Performance and Flexibility in Multiple-Processor SOC

Chris Rowen, Ph.D.
President and CEO
Tensilica, Inc.
Outline

A Brief Introduction
What Drives System on Chip?
Direct Impact of Automatic Processor Generation
  Size
  Performance
  Dimensions of Configurability
  Bandwidth and Interface
  Complete Software
Implications of Automatic Processor Generation
  High-level View: Multiple Processors
  High-level View: Processors as RTL Alternative
  Detailed View: Tools for MPSOC Development
  SOC Development Productivity
Two Examples
The Sea of Processors
Tensilica: Configurable Processor Leader

Leader in configurable processor technology
  Broad patent protection
  Successful IP business model: semiconductor design + software

Strong customer base (50 licensees, > 100 designs)
  System OEMs and semiconductor suppliers
  Full range of communications and consumer applications

Founded in 1997, product introduced 1999
Tensilica’s Complete Ecosystem

- Silicon Fabrication
  - TSMC
  - IBM

- Software Development
  - Cadence
  - Synopsys
  - Altera
  - Artisan Components
  - Avanti!
  - Mentor Graphics
  - CoWare
  - VRage 3D
  - Nurlogic Design Inc.

- Chip Construction
  - Sonics Inc.
  - Intrinsix
  - SPIKE Technologies

- Verification
  - Avanti!
  - Sophia Systems
  - InCOMM
  - SPI Group
  - MontaVista Software
  - Accelerated Technology

© 2002 Tensilica Inc.
Customers Lead in Growth Markets

- Cisco Systems
- Broadcom
- NEC
- Fujitsu
- Conexant
- Hughes Network Systems
- Fujifilm
- JVC
- Avision
- ST Microelectronics
- NTT
- Marvell
- National Semiconductor
Why System On Chip?

Needs

Lower Cost/Size/Power

Higher Performance

Time to Market

More Functionality

Triggers

0.18μ - 0.13μ Silicon (speed, density)

Foundry Proliferation (affordable silicon)

Advanced EDA Tools (portable design)

Memory

CPU

I/O

Signal processing

Protocol processing

Audio

Application accelerator

Encrypt

Imaging

Needs Triggers

Lower Cost/Size/Power

Higher Performance

Time to Market

More Functionality

0.18μ - 0.13μ Silicon (speed, density)

Foundry Proliferation (affordable silicon)

Advanced EDA Tools (portable design)
System On Chip Designer’s Dilemma

Performance

Custom Logic

Flexibility

FPGA

Standard Processors
Automated, Optimized Processor Design

Cost
small size (<0.25mm² in 0.13μ)

Performance
application extensions

Productivity
rapid hardware and software

Building Block Use
processors everywhere
Direct Impact: Performance

Source: Tensilica
Performance of Tensilica vs. RISC in core algorithms

© 2002 Tensilica Inc.
Direct Impact: Bandwidth

- Processor Data Bandwidth (MB/s)
- Effective Operations Throughput (Application MIPS)

Xtensa configurations 0.13µ

- High-end RISC 0.13µ (e.g. MIPS64)
- Std RISC 0.13µ (e.g. ARM9E, MIPS32)

Wide data-paths

System-specific instructions
Direct Impact: Complete Hardware and Software in One Hour

Electronic Specification

Tensilica Processor Generator

Hardware Design
- RISC
- OCD
- DSP
- Cache
- Timer
- Register File
- FPU

Customized Software

Build using any IC process

Design processor in one hour
Direct Impact: Configurability and Extensibility

- Configuration and Extensibility
- Optional & Configurable
- Optional Function
- Configurable Function
- Base ISA Feature
- Defined Coprocessors

**Processor Controls**
- Interrupt Control
- Timers 0 to n
- Exception Support

**Align and Decode**
- Register File
- ALU
- MAC 16
- MUL 16
- MUL 32
- FPU
- Vectra DSP

**Instruction Fetch / PC Unit**
- Instruction Cache
- Instruction ROM
- Instruction RAM

**External Interface**
- Write Buffer (1 to 32 entries)
- Xtensa Processor Interface (PIF)

**Data Load / Store Unit**
- Data Cache
- Data ROM
- Data RAM

**Data Address Watch 0 to n**
- Instruction Address Watch 0 to n
Implication: 
Processor as Universal Building Block
Implication: Interface Flexibility Makes MP Natural

Direct interface among processors and to logic at >5GB/s
Flexible memory sharing
Comprehensive global debug and trace
Bridges to standard on-chip buses

Global Trace/Debug

Processors
Rigid RTL
External Interfaces
Debug Chain

Processors
On-Chip Buses
Rigid RTL
Debug Chain

Shared Memories and Queues

Processors
Rigid CPU cores cannot be modified to fit the application

So designers must add coprocessors or custom RTL blocks to reach performance and power goals.
Fixed RTL Logic: The Risk Factor

RTL Logic Increases Risk, Slows Time To Market
Verification of complex state machines
Costly silicon respins to make changes
Obsolescence as markets or standards change

10% of Gates, 90% of Risk
90% of Gates, 10% of Risk
Good integer performance for multi-purpose computing

Real-life SOC applications do not crunch 32b integer data on a general purpose CPU

“Typical” RISC Processor Pipeline

- Instruction Fetch
- Instruction Decode / Issue
- Execute
  - ALU
- Load / Branch / Store Address
- Memory / Register Write
What if you could transform the processor, with custom instructions, custom execution units, custom register files and state variables?

“Typical” RISC Processor Pipeline

- Instruction Fetch
- Instruction Decode / Issue
- Datapath
- Execute
- ALU
- Load / Branch / Store Address
- Memory / Register Write
What Once Was Built In RTL…

... is implemented as designer-defined execution units; designer-defined instructions, and processor registers & state.
Xtensa Reshapes SOC Design

Optimized Processors as Building Blocks
Task-specific, optimized processors with RTL-like performance

Lower Risk
Verify Algorithm in Software within hours

Higher productivity

Bus topology shown for illustration purposes only
Numerous topologies and configurations possible
Multiple Processor Debug: Network Modeling, SW Dev, Bring-up

Before silicon

Host Workstation
- Target Debugger
- Target Debugger
- Target Debugger

TCP/IP

Xtensa Modelling Protocol API
- Xtensa ISS #1
- Xtensa ISS #2
- Xtensa ISS #3

Host Workstation

After silicon

Host Workstation
- OCD Server
- Topology File

Wiggler

TAP
- Custom TAP
- TAP
- TAP

Break Control Logic

PIF OCD XLMI FIFO
- Xtensa Core
- Xtensa Core
- Xtensa Core
Use XTMP API calls to:

- Instantiate the simulation components
- Connect components together to form a system
- Start and control execution of the simulator
  - Load target programs into memory
  - Enable debugging for any or all cores
  - Start the simulation

**Full source-level debugging on each processor**

- The XTMP API allows you to connect a debugger to each core
- Each core communicates independently with the debugger
- Multiple debug interfaces: xt-gdb and xt-ddd
XTMP pre-defined components

**XTMP_core**: Xtensa processor

**XTMP_memory**: simple memory device that uses pages to avoid allocating regions of memory that are not used

**XTMP_connector**: device for connecting cores to other devices via the Processor Interface (PIF). Single global address map or distinct maps for each processor

**XTMP_device**: user-defined device that interacts with the rest of the simulator via callback functions

**XTMP_procId** and **XTMP_lock**: devices to help with multiprocessor synchronization

**XTMP_connectToPort()**: direct connection of devices to Xtensa Local Memory Interface (XLMI)

You can instantiate multiple independent objects of each component
XTMP simulation commands

**XTMP_loadProgram()** loads target program into memories

**XTMP_start()** runs the simulator until finished, or for a fixed number of clock cycles

**XTMP_disable(), XTMP_enable()** and **XTMP_reset()** disables, enables and resets a particular core

**XTMP_step()** step thru one core, while other cores are disabled
Uniprocessor Example

Single-processor system with ROM and RAM connected to its PIF

Observe general program layout
1. Instantiate
2. Connect
3. Start Simulation

```c
XTMP_main()
{
    declare objects;
    instantiate core;
    instantiate memories;
    instantiate connector;
    connect component;
    load program;
    start simulation;
}
```

```c
main()
{
    ...
    process packets;
    ...
}
```
Uniprocessor Example Code

#include "iss/mp.h"

int XTMP_main(int argc, char **argv) {
    /* Declare simulation objects */
    XTMP_params p;
    XTMP_core core;
    XTMP_singleAddressMapConnector ether;
    XTMP_memory rom, ram;

    /* first, load "s1-params" file from registry */
    p = XTMP_paramsNew("s1", NULL);

    /* Create objects (core, memory, connector) */
    core = XTMP_coreNew("cpu", p, NULL);
    rom = XTMP_sysRomNew("rom", p);
    ram = XTMP_sysRamNew("ram", p);
    ether = XTMP_singleAddressMapConnectorNew("ether",
                                           XTMP_byteWidth(core));

    /* Connect objects to each other via the connector */
    XTMP_connect(ether, core);
    XTMP_connect(ether, rom);
    XTMP_connect(ether, ram);

    /* Load target program and start simulation */
    XTMP_loadProgram(core, "sieve.out", NULL);
    XTMP_start(-1);
    return(0);
}
Multiprocessor Example

core1

Proc. ID

Shared RAM

core2

lock
```c
#include "iss/mp.h"
int XTMP_main(int argc, char **argv) {
    XTMP_params p;
    XTMP_core core1, core2;
    XTMP_multiAddressMapConnector router;
    XTMP_memory rom1, rom2, ram1, ram2;
    XTMP_memory shared_mem;
    XTMP_procId pid;
    XTMP_lock lock;

    char *simulation_arg[] = {"--profile=gmon.out", NULL}; /* Args to ISS */
    /* Arguments to be passed to program running on the Xtensa core */
    char *program_arg[] = {"program_name", "-verbose", NULL};
    char *tie_path[] = {"/usr/xtensa/project/tdk", NULL}; /*path to tdk dirs */

    unsigned int dontcare = 0x0;
    p = XTMP_paramsNew( "s1", tie_path );
    core1 = XTMP_coreNew( "cpu1", p, simulation_arg );
    rom1 = XTMP_memoryNew( "rom1", p, dontcare, 0x00100000);
    XTMP_setBooleanOption( rom1,XTMP_BO_readOnly,1);
    ram1 = XTMP_memoryNew( "ram1", p, dontcare, 0x01800000);
    core2 = XTMP_coreNew( "cpu2", p, NULL );
    rom2 = XTMP_memoryNew( "rom2", p, dontcare, 0x00100000);
    XTMP_setBooleanOption( rom2,XTMP_BO_readOnly,1);
    ram2 = XTMP_memoryNew( "ram2", p, dontcare, 0x01800000 );

    return 0;
}
```
pid = XTMP_procIdNew("Processor ID Device", XTMP_byteWidth(core1),
XTMP_isBigEndianFromParams(p), dontcare);
lock = XTMP_lockNew("lock", XTMP_byteWidth(core1),
XTMP_isBigEndianFromParams(p), dontcare, 1);
router = XTMP_multiAddressMapConnectorNew("router", XTMP_byteWidth(core1));

/* Create 64K of shared memory */
shared_mem = XTMP_memoryNew("shared_mem", p, dontcare, 0x10000);
XTMP_connect(router, core1);
XTMP_connect(router, rom1);
XTMP_connect(router, ram1);
XTMP_connect(router, core2);
XTMP_connect(router, rom2);
XTMP_connect(router, ram2);
XTMP_connect(router, pid);
XTMP_connect(router, shared_mem);
XTMP_connect(router, lock);
XTMP_mapEntryAdd(router, core1, 0x20000000, rom1);
XTMP_mapEntryAdd(router, core1, 0x40000000, ram1);
XTMP_mapEntryAdd(router, core1, 0x01000000, pid);
XTMP_mapEntryAdd(router, core1, 0x01000020, lock);
XTMP_mapEntryAdd(router, core1, 0x01010000, shared_mem);
XTMP_mapEntryAdd(router, core2, 0x20000000, rom2);
XTMP_mapEntryAdd(router, core2, 0x40000000, ram2);
XTMP_mapEntryAdd(router, core2, 0x01000000, pid);
XTMP_mapEntryAdd(router, core2, 0x01000020, lock);
XTMP_mapEntryAdd(router, core2, 0x01020000, shared_mem);
XTMP_loadProgram( core1, "procl.out", NULL );
XTMP_loadProgram( core2, "proc2.out", program_arg );
XTMP_start(100000); /* run the simulation for 100k cycles */
printf("debug port number for core1: target xtensa-remote localhost:%d\n");
XTMP_enableDebug(core1,0));
printf("debug port number for core2: target xtensa-remote localhost:%d\n", 
XTMP_enableDebug(core2,0));
XTMP_setWaitForDebugger(core1,1);
XTMP_setWaitForDebugger(core2,1);
XTMP_setTraceLevel( core1, 1);
XTMP_setTraceFile( core1, "core1.trace");
XTMP_start(-1); /* continue simulation */
exit(0); 
}
## Result: Software-Based System LSI Design

<table>
<thead>
<tr>
<th>Application-tuned data-paths</th>
<th>Standard CPU</th>
<th>RTL Logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>No: one-size-fits-all instruction set</td>
<td>Yes: specify in high-level TIE</td>
<td>Yes: specify in low-level RTL</td>
</tr>
<tr>
<td>Programmable in C/C++</td>
<td>Programmable in C/C++</td>
<td>Hardwired RTL state machines only</td>
</tr>
<tr>
<td>Fast simulators or development boards</td>
<td>Fast simulators or development boards</td>
<td>RTL simulation: &gt;100x slower</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Task control and sequencing</th>
<th>Standard CPU</th>
<th>RTL Logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programmable in C/C++</td>
<td>Programmable in C/C++</td>
<td>Hardwired RTL state machines only</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Multiple engines working together</th>
<th>Standard CPU</th>
<th>RTL Logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Little interface, debug or modeling support</td>
<td>Simple direct interface, debug and simulation</td>
<td>Possible, but hard to design and model</td>
</tr>
</tbody>
</table>

- Low risk like software, high performance like RTL
- Simpler design, integration, verification and upgrade of complex chips
Example: Multiple Processors, Unified Design

Voice Gateway

Five Tensilica cores in common development system
Example: Time to Market, Long Product Life

Broadband Satellite Gateway

All real-time processing in seven Tensilica elements

- **tensilica Control**
- **tensilica Data 1**
- **tensilica Data 2**
- **tensilica Data 3**
- **tensilica Data 4**
- **tensilica Data 5**
- **tensilica Debug**

- **Network**
- **Network**
- **Peripheral**
- **Memory**
Looking Forward

Performance: Millions operations/sec per multiple-processor SOC
Cost: Raw silicon cost per processor core
The Sea of Processors
Within a few years...

- Raw processor cost measured in milli-cents
- Processors dominate almost all logic functions (control, data, I/O)
- Automated processor architecture by application-specific optimization
- The typical communication or consumer system uses 20 to 2000 processors per chip [>100 even today!]
- Processor production exceed 100 billion cores. 99% are in System LSI devices

The extensible processor is "the new transistor"