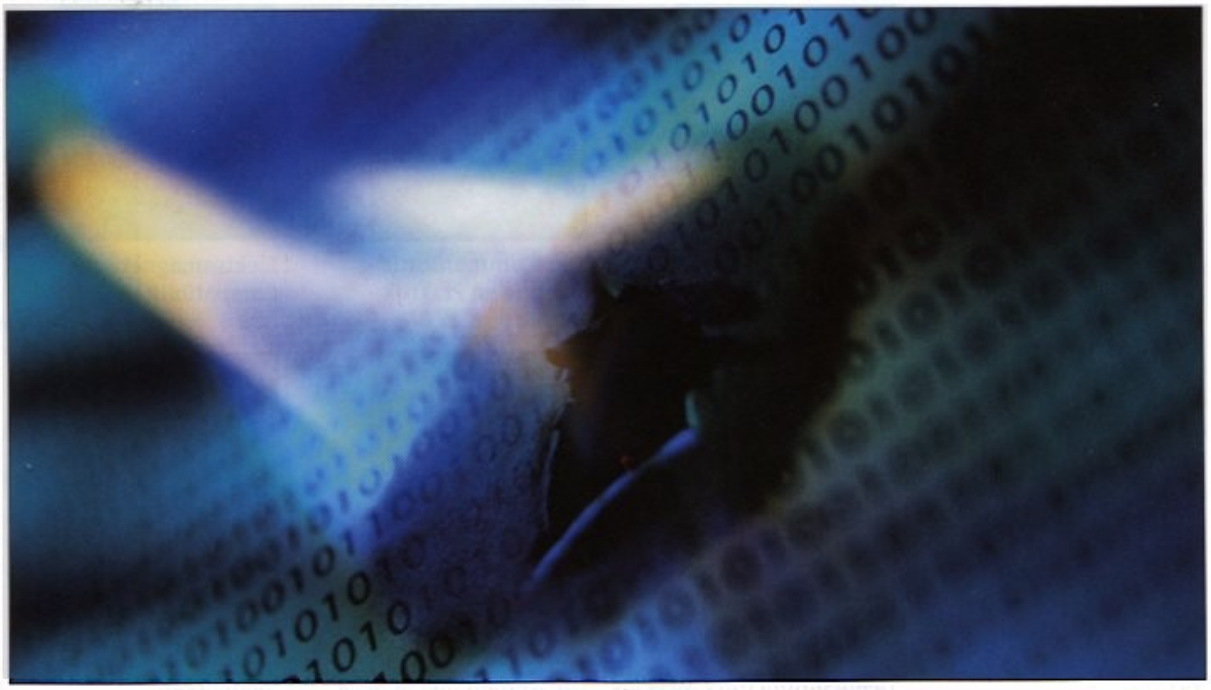
Formele verificatie voorkomt ontwerpfouten

Bewijzen is Weten



Elektronische hardware zou altijd en overal correct moeten functioneren, maar het waarmaken van die verwachting wordt met de toenemende complexiteit wel steeds moeilijker. Ontwerpfouten zijn nog het ergst omdat die, als ze niet tijdig worden ontdekt, per definitie tot elk geproduceerd exemplaar doordringen. Formele verificatie, het bewijzen van de correctheid van ontwerpstappen, kan echter bepaalde ontwerpfouten helpen voorkomen. Weliswaar zijn automatische bewijsprocedures beperkt toepasbaar maar chipontwerpers melden dat zij de gevestigde validatiemethoden goed aanvullen. Zo kon Intel door formele verificatie meer dan honderd logische fouten vermijden in de Pentium 4.

Fouten maken is menselijk, zo luidt het gezegde, en ook in de elektronica heeft men er dus mee te maken. Als een elektronische component niet goed werkt kan dat verschillende oorzaken hebben en verschillende gevolgen. In het algemeen zijn er drie soorten fouten te onderscheiden: fabricagefouten, defecten die tijdens het gebruik ontstaan en ontwerpfouten.
Fabricagefouten die niet worden veroorzaakt door eenmalige verstoringen van het productieproces zijn statistisch van aard. Zij kunnen meestal worden opgespoord door te testen en vervolgens kan het slechte exemplaar gerepareerd worden of verwijderd.

Defecten die tijdens het gebruik ontstaan kunnen niet in de fabriek worden opgespoord, maar de gevolgen kunnen wel worden beperkt door er in het ontwerp rekening mee te houden. Componenten kunnen dubbel worden uitgevoerd en dusdanig ingebouwd dat uitval van de een automatisch wordt gecompenseerd door de

ander (*redundancy*). Dit helpt ook bij fabricagefouten die op de een of andere manier door alle productiecontroles waren geglipt.
De derde klasse fouten is wel het gemeenst omdat die, als ze niet worden ontdekt en verholpen, in alle exemplaren van een product voorkomen. Het zijn de ontwerpfouten. Als beproevingen van het prototype zo'n fout niet hebben aangetoond zullen immers ook de controles bij massaproductie dat niet doen. Dubbele of driedubbele uitvoering van de component in een toepassing helpt niet. Als een onvoorziene ontwerpfout het falen van een exemplaar tot gevolg heeft zijn alle identieke alternatieven ook nutteloos. Vooral in kritische systemen (medische apparatuur, lucht- en ruimtevaart) zal men ontwerpfouten dus uit alle macht proberen te vermijden.

Maar ook bij massaproducten kunnen ontwerpfouten enorme gevolgen hebben. Klassiek is inmiddels de FDIV-fout in de Pentium-processor van 1994 die Intel $475 miljoen heeft gekost.
Strikt genomen is het probleem met fouten niet dat zij niet ontdekt worden. Een defect dat tijdens de levenscyclus van een product onopgemerkt blijft is geen defect. Het wordt pas een probleem als de fout zich kenbaar maakt door de consequenties ervan. Vaak gaat het dan om een combinatie van twee of meer omstandigheden die elk op zich al ongewoon zijn. De overgrote meerderheid van mogelijke systeemtoestanden is immers echt niet aan de aandacht van het ontwerpteam ontsnapt.
Het is een bekende uitspraak: testen kan wel de aanwezigheid van fouten aantonen maar niet de afwezigheid ervan. Bovendien zijn voor het testen van hardware werkende prototypen nodig en die kosten geld.
Simulatie is natuurlijk een belangrijk hulpmiddel voor hardware-ontwerpers, maar daarvoor geldt hetzelfde als voor testen. Niet alle potentiële combinaties zijn volledig af te dekken. Eigenlijk zou men de juistheid van alle uitgevoerde ontwerpstappen met wiskundige zekerheid willen kunnen bewijzen. Meten is weten, maar bewijzen is zeker weten. Er zijn thans technieken en tools waarmee dit, voor sommige onderdelen van het ontwerp, inderdaad mogelijk is.

Formele Verificatie

Om exact te kunnen bewijzen zijn allereerst formele representaties nodig, met bijbehorende modellen, voor elke stap van het ontwerpproces. Thomas Kropf (ref. 1) geeft in tabel 1 een globale karakterisering van hardware-ontwerp. Hij onderscheidt vier aspecten: gedrag, structuur, gegevens en tijd. Elk van die deelgebieden wordt ontwikkeld met behulp van vier abstractieniveaus: architectuur, registeroverdracht (RTL), poorten en tenslotte transistoren. De laatste fase in het ontwerp is dan nog de layout.
Elke hogere ontwerpstap is nu op te vatten als een specificatie van een lagere implementatie, die zelf weer specificatie is van een nog lagere stap. Het ontwerpproces bestaat dus uit een reeks stappen van specificatie naar implementatie, de synthese of verfijning (refinement).
Verificatie is dan het omgekeerde: aantonen dat elke implementatie aan de bovenliggende specificatie voldoet. Zo zal men willen bewijzen dat de beschrijving op poortniveau overeenkomt met de beschrijving op registerniveau en dat deze (nu beschouwd als implementatie) op zijn beurt voldoet aan de architectuurbeschrijving. In het algemeen kunnen er diverse implementaties zijn die aan eenzelfde specificatie voldoen.
Om een formele verificatie uit te kunnen voeren moeten allereerst de te vergelijken beschrijvingen in een formele notatie staan. Ruwweg (zie ook ref. 2 en 3) is dit een logisch- mathematisch symbolisme dat het mogelijk maakt om over modellen te redeneren. Om implementatie en specificatie met elkaar te kunnen vergelijken is verder nog een correctheidsrelatie vereist.
Op dezelfde manier kan men ook implementaties onderling vergelijken, bijvoorbeeld de RTL-structuurbeschrijving met de eindige toestandsmachine (FSM) die het gedrag modelleert. Tenslotte kunnen binnen eenzelfde tabelvak implementaties, bijvoorbeeld optimalisaties, op verschillen worden gecontroleerd.
Uit de tabel wordt wel aannemelijk dat er niet één formalisme bestaat dat alle deelaspecten kan beschrijven (zie ook figuur 1). Kropf geeft een uitgebreide behandeling van de verschillende achterliggende logische systemen met hun mogelijkheden en beperkingen.
Of formele verificatie in de praktijk uitvoerbaar is wordt echter behalve door de achterliggende logica bepaald door de beschikbaarheid van tools. Handmatig bewijzen is meestal door de complexiteit niet haalbaar en bovendien zelf een mogelijke bron van fouten!

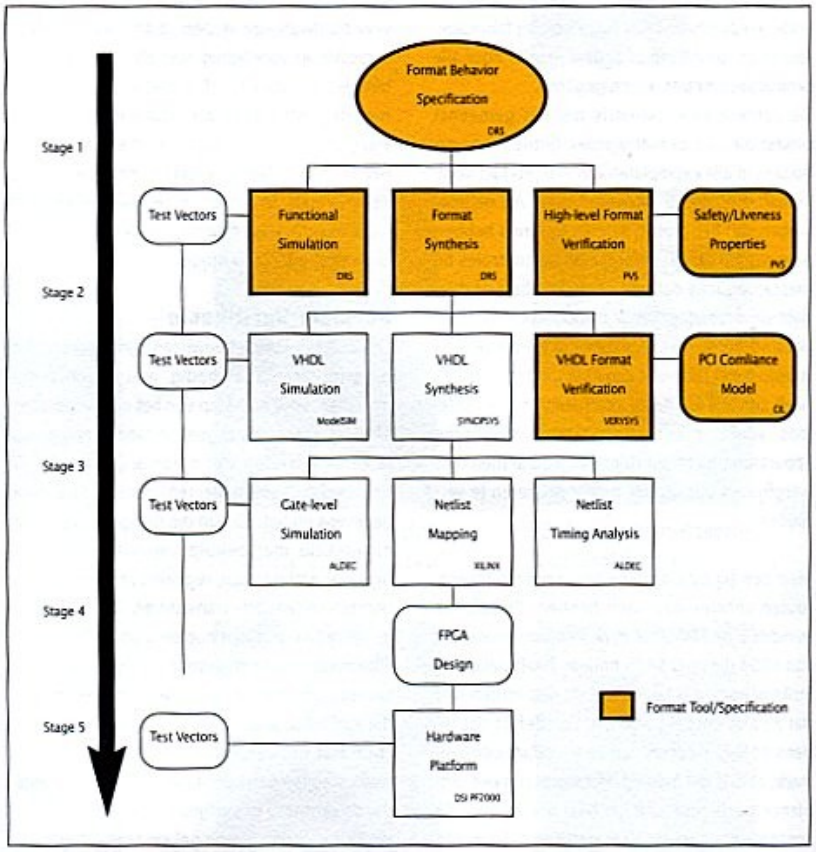
EDA ontwerp

Formele verificatie is dus altijd onderdeel van een EDA-ontwerptraject. Bose et al. (ref. 4) beschrijven zo'n geintegreerd ontwerp proces voor een synthetiseerbare VHDL 32 bit, 33 MHz PCI-interfacekern die gericht is op programmeerbare hardware. De PCI Local Bus is een bekende en welomschreven 32 of 64 bit bus met gemultiplexte adres- en datalijnen die perifere componenten verbindt met processor/geheugendeelsystemen. De onderzoeksgroep is, enigszins verschillend van Kropf, uitgegaan uit van vijf ontwerpstappen: specificatie ontwikkeling, formele synthese, VHDL validatie en synthese, netlijst validatie en afbeelding, en post-ontwerp validatie. Figuur

Tabel 1: Abstractieniveaus bij IC-ontwerp. Digitaal IC ontwerp bestrijkt verschillende aspecten en verschillende niveaus van abstractie. Het RTL (Register Transfer Level) gedrag wordt als FSM (Finite State Machine) gemodelleerd maar op poort niveau zijn Boole functies vereist. (Bron: ref. 1)

Niveau	Gedrag	Structuur	Data	Tijd
architectuur	algoritme	processen	getallen	oorzaak, volgorde
register overdracht	data stroom, FSM	register, ALU	bit vectoren	klok cyclussen
poort	Boole-functies	flipflops, poorten	bits	discrete vertragingstijden
transistor	differentiaal vergelijkingen	capaciteiten, transistoren	spanning, stroom	continue tijd
layout		oppervlak		

1 toont de fasen en de op elk bijbehorend niveau gebruikte tools. Gearceerde blokken verwijzen naar toepassing van formele methoden, de witte naar traditionele ontwerptools.
Uit de figuur is goed te zien hoe de gebruikte hulpmiddelen in onderlinge samenwerking moesten worden geïntegreerd. Zo werd een formele specificatie van het gedrag gemaakt met DRS (Derivational Reasoning System) van Derivation Systems. Hieruit werden in dezelfde omgeving een formele synthese, een functionele simulatie en PVS-theorieën gegenereerd.
Met dat Prototype Verification System van SRI werden bepaalde kritische eigenschappen (*liveness, safety*) geverifieerd. Voor verificatie op andere niveaus werden Verisys Property Prover, Structure Prover en de Verisys Circuit Interface Language (CIL) gebruikt. In deze CIL werd het PCI Compliance Model beschreven om de VHDL kern te valideren.
Het arsenaal aan tools werd gecompleteerd met ModelSim van Model Technologies en Xilinx Foundation Express (met daarin de Synopsys Express VHDL compiler en Aldec poort niveau analysator en simulator).



Ontwerp- en verificatiestappen van een 33 MHz PCI-interfacekern. Automatisch ontwerp en verificatie vereist een diversiteit van tools en een mix met traditionele hulpmiddelen. In dit ontwerpschema van een PCI-kern verwijzen de gearceerde vierkanten naar formele tools, de gearceerde ovalen naar formele specificaties. In de witte vakken staand traditionele ontwerptools. (bron: ref. 4)

Resultaten

Het project aan de FormalCore PCI/32 had als belangrijkste resultaat dat, als het ontwerp eenmaal op deze manier was geconstrueerd en gevalideerd, veranderingen in de specificatie veel beter waren door te voeren (en met meer vertrouwen in de correctheid van de uiteindelijke implementatie). Formele verificatie kan dus met name hergebruik van IP bevorderen.
De onderzoekers zeggen niets over eventuele compatibiliteitsproblemen tussen de vele tools van verschillende leveranciers. Wel melden zij dat de VHDL standaard IEEE 1076 -1993 ed. als basis diende en dat, waar van toepassing, de gebruikte tools daarmee conform waren.
Formele verificatie is inmiddels ook tot de industriële praktijk doorgedrongen. Op de recente DAC conferentie in Las Vegas (ref. 5) kwamen zowel Intel als Motorola met berichten daarover naar buiten.
De Pentium 4 is de eerste processor van Intel waarbij op grote schaal gebruik is gemaakt van formele verificatie. Daarbij zijn eigen hulpmiddelen toegepast maar Intel overweegt voor de toekomst ook aanschaf van commerciële producten. Overigens werd formele verificatie steeds ingezet ter aanvulling van de gebruikelijke validatiemethoden en nooit als vervanging daarvan. Met deze formele methode werden echter meer dan 100 'high-level logic bugs' gevonden die anders waarschijnlijk onopgemerkt waren gebleven, aldus Intel. Twee daarvan waren vergelijkbaar met de beroemde FDIV-fout in de Pentium van 1994 en hadden tot hetzelfde, voor het bedrijf rampzalige, gevolg kunnen leiden.
In de FADD-instructie van de Pentium 4 werd bij een bepaalde invoercombinatie een carry-out bit op 1 gezet terwijl er geen carry was. In de FMUL-instructie werd in afrondingsmodus 'round up' het 'sticky bit' onjuist gesteld bij sommige combinaties van mantissewaarden.
Bij Motorola is men overtuigd geraakt dat sommige verificatietools die uit de onderzoekswereld komen nu ook in de industriële praktijk kunnen worden toegepast. Het gaat dan vooral om abstractieniveaus boven RTL. Anders dan Intel meldde Motorola echter nog geen concrete resultaten. Formele verificatie lijkt dus een nuttige aanvulling maar het is niet de oplossing van alle validatieproblemen. Al kan men bij hardware-ontwerp dan thans misschien zeggen dat bewijzen weten is, zeker weten is dat nog niet. Kropf zelf wijst nadrukkelijk op een principiële beperking: de formele specificatie op het allerhoogste abstractieniveau is zelf niet formeel verifieerbaar. Of die ook echt uitdrukt wat de ontwerpers eigenlijk willen, dat blijft mensenwerk.

Referenties

1. Kropf, T., *'Introduction to Formal Hardware Verification'*, Springer, Berlijn (1999)
2. Thiel, J.M. van, *'Formeel Formuleren I: Mathematische Logica en Modellen'*, Elektronica 92/5, p.27 - 31 (1992)
3. Thiel, J.M. van, *'Formeel Formuleren II: Specificeren, predikatenlogica en betekenis'*, Elektronica 92/6, p. 21 - 25 (1992).
4. Bose, B., Tuna, E.M., Cyliax, I., 'FormalCORE PCI/32: *A Formally Verified VHDL Synthesizable PCI Core*'. Fifth NASA Langley Formal Methods Workshop (13 - 15 June 2000).
   Zie: http://shemesh.larc.nasa.gov/Lfm2000/Proc/
5. Nicolas Mokhoff, *'Intel, Motorola report formal verification gains'*, EE Times, 21 Juni 2001 www.eetimes.com