

#3 / 1999 NYHETSBRÄV FRÅN SILINOVA AB SILINOVA PROCESS

Inledning

I detta "julnummer" av Silinova Process kommer vi som vanligt med ett antal intressanta artiklar där vi ger vår expertsyn på trender och ny teknik.

Vårt syfte med detta är att vi vill visa att SiliNova är ett företag som hänger med i den senaste utvecklingen inom de områden där vi verkar, dvs ASIC/FPGA-utveckling, analog och digital elektronikkonstruktion samt realtidsprogrammering och objektorientering. Dessutom vill vi naturligtvis att våra kunder och samarbetspartners ska få ta del av våra kunskaper och förhoppningsvis ha nytta av dem i sitt arbete.

I det här numret finns artiklar som handlar om:

- Specman, verktyget som fokuserar på effektiv funktionell verifiering.
- Superlog - kanske framtidens nya systembeskrivande språk.
- Bättre och effektivare simulering utan X

- System C, framtida systemdesign från Motorola.

- SOI, som kan komma att ersätta CMOS i framtidens ASIC:ar.

Mycket nöje med artiklarna. Vi presenterar också två nya medarbetare med gedigen och vitsordad kompetens, Peter Friberg och Patrik Sandell.

Vi önskar alla en GOD JUL och ett GOTT NYTT ÅR.

Leif Ahlbom, VD

Silicon-On-Insulator

Av Tomas Jonsson

SOI är en ny geometri för transistorerna på chippen. Istället för att använda vanlig CMOS har transistorkroppen isolerats med ett extra oxidlager. Detta reducerar RDS_(ON) när transistoren är påslagen. Detta leder till en lägre effektförbrukning (20%). Det reducerar också de parasitiska kapacitanserna C_{SB}

System C

Av Tomas Jonsson

På ICCAD99 presenterade Motorola bl.a. en turtorial om framtida system-

design på SoC. Som exempel tog de en konstruktion för trådlös kommunikation. Konstruktionen är svår pga att man måste integrera RF, mixed signal och digitala system på samma chip. Trenden är att mer och mer av filtreringen görs i signalbehandlingsalgoritmer.

Konstruktionen av det digitala systemet inkluderar också, förutom HW, att mappa många SW processer på ett antal DSP:er, processorer och ASIC:ar.

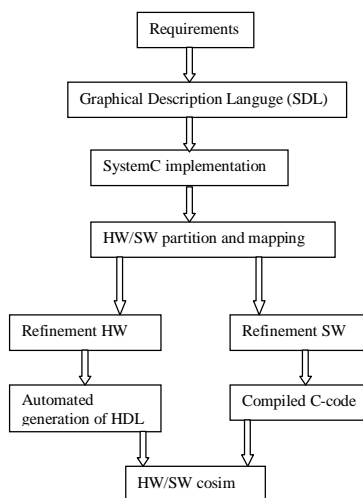
För att förenkla konstruktionsarbetet vill Motorola införa ett språk (SystemC) som beskriver HW och SW. Under

systemeringsarbetets tidiga faser kan då hela systemet simuleras utan att uppdelning mellan HW och SW gjorts. Detta leder också i sig själv till exekverbara specifikationer. Under

utvecklingsarbetets senare delar görs sedan en uppdelning av HW och SW, samt en förfining av systemkoden.

Det språk Motorola förespråkar är C++ med ett antal klassbibliotek. Dessa bibliotek stödjer de asynkrona och synkrona händelser som finns i HDL. Anledningen till att de valt C++ är att språket redan är en de facto standard och att det finns gott om utvecklingshjälp-

medel för det språket. Dessutom går C++ relativt snabbt att exekvera jämfört med HDL-beskrivna språk.



Föreslaget framtida konstruktionsflöde från Motorola.

och C_{DB}, vilket gör det enklare att konstruera med transistorerna. En tredje fördel är den högre packningstätheten (20-30%).

Det är inte bara fördelar med SOI. Verktygen måste anpassas till det. Många jämför bytet till SOI med bytet från aluminium till kopparledare på kisel, eller bytet från NMOS till CMOS.

Nackdelen med den flytande transistorkroppen är en högre variabilitet i processen. Detta leder till en ökad osäkerhet. En fördröjning t.ex. genom nät av SOI är olika beroende på

historiken av de tidigare händelserna i nätet. Detta till följd av att transistorkroppen flyter.

Enligt Sani R. Nassit och Tuyen V. Nguyen från IBM Austin Research Lab på ICCAD99 skall dessa effekter vara överkomliga. Osäkerheten kommer att öka med ca 10%. Det är överkomligt med de fördelar SOI ger. Vi kommer med andra ord att inom en snar framtid se nya processer med SOI teknik istället för CMOS i ASIC:ar.

Philips erbjuder redan idag en dubbelmetallprocess för tillämpningar mellan 12V och 60V. Det är bara början.

Effektiv funktionell verifiering

Av Peter Friberg

Funktionell verifiering av ASIC är idag en besvärlig och tidsödande process som ofta tar mer än halva konstruktionstiden och resurserna i anspråk i ett ASIC projekt.

De buggar som upptäcks efter prototyp-tillverkning beror sällan på att den funktionella verifieringsplanen är dålig eller att den inte har följts fullständigt. Felen beror snarare på oklarheter och missuppfattningar av konstruktions- eller gränssnitts-specifikationerna. Ett annat vanligt fel är oförutsedda användningsfall.

Ett annat problem är hur man ska mäta verifieringen med hjälp av metrics.

Det finns diverse verktyg på marknaden som kan mäta code coverage, state och branch coverage mm. Dessa verktyg är bra hjälpmedel för att få fram bättre stimuli och verifiering. Problemet är dock att stimulit mer tenderar att täcka t.ex. 100% av den kod som redan är implementerad, inte att täcka den funktionalitet som avses. Trots att man har 100% täckning, så finns det ingen garanti för att själva funktionen är korrekt. Det räcker således inte med att bara mäta täckning av toggle och kod, utan man måste också fokusera på att mäta användning av funktionerna under olika förhållanden.

Ibland kalkyleras i tidsplanen med att man måste göra flera kiselväändor innan alla buggar är borta. Detta innebär en stor extra kostnad i både tid, pengar och resurser. Idag är detta en lyx som få kan unna sig, ASICen måste helt enkelt fungera på första försöket.

Hittills har konstruktörer fått tillgång till allt kraftfullare verktyg för ASIC/FPGA framtagning, inom t.ex. simulering, syntes, layout osv. Ett område där väldigt lite har hänt är metoder och verktyg för verifiering.

Det finns ett antal anledningar till att verifieringen mer och mer blir den största flaskhalsen:

-Marknaden kräver att produkterna utvecklas snabbare med högre kvalitet.

-Komplexiteten på verifieringen växer exponentiellt med komplexiteten hos

konstruktionen.

-Det går inte att hyra in och träna upp verifieringspersonal på hur kort tid som helst.

-Kostnaden för flera kiselväändor är oacceptabelt dyrt och tidsödande.

-Det är svårt att veta närverifieringen är klar.

-Integrering av IP samt SOC ställer nya och högre krav på systemverifieringen.

En del av lösningen är att extrahera reglerna i konstruktionspecifikationer i en exekverbar form. Ny verifieringsmetodik måste till för att hitta buggarna som man inte tänkt på.

Med Specman Elite från Verisity Design, Inc. kan man som konstruktör nu tänka mer i termer av specifikationer i stället för register, signaler, bussar, paket och instruktioner. Konstruktören beskriver sina funktionella tester i ett mycket kraftfullt språk som heter "e". Språket är konstruerat på ett objektorienterat sätt vilket gör det modulärt, flexibelt, lättförståeligt och mycket kraftfullt. Man har plockat alla godbitar från andra språk, samt lagt till mängder med nya finesser som gör det möjligt att beskriva komplexa konstruktionsverifieringar på ett enkelt sätt.

Specman fokuserar på funktionell verifiering, dvs svarar på frågan "fungerar konstruktionen i sin omgivning?". I Specman görs detta genom simulering av konstruktionen.

Funktionell verifiering är den process som kontrollerar att en design fungerar enligt dess specifikation. Specman fokuserar verifieringen på systemnivå, men det går naturligtvis också bra att använda verktyget på blocknivå.

Vad är Specman?

Ett integrerat system för:

-Testgenerering (generering av data och stimuli)

-Checker och prediktering

-Funktionell täckningsanalys

-Referensmodeller

Det riktiga mervärdet med Specman är den automatiska stimuligeneratorn. Den genererar automatiskt slumpade funktionella test, samt tillhandahåller kraftfulla möjligheter att skapa och kontrollera data. Dessutom analyseras kontinuerligt den funktionella täckningen för att mäta fortskridandet i verifieringen gentemot verifieringsplanen.

"Constraint-driven" testgenerering är

en av Specmans kraftfulla funktioner. Det betyder i praktiken att konstruktören kan dirigera testgeneratorn att automatiskt generera testmönster som styr verifieringen till intressanta områden, i stället för att skapa testmönster för hand vilket är mycket tidsödande.

Testgeneratorns dirigeringsfunktion har två syften. Det ena är att specificera tillåtna värdemängder på framslumpat testdata och det andra är att dirigera verifieringen till att generera intressanta scenarier. Dirigeringen har full tillgång till interna signaler i konstruktionen (DUT), vilket möjliggör automatisk generering av testfall som styr konstruktionen till de funktionella extremfallen "corner-cases". Denna generering sker helt dynamiskt vartefter verifieringen fortskrider.

Några fall av dirigering skulle kunna vara att 1) generera extra många avbrott när exempelvis ett FIFO börjar bli fullt, 2) generera instruktioner som nyttjar carryflaggan då carryflaggan är satt eller 3) mer generellt ändra fördelningen av värden på slumpat data då ett visst villkor uppfylls.

Tack vare tillgången till alla signaler i konstruktionen kan konstruktören skräddarsy testerna till att vara av typen "black", "gray" eller "whitebox" verifiering.

En annan möjlighet är att använda funktionen "Temporal checking" för datakontrollverifiering av timing. Kontrollen utförs under hela verifieringen, varför granskning verifieringsloggen inte är nödvändig. Kontrollen minskar också risken för att fel inte upptäcks.

Det går även att skapa kontroll av protokoll, dvs sekvenser av data och händelser. Generering av testdata och kontroll av resultat kan således vävas ihop till att exempelvis skicka data, vänta på någon händelse, kontrollera data och sedan skicka nytt data etc. Detta medför att man hela tiden vet exakt vad som skickas och vad det förväntade resultatet ska vara.

Med hjälp av den inbyggda täckningsanalysen kan man skräddarsy hur täckningen över exekveringen ska mätas. Den funktionella täckningsanalysen kan också samla data om exempelvis tillståndsmaskiner (både tillstånd och övergångar), data samt händelser som sträcker sig över flera cykler. Med hjälp av denna täckningsanalys kan man lätt styra genereringen av testdata så att 100% täckning kan uppnås.

Med funktionell täckning kan man

exekvera testplanen och visualisera framåtskridandet. En fördel med funktionell täckning är att det är lättare att förutsäga hur lång tid verifieringsplanen kommer ta att exekvera dvs förutsäga när verifieringen är klar.

Specman går att koppla ihop med de flesta VHDL och Verilog-simulatorerna på marknaden och därigenom få 100% kontroll och insyn i konstruktionen vid verifieringen.

Sammanfattningsvis kan sägas att ett formellt verktyg med funktionell verifiering som bygger på specifikationer idag gör det möjligt att snabbt och effektivt hitta buggar i ett tidigt skede och därför dramatiskt minskar tiden för verifieringen.

Superlog på DAC-99

Av Michael Greting

Årets största överraskning bjöd start-up företaget Co-Design på. I en liten och undangömd monter träffade vi på Simon Davidmann, som vi träffat på en tidigare DAC, se artikel i SiliNova Process 2-1997.

Alla är överens om utmaningarna när det gäller SOC- System On a Chip. Hela system med hårdvara, mjukvara, processorer, minnen etc. ska klämmas in på en enda kiselbricka - och det hela ska dessutom fungera!

Nuvarande lösningar och förslag som inkluderar utökningar av C/C++, Java, nya verktyg och språk för funktionell verifiering (Specman/E och Vera) samt systembeskrivningsspråk, löser egentligen inte hela SOC-problematiken. C och Java med utökningar för att bl.a. hantera tid och parallelism) tillåter t.ex. inte att delar som redan är skrivna i VHDL/Verilog återanvänds. Det kan också vara svårt att använda IP-block från tredje-parts leverantörer. Specman och Vera i sin tur inriktar sig enbart på den funk-

tionella verifieringen och inte på modelleringen av designen.

Nytt för i år på DACen var också att det mycket riktigt finns många företag som försöker lösa problemen med SOC. Det fanns en uppsjö av olika verktyg med utökningar av C, C++ och Java, verktyg som konverterar mellan RTL och C och ett primitivt försök till syntes från C-kod. Alla dessa "verktyg" lider av ett stort problem, nämligen att de inte tar något helhetsgrepp. Oftast riktar de bara in sig på någon liten del av problematiken. Utökningarna är inte standard och dessutom passar verktygen inte in i den nuvarande metodiken och designflödet. Om man t.ex. beskriver sitt system i C (med utökningar), så kan man inte få ut Verilog eller VHDL kod, så på så sätt finns det inget enkelt sätt att syntetisera en sådan implementation.

Davidmann presenterade Co-Designs förslag på lösning: Superlog, ett nytt språk som plockar godbitarna ur C, Verilog och test/systembeskrivningsspråk. Med Superlog försöker man angripa SOC-problematiken på ett helhetssätt. Till sin hjälp har man bl.a. haft Peter Flake, som tog fram Hilo-simulatorens, och Phil Moorby, som skapade Verilog. Co-Design jobbar för fullt och planerar att presentera sina verktyg för SOC och Superlog någon gång i början av nästa år.

Davidmann använder begreppet USE (Unify, Speed up, Evolution) för att illustrera vad som krävs för att lösa SOC-problemet:

- * Unify, förena hela SOC designflödet.
- * Speed up, snabba up design processen.
- * Evolution, tillåter migration från gammal metodik till ny.



Christian Burrish och Simon Davidmann, Co-design.

Superlog skapades för att förena de olika delarna av designflödet genom att tillhandahålla ett gemensamt språk för arkitektur/systemnivå-design, hårdvara/mjukvara co-design, implementering och funktionell verifiering. Genom att förena alla dessa områden vill man eliminera behovet av komplexa designflöden med många olika verktyg och lång-

samma krångliga kopplingar mellan dem. Målsättningen är också att minska klyftorna mellan olika design-team (systemarkitektur, hårdvara och mjukvara) och istället få dem att samarbeta på en högre nivå. Superlog är ett front-end verktyg i SOC-designflödet och det är inte meningen att dagens backend-flöde ska ersättas. Från kommande verktyg kommer man att få ut syntetiserbar VHDL/Verilog kod som man kan gå vidare med till simulering och syntes.

Viktigt är också planerna på att göra Superlog public domain, så att andra företag kan utveckla verktyg för Superlog. Däremot vill Co-Design själva behålla kontrollen över språket och inte överlåta specifikeringen/utvecklingen av Superlog till något standardiseringsorgan. Istället är målet att Superlog ska bli en ny industristandard.

Länk: www.co-design.com

Bättre simulering utan X

Av Tobias Roos

Slösa inte bort tid på att simulera med X i RTL-simuleringar. Hantering av X i RTL-kod är sällan konsekvent och kan ge både för pessimistiska och optimistiska resultat. Detta hävdades av konstruktörer från Hewlett Packard i en presentation under DAC-99.

Tidigare artiklar om verifiering i SiliNova Process har avhandlat verifiering på relativt hög nivå. Nu är det dags att dyka på djupet och bli lite mer jordnära, hur jordnära det nu är med X i simuleringar.

Hur många timmar har inte slösats bort på att eliminera X i RTL-simuleringar? X är det värde som samtliga register sätts till vid tidpunkten noll i en simulering. X har ingen motsvarighet i det verkliga systemet där enbart 1:or och 0:or förekommer.

Det är framför allt i uppstartssekvenser som problem med X förekommer. En del hävdar att simulering med X visar att man tagit hand om eventuella initieringsproblem på ett riktigt sätt. Analyserar man detta mer noggrant visar det sig ofta att så inte är fallet.

I konstruktioner med en global reset, som går till samtliga register och minnen, är detta med X inte ett problem - de försvinner ju nämligen direkt

via reset. Tyvärr fungerar inte alla konstruktioner på detta sett. Det är inte alltid man har råd med en global reset då det bl.a. kostar i form av ledningsdragning i den fysiska implementeringen. I vissa system existerar det inte heller en global reset som man kan förlita sig på. Ibland förekommer olika resetsignaler vilka inte kan vara aktiva samtidigt etc.

Vid simulering av RTL-koden för sådana system hanteras X sällan konsekvent. För en adderare räcker det med att en bit är X för att varje bit i summan antar värdet X. Man får sk X-spridning. I andra fall sker sk X-eliminering, t.ex. via default tilldelningar i if-else och case satser. I if-else satser hanteras X av else satsen, och i case satsen hanteras X av default satsen. Detta kan leda till att ett X på en signal hanteras som en 0:a i ett block och som en 1:a i ett annat. Se följande två exempel där signalen A initialt är 2'b0X:

```
if(A==2'b11)
  OUT = 4'b0000;
else
  OUT = 4'b1010;

case(A)
2'b01: OUT = 4'b1010;
2'b10: OUT = 4'b1010;
2'b11: OUT = 4'b1010;
default: OUT= 4'b0000;
endcase
```

Alla förstår att detta inte är en bra modell av det verkliga systemet!

Istället för att simulera med X rekommenderas att enbart simulera med värdena 0 och 1, sk 2-state simulering, vilket de flesta cykelbaserade simulatorer stödjer. Uppstartsproblem hanteras via slumpgenerering av initialvärden för samtliga register, där slumpgenerering av initialvärden har implementeras via PLI-rutiner. Förutom en bättre hantering av uppstarts-

problem får man en snabbare simulering på köpet, då enbart två värden för varje register/signal behöver hanteras av simulatören.

Den konstruktion som gruppen från Hewlett Packard arbetade med bestod av 5 st. ASICar på ca. 500k grindar per ASIC. Via PLI-rutinerna initierades varje register slumpvis till 1 eller 0 beroende på vilket frö som angavs till slumpgeneratorn. Normalt angavs systemklockan som frö till slumpgenereringen under den 6 månader långa verifieringsperioden. Detta gjorde att varje testfall, för varje gång det kördes, startade med ett unikt initialvärde. Under 6 månaders verifiering hade gruppen startat ca. 1/2 miljon testfall, mao hade man därigenom verifierat 1/2 miljon olika uppstartstillstånd av totalt $2^{200,000}$ möjliga. Ingen av de kretsar som tillverkades hade några uppstartsproblem, vilket hade varit fallet för kretsar konstruerade tidigare med traditionell hantering av uppstartsproblem.

Patrik Sandell



Patrik Sandell har tillhört SiliNova i juni. Han tog examen från Data-tekniklinjen i Linköping, med inriktning på elektronik 1991. Sedan dess har Patrik jobbat uteslutande med ASIC-nära saker. Patrik har deltagit i ett ASIC-projekts samtliga faser och har därför en mycket bred kunskapsbas att stå på. De senaste två åren har Patrik tillbringat som applikationsingenjör hos en ASIC leverantör.

De områden inom ASIC som Patrik har jobbat mest med är design, implementering, testvektorgenerering och tidsanalys. Om behovet skulle finnas kan han även tänka sig att skriva lite programvara också.

Patrik är sambo med Britt-Marie och gillar att njuta av livets goda utanför kontorstid. Ett utdrag av det han finner njutbart är natur, fiske, skidåkning, bilar, resor m.m.

Peter Friberg



Peter har 10 års yrkeserfarenhet inom design av inbyggda realtidssystem.

Han har deltagit i samtliga faser i olika utvecklingsprojekt.

Peters starkaste sidor är gränslandet mellan elektronik och programvara.

Han konstruerar både elektronik och programvara med tonvikt på det senare.

Närmast kommer Peter från ett annat stort konsultbolag där han främst arbetat med hårdvarunära realtidssystemprogrammering av signalprocessorer i assembler.

Bland Peters intressen utanför kontorstid finner man saker som:

windsurfing, dykning, privatflygning, segling, skidåkning,

...listan kan göras lång. Nappar på det mesta som är kul.

Peter är sambo med Sofie och bor i en lägenhet i Bergshamra i Solna.

SiliNova Process

Redaktör: Thomas Magnusson
Ansvarig utgivare: Thomas Magnusson
Layout: Jens Aronsson
Adress:
SiliNova AB
Ekbacksvägen 28
SE - 168 69 Bromma
08 - 555 36 270 (telefon)
08 - 555 36 108 (fax)
silinova@silinova.se
<http://www.silinova.se>

Copyright © 1999 SiliNova AB

SiliNova är ett oberoende konsult företag med fokus på ASIC.

Med över 70 års samlad erfarenhet av ASIC och FPGA är SiliNova ett av de ledande företagen inom branschen. Realtidssystem och objektorienterad programmering är andra områden inom vilka SiliNova har bred kunskap. Inom dessa områden erbjuder SiliNova tjänster för utveckling av kompletta system, antingen som uppdrag eller i samarbete med kunden.

SiliNova flyttar kontinuerligt fram sin position genom individuella utbildningsprogram och selektiv rekrytering.

Med vår erfarenhet i kombination med utbildning har vi som mål att samtliga uppdrag skall utföras effektivt, med hög kvalitet, till kundens fulla belåtenhet.