VLIW DSP Processor Design for Mobile Communication Applications

Contents crafted by
Dr. Christian Panis
Catena Radio Design
Agenda

- Trends in mobile communication
- Architectural core features with significant impact on performance
- Case study: 3a – a scalable VLIW architecture
- Design space exploration
- Challenges of scalability
- Summary
Trends in Mobile Communication

**TELECOM RULES:** Wireline, nomadic, and wireless technologies improve in a manner reminiscent of Moore’s Law. Soon, even slower communications channels like cellphones and radio modems will eclipse the capacity of early Ethernet, thanks to upcoming standards known as UMTS and MIMO, which will boost bandwidth by maximizing antenna usage. [Time axis shows dates of first use.]
Embedded systems emerging increasingly

- Bandwidth demands leads to significant increase in computational requirements
  - Trade-off: power dissipation vs. flexibility vs. performance

- Cost pressure, feature size, application space
  - Multistandard solutions

- Application-specific and customizable processors
How to Tackle the Problem?

Application specific processor architectures
- provides support for application specific requirements
- provides domain specific problem solutions
- provides trade-off power vs. area effort vs. flexibility

Domain Specific Processor Architectures

Things to be considered
- For each core a separate tooling/tool-chain?
- How to analyse the application specific requirements?
- How to analyse the gain compared with a standard core solution?
- How to deal with additional verification effort caused by flexibility?
Focus / Why VLIW?

**Focus**
- Embedded processing of lower communication layers

**Characteristica**
- Mix of traditional loop-centric DSP algorithms with control code

**load/store VLIW is one possible solution**
- Real time requirements for signal processing algorithms (+)
- High ILP support allows efficient execution of inner loops (+)
- Code density drawback of VLIW (-)
- Poor cache support (+/-)
Architectural Key Characteristika

- **Register file**
  - size, number of entries, structure
- **Data path(s)**
  - number, parallel availability, type
- **Memory ports / bandwidth**
  - number, data width, supported granularity
- **ISA, binary coding**
  - instruction mapping, binary coding, native instruction word size
- **Pipeline structure**
  - number of stages, exceptions
### Register file

size, number of entries, structure

<table>
<thead>
<tr>
<th></th>
<th>d1</th>
<th>d0</th>
<th>a0/l0</th>
</tr>
</thead>
<tbody>
<tr>
<td>d3</td>
<td></td>
<td>d2</td>
<td>a1/l1</td>
</tr>
<tr>
<td>d28</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a15/l15</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d29</td>
<td></td>
<td></td>
<td>a14/l14</td>
</tr>
<tr>
<td>a15/l15</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d31</td>
<td></td>
<td>d30</td>
<td></td>
</tr>
<tr>
<td>d28</td>
<td></td>
<td></td>
<td>a14/l14</td>
</tr>
<tr>
<td>a15/l15</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

D [0..31]
L [0..15]
A [0..15]

D [0..15]
L [0..15]
A [0..15]
Architectural Key Characteristika

- **Data path(s)**
  number, parallel availability, type of supported functions

---

**SIMD MUL**

```
+    ALU
|_   |_   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
|  |   |   |
Architectural Key Characteristika

- Memory ports / bandwidth
  number, data width, supported granularity

![Diagram showing connections between AGUs and ports](image-url)
Architectural Key Characteristika

- **ISA, binary coding**
  - Instruction mapping, binary coding, native instruction word size

  ![Diagram showing ISA operations and word sizes]

  - **ISA**
    - `add op1, op2`
    - `add op1, op2, op3`
    - `sub op1, op2`
    - `sub op1, op2, op3`
    - ...

  - **Word sizes**
    - 20 bits
    - 16 bits

  - **Number of instructions**
    - 710
    - 164
    - 2185

  - **Long instructions**
    - 710
    - 215
    - 1850
Architectural Key Characteristika

- **Pipeline structure**
  - number of stages, exceptions

Load/store operation

- Instruction Fetch
- Alignment
- Instruction Decode
- Execute 1
- Execute 2

  - read op1
  - address calculation

  - write back

Load/store operation accu

- Instruction Fetch
- Alignment
- Instruction Decode
- Execute 1
- Execute 2
- Execute 3

  - read op1
  - address calculation 1

  - read op2a
  - address calculation 2

  - write back 1
  - read op2b

  - write back 2
3a: case study

Key architectural aspects
- Modified Dual Harvard load-store architecture
- RISC instruction set
- xLIW (scalable long instruction word)
- Orthogonal register file and ISA
- Destination register based predicated execution
- Instruction buffer for power efficient inner loop processing
- Scalable and configurable core architecture
- Architecture specified considering an optimizing C-compiler
3a: case study

C-compiler aspects

- Load/store architecture
- Orthogonal ISA
- Large uniform register files
- Functionality stored in ISA
- Simple issue rules

- Mode dependent instructions
- Irregular instructions
- Implicit dependencies
- Modes for instruction sets
3a: case study

xLIW – scalable long instruction word

program memory

<table>
<thead>
<tr>
<th>inst0</th>
<th>inst1</th>
<th>inst2</th>
<th>inst3</th>
</tr>
</thead>
<tbody>
<tr>
<td>inst4</td>
<td>inst5</td>
<td>inst6</td>
<td>inst7</td>
</tr>
<tr>
<td>inst8</td>
<td>inst9</td>
<td>inst10</td>
<td>inst11</td>
</tr>
<tr>
<td>inst12</td>
<td>inst13</td>
<td>inst14</td>
<td>inst15</td>
</tr>
<tr>
<td>inst16</td>
<td>inst17</td>
<td>inst18</td>
<td>inst19</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>inst n-3</td>
<td>inst n-2</td>
<td>inst n-1</td>
<td>inst n</td>
</tr>
</tbody>
</table>

align unit

cycle m

<table>
<thead>
<tr>
<th>inst0</th>
<th>inst1</th>
</tr>
</thead>
<tbody>
<tr>
<td>inst2</td>
<td>inst3</td>
</tr>
<tr>
<td>inst4</td>
<td>inst5</td>
</tr>
<tr>
<td>inst6</td>
<td>inst7</td>
</tr>
</tbody>
</table>

LD/ST

CMP

PSEQ

decoder ports
3a: case study

Destination register based predicated execution

flag register file

load/store  load/store  arithmetic  arithmetic  predicated execution
3a: case study

Orthogonal register file incl. flag register file

- **Register File**: Contains data, address, and flag sections.
- **Data Register**: gb, d1, d0
- **Long Register**: l0
- **Accumulator Register**: a0
- **Address Register**: m0, r0
- **Modifier Register**: Connected to address register
3a: case study

"3-phase" pipeline

<table>
<thead>
<tr>
<th>Instruction Fetch</th>
<th>Alignment</th>
<th>Instruction Decode</th>
<th>Execute 1</th>
<th>Execute 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phase 1: fetch</td>
<td>Phase 2: decode</td>
<td>Phase 3: execute</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Things to be considered

- For each core a separate tooling/tool chain?
- How to analyse the application specific requirements?
- How to deal with additional verification effort caused by flexibility?
- How to analyse the gain compared with a standard core solution?

- Scaleable core architecture –
  - Adaptable to application specific requirements
  - Scalable in performance
- "one core" – "one tool chain"
Design Space Exploration – Static Analysis

- **Code Size**
  Measure how efficient the application can be mapped on a processor in term of required code space

- **Parallelism**
  Measure how efficient the application code can be mapped on a parallel architecture

- **Instruction histogram**
  Measure how frequent instructions are used during mapping of the application code onto the chosen ISA
Program memory fetch
Measure of efficient use of memory fetches, mainly influenced by pipelined processors and application code with low branch distance and high branch frequency

Execution count per bundle
Measure how often a certain execution bundle will be executed

Execution count per instruction
Measure how frequent a certain instruction will be executed
Design Space Exploration – Example
### Design Space Exploration – Statistics

#### Instructions

<table>
<thead>
<tr>
<th>Index</th>
<th>opcode</th>
<th>op1</th>
<th>op2</th>
<th>sync shift</th>
<th>sync src</th>
<th>sync dst</th>
<th>imm</th>
<th>inst</th>
<th>imm_idx</th>
<th>count</th>
<th>percent</th>
<th>cum %</th>
<th>cum %</th>
</tr>
</thead>
<tbody>
<tr>
<td>2169</td>
<td>13%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>LOAD_OFFSET</td>
<td>13%</td>
<td>13%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1751</td>
<td>10%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>COMPUTATIONAL_NO_OPERATION</td>
<td>23%</td>
<td>36%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1684</td>
<td>10%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>STORE_OFFSET</td>
<td>32%</td>
<td>63%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1026</td>
<td>6%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>MOVE_REGISTER</td>
<td>38%</td>
<td>71%</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Dynamic Instruction Counts

<table>
<thead>
<tr>
<th>Index</th>
<th>opcode</th>
<th>op1</th>
<th>op2</th>
<th>sync shift</th>
<th>sync src</th>
<th>sync dst</th>
<th>imm</th>
<th>instruction</th>
<th>imm_idx</th>
<th>count</th>
<th>percent</th>
<th>cum %</th>
<th>cum %</th>
</tr>
</thead>
<tbody>
<tr>
<td>101573978</td>
<td>10%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ADDITION</td>
<td>10%</td>
<td>10%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>95342213</td>
<td>10%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>PREDICATED_EXECUTION</td>
<td>20%</td>
<td>30%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>72279857</td>
<td>7%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>LOAD_OFFSET</td>
<td>27%</td>
<td>47%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>66670119</td>
<td>7%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>STORE_OFFSET</td>
<td>34%</td>
<td>81%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>62711869</td>
<td>6%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>LOAD</td>
<td>40%</td>
<td>120%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>53986000</td>
<td>5%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>INVERSION_OPERATION</td>
<td>45%</td>
<td>165%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>47918438</td>
<td>5%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>COMPUTATIONAL_NO_OPERATION</td>
<td>50%</td>
<td>215%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>47410313</td>
<td>5%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ARITHMETIC_SHIFT_RIGHT_IMMEDIATE</td>
<td>55%</td>
<td>370%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>35130283</td>
<td>4%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>STORE</td>
<td>58%</td>
<td>422%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>34684361</td>
<td>3%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CLEAR</td>
<td>62%</td>
<td>684%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32946383</td>
<td>3%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SUBTRACTION</td>
<td>65%</td>
<td>747%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31720118</td>
<td>3%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ADD_IMMEDIATE</td>
<td>68%</td>
<td>815%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31012492</td>
<td>3%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>COMPARE_LESS_THAN</td>
<td>71%</td>
<td>886%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29017052</td>
<td>3%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>MULTIPLY_AND_ACCUMULATE</td>
<td>74%</td>
<td>959%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27460037</td>
<td>3%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ADD_ADDR_REGISTER_IMMEDIATE</td>
<td>77%</td>
<td>1026%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24242852</td>
<td>2%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>MOVE_REGISTER</td>
<td>79%</td>
<td>1075%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23697128</td>
<td>2%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>MOVE_Register_DATA_REGISTER_ACCU_REGISTER</td>
<td>82%</td>
<td>1157%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16589826</td>
<td>2%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>OR_OPERATION</td>
<td>83%</td>
<td>1230%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16273273</td>
<td>2%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>LOGICAL_SHIFT</td>
<td>85%</td>
<td>1285%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15932397</td>
<td>2%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>MINIMUM</td>
<td>87%</td>
<td>1472%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15169027</td>
<td>2%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>MOVE_IMMEDIATE</td>
<td>89%</td>
<td>1561%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14803476</td>
<td>1%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>BRANCH_CONDITIONAL</td>
<td>90%</td>
<td>1651%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13259175</td>
<td>1%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>MULTIPLE</td>
<td>91%</td>
<td>1742%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12676506</td>
<td>1%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>LOGICAL_SHIFT_IMMEDIATE</td>
<td>92%</td>
<td>1834%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10273608</td>
<td>1%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AND_OPERATION_IMMEDIATE</td>
<td>93%</td>
<td>1927%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9290700</td>
<td>1%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>PREDICATED_EXECUTION_LONG</td>
<td>94%</td>
<td>2021%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>79002923</td>
<td>1%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>VECTOR_MULTIPLY</td>
<td>95%</td>
<td>2116%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6123816</td>
<td>1%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NEGATION</td>
<td>98%</td>
<td>2125%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6091735</td>
<td>1%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>MULTIPLE</td>
<td>99%</td>
<td>2126%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5577068</td>
<td>1%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>MOVE_IMMEDIATE_UNSIGNED</td>
<td>97%</td>
<td>2127%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5030512</td>
<td>1%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>COMPLEX_MULTIPLY</td>
<td>97%</td>
<td>2128%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4276972</td>
<td>0%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>MULTIPLE_AND_ACCUMULATE_IMMEDIATE</td>
<td>98%</td>
<td>2129%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3850143</td>
<td>0%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ARITHMETIC_SHIFT_RIGHT</td>
<td>98%</td>
<td>2129%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2721230</td>
<td>0%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AND_OPERATION</td>
<td>98%</td>
<td>2129%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2715699</td>
<td>0%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ADDITION_SATURATE</td>
<td>99%</td>
<td>2129%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2157497</td>
<td>0%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>COMPARE_LESS_THAN_IMMEDIATE</td>
<td>99%</td>
<td>2129%</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
3a: case study

**Things to be considered**
- For each core a separate tooling/tool chain?
- How to analyse the application specific requirements?
- How to analyse the gain compared with a standard core solution?
- How to deal with additional verification effort caused by flexibility?

- Application code analysis
  - Detailed static & dynamic analysis
  - Quantitative analysis of different core based platforms
  - Balance different core features against area/power consumption
  - Quantitative support to optimize HW/SW partitioning
  - Identify "Hot Spots"
3a: case study

Benchmarking?

Does the application requirements fit to standard benchmarks?

- Application Benchmarking: Benchmarking of the Target Architecture
Design Space Exploration

- MSC: 39% (1%) 39.7 3x
- SDC: 1% (0%) 1.5 45x
- f_DrmRcvProcess: 100% (2%) 102.5 75x
- f_demProcessBlock: 57% (2%) 58.2 75x
- f_postFFTProcess: 3% (1%) 2.9 45x
- f_preFFTProcess: 8% (2%) 8.2 45x
- f_rsmprResample: 7% 6.7 75x
- f_vdDecode: 31% 31.3 15x
- f_vdWeightMetrics: 1% 1.1 15x
- f_vdDecode: 1% 1.2 3x
- FFT: 2% 1.6 45x
- SDFT: 15% (15%) 15.7 45x
- f_channelProcess: 21% (5%) 21.0 45x
- channel_EQ: 8% (5%) 7.8 45x
- f_trfFindRefs: 2% 2.1 45x
- f_fcCorrect: 2% 1.8 45x
- f_tpCorrelateTime: 1% 1.0 45x
- f_tpRestore: 2% 2.0 45x
- f_tpSmother: 1% 1.2 45x

Other notes:
- ARM - B10_6411_warp_1xWFS1x12
- M cycles / second
- Sliding average over 75 slices
- Time slice = f_DrmRcvProcess (62.5 Hz)
- Folded nodes with value < 0.01 * total
Design Space Exploration

- **f_DrmRcvProcess**: 100% (3%) 51.0 75x
- **f_demProcessBlock**: 62% (1%) 31.7 72x
- **MSC**: 35% (0%) 17.7 3x
- **cell_deinterleaver**: 1% 0.6 3x
- **f_micdDecode**: 33% (3%) 16.9 3x
- **f_vdDecode**: 27% 13.7 15x
- **f_vdWeightMetrics**: 2% 0.8 15x
- **CSI**: 4% 1.9 90x
- **IR**: 14% (2%) 7.1 45x
- **channel_EQ**: 5% 2.5 45x
- **f_irFindInpRespCoM**: 1% 0.6 45x
- **f_irSearch**: 2% 1.2 45x
- **f_trfFindRef**: 2% 1.2 45x
- **f_fcCorrect**: 1% 0.5 45x
- **f_tpRestore**: 2% 0.8 45x

3a = B10_0411 unwrap_100MBPS_1q12
M cycles / second
sliding average over 75 slices
time slice = f_DrmRcvProcess (62.5 Hz)
folded nodes with value < 0.01 * total
Design Space Exploration

Things to be considered
- For each core a separate tooling/tool chain?
- How to analyse the application specific requirements?
- How to analyse the gain compared with a standard core solution?
- How to deal with additional verification effort caused by flexibility?

- Analysis application code on "function" level
- Compare MIPS/Memory requirements for different application setup’s and for different core architecture
- Benchmarking for the target architecture
Challenges of scalability

Verification effort versus Flexibility

- XML based configuration file
- Binary code generator
- Documentation generator/adaptation
- HW code generator
- Testcase generator
Summary

- Application Specific Processors allows to meet area and power dissipation requirements in SoC’s for mobile communication platforms.

- Multistandard requirement leads to domain specific processor architectures.

- "one core" – "one tool chain".

- Design Space Exploration is required to analyse domain specific requirements on core subsystem.