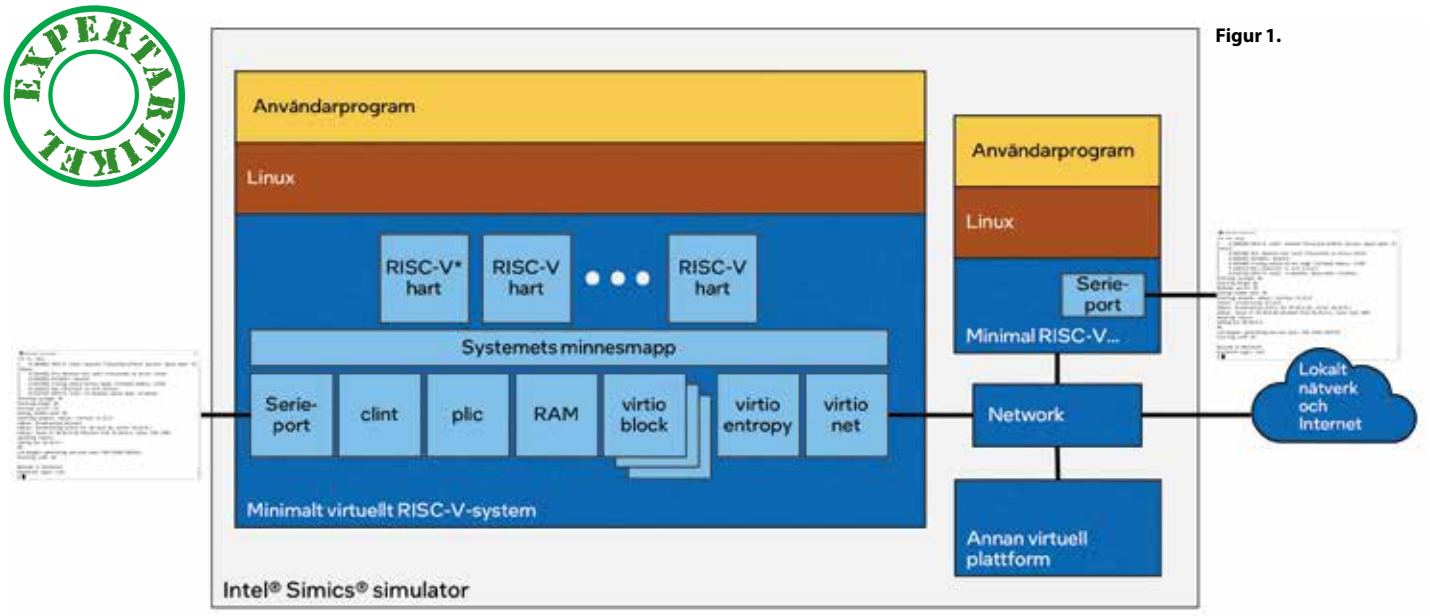


Figur 1.

RISC-V i Intels svenska verktyg

Intels Simics simulator är en helsystemssimulator som används av Intel och andra företag för att korta ner utvecklingstider och tidigarelägga utveckling av mjukvara till nya generationer av hårdvara. Simulatorn är också tillgänglig i en publik version för akademiker och allmänhet. Vi lade nyligen in stöd för RISC-V-arkitekturen. RISC-V ger oss en enkel arkitektur i ett enkelt system, vilket öppnar upp en del nygamla användningsfall för Intel Simics-simulatorn.

Av Jakob Engblom och Peter Nyström, Intel

Jakob Engblom jobbar med att främja ekosystemet för Simics-simulatorn hos Intel i Stockholm. Han har jobbat med Simics och andra simulatorlösningar de senaste tjugo åren, först på startupbolaget Virtutech, sedan på Wind River, och nu på Intel.

Peter Nyström jobbar med att utveckla Intel Simics-simulatorn, framför allt olika CPU-modeller. Han har jobbat med programmeringsverktyg under de senaste tjugofem åren, varav de sista fem åren med Simics på Intel.

Systemmodellen

RISC-V-systemet i den publika simulatorn är en rent virtuell modell som inte motsvarar någon specifik hårdvara. Figur 1 visar systemets innehåll, vilket är tillräckligt för att boota en Linux med ett obegränsat filsystem och koppla upp systemet till lokala nätverk och Internet. Liksom alla andra Intel Simics modeller är detta en snabb funktionell modell som inte modellerar hårdvaruprestanda i någon större utsträckning.

Som Figur 1 indikerar kan man simulera flera maskiner i samma simulatorprocess, både flera kopior av RISC-V-systemet och andra system. Ett exempel på det visas i Figur 2: här simuleras en standard-PC tillsammans med ett RISC-V-system. En ssh-klient på PCn kopplar upp sig till RISC-V-systemet över nätverket inne i simulatorn. Hela nätverket kontrolleras som en enhet, och kan också kopplas upp till det externa labbnätverket eller Internet.

En kul sak som vi lärde oss när vi satte igång nätverket på RISC-V-modellen var att secure shell (SSH) kräver en viss mängd slumpdata för att fungera. Eftersom modellen är väldigt enkel så producerar Linuxen väldigt lite slump av sig själv, och det tog väldigt lång tid. Lösningen var att lägga in en simulerad hårdvaruslumpgenerator (entropy) i modellen, som snabbt (i virtuell tid) producerade de slumpdata som krävdes. Eftersom det är en simulering är de data som produceras reproducerbara, men det ser ut som bra slump från Linux perspektiv.

Lågnivåkodning

RISC-V-modellen måste inte köra Linux. RISC-V är en enkel och trevlig arkitektur som är lätt att köra ”på metallen”. Simulatorn kan ladda vanliga elf-filer som är statiskt länkade direkt till minnet. Exemplet i Figur 3 visar en liten textbaserad applikation som kör direkt på hårdvaran, utan ett operativsystem.

Ett problem med den här typen av interaktiva realtidsapplikationer är att simulatorn oftast går för fort när man kör enkel kod som det här. Simulatorns ”realtidsläge” ser till att den virtuella tiden inte går fortare än verklig tid och gör att det faktiskt går att spela ”snake” på simulerad RISC-V.

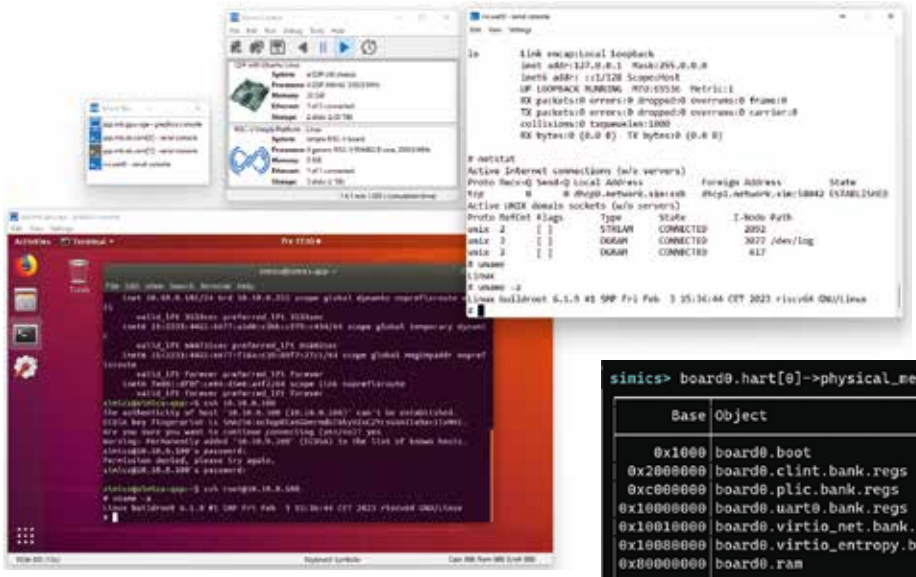
När programmet ska prata direkt med hårdvaruenheter gäller det att veta var i minnet de finns och av vilken typ de är. Figur 4 visar minnesmappen för systemet. Figur 5 visar programmeringsregistren för serieporten.

Konfiguration

RISC-V-arkitekturen är väldigt flexibel, och en viss kärna kan välja att implementera endast vissa valda grupper av instruktioner eller ’extensions’. Vår modell av RISC-V är konfigurerbar så att extensions kan slås av eller på för att bättre likna den kärna man vill simulera. Till exempel slår man av flyttalsstödet med en kodsnutt som i figur 6.

Som alla andra Simics-modeller kan man också konfigurera och bygga om hårdvaran. Aspekter som antalet kärnor och storleken på minnet och diskar ändras med enkla parametrar till systemet. För mer djupgående ändringar kan användare bygga egna hårdvarumodeller (till exempel med Intels Device Modeling Language, se Elektroniktidningen NNN) och lägga till i minnesmappen.

Eftersom RISC-V-arkitekturen är öppen kommer det hela nya förslag på utökningar och nya versioner av befintliga instruktioner. Detta gör att modellen aldrig blir ”klar”. Precis vilka utökningar som implementeras i modellen beror på precis vilka tillämpningar som RISC-V-modellen får. ■



Figur 2. En enda simulatorprocess som innehåller ett nätverk med två maskiner, en standard-PC och en RISC-V.

Figur 3. ”Bare-metal”-exempel som kör på RISC-V.

```
simics> board0.hart[0]->physical_memory.map
```

Base	Object	Fn	Offset	Length	Target	Prio	Align	Swap
0x1000	board0.boot		0x0	0x40000		0		
0x2000000	board0.clint.bank.regs		0x0	0xc000		0		
0xc000000	board0.plic.bank.regs		0x0	0x4000000		0	8	
0x10000000	board0.uart0.bank.regs		0x0	0x11		0	8	
0x10010000	board0.virtio_net.bank.mmio		0x0	0x10000		0	8	
0x10080000	board0.virtio_entropy.bank.mmio		0x0	0x10000		0	8	
0x80000000	board0.ram		0x0	0x100000		0		

Figur 4. Adresser till de olika enheterna i minnet.

```
simics> print-device-regs board0.uart0.bank.regs
Offset  Name     Size  Value | Offset  Name     Size  Value | Offset  Name  Size  Value | Offset  Name  Size  Value
------------------------------+------------------------------+-------------------------+-------------------------
     0  rtb_drl     1      0 |      2  iir_fcr     1      1 |      4  mcr     1      0 |      6  msr     1    187
     1  ier_drm     1      0 |      3  lcr         1      0 |      5  lsr     1     96 |      7  scr     1      0
```

Figur 5. Serieportens programmeringsregister.

```
foreach $h in (list-objects class = riscv-rv64 -all) {
   $h.extensions->F = FALSE
   $h.extensions->D = FALSE
   $h.info   ## generate output showing the ISA state of the core
}
```

Figur 6. Kod för att slå av flyttal i RISC-V.