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, $)

ASIC

Instruction-set Configurable Processor

$\Delta \sim 10^x$

FPGA

$\Delta \sim 10^x$

FPGA + Processor

General Processor

Flexibility/modularity (e.g. time-to-market)
Computing using temporal connection

Processor Solution

Memory (Program)  Control  

Registers  Datapath

Correct  Efficient

Processor

✓  X
Computing using spatial connection

Processor Solution

Memory (Program) -> Control -> Registers -> Datapath

ASIC Solution

FSM -> Storage

Correct: ✓  Efficient: ✗
Configurable Processors: best of both

Processor with Application-specific Instructions

Processor Solutions

ASIC Solutions

Memory (Program)  Control

Registers

Datapath

FSM

Storage

Correct

Efficient

Processor

ASIC
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.

- Source routing
- Result routing
- Decoder
- RF0

- 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
**Opcode**

- **Decoder**
- **Control**
- **RF0**
- **FU0**

**opcode**: MAC \( op2=5 \) CUST0
TIE Language – regfile / state

Decoder

RF0

S0

FU0

• Opcode
• Register file / State

... as needed

state

ACC 40
**TIE Language – semantics**

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

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

\begin{itemize}
  \item Opcode
  \item Register file / state
  \item \textbf{Instruction class}
  \item \textbf{schedule}
\end{itemize}

\[ \text{schedule} \{ \text{MAC}\} \{ \text{use ars 1; use art 1; use ACC 2; def ACC 2;} \} \]
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;
}
Productivity Gain – language + compiler

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

Using the Xtensa processor generator, create...

Selct processor options

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

Describe new instructions

Tailored, synthesizable HDL uP core

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

- Assembler
  Loop a2, .L1
  116si  a10, a3, 0
  116si  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);
```
Software Support – register allocation

- Assembler
- Custom data type
- Register allocation

Spilling around a call:

```
sat_add s3, s1, s2
sat_store s3, a1, 0
call8 foo
sat_load s3, a1, 0
```

- Code Scheduling
- RTOS
- Simulator/debugger
Software Support – code scheduling

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

```
t = sat_mult(x, y);
z = sat_add(z, t);
t2 = 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

Decoder

Control

FU0

RF0

ACC1

ACC2

\[ + \]

\[ \ast \]

Context Switch

Task0
S0, S1, ...
s15

sat_store

Memory

Task1
S0, S1, ...
s15

sat_load
Software Support – simulator/debugger

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

- Simulator/debugger

```
gdb> break ...
gdb> cont
```
Outline

- Configurable processors
  - Architecture
  - Instruction extension
  - Software support
- An Example
- Results
- Summary
Initial step
\[(R, L) = \text{Initial\_permutation}(\text{Din}_{64})\]

Iterate 16 times

Key generation
\[(C, D) = \text{PC1}(k)\]
\[n = \text{rotate\_amount \ (function\ of\ iteration\ count)}\]
\[C = \text{rotate\_right}(C, n)\]
\[D = \text{rotate\_right}\ (D, n)\]
\[K = \text{PC2}(D, C)\]

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

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
DES: DES instruction

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

DES immediate
DES: GETDATA instruction

Initial Permutation

Expansion Permutation

S Boxes

P Permutation

⊕

Final Permutation

Key Generation

State Machine
DES: Putting it together

GETDATA ars, hilo

SETDATA ars, art

Initial Permutation

Expansion Permutation

S Boxes

P Permutation

Final Permutation

Key Generation

State Machine

SETKEY ars, art

DES immediate

GETDATA ars, hilo
DES: Improved Program

Encryption

```c
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(ENCRYPT2);
    DES(ENCRYPT2);
    DES(ENCRYPT1);
    DES(ENCRYPT2);
    DES(ENCRYPT2);
    DES(ENCRYPT2);
    DES(ENCRYPT2);
    DES(ENCRYPT1);
    E_hi = GETDATA(hi);
    E_lo = GETDATA(lo);
    ... /* write encrypted data */
}
```

Decryption

```c
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(DECRYPT1);
    DES(DECRYPT2);
    DES(DECRYPT2);
    DES(DECRYPT2);
    DES(DECRYPT2);
    DES(DECRYPT1);
    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

![Bar chart showing DES performance with block size in bytes and speedup in X. The chart indicates speedup values of 43, 50, 72, and 53 for block sizes of 1024, 64, 8, and mean, respectively.](chart.png)
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
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
EEMBC Networking Benchmark

- Comparable in Netmark to high-end desktop CPUs
- 2x in Netmark/MHz
- 59K total gates at 200MHz

Colors: Blue-Xtensa, Green-Desktop x86s, Maroon-64b RISCs, Orange-32b RISCs
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

Colors: Blue-Xtensa, Green-Desktop x86s, Maroon-64b RISCs, Orange-32b RISCs
Summary

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

 ASIC

Instruction-set Configurable Processor

FPGA

FPGA + Processor

Traditional Processor

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

$\Delta \sim 10^x$

$\Delta \sim 10^x$
Summary

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

ASIC

Instruction-set Configurable Processor

FPGA + Processor

FPGA

General Processor

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

$\Delta \sim 10^x$

$\Delta \sim 10^x$
Summary

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

- ASIC
- FPGA
- Instruction-set Configurable Processor

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
Thank You!