Long Words and Wide Ports: Reinventing the Configurable Processor

Dhanendra Jani
Gulbin Ezer
James Kim

SOC Design Challenges

1. Ever more complex requirements...
2. ...at ever higher performance
3. ...in ever tighter budgets

Source: STMicroelectronics
Tensilica’s Solution: Next Generation Configurable Processor

**Xtensa LX innovations:**
- High computation performance with long instruction words
- High I/O bandwidth with new type of wide processor I/O port
- Low-power implementations with fine-grained clock gating

**Overall impact is three-fold:**
- Significant increase in application data throughput
- Broader generality of application-specific instruction sets
- Greater automation in creation of optimized hardware and software

Xtensa LX opens up ISA automation to a wider audience of engineers

---

Xtensa LX Continues Automated HW+SW Generation Tradition

Original C/C++ Code

Calculates millions of possible extensions using SIMD/vector operations, operator fusion and parallel execution

Configuration Selection

Designer selects “best” configuration

Xtensa Processor Generator

Synthesizeable RTL

Scheduling assembler, C/C++ compiler, ISS, debuggers, XTMP System modeling API, BFM, RTOS

Embedded Code

XPRES Compiler

Customized Software Tools

Embedded Code
Traditional options for improving instruction-level parallelism
- Superscalar and VLIW

Issues for user-extensible embedded processors
- Superscalar: high hardware complexity and large gate count
- VLIW: code bloat due to instruction-length and encoding inflexibility

Xtensa LX’s FLIX (Flexible Length Instruction eXtensions) technology
- Allows multiple instruction lengths which can be freely intermixed
- Helps compiler achieve significantly better code size
- Can reduce bus bandwidth and power dissipation
Instruction Set Definition using FLIX

- Xtensa base ISA provides 16- and 24-bit instructions
- FLIX technology introduces long (32- or 64-bit) words
- 16-, 24-bit and FLIX instructions can be intermixed
- FLIX instructions can define multiple operation slots
  - Each independently encoded and scheduled
- Designers assign Xtensa ISA or own operations to each slot

<table>
<thead>
<tr>
<th>63</th>
<th>Operation 1</th>
<th>Operation 2</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example 1 – 64b Instruction Format, shows 2 Big Slots that use deep register files or large number of operands

<table>
<thead>
<tr>
<th>63</th>
<th>Operation 1</th>
<th>Operation 2</th>
<th>Op 3</th>
<th>Op 4</th>
<th>Operation 5</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example 2 – 64b Instruction Format, shows 2 Small Slots that use shallow register files or implicit operands

<table>
<thead>
<tr>
<th>0x0</th>
<th>64b instr</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x8</td>
<td>24b</td>
</tr>
<tr>
<td>0x10</td>
<td>64b (con't)</td>
</tr>
</tbody>
</table>

Example 3 – shows how instructions of various lengths can be freely intermixed without code bloat

Tensilica Instruction Extension (TIE)
Support for FLIX

- TIE language enables concise description of FLIX instructions
  - Designers only specify new state and operation semantics
- TIE Compiler automatically generates complex multiple operation encoding and scheduling information for compiler/assembler
- Processor implementations with high data parallelism possible
  - Up to 7 instruction formats, 15 slots per format, 10 operands per operation
  - Up to 2 load/store units and 32/64/128-bit memory datapath width support

TIE Code Example: FLIX Instruction Definition

```plaintext
format  x64  64 {slotMAC, slotALU}

slot_opcodes slotMAC  {MUL16U, CustMAC}
slot_opcodes slotALU  {ADD, SUB, MIN, MAX}

state  ACC  40
operation CustMAC  {in AR m0, in AR m1} {inout ACC}
  { assign ACC = ACC + m0[15:0] * m1[15:0] }
```
**Hardware Resource Sharing Flexibility**

- **24-bit instruction format**
  - slot inst
  - slot load
  - slot mulA
  - slot ptr

- **64-bit instruction format w/ 3-slots**
  - exec slot 0
  - exec slot 1
  - exec slot 2

- **64-bit instruction format w/ 2-slots**
  - slot store
  - slot mulB

- **Semantics or slot shared functions**
  - Share hardware within an execution slot between all slots that map to it

- **Globally shared functions**
  - Shared by all slots, creates some resource conflicts, i.e., it is illegal to bundle multiple operations that use same globally shared function

**FLIX Micro-architecture Support**

- **Instruction Memory**
  - 0x0
  - 0x8
  - 0x10

- **Data Memory**
  - Variable depth pipeline
  - Optional memory stage
  - TIE generated
  - Portion of core configurable
  - ISA implemented in TIE
  - Portion of core configurable
  - ISA implemented in RTL

- **Interlock**
- **Bypass**
- **Commit Point**
- **Exception and Interrupt Control**

- **Variable long latency pipeline, up to 32 stages**
Xtensa Core ISA Implementation in TIE

- Core ISA, originally implemented in RTL, re-implemented in TIE
- TIE language evolved to allow full processor description:
  - exceptions and interrupts
  - branches
  - register window operands
  - variable commit points
  - variable pipeline support
- TIE Compiler automatically generates RTL and ISS for the processor datapath and control

Comparable Timing and Area

Timing Comparison

Area Comparison

Power Comparison

Reduced Power due to Fine-grained Clock Gating

Simple FLIX Example

3 line TIE description generates “static superscalar” machine
- Many variations possible

2 slots of Xtensa base ISA Operations
- No new designer-defined functional units

XCC compiler statically pairs base ISA operations

ACTUAL TIE CODE FOR STATIC SUPERSCALAR MACHINE

```
format f64 64 | slot0, slot1

slot_opcodes slot0 { L32I, L32B, L16S, L16U, L8U, S32I, S16I, S8I, MUL16S, MUL16U, MULL, ABS, MAX, MIN, MINU, NEG, EXTUI, SEXT, AND, OR, XOR, ADD, ADDI, ADDU, ADDUX, SUB, SUBU, SUBUX, SUBX2, SUBX4, SUBX8, SLL, SLLI, SRA, SRAI, SRC, SRL, SRLI, J, JX, BNE, BAEQ, BNEZ, BEQZ, BLT, BGE, BGEZ, BLTZ, BGEU, BLTU, MOV, MOVU, MOVN, MOVN }
```

```
slot_opcodes slot1 { ABS, NEG, EXTUI, SEXT, ADD, ADDI, ADDU, ADDUX, SUB, SUBU, SUBUX, SUBX2, SUBX4, SUBX8, SLL, SLLI, SRA, SRAI, SRC, SRL, SRLI, J, JX, BNE, BAEQ, BNEZ, BEQZ, BLT, BGE, BGEZ, BLTZ, BGEU, BLTU, MOV, MOVU, MOVN, MOVN }
```

RESULTS

- 46% average performance increase compared to base-case (scalar) Xtensa LX on EEMBC Consumer Suite
- 12,500 gates more than base Xtensa LX
  - Approx 40K total gates
  - Includes 5-port base register file
- 325 MHz in 0.13 TSMC LV (worst case)
- Power: 160 µW/MHz
### Vectra LX: High Performance DSP Engine

- Based on FLIX Technology
  - General DSP instruction set >210 ALU, MAC and Load/Store instructions
  - 3 vector operations per 64-bit instruction bundle. 2 load store units
- User Extensible: Designer-defined instruction extensions can be added
- Complete software support
  - Pipeline accurate ISS
  - Vectorizing compiler
  - Scheduling assembler
- Configuration option to Xtensa LX processor
  - Additional 200K to 250K gates

**130 nm Technology**

<table>
<thead>
<tr>
<th>Frequency</th>
<th>Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>370MHz</td>
<td>0.50mW/MHz</td>
</tr>
<tr>
<td>100MHz</td>
<td>0.25mW/MHz</td>
</tr>
</tbody>
</table>

### BDTI Benchmark: Xtensa LX Processor with Vectra LX engine

**Xtensa LX Configuration includes Vectra LX DSP Engine + 11 custom extensions**

- Viterbi trellis decode (3)
- Viterbi trellis traceback (2)
- Bit stream unpacking (2)
- Multiply intensive filter (4) - add a total of 30K additional gates

**BDTIsimMark2000™ Scores**

- Xtensa LX (370 MHz in 0.13um)
- CEVA-X 1620 (450 MHz in 0.13um)
- StarCore SC1400 (305 MHz in 0.13um)
- ARM9E (250 MHz in 0.13um)

The BDTIsimMark2000™ is a summary measure of DSP speed. See www.BDTI.com for info. Scores © 2004 BDTI.
Xtensa LX Wide Ports
10-100x bandwidth advantage

32 bit processor interface: < 1 data byte/cycle sustained

Wide processor ports: 10-100+ data bytes/cycle sustained

Xtensa LX adds wide I/O ports to address this gap
• Provides a flexible way for execution units to directly access external devices
• Enables low-latency, high bandwidth communication
• Enables Xtensa LX to take on data-intensive roles previously reserved for hardwired RTL blocks

TIE Support for I/O Ports

TIE language supports 3 type of I/O ports
• Control
  • Input Wire: Allows TIE instruction to directly sample an external signal
  • Export State: Allows TIE internal state to be visible as an external signal
• Data
  • Queue: Allows TIE instruction to directly read/write an external queue

Per processor I/O bandwidth is unlimited
• Up to 1024 ports of up to 1024 signals per port are allowed

TIE language enables concise descriptions
• Designers only specify queue type and width

TIE Compiler automatically generates:
• Interface signals to connect Xtensa LX to external logic
• Stall logic when reading from an empty input queue or writing to a full output queue
• Logic to handle the speculative nature of a processor pipeline
Simple RTL Replacement Example:
RTL Implementation of Algorithm

InData1[255:0]
InData1_Empty
InData1_PopReq
InData2[255:0]
InData2_Empty
InData2_PopReq
Count[7:0]
OutData[255:0]
OutData_PushReq
OutData_Full

 Done

Get count
Add input data
Iterate

Done

Flow Control & Speculation Handling (FCS)

Pipeline Control

States & Register Files

[Other] Execution Units

void process_queue()
{
   // get count from input wire
   int count = GETCOUNT();
   // perform queue operations
   for (int i = 0; i < count; i++)
   {
      QADDA();
      ...
   }
   // set done and exit
   #pragma flush
   WR_DONE();
}

import_wire COUNT;
operation GETCOUNT (out AR val)
{assign val = COUNT;}

Simple RTL Replacement Example:
Xtensa I/O Ports Implementation

void process_queue()
{
   // get count from input wire
   int count = GETCOUNT();
   // perform queue operations
   for (int i = 0; i < count; i++)
   {
      QADDA();
      ...
   }
   // set done and exit
   #pragma flush
   WR_DONE();
}

pipe_i ut2[256 in
pipe_in[256 in
pipe_out[256 out
operation QADDA()
{In InData1, In InData2, out OutData}
{assign OutData = InData1 + InData2;}

Flow Control & Speculation Handling (FCS)

Pipeline Control

States & Register Files

[Other] Execution Units

Done

state DONE 1
pragmas write export
**SOC Design Example: Secure Digital Audio Recorder**

- Portable device to record/store/playback compressed/encrypted audio
  - AES - 128 bit open encryption standard
  - AAC - MPEG4 audio compression algorithm

---

**Two Implementation Options**

- **Option 1: Bus Architecture**
  - 115M cycles for 1 sec audio
  - All data traffic shares the common system bus, resulting in memory bottleneck

- **Option 2: Dataflow Architecture**
  - 60M cycles for 1 sec audio
  - Direct queue connections eliminate classical memory bus bottlenecks
Summary

- Xtensa LX provides the foundation to increase ISA automation by enabling
  - High compute throughput with long instruction words
  - High I/O bandwidth with wide ports
- The new features make Xtensa LX ideal for data intensive applications that require low power dissipation and easy programming from C/C++

Xtensa LX
Small 7-stage pipe configuration
0.16mm² in 90nm