The Configurable Processor Company

The End of ISA Design
*Power Tools for Optimal Processor Generation*

Hot Chips 16
August 22-24, 2004

David Goodwin
Chief Engineer

---

**Instruction Set Architecture**

- Contract between software and hardware
- Semiconductor and software progress cause ISA design to evolve
- But always constrained by conflicting goals
  - Generality for future (unknown) applications
  - Improve performance, code-size, area, power for known applications
Application-Specific ISA

- High performance requirements
  - Wide variety of specialized ISAs

- Application must be specifically targeted to exploit ISA
  - C/C++ compiler cannot efficiently generate code for peculiarities of ISA
  - Developer must often use C intrinsic functions or assembly

- ASIP better than a coprocessor
  - No data transfer / communication overhead
  - No function unit or memory duplication
  - ASIP based on general core

Meta-ISA: A Better Target for Application Development

- XPRES Compiler: new commercially available tool for automatic ASIP generation
- Exposes meta-ISA that supports varying degrees of
  - Instruction-level parallelism
  - Data parallelism
  - Pipeline parallelism
- Generates variety of ISAs to exploit parallelism
**Meta-ISA Parallelism**

**Maximum parallelism:** $L \times M \times N$

**Meta-ISA:**
- VLIW
  - Instruction size
  - Number of operation slots per instruction
  - Operation mix in each slot
- SIMD
  - Types of vector operations
  - Number of vector elements
  - Number and size of vector register files
- Fusion
  - Number and type of composing operations
  - Number of inputs and outputs
  - Latency

**Targeting the Meta-ISA**

- **Attributes of Meta-ISA**
  - Exploits data parallelism with vector (SIMD) operations
  - Exploits instruction parallelism with multiple issue instructions
  - Exploits pipeline parallelism with long latency fused operations

- **Application developers focus on clean algorithm implementation**
  - Expose parallelism
  - Don’t tune for peculiarities of specific ISA
  - No intrinsics or assembly

- **Application code more directly implements algorithm**
  - Easier implementation and maintenance
  - Easier debugging
XPRES Backend: Implementing an Application-Specific Processor

XPRES uses Xtensa processor generator flow as backend for creating ASIPs

- Application-Specific ISA
  - Configuration selection and custom-instruction description (TIE)
- Xtensa Processor Generator
- Use standard techniques and libraries for any IC fabrication process
- Complete Hardware Design
  - Verilog/VHDL RTL, EDA scripts, test suite

* TIE: Tensilica Instruction Extension Language

Implementing a Family of Meta-ISA ASIPs Using XPRES

- Application Source Code
- XPRES Compiler
- ASIP Implementations

```c
int main()
{
    int i;
    short c[100];
    for (i=0;i<100;i++)
    {
```

- Generation
- Use

```c
int main()
{
    int i;
    short c[100];
    for (i=0;i<100;i++)
    {
```

- Customized Software Tools
- Application Binary

* TIE: Tensilica Instruction Extension Language

© 2004, Tensilica, Inc.
XPRES Compiler Flow

- **Profile / Analyze application**
  - Identify performance critical functions / loops
  - Collect operation mix, dependence graphs
  - Provide feedback to user – code transformations to better target meta-ISA (esp. vectorization)

- **Generate sets of ISA extensions**
  - Each set implements some dimension of meta-ISA
  - Evaluate each set across all functions / loops
  - Performance and cost estimates allow exploration of large design space

- **Collect ISA extensions that together provide maximum performance improvement**
  - Each collection of ISA extensions forms ASIP
  - Generate family of ASIPs for varying hardware cost budget

XPRES Output Showing Application Performance vs. Hardware Cost
Example: Family of ASIPs

```c
for (i=0; i<n; i++)
    c[i] = (a[i] * b[i]) >> 4
```

Total core size (130 nm) =
0.25 mm² (base Xtensa) +
gates / 125,000 mm²

VL 8, IW 3, 2L/S
VL 4, IW 3, 2L/S
VL 4, IW 2, 2L/S
VL 2, IW 3, 2L/S
VL 2, IW 3, 2L/S and VL 4, 1 L/S
VL 2, IW 2, 1L/S
VL 2, IW 2, 1L/S
VL 2, 1L/S

VL – vector length
IW – issue width
L/S – memory units

Instruction-Level Parallelism

Meta-ISA supports ILP with multiple-issue
- 32-bit or 64-bit instructions
- Each instruction can contain 1-15 slots
- Each slot can contain arbitrary mix of operations

Implemented in Xtensa using FLIX (Flexible Length Instruction Xtensions)
- Similar to VLIW without code size increase

Xtensa C/C++ compiler exploits automatically
- Software pipelining for loops
- List scheduler for other code
XPRES: Instruction Parallelism (exploited using FLIX)

Original C Code

```
for (i=0; i<n; i++)
    c[i] = (a[i] * b[i]) >> 4
```

64-bit 3-issue FLIX

```
loop:
    { l8ui   a4,a11,12; addi  a13,a13,4 }  
    { l8ui   a2,a9,12; mul16u a3,a3,a2 }  
    { l8ui   a14,a9,4; srai  a12,a14,4}  
    ...  
```

Generated Assembly

(1 iteration in 3 cycles, 2.6x)

64-bit 2-issue FLIX

```
loop:
    { l8ui a8,a9,0; mul16u a12,a10,a8; addi a9,a9,1 }  
    { l8ui a10,a11,0; nop; addi a11,a11,1 }  
    { a8i a13,a14,508; srai a13,a12,4; addi a14,a14,1 }  
```

(2 iterations in 8 cycles, 2x)

---

Results: Instruction Parallelism

![Graph showing application speedup vs. estimated area for various applications like idwt, autocor, rgbcmyk, fft-radix4, gsm encode, gsm decode, compress95, djpeg, and m88ksim. The x-axis represents estimated area (gates), and the y-axis represents application speedup. The graph illustrates the trade-off between area and speed for different applications, with some showing significant speed improvements at the cost of area.](image)
**Data Parallelism**

- **Meta-ISA supports data parallelism with SIMD operations**
  - Vectors of length 2, 4, 8, and 16 (up to 256 bits)
  - Support for unaligned vector loads and stores
  - Vector C–operators, MIN, MAX, ABS, reductions
  - Vector user-defined operations

- **Xtensa C/C++ compiler exploits automatically**
  - Automatic loop vectorization
  - Source attributes and compiler options specify aliasing and alignment directives to enable additional vectorization opportunities

---

**XPRES: Data Parallelism (exploited using SIMD)**

**Original C Code**

```c
for (i=0; i<n; i++)
c[i] = (a[i] * b[i]) >> 4
```

**Vector Length 8**

```c
regfile vrs8x8 64 4 v
regfile vr16x8 128 2 x
operation ashr16x8 { out vr16x8 a, _ }
```

**Vector Length 2**

```c
regfile vrs8x2 16 4 v
regfile vr16x2 32 2 x
operation ashr16x2 { out vr16x2 a, _ }
```

**Generated Assembly**

(8 iterations in 6 cycles, 10.6x)

```
loop:
liu8x8    v0, a, 8
ashr16x8  x0, x, 4
liu8x8    v1, a, 8
cvts16x8sr8x8s v2, x0
mpy8r16x8u x0, v0, v1
siu8x8    v2, a10, 8
```
Exploiting Pipeline Parallelism

- Meta-ISA supports pipeline parallelism with fused operations
  - Fused operation composed of two or more other operations, plus possibly constant values
  - Latency of fused operation usually less than combined latency of composing operations
  - Limits on input/output operands, number of composing operations, hardware cost, max latency, etc.
  - Graphical support for manual fused operation generation

- Tradeoff performance and generality
  - Performance: large fusions w/ fixed constants
  - Generality: smaller fusions, fewer fixed constants

- Xtensa C/C++ compiler exploits automatically
  - Fused operations automatically replace sequences of composing operations
Original C Code

```c
for (i=0; i<n; i++)
    c[i] = (a[i] * b[i]) >> 4
```

Performance
- Decrease instruction count
- Decrease computation latency

Hardware cost
- Logic to share function units
- Register file ports

Generated Assembly
(1 iteration in 5 cycles, 1.6x)
```
loop:
  l8ui    a12,a11,0
  l8ui    a13,a10,0
  addi.n  a10,a10,1
  addi.n  a11,a11,1
  fusion.mul16u.srai.s8i.addi a9,a12,a13
```

Results: Pipeline Parallelism
Exploiting Multiple Types of Parallelism

XPRES combines FLIX, SIMD, and Fusion

Original C Code

```
for (i=0; i<n; i++)
    c[i] = (a[i] * b[i]) >> 4
```

FLIX and Vectorization (16 iterations in 4 cycles, 32x)

```
{ cvt16x8sr8x8s v2,x1; liu8x8 v0,a8,8; mpy8r16x8u x2,v1,v2 }
{ si8x8 v2,a10,8; liu8x8 v1,a9,8; ashr16x8 v1,v0,4 }
{ siu8x8 v3,a10,16; liu8x8 v2,a9,8; ashr16x8 v3,v2,4 }  
{ cvt16x8sr8x8s v3,x3; liu8x8 v1,a8,8; mpy8r16x8 x0,v0,v1 }
```

FLIX, Vectorization, and Fusion (16 iterations in 3 cycles, 42.6x)

```
{ si8x8 v2,a10,8; liu8x8 v0,a8,8; fusion.mpy.ashri.cvtx8 v2,v0,v1 }
{ siu8x8 v5,a10,16; liu8x8 v1,a9,8; fusion.mpy.ashri.cvtx8 v5,v3,v4 }
{ liu8x8 v3,a8,8; liu8x8 v4,a9,8; nop }
```

Results: Instruction and Data Parallelism

- `idwt`
- `autocor`
- `rgbcmymk`
- `fft-radix4`
- `gsm encode`
- `gsm decode`
Results: Instruction, Data, and Pipeline Parallelism

Conclusion

- XPRES exposes meta-ISA, a better target for embedded application development
  - Instruction, data, and pipeline parallelism
  - Developers focus on clean algorithm development, not particulars of specific ISA

- Implementations of meta-ISA span range of hardware costs
  - Early feedback on potential performance and hardware cost for system applications / tasks
  - Quick exploration of performance / cost tradeoffs
Manual Fusion Selection

Dataflow Graphs (1 of 1)

Manually Fused Graphs

Vector Length: Scalar

Evaluate Add Clear

Est. Area: <none>  Est. Latency: <none>

Name

Fusion Evaluation

The manual fusion is valid.

Estimated Area: 474 gates

Estimated Latency: 1 cycle