Hardware/Software Instruction Set Configurability for System-on-Chip Processors

Albert Wang, Chris Rowen, Dror Maydan, Earl Killia
Landscape of reconfigurable computing

Optimality/integration
(e.g. mW, $)

Optimality/integration

ASIC

Instruction-set Configurable Processor

Δ ~10x

Δ ~10x

Flexibility/modularity
(e.g. time-to-market)

Δ ~10x

FPGA

FPGA + Processor

General Processor
Computing using temporal connection

**Processor Solution**

<table>
<thead>
<tr>
<th>Memory (Program)</th>
<th>Control</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="Registers" alt="Image" /></td>
<td><img src="Datapath" alt="Image" /></td>
</tr>
</tbody>
</table>

**Table: Processor**

<table>
<thead>
<tr>
<th>Correct</th>
<th>Efficient</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>X</td>
</tr>
</tbody>
</table>
Computing using spatial connection

Processor Solution

Memory (Program)  Control  Registers  Datapath

ASIC Solution

FSM  Storage

Correct  Efficient

ASIC  X  ✓
Configurable Processors: best of both

Processor with Application-specific Instructions

Processor Solutions

ASIC Solutions

Memory (Program)

Control

Registers

Datapath

FSM

Storage

Processor

ASIC

Correct

Efficient
Outline

- Configurable processor solution
  - Xtensa™ processor Architecture
  - Instruction extension automation
  - Software development tools
- An Example
- Results
- Summary
Conventional Architecture

- More registers
- More FU's
- Deeper pipeline
- Bypass/forward
Conventional Architecture - cont.

• More FU’s
Conventional Architecture – cont.

- More FU’s
- More registers
Conventional Architecture – cont.

- More registers
- More FU’s
- Deeper pipeline
Conventional Architecture – cont.

- More registers
- More FU’s
- Deeper pipeline
- Bypass/forward
Problem with fixed processor:

- Waste silicon
  - There is no universal extensions, or even one for each application class
- Not fast enough, compared with hardware implementation
- Waste power

The Tensilica solution:

- Small core processor
- Allow easy and efficient application-specific instruction extensions
Xtensa Architecture – Base

- Good performance
  - Comparable to any embedded 32-bit RISC
- Good code density
  - Much better than 32-bit RISC
  - Use 16b/24b instructions
- Small
  - .7mm² in .18
- Low power
  - .37mw / MHz
- Easy extension
  - With Tensilica Instruction Extension (TIE) language – ISA level
- Efficient extension
  - TIE compiler generates efficient pipelined implementation
  - TIE compiler extends all software development tools
TIE language - opcode

Decoder

Control

RF0

FU0

Opcode

Opcode MAC op2=5 CUST0
TIE Language – regfile / state

- Opcode
- Register file / State

Decoder

RF0

S0

… as needed

FU0

Control

state ACC 40
TIE Language – semantics

Decoder

Control

Source routing

Result routing

• Opcode
• Register file / state
• semantics

**semantic** sem1 {MAC} {assign ACCL=ACCL+ars[16:0]*art[15:0];}
TIE Language – iclass

Decoder

Source routing

Control

Result routing

Opcode
Register file / state
Semantics
Instruction class

iclass cl \{MAC\} \{in ars, in art\} \{inout ACC\}
TIE Language - schedule

```plaintext
schedule s1 {MAC}{use ars 1; use art 1; use ACC 2; def ACC 2;}
```
**A Complete Example – parallel MAC**

**opcode** PMAC op2=0 CUST0  
**state** ACC1 40  
**state** ACC2 40  
**iclass** rr {PMAC}{in ars, in art}{inout ACC1, inout ACC2}  
**semantic** pmac_sem {PMAC} {  
    assign ACC1 = ACC1 + ars[15:0] * art[15:0];  
    assign ACC2 = ACC2 + ars[31:16] * art[31:16];  
}  
**schedule** pmac_schd {PMAC} {  
    use ars 1; use art 1;  
    use ACC1 2; use ACC2 2;  
    def ACC1 2; def ACC2 2;  
}
Select processor options

********
****
********
***

Describe new instructions

Using the Xtensa processor generator, create...

In Minutes!

Tailored, synthesizable HDL uP core

Customized Compiler, Assembler, Linker, Debugger, Simulator
Productivity Gain – Software Tools

Select processor options

******
****
*******
***

Describe new instructions

Using the Xtensa processor generator, create...

Tailored, synthesizable HDL uP core

Customized Compiler, Assembler, Linker, Debugger, Simulator
Software Support – Assembler

• Assembler
  Loop a2, .L1
  l16si a10, a3, 0
  l16si a11, a3, 2
  addi.n a3, a3, 2
  PMAC a10, a11
  .L1:

• Custom data type
• Register allocation
• Code Scheduling
• RTOS
• Simulator/debugger
Software Support – custom data type

- Assembler
- Custom data type
- Register allocation
- Code Scheduling
- RTOS
- Simulator/debugger

C Code:
```
sat_int x, y, z;
z = sat_add(x, y);
```
• Assembler
• Custom data type
• Register allocation

Spilling around a call:

sat_add   s3, s1, s2
sat_store  s3, a1, 0
call18    foo
sat_load  s3, a1, 0

• Code Scheduling
• RTOS
• Simulator/debugger
Assembler
Custom data type
Register allocation
Code Scheduling

\[
t = \text{sat\_mult}(x, y);
\]
\[
z = \text{sat\_add}(z, t);
\]
\[
t2 = \text{sat\_mult}(x2, y2);
\]

sat\_mult s3, s1, s2
sat\_mult s6, s5, s4
sat\_add s7, s7, s3

RTOS
Simulator/debugger
Software Support - RTOS

- Assembler
- Custom data type
- Register allocation
- Code Scheduling
- RTOS

- Simulator/debugger
Software Support – simulator/debugger

- Assembler
- Custom data type
- Register allocation
- Code Scheduling
- RTOS
- Simulator/debugger

```
gdb> break ...
gdb> cont
```
```
gdb> step
gdb> display ...
```
Outline

- Configurable processors
  - Architecture
  - Instruction extension
  - Software support
- An Example
- Results
- Summary
Data Encryption Standard (DES)

**Initial step**

\[(R, L) = \text{Initial\_permutation}(\text{Din}_{64})\]

**Iterate 16 times**

**Key generation**

\[(C, D) = \text{PC1}(k)\]

n = rotate\_amount (function of iteration count)

\[C = \text{rotate\_right}(C, n)\]

\[D = \text{rotate\_right}(D, n)\]

\[K = \text{PC2}(D, C)\]

**Encryption**

\[R_{i+1} = L_i \oplus \text{Permutation ( S\_Box ( K \oplus \text{Expansion ( R ) } ) )}\]

\[L_{i+1} = R_i\]

**Final step**

\[\text{Dout}_{64} = \text{Final\_permutation}(L, R)\]
static unsigned permute(
    unsigned char *table,
    int n,
    unsigned hi,
    unsigned lo)
{
    int ib, ob;
    unsigned out = 0;
    for (ob = 0; ob < n; ob++) {
        ib = table[ob] - 1;
        if (ib >= 32) {
            if (hi & (1 << (ib-32))) out |= 1 << ob;
        } else {
            if (lo & (1 << ib)) out |= 1 << ob;
        }
    }
    return out;
}
static unsigned permute(
    unsigned char *table,
    int n,
    unsigned hi,
    unsigned lo)
{
    int ib, ob;
    unsigned out = 0;
    for (ob = 0; ob < n; ob++) {
        ib = table[ob] - 1;
        if (ib >= 32) {
            if (hi & (1 << (ib-32))) out |= 1 << ob;
        } else {
            if (lo & (1 << ib)) out |= 1 << ob;
        }
    }
    return out;
}
DES: Hardware Implementation

- Initial Permutation
- Expansion Permutation
- S Boxes
- P Permutation
- Key Generation
- State Machine
- Final Permutation
DES: Hardware Implementation

Initial Permutation

Expansion Permutation

S Boxes

P Permutation

Final Permutation

Key Generation

State Machine

Complicated control logic!
DES: SETDATA instruction

SETDATA ars, art

Initial Permutation

Expansion Permutation

S Boxes

P Permutation

Final Permutation

Key Generation

State Machine
DES: SETKEY instruction

- Initial Permutation
- Expansion Permutation
- S Boxes
- P Permutation
- Final Permutation
- Key Generation
- State Machine

SETKEY instruction
DES: DES instruction

- Initial Permutation
- Expansion Permutation
- S Boxes
- P Permutation
- Final Permutation
- Key Generation
- State Machine

DES immediate
DES: GETDATA instruction

GETDATA ars, hilo
DES: Putting it together

GETDATA  ars, hilo

SETDATA  ars, art

SETKEY  ars, art

DES  immediate
DES: Improved Program

Encryption

```
SETKEY(K_hi, K_lo);
for (;;) {
  ... /* read data */
  SETDATA(D_hi, D_lo);
  DES(ENCRYPT1);
  DES(ENCRYPT1);
  DES(ENCRYPT2);
  DES(ENCRYPT2);
  DES(ENCRYPT2);
  DES(ENCRYPT2);
  DES(ENCRYPT2);
  DES(ENCRYPT2);
  DES(ENCRYPT1);
  DES(ENCRYPT2);
  DES(ENCRYPT2);
  ... /* write encrypted data */
}
```

Decryption

```
SETKEY(K_hi, K_lo);
for (;;) {
  ... /* read encrypted data */
  SETDATA(D_hi, D_lo);
  DES(DECRYPT1);
  DES(DECRYPT1);
  DES(DECRYPT2);
  DES(DECRYPT2);
  DES(DECRYPT2);
  DES(DECRYPT2);
  DES(DECRYPT2);
  DES(DECRYPT2);
  DES(DECRYPT1);
  DES(DECRYPT2);
  DES(DECRYPT2);
  E_hi = GETDATA(hi);
  E_lo = GETDATA(lo);
  ... /* write data */
}
```
Add 4 TIE instructions:
- 80 lines of TIE description
- No cycle time impact
- ~1700 additional gates
- Code-size reduced

DES Performance

<table>
<thead>
<tr>
<th>Block Size (Bytes)</th>
<th>Speedup (X)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024</td>
<td>43</td>
</tr>
<tr>
<td>64</td>
<td>50</td>
</tr>
<tr>
<td>8</td>
<td>72</td>
</tr>
<tr>
<td>Mean</td>
<td>53</td>
</tr>
</tbody>
</table>
Outline

- Configurable processors
  - Architecture
  - Instruction extension
  - Software support
- An Example
- Results
- Summary
Improvement over general purpose 32b RISC

- **FIR filter** (signal processing) Base + 6500 gates
- **JPEG** (image compression) Base + 7500 gates
- **Viterbi Decoding** (wireless communication) Base + 900 gates
- **Motion Estimation** (video conferencing) Base + 1000 gates
- **DES** (content encryption) Base + 1700 gates

MIPS or MIPS/Watt

1x 2x 4x 6x 8x 10x 55x
What is “EEMBC”?  

- EDN Embedded Microprocessor Benchmark Consortium  
- Pronounced “Embassy”  
- Non-profit consortium, funded by over 40 members  
  - Including: ARM, AMD, IBM, Intel, LSI Logic, MIPS, Motorola, National Semi, NEC, TI, Toshiba…Tensilica, and more…  
- Objective: Provide independently certified benchmark scores relevant to deeply embedded processor applications  
  - Independent laboratory recreates and certifies all benchmark results - no tricks  
- Five different benchmark suites:  
- Each suite comprised of a range (five to sixteen) of benchmarks representative of that product category  
  - Example: Consumer: image compression, image filtering, color conversion
 comparator in Netmark to high-end desktop CPUs
  - 2x in Netmark/MHz
  - 59K total gates at 200MHz
Beats all processors, including hand-optimized TI C6x

180K total gates at 200MHz
6x in Consumermark and 12x in Consumermark/MHz
127K total gates at 200MHz
Summary

*Optimality/integration* (e.g. mW, $)

$\Delta \approx 10^x$

*Flexibility/modularity* (e.g. time-to-market)

$\Delta \approx 10^x$

- ASIC
- Instruction-set Configurable Processor
- FPGA
- FPGA + Processor
- Traditional Processor
Summary

Optimality/integration (e.g. mW, $)

Δ ~10x

ASIC

Instruction-set Configurable Processor

Δ ~10x

Flexibility/modularity (e.g. time-to-market)

FPGA

FPGA + Processor

General Processor
Optimality/integration (e.g. mW, $)

\[ \Delta \sim 10^x \]

- **Benefit of SoC integration**
  - Higher Bandwidth
  - Lower Cost
  - Lower Power

- **Benefit of IS configuration**
  - A cost-effective computing platform

- **Benefit of TIE compiler and SW tools**
  - Faster time-to-market
  - Lower development cost
  - Lower risk

Flexibility/modularity (e.g. time-to-market)
Thank You!