

Gebruik van IP heeft

Soc-technologie (system on chip) die gebaseerd is op standaard fpga's en IP-blokken (intellectual property, red.) is niet alleen voor consumentenelektronica en de automobielenindustrie van belang; zij is ook toepasbaar voor technologisch minder geavanceerde applicaties, zelfs bij lage volumes. Vervanging van meerdere discrete componenten als microcontroller, periferen en verbindinglogica door een fpga kan productbetrouwbaarheid, stroomverbruik, printkaartafmeting, componentkosten en de kosten en inspanningen van voorraadbeheer aanzienlijk beïnvloeden.

Eda- en embedded-software-toolleveranciers (electronic design automation, red.) zijn gaan beseffen dat de toeneemende trend naar soc-ontwerp gepaard moet gaan met een gelijkwaardige integratie van hardware- en software-ontwikkeltools. Hoewel de huidige ontwikkeltools de meeste individuele componenten wel bestrijken, zijn door het effect van de integratie van deze componenten op een chip, en de invloed van herconfigureerbare kernen op software-ontwikkeltools, de huidige toolsets minder doelmatig geworden.

Tools integreren

Het integreren van eda- en software-ontwikkeltools in een gebruiksvriendelijke omgeving is vooral relevant voor ontwerpen die in fpga-technieken zijn geïmplementeerd. Asic-ontwerp is nog steeds een hooggespecialiseerde activiteit waarin engineers zich concentreren op of hardware- of alleen software-ontwikkeling. Engineers die fpga-gebaseerde systemen ontwikkelen met gebruik van standaard IP-blokken hebben vaak een bredere, minder gespecialiseerde kennis, werken in kleine teams, en zijn betrokken bij zowel hardware- als software-aspecten.

Gegeven de overeenkomsten tussen HDL-gebaseerde (hardware description language, red.) hardware-ontwikkeling en software-engineering kunnen de tools voor tekstinvoer, bestandsbeheer, versiebeheer en systeembouw worden gedeeld door software- en hardware-ontwikkelteams. Dat vermindert licentiekosten en vergemakkelijkt de integratie tussen beide ontwikkelteams.

Tot voor kort werden *system-on-chips* gezien als het exclusieve domein van hightech bedrijven. De benodigde expertise, de hoge kosten van ontwikkeltools en de noodzakelijke afzetvolumes om de kosten van de ontwikkeling te rechtvaardigen, vormden een hoge belemmering. Met de introductie van een nieuwe generatie fpga's en de beschikbaarheid van IP is deze technologie echter bereikbaar geworden voor een veel bredere gemeenschap.

GERARD VINK Tasking

Illustratie: Dan Geerlings

effect op ontwikkeltools

Simulatie, noodzakelijk voor zowel ontwerp van hardware als software, wordt gebruikt om de individuele componenten van een systeem, haar interfaces, en het systeem als geheel te verifiëren. Hoewel omgevingen voor co-simulatie beschikbaar zijn vanaf halverwege de jaren negentig hebben we om de brug tussen hardware- en software-ontwerp af te maken een hardwaresimulatie en software-debug-omgeving nodig die in veel hogere mate geïntegreerd is dan thans.

Geen enkel product dat op dit moment verkrijgbaar is combineert, in een omgeving, de signaalweergave en tijdsanalyse van hardwaresimulatoren met de kenmerken van traditionele software-debuggers om zowel de hardwarebeschrijving als de applicatiesoftware te analyseren. Een simulatie- en debug-omgeving die zowel de hardware- als de software-aspecten van een ontwerp ondersteunt is noodzakelijk om complexe soc-ontwerpen te ondersteunen die meerdere zachte processorkernen, randlogica en andere fpga-gebaseerde logica bevatten.

Welnu, dit zijn de benodigde tools voor hardware- en software-ontwikkeling. Laten we nu een blik werpen op traditionele software-ontwikkeltools: de compiler, assembler, linker en de debugger.

Compiler/assembler

Processorkernen voor soc-implementatie bestrijken een breed gebied. Aan de ene zijde van het spectrum worden bestaande 8- en 16-bit processoren geoptimaliseerd voor implementatie in fpga en die zijn beschikbaar als synthetiseerbaar HDL-model. Merk op dat deze kernen hun originele ontwerpen met een orde van grootte kunnen overtreffen; een 8051 IP-kern kan geklokt worden op 50 MHz (ter vergelijking ~ 10MHz) en voert een instructie uit per klokcyclus (in plaats van een op drie). Aan de andere zijde van het spectrum bedienen high-performance architecturen de bovenkant van de markt. Deze kernen worden getypeerd door een VLIW-architectuur (very large instruction word, red.), ondersteunen mogelijk zelfs SIMD-operaties (single instruction multiple data) en zijn gebruiker-

configureerbaar om de architectuur af te stemmen op een bepaalde applicatie.

Een raamwerk dat de constructie van C/C++-compilers voor dit bereik van architecturen faciliteert moet heel modulair zijn. Wanneer een compiler voor een bepaalde target wordt gebouwd moeten er individuele code-generatie en optimalisatiefasen worden toegevoegd, verwijderd of vervangen voor een andere implementatie.

Configureerbare VLIW-kernen kunnen typisch op maat worden gebracht in termen van:

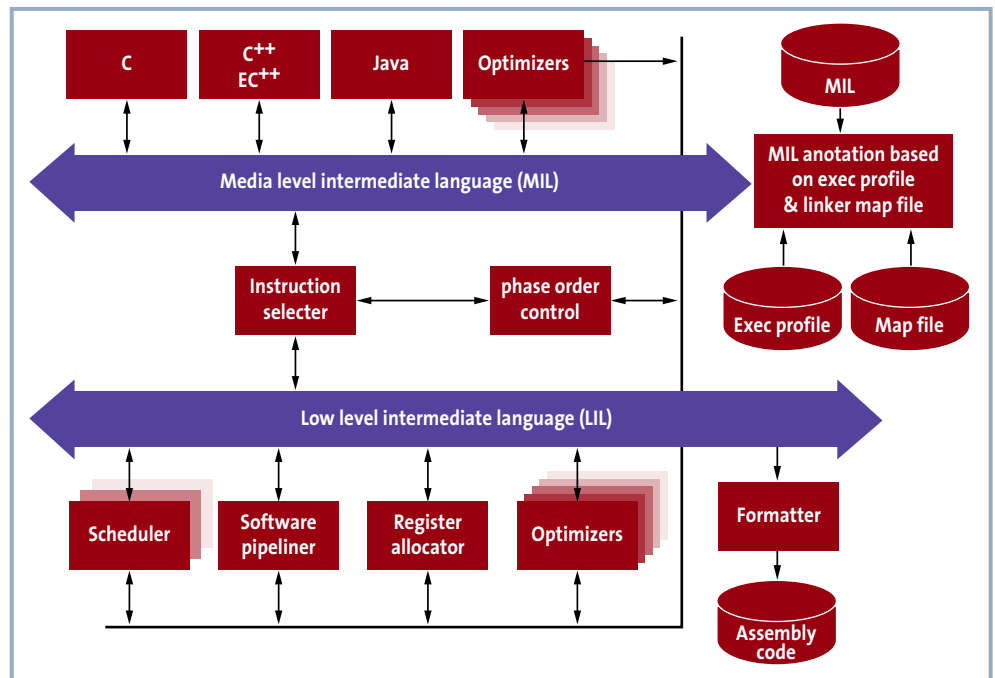
- de beschikbaarheid en grootte van instructie en data-caches;
- veranderingen in aantal en verhouding van functionele eenheden;
- veranderingen in registers en adresseer-modi;
- veranderingen in de instructie-set en hardwareversnellers.

Deze kernen vereisen de ondersteuning van compilers en assemblers wier gedrag tijdens runtime kan worden veranderd om tegemoet te komen aan veranderingen in, en varianten van,

hardware. Grote architectuurveranderingen – zoals de invoering van nieuwe instructies, nieuwe adresseer-modi, of een nieuwe pijplijnarchitectuur – noodzakelijk tot de bouw van een nieuwe compiler. Weglaten zoals het verwijderen van opcodes, registers en adresseer-modi kunnen, evenals veranderingen in aantal en verhouding van functionele eenheden, ondersteund worden tijdens runtime.

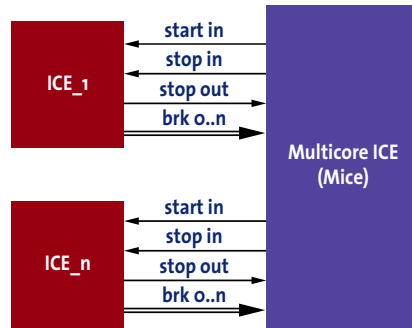
Signaal-verwerkingsalgoritmen kunnen worden geïmplementeerd als of drijvende komma of vaste-komma-berekeningen. Consumenten- en andere groot-volume-toepassingen gebruiken vaste-komma-berekening, omdat vaste-komma-operaties kunnen worden geïmplementeerd met minder poorten, wat hardwarekosten verlaagt en een snellere doorvoer van berekeningen oplevert. Omdat ISO-C geen vaste-komma-datatype ondersteunt introduceert de compilerontwerper een nieuw (bijvoorbeeld vaste komma) datatype ofwel ondersteunt vaste-komma-operaties door intrinsieke functies.

Figuur 1 illustreert de basisarchitectuur van Viper, dat een soc-compiler-raam- >



1. De front-ends van Tasking's Viper-compiler vertalen hun invoer in een medium-level intermediate language (MIL) waarop target-onafhankelijke front-end optimaliseerders werken. De instructie-selector brengt de MIL over naar een low-level intermediate language (LIL) waar instructie-inroosting, registertoekenning en back-end-specifieke optimaliseerders opereren.

2. De debug-topologie wordt gedownload in de multi-core ICE. De single-core on-chip debug eenheden zijn uitgebreid om de multi-core ICE in werking te zetten wanneer hun geassocieerde kern stopt. Gegeven de debug topologie en de oorzaak van het stoppen stuurt MICE een stop signaal naar de toepasselijke kernen.



werk vormt dat de uitdagingen van configureerbare IP-kernen behandelt. De tussenliggende talen kunnen worden beschouwd als bussen waar de codegeneratie-fasen op kunnen worden aangesloten. De front-end optimalisatoren zijn onafhankelijk van de target, terwijl de back-end optimalisatoren zijn geparametriseerd door karakteristieken van een gegeven target.

In tegenstelling tot traditionele compilers waar de codegeneratie fasen in sequentiële volgorde verlopen, gebruikt Viper fase-ordeningbeheer om een bepaalde optimaliseerder op de juiste momenten te executeren en om mogelijk een gegeven optimalisatie terug te draaien indien later in het codegeneratieproces blijkt dat de optimalisatie een negatief effect heeft op de uiteindelijke codekwaliteit.

Moderne assemblers zijn niet handgemaakt maar worden gegenereerd uit de instructieset-database van een processor. Ondersteuning van herconfigureerbare kernen wordt geïmplementeerd door attributen die specificeren welke instructies door welke variant van de kern worden ondersteund. Dus kan de assembler zijn gedrag aanpassen tijdens runtime in plaats van compileer-tijd.

Maar, VLIW-architecturen kennen meestal beperkingen voor instructies die parallel opereren. Momenteel is er geen formele manier om deze beperkingen te beschrijven, dus is de restrictie-checker nog steeds een handgemaakt deel van de assembler en moet hij worden geparametriseerd door de veranderbare architectuureigenschappen.

Linking & locating

Herconfigureerbaarheid heeft geen effect op de linker, maar multicore-aspecten wel. Een geavanceerd soc-ontwerp kan meerdere, ongelijke processor-kernen bevatten en hoewel de huidige linkers kunnen worden gebruikt om applicaties voor zulke omgevingen te bouwen, is het proces eentonig en foutgevoelig.

Ten eerste zijn linkers target-afhanke-

lijk, dus zijn er vaak verschillende link-tools vereist. Ten tweede wordt elke executable onafhankelijk gelinked van de andere executables, waardoor de gebruiker meerdere linker-controlbestanden moet schrijven. Fysiek geheugen dat door meerdere kernen wordt gedeeld moet worden gedefinieerd in elk linker-controlbestand, en omdat de software voor elke kern wordt gelinked in een aparte run van de link-tool, moet de gebruiker expliciet aangeven welke programma-secties zich bevinden op welk fysiek adres.

Een linker die heterogene multicore-systemen ondersteunt moet targetonafhankelijk zijn. Alle target-specifieke informatie hoort in runtime te worden gelezen uit een linker-scriptbestand.

De benodigde inspanning om een target-onafhankelijke linker te ondersteunen hangt af van de objectformaten die worden gebruikt – bijvoorbeeld, IEEE-695 objecten zijn target-onafhankelijk terwijl ELF-objecten target-afhankelijk zijn. Niettemin kan een linker die ELF-objecten verwerkt ongelijke kernen ondersteunen, maar hij moet worden aangepast wanneer er ondersteuning voor een nieuwe kern wordt vereist.

Vanuit het perspectief van de linker wordt de architectuur van een soc beschreven in termen van:

- de kernen binnenin het systeem;
- per kern de logische adresruimten, adresomzettingen tussen logische adresruimten (voor het geval dat een adresruimte een deelverzameling is van een andere ruimte, zo is de nabije dataruimte typisch een deelverzameling van de dataruimte), en de afbeelding van logische adresruimte naar interne data- en adresbussen;
- de adres- en databussen in het systeem en adresomzettingen tussen bussen;
- de connecties tussen interne en systeem adres- en databussen;
- het fysieke geheugen en met welke bus het geheugen is verbonden.

Deze informatie wordt typisch hard ge-

codeerd (hoewel op een lager abstractieniveau) in een linker. Een multicore-linker haalt deze informatie uit de linker-scriptfile.

Debugger

Herconfigureerbaarheid heeft een zeker effect op de software-debugger. Ondersteuning voor de eerder genoemde weglatende veranderingen – veranderingen in grootte van instructie- en data-caches, aantal en verhouding van functionele eenheden en verwijdering van opcodes – is op triviale manier te implementeren in een embedded-systems-debugger.

Maar het effect van het toepassen van meerdere ongelijke processor-kernen in een soc-ontwerp is een technologische, maar bovendien een economische uitdaging, gezien het betrekkelijk klein aantal ontwikkelaars. Met uitzondering van eerdere opmerkingen over simulatie wil ik de discussie over software-debugging beperken tot hardware-omgevingen.

Heden ten dage worden bijna alle IP processor-kernen, zowel voor asic- als fpga-implementatie, gedistribueerd in combinatie met een on-chip debug (OCD) eenheid die in-circuit emulator functionaliteit levert op de chip. De debugger heeft, typerend, toegang tot de OCD-eenheid via een JTAG-interface.

Wanneer een debug-sessie wordt begonnen is het eerste probleem dat opdoemt, dat de JTAG-poort wordt gebruikt door de eda-tools om de hardware-reconfiguratie in de fpga te laden en ook door de software-debugger wordt gebruikt om de OCD-eenheid aan te spreken. In een engineer-vriendelijke omgeving zouden zowel de eda- en software-ontwikkeltools hetzelfde interface-device behoren te gebruiken om de JTAG-poort aan te spreken, maar helaas is dat op dit moment niet het geval. De technologie om meerdere ongelijke processor-kernen te debuggen is snel volwassen aan het worden. Bijvoorbeeld, het Multicore Debug System (MDS) van Tasking gebruikt single-core-debuggers die zich niet bewust zijn van elkaars bestaan. De debuggers interfacen met een mediator, die de processor-kernen coördineert. De kernen kunnen geïmplementeerd zijn als een soc, verschillende processoren, of zelfs simulatiemodellen zijn. Voor elke debugger schept de mediator de illusie dat de debugger totale controle heeft over de processor waar hij mee is verbonden iets dat als vanzelfsprekend

werd beschouwd toen single-core-debuggers werden ontworpen.

De MDS-manager is het gebruikers-interface-deel en dient drie doelen: initialisatie van de executie-omgeving, instanties van de debugger opzetten en de debug-topologie definiëren. De debug-topologie staat gebruikergedefinieerde scenario's toe zoals:

- wanneer processor A de executie begint dan start processor B ook de executie;
- wanneer processor B de executie stopt (bijvoorbeeld door een breakpoint) gaat processor B door met de executie;

Hoewel de MDA en andere multicore-architecturen, op het eerste gezicht nogal gelijksoortig zijn, maken de unieke kenmerken van de mediator het de gebruiker mogelijk alle debug-kenmerken die in een single-core-omgeving beschikbaar zijn te gebruiken in een multicore-omgeving (bijvoorbeeld: elke single-core debugger is in staat zijn breakpoint-scripts te draaien die de executie opnieuw opstarten).

Single-core-debuggers interfacen met de target door het Generic Debug Interface (GDI), dat wordt ondersteund door diverse debugger- en chip-leveranciers. Als gevolg daarvan is het gemakkelijk een third-party debugger te integreren in de MDS-managed multicore debugging-omgeving.

Het debug -topologieconcept van de MDS ondersteunt zowel gesimuleerde als reallife hardware executie-omgevingen. De Multicore ICE (MICE) breidt de functionaliteit die de debug-eenheid van elke single-core levert verder uit. Invoersignalen *start_in* en *stop_in* zijn geïntroduceerd om MICE in staat te stellen een kern te starten of te stoppen door zijn specifieke ICE. Uitvoer signaal *stop_out* wordt actief gemaakt wanneer een kern in debug-modus komt; de conditie die dit event heeft veroorzaakt wordt doorgegeven via de *brk*-bus. De debug-topologie wordt in MICE gedownload via de JTAG-scanketen.

Conclusie

In wezen wordt soc-ontwerp gestuurd door goedkoop silicium, IP en tools. Ge-

geven dat fpga's thans verkrijgbaar zijn voor \$0,00015 per poort kunnen soc-ontwerpen kosteneffectief zijn in applicaties die in kleine volume worden geproduceerd. Om efficiënt een soc te ontwikkelen heb je geen generieke soc-methodologie nodig, maar een groep IP-blokken die samenwerken en een leverancier die hun samenwerkbaarheid garandeert.

Herconfigureerbare processor-kernen, vooral complexe VLIW-architecturen, vereisen compilers die ontworpen zijn – niet aangepast – om dergelijke kernen te ondersteunen. Hetzelfde, hoewel in mindere mate, gaat op voor de assembler en linker/plaatser. Ook is het mogelijk om kosten-effectief een professioneel heterogeen multicore debug systeem te construeren vanuit bestaande single-core debuggers.

Tot er goed geïntegreerde, gebruikersvriendelijke, en betaalbare hardware- en software-ontwerptools beschikbaar komen zal fpga IP-technologie niet algemeen worden geaccepteerd door de algehele microelektronica-markt. ■