

Automatiserat test av SoC med System Verilog

Metoderna att verifiera systemkomponenter förbättras snabbt och System Verilog innebär ett stort steg framåt. Bindesh Patel och Rex Chen från SpringSoft, samt J. L. Gray från Verilab beskriver här hur testautomatiseringen i System Verilog kan lyftas till en högre abstraktionsnivå och göras mera strukturerad.

Att verifiera dagens förfinade chip och system är smärtsamt. Av konstruktionscykeln utgör verifieringen den största tiden. Därför tvingas konstruktörsteam över hela världen att söka efter kreativa sätt att förkorta verifieringstiden. Många nya metoder har tagits fram för att lösa detta problem.

SystemVerilog med tillhörande

de bibliotek och verifieringsmetoder är ett exempel på en möjlig lösning. Eftersom lösningen arbetar med en skiktbaserad ansats för generering av testbäddar och stimuli, använder den testbäddar uppbyggda kring transaktioner och sekvenser av transaktioner.

Därför ger lösningen flexibla och återanvändbara testbäddar,

och det är enklare att generera förfinade och realistiska scenarier av stimuli. Eftersom dessa scenarier ger en mer genomträngande exercis av konstruktionerna ger de också högre täckningsgrad, vilket ytterligare lättar verifieringsbördan.



fortsättning på sid 27.

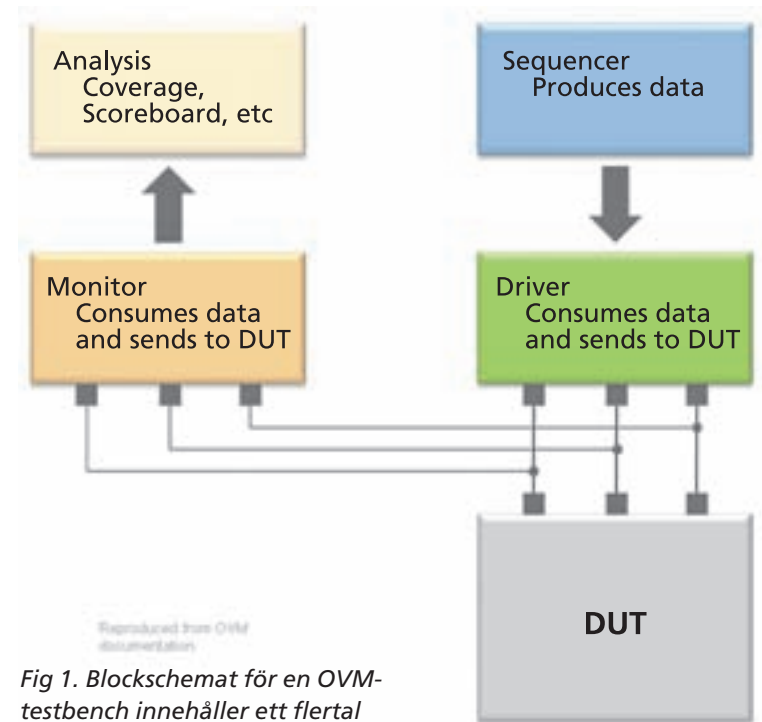


Fig 1. Blockschemat för en OVM-testbench innehåller ett flertal baskomponenter.

Visserligen utgör denna ansats ett betydande steg framåt vad gäller automatisering av testbäddar. Men det är en svår och tröttsam utmaning att avbugga och analysera de abstrakta datatyper som finns i dessa testbäddar. De flesta ingenjörer tvingas gå tillbaka till att skriva ut meddelanden till textfiler för att kunna hitta problemen. Därför krävs det nya avbuggningsmöjligheter för att automatisera denna mödosamma avbuggningsprocess i den miljö som en typisk skiktad lösning för stimuli utgör.

FÖRSTÅ VERIFIERING PÅ TRANSAKTIONSIVÅ

För att tydligare kunna förstå behovet av förbättrad testbäddavbuggning, dataregistrering och virtualisering måste vi först se närmare på en typisk lösning med skiktade stimuli och skaffa oss en bättre förståelse för generering av stimuli i miljöerna Open Verification Methodology (OVM) och Verification Methodology Manual (VMM).

Fig 1 visar baskomponenterna i en OVM-baserad miljö för verifiering på transaktionsnivå. VMM-baserade lösningar är av samma natur. Här ingår:

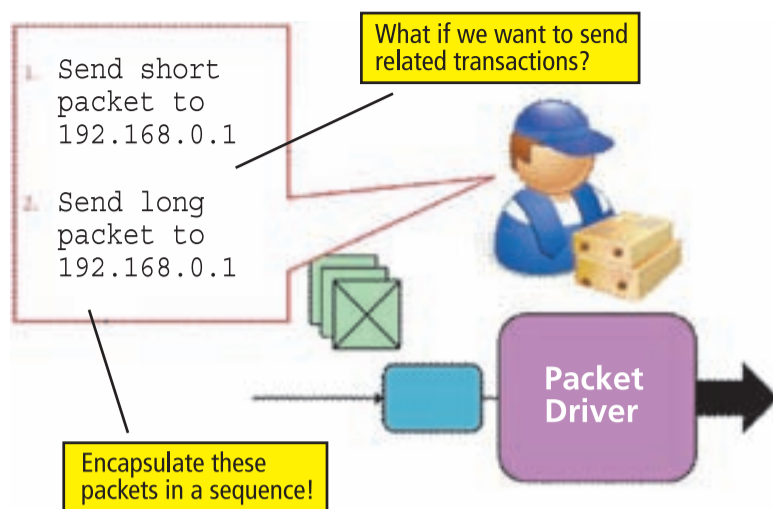
- En stimulusgenerator (sekvenserare) som skapar trafik på transaktionsnivå till testobjektet (DUT).
- En drivenhet som omvandlar transaktioner till stimuli på signalnivå på DUT-gränssnittet.
- En monitor som kan känna igen aktiviteter på signalnivå på DUT-gränssnittet och omvandla dessa till transaktioner.
- En analyskomponent, t ex en "täckningsinsamlare" (coverage collector) eller en "resultattavla" (scoreboard), för analys av transaktionerna.

Låt oss nu se närmare på varje komponent, och vi börjar med stimulusgeneratoren. Innan "constrained random testing" (framtingat, slumpmässigt test) hade utvecklats, krävde genereringen av stimuli att ingenjören skrev riktade test för att lirka med pinnarna på testobjektet eller för att fylla ett minne med bytes av maskinkod.

Genom att tänka på en högre abstraktionsnivå fick man en lämpligare ansats till generering av stimuli. Istället för att skriva riktade test som lirade med pinnar, kunde testkonstruktören istället helt enkelt exekvera en transaktion med registerläsning eller -skrivning, som innehöll all pinnlirkning som krävs för att exekvera kommandot. Härigenom avlastades testförfattaren den långgrandiga mödan med alla detaljer på låg nivå som förknippas med olika operationer.

SLIPPER BETEENDET

Transaktioner besparade ingenjören mödan att behöva ta sig an det underliggande testobjektets beteende. Men de öppnade också



upp för en annan utveckling: att skapa en samling av sekvenser, liksom den underliggande sekvenseraren, för att kapsla in viktiga fall/beteenden av systemanvändning som slumpmässigt kunde väljas under en simulering.

Sekvenser definierar meningsfulla strömmar av meningsfulla transaktioner. De används för att modellera samlingar av transaktioner på högre nivå, t ex att sän-

da en viss ström av registerkommandon om läsning eller skrivning för att programmera ett testobjekt eller öppna en TCP/IP-förbindelse med hjälp av en ström av Ethernet-paket. Ett exempel på en sekvens visas i fig 2.

Samlingar av sekvenser hanteras med hjälp av en testbäddkomponent. I en OVM-miljö kallas denna komponent för en sekvenserare, och den har flera

Fig 2. I detta exempel på en sekvens testar ingenjören en nätverkskomponent i ett scenario, där en användare behöver öppna en TCP/IP-förbindelse.

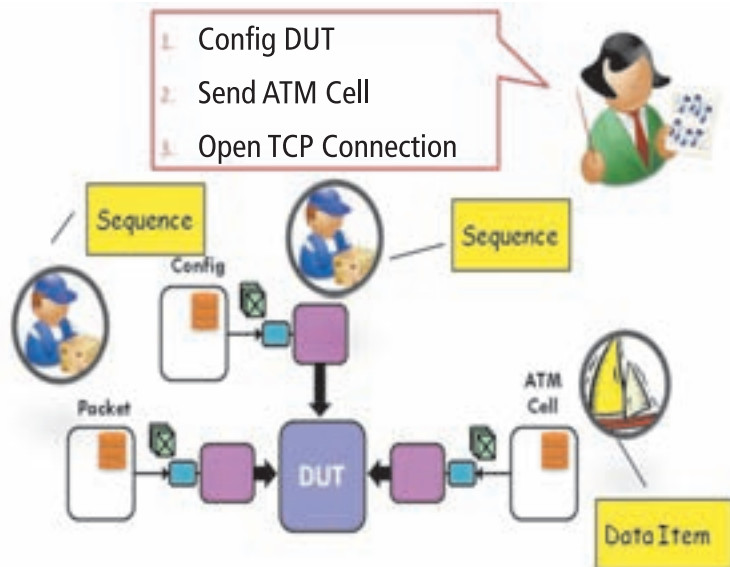


Fig 3. I denna instans har ingenjören konfigurerat configurationen, ATM och paketsekvenserna tillsammans.

funktioner inom en OVM-testbädd. Dessa funktioner är:

- Hantera den samling sekvenser som är relevant för den specifika typen och instansen av sekvenserare. Individuella sekvenserare kan anpassas av testskrivare och miljöintegratorer för att uppträda på sådant sätt som den aktuella applikationen kräver.
- Hantera interaktionen mellan sekvenser och den underliggande drivenheten. Sekvenserare ger också en möjlighet att styra vilka sekvenser som skall väljas. Både VMM och OVM gör det möjligt att skapa en uppsättning av viktade begränsningar (constraints) som kan användas i denna process.
- Fungera som ett ankare för

att hålla den anpassade uppsättningen av sekvenser och underliggande konfigurationsparametrar för sekvenser i sekvensbiblioteket.

- Fungera som en resurs i OVM som kan nås av andra testbäddkomponenter (sekvenserare eller virtuella sekvenser).
- Fungera som en läsbar resurs i VÅM, även om scenariegeneratorn har ansvaret för att underhålla ett register av scenarier som kan köras i scenariegeneratorns kontext.

Virtuella sekvenserare ger ingenjören möjlighet att koordinera vad som sker på flera sekvenserare tillsammans (se fig 3). Virtuella sekvenser som exekveras av den virtuella sekvenseraren klarar

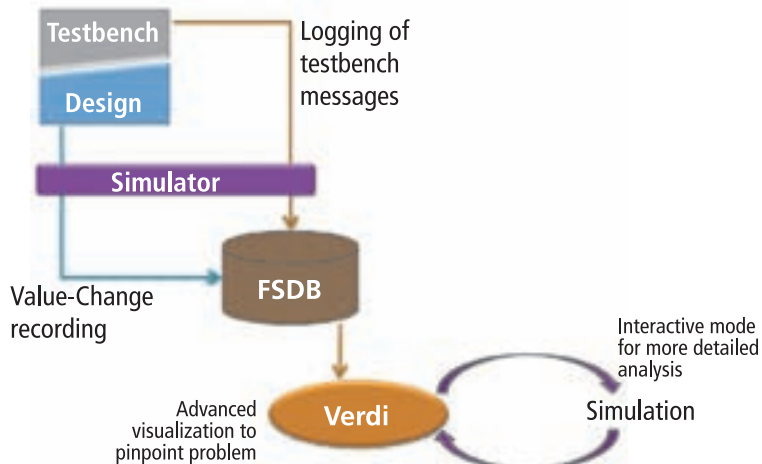


Fig 4. Ett flöde bestående av avancerad loggning, sammankopplad med interaktiv inspektion.

att exekvera sekvenser från valfri annan sekvenserare eller virtuell sekvenserare i verifieringsmiljön. De ger användaren en noggrann kontroll över testbäddaktiviteter och möjliggör koordinering mellan ett flertal olika gränssnitt.

Visserligen är testbäddstimuli en viktig del av varje modern verifieringsmiljö, men det är också viktigt att ha en möjlighet att observera beteendet hos en testbädd. Vanligen sker observationerna genom att man skapar en komponent kallad en monitor, som övervakar testobjektets gränssnitt.

När monitorer observerar transaktioner överför de denna information vidare till andra testbäddkomponenter med hjälp av analysportar i OVM-miljöer eller "callbacks" i VMM-miljöer. Informationen kan sedan användas som indata till "scoreboards", "checkers" och "coverage collectors".

Oberoende av det underliggande biblioteket är de data, som är mest intressanta och användbara för ingenjören, transaktionerna mellan sekvenseraren och drivenheten samt mellan monitorn och analysatorer. I idealfallet skall denna trafik spelas in i ett format som kan användas för analys och avbuggning efter simuleringen. En transaktion är en mycket mer praktisk inkapsling på högnivå av data för OVM- och VMM-baserade miljöer, och därför är avbuggningsvisualisering och analys på transaktionsnivå helt klart något önskvärt.

TILLKORTAKOMMANDEN

Idag finns det två huvudstrategier som ingenjörer använder för att bättre kunna förstå, analysera och avbugga testbäddmiljöer i SystemVerilog. Tyvärr har båda egenskaper som gör dem långt från idealiska. Dessa strategier är:

- Att använda standardbegreppet \$display i SystemVerilog för att logga användarspecifik information som spelas in i textfiler. Ingenjörerna kan använda \$display, eller klasser på högre nivå byggda på \$display, för att spela in sekvenser av transaktioner till loggfiler. Dessa kan sedan analy-

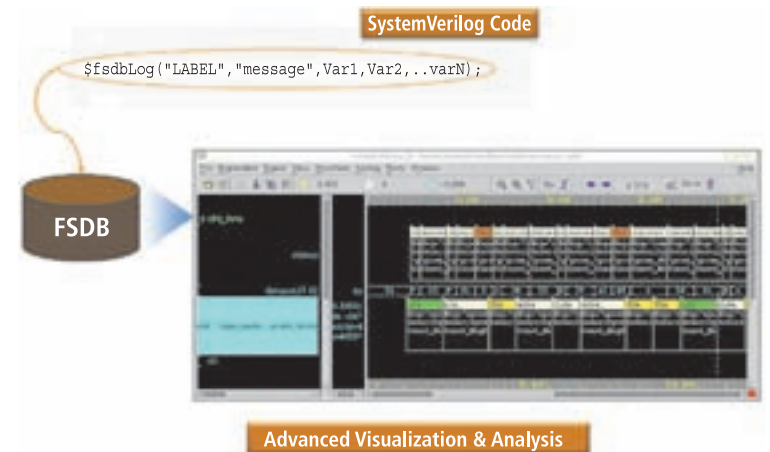


Fig 5. Här visas det föreslagna flödet och användandet av avancerad loggning. I denna bild används avancerad loggning till FSDB-databaser, anropade av SpringSoft-systemen Verdi Automated Debug och Siloti Visibility Automation, för att möjliggöra avancerad visualisering och analys.

seras efter simuleringen. Det största problemet med denna ansats är att dessa textbaserade lågnivå-loggfiler manuellt – och ofta mödosamt – måste korreleras till vägformer som visar vad konstruktionen gör på tidsaxeln.

- Att använda simulatorns interaktiva möjligheter, ungefär som man använder en GDB-avbuggare. Detta ger ingenjören möjlighet såväl att sätta brytpunkter som att inspektera variabelvärden och stackspår vid en viss tidpunkt. Men det finns många nackdelar med denna ansats. Ingenjören måste veta när och var brytpunkterna skall sättas, så att simulatoren stannar på rätt plats för ytterligare probning. Dessutom kanske simulatoren behöver köras i timmar, eller rent utav i dagar, för att komma till brytpunkten.

För att effektivt kunna uppfylla kraven och utmaningarna i att förstå, analysera och avbugga testbäddmiljöer i SystemVerilog krävs en annan ansats. Att bara utöka de traditionella metoderna för hårdvaruavbuggning till testbäddavbuggning är varken tillräckligt eller genomförbart.

EN MER

STRUKTURERAD ANSATS

En ny ansats för testbäddavbuggning bygger på befintlig loggning och interaktiva mekanismer som

ger ingenjören en insikt i vad som försiggår i en testbädd under simuleringen (se fig 4). Detta gör loggningsprocessen mycket mer förfinad och automatiserad, så att merparten av avbuggningen och analysen av testbäddaktiviteter kan göras på denna nivå.

Ansatsen – avancerad loggning – kan användas för att direkt hitta problem. Och om det visat sig att ett problem finns på testbäddsidan, och mer detaljerad information krävs, kan den också driva den interaktiva inspektionen.

Att avbugga konstruktion och testbädd tillsammans kan vara en både praktisk och effektiv process, men den kräver att loggningsmekanismen är flexibel att använda. Och – som visas i fig 5 – de erhållna resultaten måste automatiskt lagras i samma avbuggningsdatabas som konstruktionsresultaten (t ex i defacto-standardformatet FSDB). Detta har avgörande betydelse för att skapa avancerade funktioner för visualiserings, avbuggning och analys.

Med ett enhetligt system, drivet av avancerad loggning, kan ingenjören observera vad som pågår i hela miljön. Systemet samlar in och visar upp för ingenjören:

- Meddelanden på transaktionsnivå
- "Severities", variabeltillstånd o dyl – t ex tillhörigheter eller att-

ribut till meddelandet

- Anropsstacken, som senare kan utnyttjas för ytterligare automatisering av avbuggningen

Det går att till dessa vyer lägga till funktioner för speciella ändamål, som hjälper ingenjören att lätt upptäcka intressanta meddelanden från loggade data. Avancerad filtrering och markering kan användas för att filtrera eller färgmärka specifika meddelanden baserat på något visst villkor (t ex rödmärkning av alla meddelanden som har "WRITE" i etiketten och "address=5"). Applikationer som visar loggade meddelanden kan också ge ingenjören möjlighet att snabbt söka efter och hitta meddelanden som matchar användarspecificerade sökkriterier.

PÅGÅENDE INNOVATION

Avancerad loggning, kopplad till interaktiv inspektion, ger en klar fördel över andra idag använda ansatser för att förstå, analysera och avbugga testbäddmiljöer i SystemVerilog. Men ingenjören behöver fortfarande undersöka koden för att logga intressant information. OVM- och VMM-bibliotek kan ge en viss grad av automatisering av denna uppgift, och på så sätt avlasta ingenjören avsevärt arbete. Istället för loggning utförd av användaren, kan man istället utnyttja en metod kallad automatisk sekvensinspelning (automatic sequence recording).

Idén bygger på att man utnyttjar strukturen och den transaktionsbaserade naturen hos OVM- och VMM-bibliotek, och dessutom den objektorienterade naturen hos SystemVerilog. Polymorfism kan användas för att ge både automatisering och flexibilitet, så att användaren kan logga de data som flyter mellan testbäddkomponenter.

Pågående forskning är nu inriktad på att i förväg bädla in relevant loggningsinformation i de standardfunktioner och makron som används för att överföra data mellan olika skikt i en testbädd. Detta skulle ge möjlighet att automatiskt spela in alla data på transaktionsnivå, som skickas mellan de olika testbäddskikten. Dessa kan sedan analyseras i traditionella vyer som vågformer. Eftersom dessa data är så kompletta och lättillgängliga, kan de även potentiellt driva fram nya vyer baserade på UML-liknande sekvensdiagram.

FÖRFINING

SystemVerilog, liksom de verifieringsmetodiker som uppstått i dess närhet, har möjliggjort ett betydande steg framåt inom testbäddautomatisering och öppnat upp för generering av förfinade stimuliscenarier. Dessa scenarier exacerar konstruktionerna mer genomgående, vilket leder till högre täckningsgrad.

Det är tydligt att denna grad av förfining och automation kräver ett motsvarande steg för avbuggningsmöjligheterna i sådana miljöer. Avancerad loggning och interaktiv inspektion är idealiska metoder för att spela in transaktioner från en testbädd, och senare möjliggöra visualisering och analys av dessa transaktioner.

Automatisering av denna inspelning av alla intressanta transaktioner kommer att ytterligare förenkla processen genom att användarens arbetsinsats reduceras. Tillsammans kommer dessa

framväxande metoder att lätta den verifieringsbörda som vilar på de ingenjörer, som har till uppgift att utveckla dagens förfinade chip och system. ■ ■ ■

*Bindesh Patel och Rex Chen,
SpringSoft,
samt J. L. Gray, Verilab*

Om författarna:

Bindesh Patel är Technical Marketing Manager för Verification Group på SpringSoft i USA, där han ansvarar för att definiera framtida verifieringsprodukter. Han har tidigare arbetat med Design and Applications Engineering på LSI Logic, Zycad och Atrenta. Han har en examen i Computer Engineering från University of California, Santa Cruz. ■

Chen Yung Chuan (eller **Rex Chen**) är Software Engineer i Research & Development Group på SpringSoft i Taiwan. I hans ansvarsområde ingår Novas-produkterna för avbuggningsautomation, med specialkunskap om SystemVerilog Testbench och OVM. ■

JL Gray är Associate Principal på Verilab, som är baserat i Austin, Texas. JL har rest runt i världen och hjälpt företag att utveckla verifieringsmiljöer och förbättra deras verifieringsmetoder med hjälp av SystemVerilog, e och SystemC. JL är medlem i Accellera Verification IP Technical Subcommittee, som är inriktad på att utveckla en interoperabilitetslösning mellan VMM och OVM. Han är också författare till Cool Verification, en blog om hårdvaruverifiering sedd ur en konsults perspektiv. JL har en Bachelor-examen i Electrical Engineering från Purdue University i West Lafayette, Indiana. ■