulos e sinais assíncronos, o que faz com que seja possível a análise de comportamento com os atrasos dos componentes, além da avaliação do funcionamento do modelo com eventos não sincronizados. A Figura 3a mostra um esquema de como é a execução da simulação orientada a evento e Figura 3b, um exemplo da simulação em si usando Verilog (uma linguagem popular de HDL).

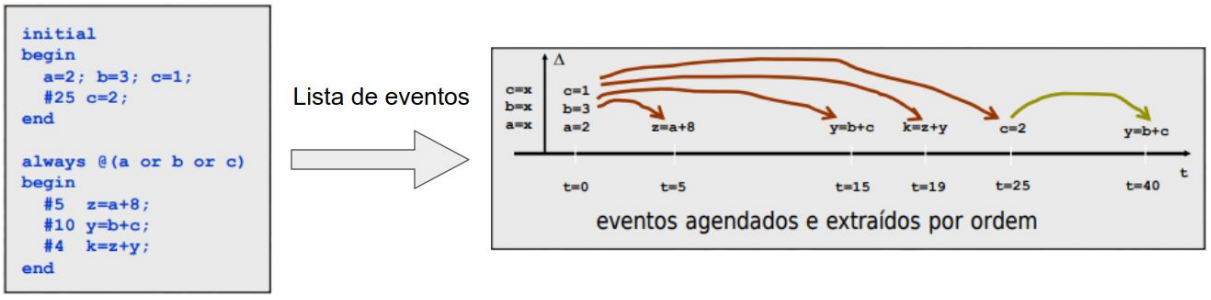
Em alguns cenários, pode ser interessante a cooperação de dois simuladores num só ambiente. Esse caso é conhecido como co-simulação. Um exemplo é a união da simulação baseado em ciclo e orientado a evento. Isso é bastante útil para simular as partes síncronas com um simulador, e com outro, as partes não síncronas. O uso de somente a simulação orientado a evento tende a ser mais lenta do que a co-simulação citada.

A fim de avaliar as partes assíncronas do sistema, existem alguns modelos de atrasos. Entre os modelos existentes destacam-se:

- Atraso de transporte: atraso na propagação do nível lógico.
- Atraso de subida e descida do clock: delay na troca do sinal (0-1 e 1-0) do clock.
- Atraso inercial: duração mínima de uma entrada para ser propagada para a saída.
- Tempo de setup e hold em Flip-Flops: o tempo de setup consiste no tempo mínimo para o Flip-Flop D estar estável antes de clock, o hold o tempo mínimo para D permanecer estável após o sinal do clock.



(a) Esquema da execução da simulação orientada a evento



(b) Exemplo da aplicação da simulação orientada a evento

Figura 3: Simulação orientada a evento

A aplicação dos métodos formais se faz mais presente da indústria de hardware, devido aos altos custos associados a erros de projeto. Um exemplo disso é um erro encontrado nos microprocessadores na família Pentium P5 da Intel, sucessor do também popular 486. Conhecido como Pentium FDIV bug (FDIV é o mnemônico na linguagem assembly para a divisão em ponto flutuante), o defeito foi descoberto pelo professor de matemática do Lynchburg College, Thomas Nicely, que ao analisar os cálculos de sua pesquisa, notou que os resultados não eram iguais aos realizados pelo processador 486. A falha foi noticiada no mundo inteiro e a Intel, também devido a erros no tratamento do caso, como a tentativa de minimizar sua importância, contabilizou um prejuízo superior a 400 milhões de dólares com a substituição das unidades afetadas, além da perda de confiança de seus clientes. O ciclo de desenvolvimento de um projeto de circuito integrado é muito extenso, complexo e de alto custo. Em todas as suas fases, desde a criação do conceito até a disponibilização do componente para o mercado, passando pelo projeto dos circuitos, do encapsulamento, testes, verificações e simulações, são empregadas diversas ferramentas e equipes altamente capacitadas, a fim de desenvolver o produto especificado e livre de erros. A Cadence Design Systems é uma companhia americana de desenvolvimento de software de projetos de circuitos integrados e ferramentas de verificação. Ela é proprietária da ferramenta JasperGold Apps, uma plataforma de verificação formal de projetos de hardware. O JasperGold Apps é dividido em basicamente duas partes: o parser, responsável pela construção do modelo, e as engines, que são as implementações de técnicas de verificação formal. Ele recebe como dado de entrada a descrição do circuito integrado a ser verificado, que na sequência é transformado em um modelo pelo parser. Em seguida o modelo é analisado pelas engines cujo objetivo é examinar o modelo em busca de condições que invalidem as asserções.

Além das técnicas convencionais de validação de sistemas digitais, novas tendências estão surgindo, como o uso de aprendizado de máquina para prever quando um sistema irá começar a dar problemas. Esse tipo de abordagem vai além da verificação convencional, uma vez que antecipa problemas futuros. Além disso, ela pode ser usada inclusive no solvers usados pelos verificadores, isso porque diferentes solvers (MiniSAT, MarchSAT) possuem qualidades e falhas, além disso, cada um possui um número de parâmetros que podem se sair bem com certos tipos de problema, vale lembrar que SAT é um problema NP completo. As técnicas de aprendizado de máquina podem entrar nesse cenário para auxiliar esse processo de escolha de parâmetros e de quais técnicas são melhores para cada problema, dada suas características.

A etapa de validação de um sistema (considerando o RTL comportamental e o RTL estrutural com todas as trilhas e conexões do sistema - netlist) é uma das partes do projeto que podem envolver mais tempo e custo de todo o processo de design de um sistema embarcado e sistemas em geral. Por esse motivo, é necessário que o circuito esteja correto (ou não tenha-se detectado erros) logo na primeira vez que for fabricado. Uma vez que o sistema passou por todas as validações, o circuito planejado é enviado para empresas para que sejam produzidos. Normalmente, há a produção de milhares, até milhões de unidades do Sistema Embarcado ou SoC. Um erro no planejamento e falhas que deixaram de ser detectadas podem ocasionar prejuízos de milhões de dólares, como aconteceu com a empresa Samsung e o smartphone Galaxy Note 7 que ocasionou um prejuízo de mais de 15 bilhões de dólares. A bateria do smartphone explodia, devido a problemas com sua bateria que aqueciam, aliada a falta de dissipação de calor. Várias unidades foram vendidas, e todas que não foram vendidas deixaram de ser comercializadas.

Disponível em: https://github.com/plsilva/Files_DigitalSystems

https://computing.ece.vt.edu/~mhsiao/papers/ats03lz.pdf

http://pdf.blucher.com.br.s3-sa-east-1.amazonaws.com/openaccess/9788580392234/08.pdf

https://paginas.fe.up.pt/~aja/PSD2005_06/slides/PSD0506-aula_6_slides-93-124.pdf

http://www.eldorado.org.br/blog/2018/04/04/verificacao-formal-e-seu-papel-no-desenvolvimento-de-sistemas-embarcados-criticos/

http://www.iwls.org/iwls2017/slides/keynote-manish-pandey.pdf

http://www.bibliotecadigital.ufmg.br/dspace/bitstream/handle/1843/ESBF-A9EGUS/monografia_fabioenrique_final.pdf?sequence=1

A verificação se dois modelos RTL/estruturais são equivalentes consistem em verificar se dois *netlists* (rede de ligações entre os elementos disponíveis na plataforma). As duas *netlist* precisam passar por algum pós-processamento, seja ele manual ou automático. Esse processo pode ser feito usando a inserção de cadeia de scan (para construir a infraestrutura de teste). Além disso, o seu foco é verificar se o netlist criado pela síntese é equivalente ao modelo RTL, este que pode ser descrito com HDL. Isso é importante, porque o modelo pode estar mal codificado, ou as ferramentas de síntese podem ter bugs. Outro ponto é que é possível verificar se já existe uma netlist para um modelo RTL.