Adding C Programmability to Data Path Design

Gert Goossens
Sr. Director R&D, Synopsys
Smart Products Drive SoC Developments

- Need for reusable SoC platforms
- SoC platforms must become software programmable, without compromising PPA (performance, power, area)
Programmable Processors in SoCs
Processor Solutions Spectrum

ASIP = Application-Specific Instruction-set Processor

Maximize performance
- Architectural specialization
- Parallelism: instruction-level, data-level, task-level

Minimize power consumption
- Architectural specialization
- Parallelism: instruction-level, data-level, task-level
- Power-optimised RTL generation
- Power-gating of cores

Programmability
- Support changing requirements, product differentiation, new features… without SoC respin!
- Quick algorithm mapping from C to silicon, with easy debugging
ASIP Architectural Optimization Space

- Architectural space beyond configurable templates
- Can be captured by processor description language
- Architectural exploration enabled by retargetable ASIP design tools
ASIP Designer – Retargetable ASIP Design Tool

Typical users: ASIC/SoC design teams

1. SDK Generation
2. Architectural Optimization
3. Hardware Generation
4. Verification

© Synopsys, Inc. All rights reserved
ASIP Designer – History

IP Designer
• Processor description language: nML
• Roots in architectural exploration and retargetable compilation

Processor Designer
• Processor description language: LISA
• Roots in modeling and fast simulation

ASIP Designer
• Processor description language: nML
• Consolidated product, combining strengths of IP Designer and Processor Designer
• Stepwise deployment in 2015-2016 time frame
• Legacy products remain available
Adding C Programmability to SoC Design

- Graph-based compilation technology combines retargetability with high code efficiency
- Instruction-set graph (ISG)
  - Graph-based optimization algorithms operate on (any) ISG
  - Closer to HW than other compilers’ machine models
    - HW resources, data types, connectivity, instruction encoding, instruction-level parallelism, instruction pipeline
    - Supports “irregular” architectures
- Enables rapid and architectural exploration with compiler-in-the-loop
- Enables algorithm development in C, even for highly specialized ASIPs
Applicable to “Any” Application Domain

<table>
<thead>
<tr>
<th>Medical</th>
<th>GN ReSound</th>
<th>ON</th>
<th>NXP</th>
<th>imec</th>
<th>Cochlear</th>
</tr>
</thead>
<tbody>
<tr>
<td>Audio</td>
<td>Texas Instruments</td>
<td>NXP</td>
<td>CONEXANT</td>
<td>SANYO</td>
<td>ETRI</td>
</tr>
<tr>
<td>Video &amp; imaging</td>
<td>Texas Instruments</td>
<td>OLYMPUS</td>
<td>BRIGHTSCALE</td>
<td>Sensata</td>
<td>e cognive</td>
</tr>
<tr>
<td>Graphics</td>
<td>dialog</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wireless</td>
<td>Texas Instruments</td>
<td>NOKIA</td>
<td>freescale</td>
<td>dialog</td>
<td>NXP</td>
</tr>
<tr>
<td>Wireline</td>
<td>ST</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Network processing</td>
<td>gennum</td>
<td>imec</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>High-perf. computing</td>
<td>Atmel</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Automotive</td>
<td>NXP</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Crypto &amp; identification</td>
<td>imec</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Industrial</td>
<td>YOKOGAWA</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Publicly announced IP Designer and Processor Designer customers
Examples: Wireless Communication

“BoT” [1]
Configurable inner-modem processor
LTE(A) + 11ac + 11ad + WPAN + GPS + DVBT...

“FlexFEC” [2]
3-standard FEC engine
LDPC + Turbo + Viterbi

“BLOX” [1]
Single-function sliced accelerators
FFT / LDPC / Matrix inv.

Examples: Wireless Communication

“BoT” [1]
Configurable inner-modem processor
LTE(A) + 11ac + 11ad + WPAN + GPS + DVBT...

“FlexFEC” [2]
3-standard FEC engine
LDPC + Turbo + Viterbi

“BLOX” [1]
Single-function sliced accelerators
FFT | LDPC | Matrix inv.

---


BOT

Configurable Inner-Modem Processor [1]

- Mixed scalar/vector processor
  - 10-slot VLIW: 3 scalar, 2 vector L/S, 3 vector compute, 2 pack/unpack
  - Vector compute units with increased specialization
    - VU1: alu, mul, shift
    - VU2: alu, cabs, interleave, shift
    - VU3: alu, recip, sqrt, tan, cexp, slope, interleave, softdemap
  - Vector packing/unpacking
  - Low power: clock gating exploits low duty cycle
  - C programmable

average: 45mW (40nm@400MHz)
Examples: Wireless Communication

“BoT” [1]
Configurable inner-modem processor
LTE(A) + 11ac + 11ad + WPAN + GPS + DVBT...

“FlexFEC” [2]
3-standard FEC engine
LDPC + Turbo + Viterbi

“BLOX” [1]
Single-function sliced accelerators
FFT / LDPC / Matrix inv.

FlexFEC
3-Standard Forward Error-Correction (FEC) Engine [2]

- Application-specific mixed scalar/vector processor
  - SIMD: n-way x 8-bit
  - VLIW: 1 scalar and 5 vector issue slots
  - App.-specific primitive functions
    - LDPC decode, Turbo decode, Viterbi decode (e.g. add-compare-select), special addressing modes
  - App.-specific complex instructions
    - “abs() + abs()”, element-wise vector shift, cross correlation with programmable spreading code
  - Transparent background memory access through lookup address generator
  - C programmable
FlexFEC
3-Standard Forward Error-Correction (FEC) Engine [2]

• Specialization: e.g. LDPC decode function “vq()”

  Standard 32-bit RISC
  3,040 cycles
  ↓ (mild) specialization

  32-bit RISC with predicated add/sub
  2,707 cycles
  ↓ data-level parallelism

  96-lane, 16-bit SIMD with vector-predicated add/sub
  32 cycles
  ↓ specialization

  96-lane, 16-bit SIMD with LDPC decode instruction
  (synthesized from C code)
  1 cycle

Note: cycle counts obtained for randomized input data
FlexFEC
3-Standard Forward Error-Correction (FEC) Engine [2]

- Instruction-level parallelism: 1 scalar + 5 vector (SIMD) slots
  - C compiler efficiently exploits VLIW issue slots

```c
vchar chess_storage(DMV2) upd[100];
vchar chess_storage(DMV2) alfa[10];
vchar chess_storage(DMV1) tmp[22];

movi R1 upd || nop || nop || nop || nop || nop || nop || nop || VR0=DMVB[R0++]
  add R1 R0 R2 || nop || nop || nop || nop || nop || nop || VR1=DMVB[R0++]
  add R1 R0 R3 || nop || nop || nop || nop || nop || nop || VR2=DMVB[R0++]
  add R1 R0 R2 || nop || nop || nop || nop || nop || nop || VR3=DMVB[R0++]
  movi R3 -32 || nop || nop || nop || nop || nop || nop || VR4=DMVB[R0++]
  add R1 R0 R2 || vset VR4 R3 || nop || nop || nop || VR5=DMVB[R0++]
  movi R6 tmp || nop || nop || nop || nop || nop || nop || VR7=DMVB[R0++]
  add R1 R0 R4 || vsb VR0 VR6 VR0 || nop || nop || nop || VR0=DMVB[R2++]
  movi R2 || vsb VR1 VR3 VR1 || vq VR4 VR0 VR3 || DMV[R6+1]=VR0 || nop || VR1=DMVB[R0++]
  movi R8 alfa || vq VR3 VR1 VR6 || DMV[R6+1]=VR1 || nop || VR2=DMVB[R0]
  movi R4 R3 || vq VR2 VR4 VR6 || VR2=DMV[R7-1] || nop || VR3=DMVB[R5-1]
  add R1 R0 R5 || vsb VR2 VR5 VR3 || vq VR0 VR4 VR0 || VR0=DMV[R7-1] || nop || VR4=DMVB[R0]
  movi R5 VR6 VR0 VR0 || vsb VR7 VR0 VR3 || vq VR6 VR3 VR3 || DMV[R6+1]=VR3 || nop || VR5=DMVB[R4]
  movi R5 R4 || vq VR3 VR4 VR6 || VR6=DMVB[R5] || nop || VR6=DMVB[R5]
  movi R6 R7 || vsb VR1 VR5 VR1 || vq VR3 VR0 VR0 || DMV[R6+1]=VR0 || nop || VR7=DMVB[R6]
  movi R8 R5 || vq VR0 VR1 VR2 || DMV[R6]=VR1 || nop || VR8=DMVB[R4]
  add R1 R2 R0 || vadd VR6 VR6 VR6 || vq VR2 VR4 VR6 || VR2=DMV[R7-1] || nop || VR9=DMVB[R5-1]
  add R1 R2 R0 || vadd VR4 VR0 VR6 || vq VR1 VR7 VR4 || VR7=DMVB[R7] || nop || VR10=DMVB[R0]
  add R1 R2 R0 || vadd VR4 VR3 VR3 || vq VR5 VR0 VR4 || VR0=DMV[R5-1] || nop || VR11=DMVB[R0]
  add R1 R2 R0 || vadd VR4 VR3 VR3 || vq VR5 VR0 VR4 || VR2=DMV[R5] || nop || VR12=DMVB[R0]
  ret || vq VR2 VR0 VR0 || nop || nop || nop || VR1=DMVB[R2]
  add R1 R2 R0 || vadd VR0 VR1 VR1 || nop || nop || nop || VR2=DMVB[R2]
```

© Synopsys, Inc. All rights reserved
Examples: Wireless Communication

“BoT” [1]
Configurable inner-modem processor
LTE(A) + 11ac + 11ad + WPAN + GPS + DVBT...

“FlexFEC” [2]
3-standard FEC engine
LDPC + Turbo + Viterbi

“BLOX” [1]
Single-function sliced accelerators
FFT / LDPC / Matrix inv.


BLOX
Single-Function Sliced Accelerators [1]

- Highly regular vector processors
  - In each SIMD lane, stack elementary operators, limited HW multiplexing
  - Low power, thanks to
    - Short active wires and modularity
    - Simple operators
    - Very wide register-files (asymmetric access)
  - Examples
    - FFT for 11ac
    - LDPC for 11ac
    - Matrix ops
    - C programmable (but requires C code refactoring)
Conclusions

• ASIP design tools introduce C programmability in SoC design
  – Better design reuse
  – Functional enhancements even after tapeout
  – Productivity increase by raising abstraction from RTL to C

• “Compiler-in-the-Loop” concept
  – Rapid architectural exploration
  – Highly differentiating architectures

• Full control on PPA (performance, power, area)

• Software development kit for end users is available automatically

• Same tool supports wide range of IP needs

• Royalty-free solutions