Applicatiespecifieke

Na de programmeerbare cpu en de programmeerbare dsp komen in dit laatste deel over processoren in embedded systemen de applicatie-domein-specifieke processoren aan de orde. Ze worden doorgaans ingezet in combinatie met een programmeerbare cpu of dsp.

GER SCHOEBER, Ordina Media, Son

In dit artikel bespreken we de applicatiedomein-specifieke processoren (ADSP's) en de applicatiespecifieke processoren (ASP's). ASP's kenmerken zich doordat deze de intrinsieke mogelijkheden van het silicium ten volle benutten. De ADSP's daarentegen bakenen weliswaar een applicatiedomein af, maar binnen dit doeldomein blijft toch enige programmeerbaarheid behouden. De ADSP wordt ook wel eens aangeduid met applicatiespecifieke instructieset-processor (ASIP).

Domeinspecifiek

Zoals gezegd zijn deze processoren geoptimaliseerd voor een bepaald applicatiedomein. Hierbij kan worden gedacht aan DECT, GSM of MPEG2. De standaard voor MPEG2 bijvoorbeeld specificeert het gebruik van DCT-transformaties op blokken van acht bij acht pixels. Door van deze voorkennis gebruik te maken kan een heel efficiënte implementatie voor een DCT worden gerealiseerd. Een duidelijk verschil met de besproken programmeerbare cpu en dsp is dat bij de realisatie van een ADSP het applicatiedomein als uitgangspunt wordt genomen. Een ADSP wordt dan ook met synthesetools ontworpen.

Het ontwerpproces kent drie fasen: de *exploratiefase, detailontwerpfase* en *applicatie-ontwerpfase*. Tijdens de exploratiefase wordt er geëxperimenteerd met de architectuur van de gewenste processor. Middels een eerste keuze van een applicatie binnen het domein en een eerste processorontwerp wordt de applicatie op de betreffende processor afgebeeld. Het doel ervan is een idee te krijgen van het aantal klokcycli dat nodig is voor het algoritme, de duur van een cyclus, de benodigde oppervlakte, de hoeveelheid vermogensdissipatie, enzovoort. Als dit niet aan vooraf gestelde criteria voldoet, zal een tweede iteratieslag volgen. Wordt wel aan de criteria voldaan, dan wordt het proces herhaald, maar nu voor een iets andere applicatie, weliswaar binnen het gestelde domein (bijvoorbeeld een andere MPEG-standaard).

Als een volledige specificatie beschikbaar is vanuit een exploratiefase, kunnen de hardware-ontwerpers aan de slag om middels gate- en layoutniveau-ontwerp een gedetailleerde database klaar te maken.

Nadat de hardware is ontworpen, is het mogelijk nieuwe code te genereren voor applicaties binnen het applicatiedomein. Deze fase lijkt sterk op de exploratiefase, met dien verstande, dat het processormodel nu stabiel blijft en dat in een volgende iteratie getracht wordt het algoritme te optimaliseren om tot een steeds beter eindresultaat te komen.

De vele, snelle iteraties vereisen veel aandacht voor de software-aspecten van de iteratie. Er zijn compilers nodig die met een minimale inspanning kunnen worden aangepast om code te genereren voor elke processor die tot een bepaalde klasse behoort. We noemen dit *retargetable compilers*.

Ook de codeselectie, registerallocatie en scheduling (zie het artikel over dsp's in de vorige editie) zijn sterk met elkaar verweven. Er moet zelfs rekening worden gehouden met routing van data. Ten opzichte van programmeerbare dsp's is het probleem alleen maar complexer geworden. Dit is een extra reden om te zien in hoeverre VLIW-architecturen (very large instruction word) een oplossing kunnen bieden.

VLIW-architecturen

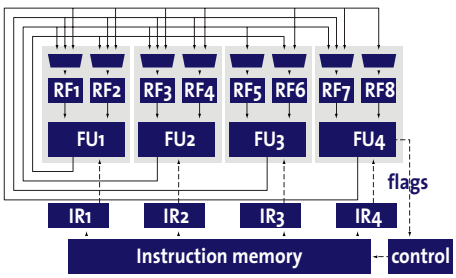
Veelal wordt uitgegaan van een VLIW-architectuurtemplate. Van dit template kunnen voor specifieke applicatiedomeinen specifieke instanties worden afgeleid. Het template is een soort model met een aantal kenmerken. Een eerste kenmerk is de aanwezigheid van meerdere *functionele units* (FU's) die gezamenlijk meerdere operaties in parallel kunnen uitvoeren. In tegenstelling tot wat bij dsp's het geval is, verschuiven de FU's hier meer richting grotere processor-kernels. Dit betekent een grotere korrel van granulariteit, complexere units, een verschuiving van homogeniteit naar heterogeniteit en een grotere pipeline-diepte. Ze worden vaak *applicatiespecifieke units* (ASU's) genoemd. De data-bitbreedte kan tot 128 bits oplopen.

Een tweede kenmerk is de aanwezigheid van meerdere registerfiles, typisch meer dan 5, met een beperkt aantal poorten en registers per registerfile.

Een derde kenmerk tot slot is dat het aantal issue-sloten gelijk is aan het aantal FU's (ASU's).

Aansluitend is het interconnect-netwerk opgebouwd uit één of meer bussen en kunnen geheugens gemodelleerd zijn als FU's. Ook kan elke FU één of meerdere vlaggen genereren die naar de controller worden gestuurd. De laatstgenoemde kenmerken zijn specifiek voor de Mistral 2 architectuurtemplate. Mistral 2 is een commercieel tool dat gebaseerd is op onderzoek van Imec en Philips. Een voorbeeld van een architectuurtemplate is opgenomen in figuur 1.

Hierin herkennen we vier FU's. Iedere

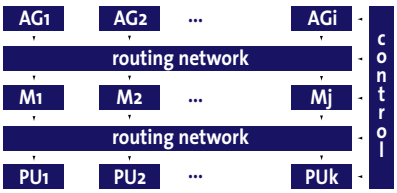


1. Voorbeeld van een architectuurtemplate. Herkenbaar zijn vier functionele units (FU's). Elke FU heeft als input twee registerfiles die weer door middel van multiplexers zijn gekoppeld aan één of meerdere bussen.

2. De instructievelden van een instructieregister voor een execution unit.

mux 1	mux 2	read address RF1	write address RF1	read address RF2	write address RF2	control FU	output drivers

processor past prima in **beeld**



3. Een generieke architectuur voor een applicatiespecifieke processor.

FU heeft als input twee registerfiles die middels multiplexers zijn gekoppeld aan één of meer bussen. Een functionele unit met bijbehorende registerfile en inputmultiplexer noemen we een *execution unit* (EXU). Van deze template kunnen architectuur-instanties worden afgeleid. Dit gebeurt over het algemeen door het middels parameters aangeven van de bitbreedte, het aantal registerfiles, aantal registers in een file, aantal inputs, enzovoort.

Het gebruik van vlaggen en controlelogica opent de mogelijkheid de processor te herprogrammeren nadat het silicium reeds beschikbaar is. Dit laatste is niet meer het geval bij applicatiespecifieke processoren die over een hard bedrade *finite state machine* (FSM) beschikken. De opgehaalde instructie wordt opgedeeld en in een viertal instructieregisters geplaatst. Een enkel instructieregister stuurt het totale datapad van een EXU. Dit is de aansturing van de multiplexers, de registeradressering, de FU zelf en de output. De instructievelden zijn getoond in figuur 2.

Het aantal bits voor een bepaald veld is overigens ook middels een parameter configureerbaar. Merk op dat de meeste velden een NOP (NOP staat voor no-operation, een instructie die meestal een byte geheugenruimte kost en 1 cyclus processortijd, maar verder geen operatie uitvoert) moeten kunnen ondersteunen. De vier instructieregisters samen vormen één enkele brede instructie (VLIW) die vier EXU's in parallel kan aansturen.

Tijdens de exploratiefase van het ontwerpproces wordt de keuze van de FU's bepaald. Deze keuze wordt eenvoudiger naarmate de FU's meer applicatiespecifiek worden. Bijvoorbeeld een dsp met daarin een ALU, een MAC, een RAM, een ACU en voor de rest aangevuld met ASU's. Algemeen kunnen we concluderen dat de VLIW-concepten niet voldoende garantie bieden om een relatief eenvoudige codeselectie mogelijk te maken, maar ze kunnen in de praktijk toch veel helpen als ze op de juiste manier worden toegepast. Denk hierbij aan een doorzichtige VLIW-architectuur in combinatie met gebruikersinteractie via pragma's.

Specifieke applicatie

De meest efficiënte oplossingen worden bereikt door een ontwerp te maken dat optimaal is voor één specifieke applicatie. Deze oplossingen zijn sterk verwant aan de klassieke *high level synthesis* (HLS); de specificatie van de te ontwerpen functie is van redelijk hoog abstractieniveau. Gegeven de functionele specificatie aangevuld met *timing requirements* moet een architectuur worden ontworpen die uiteraard aan die gegeven eisen voldoet, maar die ook zoveel mogelijk naar kosten is geoptimaliseerd. De kosten voor een hardware-ontwerp kunnen worden uitgedrukt in chip-oppervlak. Hoe groter de uiteindelijke oppervlakte, hoe groter de vermogensdissipatie en hoe groter de uiteindelijke complexiteit.

Een typisch applicatiegebied zijn de embedded multimediasystemen. Denk aan pixelniveau-processing bij video- of grafische systemen. Het zijn applicaties die sterk zijn georiënteerd op een grote hoeveelheid data waarbij alle dimensies naar de tijd worden gemapt. Processing-nodes krijgen aan de ingangen datastromen aangeboden en produceren datastromen naar de uitgangen. Een typisch voorbeeld is de conversie van interlaced beelden naar progressieve beelden waar geen onderscheid wordt gemaakt tussen even en oneven lijnen zoals bij pc-monitoren. Bij de oplossing van dit soort problemen wordt elke operatie als een aparte entiteit gezien, die op zich periodiek in de tijd plaatsvindt: het model van *periodieke operaties*.

Video-applicaties worden vaak gekenmerkt door rekenintensieve kernels, of sterk gekoppelde groepen van operaties. Ze vormen als het ware op natuurlijke wijze een hiërarchisch niveau in de specificatie, oftewel specifieke functionele units die ieder voor zich redelijk complex kunnen zijn. Ze worden ook wel *processing units* (PU's) genoemd. In figuur 3 is een architectuur weergegeven met een aantal processing-units, geheugens, adresgeneratoren en een controller. De rol van de geheugens is het datatransport te verzorgen tussen dataproducerende en dataconsumerende PU's. De controller heeft als taak de juiste PU te selecteren, de read/write-signalen voor de geheugens te verzorgen, het uitgeven van de *next-address* signalen voor de adresgeneratoren en de besturing van de netwerken. Verschillende PU's, geheugens en adresgeneratoren kunnen parallel actief zijn.

Om tot een uiteindelijke processor te komen, moet een aantal ontwerpstappen worden doorlopen die met tools worden ondersteund. De eerste stap wordt wel clustering genoemd. Dit is het 'binden' van functies uit de specificatie aan PU's, hierbij rekening houdend met de dataprecedenties tussen de clusters. Daarna worden de PU's gegenereerd. De PU's en de genoemde dataprecedenties vormen de input voor de scheduling-stap. Na deze stap is bekend hoeveel buffering er nodig is. Dit als input voor de geheugensynthese-stap. Hierna volgt de adressynthese waar binnen elk geheugen de plaats van elke variabele wordt bepaald. Ten slotte kan uit alle vorige stappen een aantal controlestromen worden afgeleid die tijdens de controllersynthese worden geïmplementeerd. Het resultaat vormt een register-transferniveau-beschrijving van de architectuur die verder door synthesetools (RT-niveau) kan worden verwerkt tot een ASP (zie ook deel 2).

Zoals gezegd wordt een applicatiespecifieke processor voor één applicatie ontworpen en is hiermee de meest efficiënte implementatie mogelijk. Een zwakke vorm van programmeerbaarheid in de vorm van parameters blijft echter mogelijk. ASP's worden vaak ingezet in die delen van een systeem die vaak een bottleneck betekenen op systeemniveau. Toepassing ervan vindt vaak plaats in combinatie met minder specifieke processoren zoals bijvoorbeeld dsp's. ▪

Achtergrond

Auteur Ger Schoeber is consultant software- en systeemarchitectuur voor Ordina Media. Voor vragen kunt u contact opnemen met ger.schoeber@ordina.nl