The Configurable Processor Company

Introducing XPRES Compiler

Synthesizing Optimized Hardware - Directly from C

Under Embargo Until July 7, 2004

XPRES Compiler Announcement

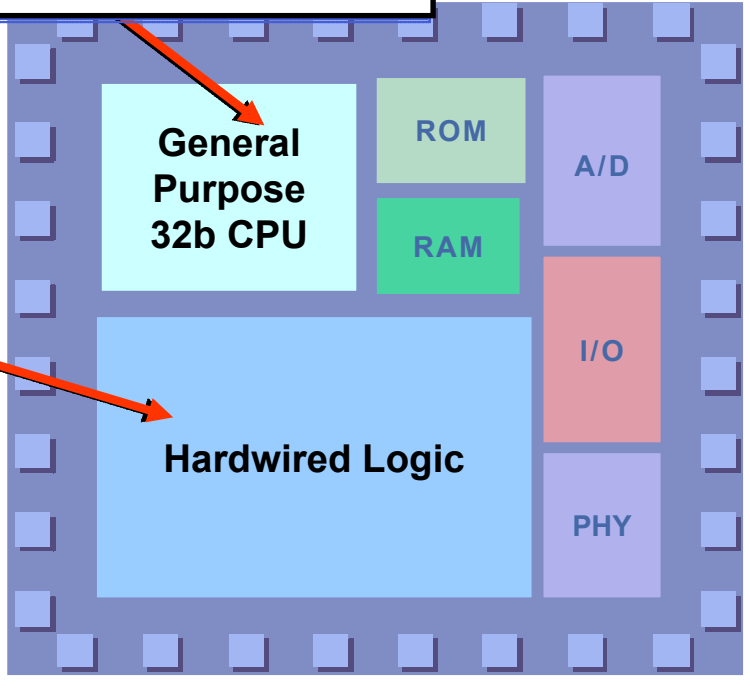
- Background
- The XPRES Compiler
- Quality of results

SOC Design Problems

General-purpose processors aren't fast enough, so custom RTL is used to accelerate data-intensive or compute-intensive tasks

RTL—the Achilles Heel for SOC ROI

- Slow to design and verify
- Inflexible after tapeout
- Not programmable
- High re-spin risk
- Slows time to market and reduces ROI

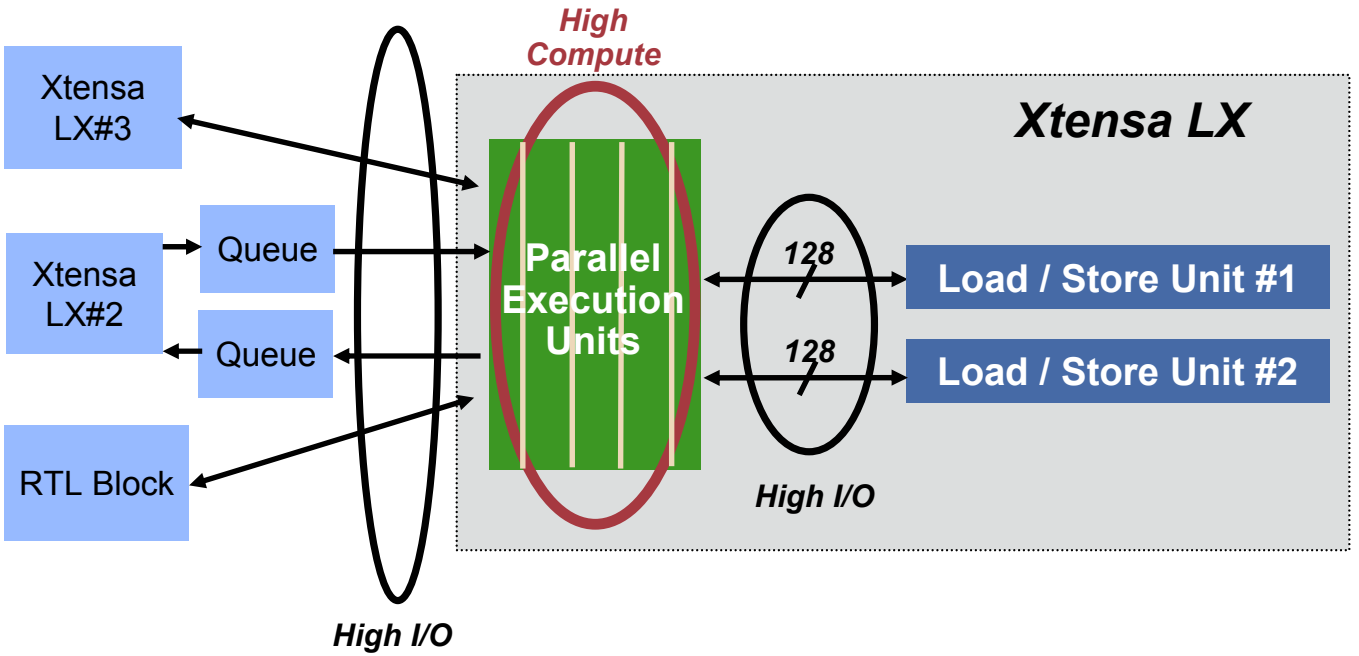


1999: Tensilica Introduced A Faster Way to Generate RTL

- **We called it a "configurable processor"**
- **You could change the processor 2 ways:**
  - Configure thousands of options
    - MMU, RAM, interrupts, floating point, DSP engines, EDA scripts & more
  - Extend for any application
    - Custom instructions enhance algorithm performance (10X - 100X or more)
- **SOC design methodology began transition from RTL-centric to processor-centric**
  - Design faster (reduced time to market, reduced design cost)
  - Lower risk (pre-verified blocks, added programmability)

Xtensa LX: Breaks Through Computation and I/O Bottlenecks

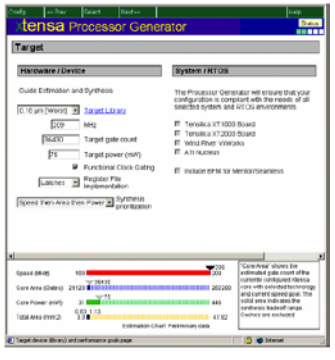
- High I/O and compute bandwidth are in balance
- Xtensa LX can process data as fast as it can get it in and out



Automation was the Key Enabler

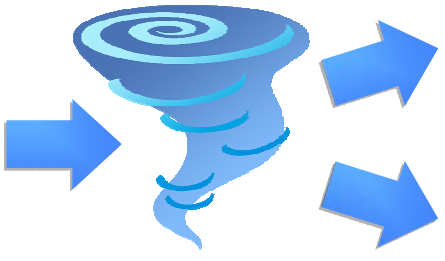
Complete Hardware Design

Verilog/VHDL RTL, EDA scripts, test suite



Electronic Specification

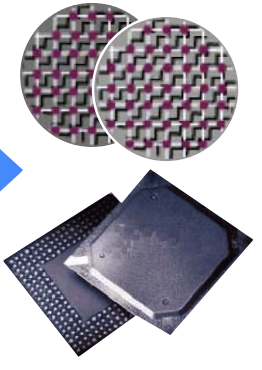
Configuration selection and custom-instruction extensions



Xtensa *Processor Generator*

Customized Software Tools

C/C++ compiler
Debuggers
Simulators
RTOSes



Use standard techniques and libraries for *any* IC fabrication process

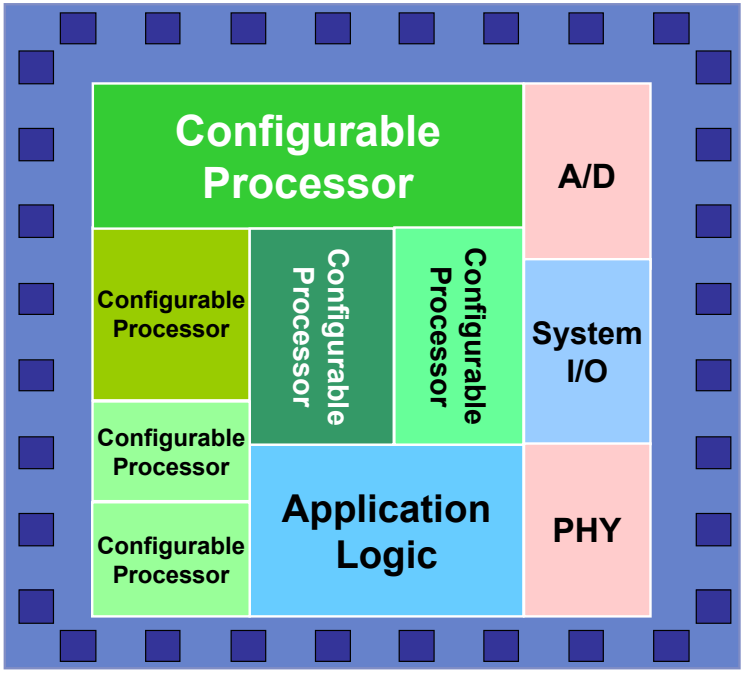
Iterate in hours!

** US Patent: 6,477,697*

Now SOCs Contain Many Configurable Processors

Tensilica customer average = ~6 processors / SOC

- **60+ leading IC and systems companies**
- **160+ design starts**
- **Alternative to RTL for:**
  - Audio [AAC, MP3, WMA, AC-3]
  - Video processing – MPEG-4
  - Packet processing – Routers, DSL
  - Storage network processing
  - Image processing – printers, scanners
  - Encryption
- **Anywhere large amounts of data need to be processed quickly**



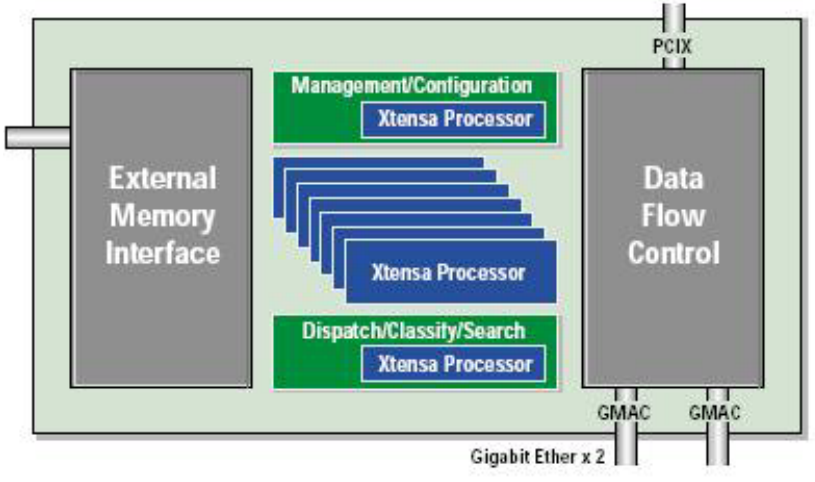
NEC TCP/IP Offload Engine

NEC iStorage NV8200 Series

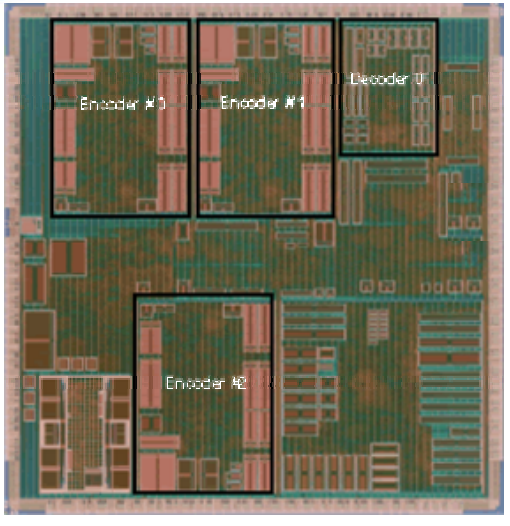
TCP/IP offload engine (TOE) chip w/ 10 Xtensa processors
One TOE engine per blade

NV8220

NV8210



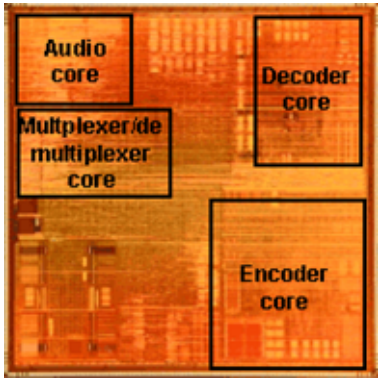
NTT Electronics: HDTV/SDTV Codecs



Single-chip MPEG-2 HDTV CODEC

Product: "VASA"

60M transistors, 4 Xtensa cores, 0.13μm



Single-chip MPEG-2 SDTV CODEC

Product: "SuperENC-III"

30M transistors, 1 Xtensa core, 0.13μm

First High-Definition "Prosumer" Camcorder

JVC GR-HD1

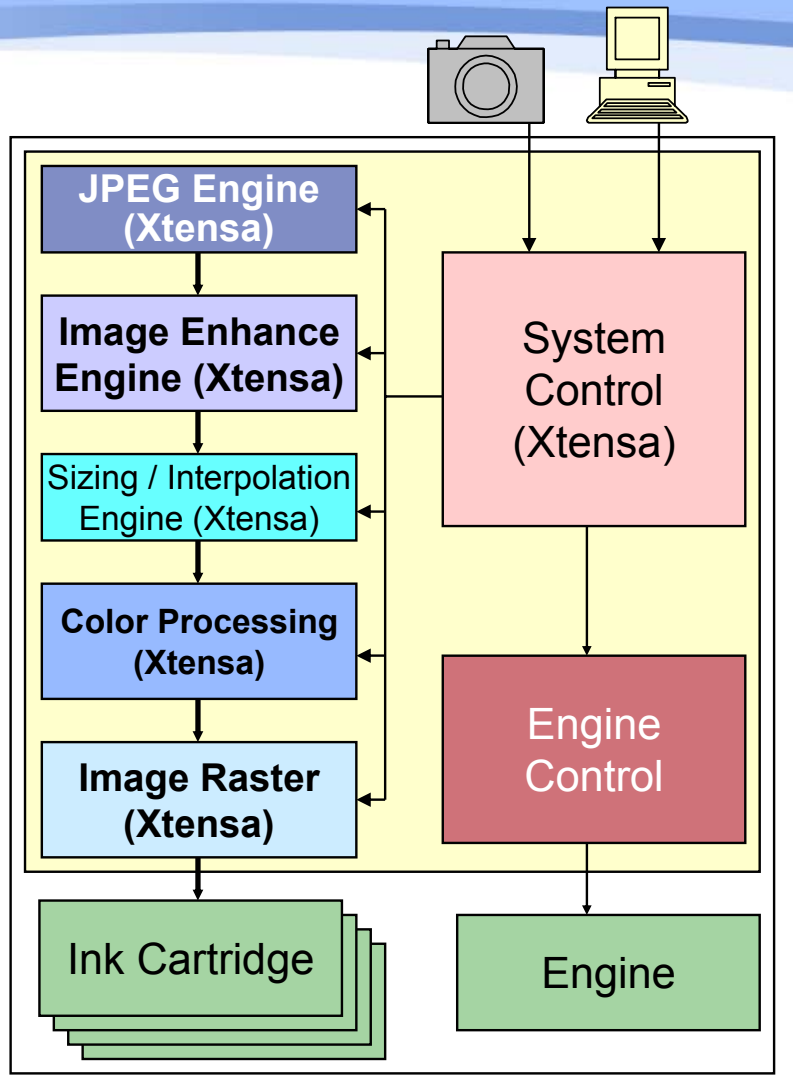
NTT Electronics Corporation

NEL®

For announcement July 7, 2004

Printers - Market Transition

- **Product challenges**
  - Processing shifting from PC to printer
  - Host-less printing (direct from camera)
  - Size and portability important
  - Multiple price points
- **SOCs with multiple configurable processors deliver**
  - Low cost
  - High speed processing
  - Programmable platform used in dozens of printer models



Designers Use Configurable Processors Because…

- **Programmability after silicon implementation**
  - Run similar programs, implement changing standards
- **Avoid re-spin**
  - Fix bugs in software
- **Provide maximum flexibility**
  - Example: Xtensa HiFi Audio Engine can run different codecs from same silicon: [AAC, AC-3, MPEG-2/4, WMA, Speech]
- **Less verification**
  - Pre-verified core, correct-by-construction RTL
- **Faster design time**
  - Fast processor generator allows exploration of alternatives

Challenge Still Remains: Converting Algorithms to Hardware

- **Fact: Many algorithms are designed in C/C++**
  - C is designed to run on processors - not to specify HW
- **No matter what the implementation choice - RTL, configurable processor - a manual conversion must first occur from C to "an architecture"**
- **Goal: Generate efficient HW directly from the algorithm**
  - Eliminate conversion of original algorithm to intermediate language

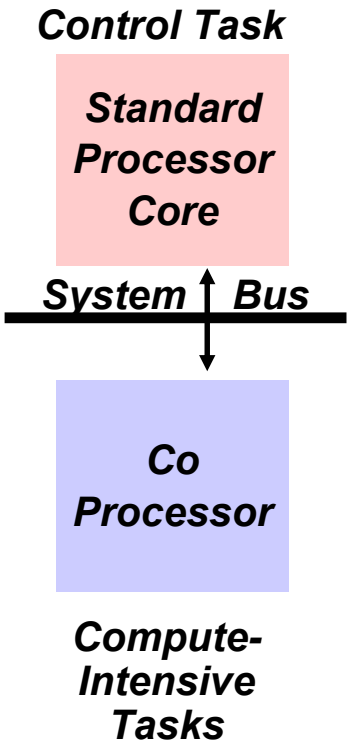
Alternative #1: Manual RTL Design

- **Proven, predictable, successful methodology**
- **Not programmable**
  - Can't make changes after chip is made
- **Much slower to design, harder to verify**
  - Verification takes 70+% of design time

Alternative #2: Higher-level Synthesis – Behavior to Gates

- **Replace Verilog/VHDL RTL with a simulation-friendly alternative hardware design language**
  - Behavioral HDL, System C, other hybrids
- **Much closer to native C than traditional Verilog/VHDL**
  - Not native, full C or C++ language
- **Simulates faster than HDL**
- **Produces non-programmable hardware**
- **QOR to date has often been poor**

Alternative #3: "Coprocessor Synthesis"



- **Starts with C code, or a subset, and maps compute-intensive software functions to hardwired RTL coprocessor**
  - Control functions stay on standard processor
  - Changes original C code
- **Accelerates current manual methodology of developing HW "assist" blocks for CPUs/DSPs**
  - Limited by communications on std CPU busses
  - Task "hand off" overhead from CPU to coprocessor limits minimum offloaded task size
- **Post-silicon changes to C code may be unable to exploit hardwired coprocessors**

XPRES Compiler

(Xtensa PRocessor Extension Synthesis)

Automatic Generation of Processor Extensions
for
Xtensa LX processors

Using XPRES Compiler

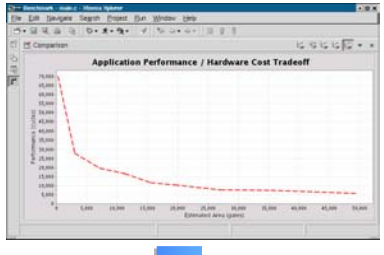
Compile Original C/C++ Code

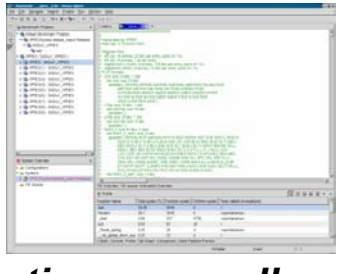
```
int main()
{
 int i;
 short c[100];
 for (i=0;i<N/2;i++)
 {
```

Run XPRES Compiler

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

Designer selects "best" configuration





Option: manually refine configuration

Run Xtensa Processor Generator

Build processor, including RTL, SW, etc.

Build chip, system

Compile & run original, unmodified C code

```
int main()
{
 int i;
 short c[100];
 for (i=0;i<N/2;i++)
 {
```

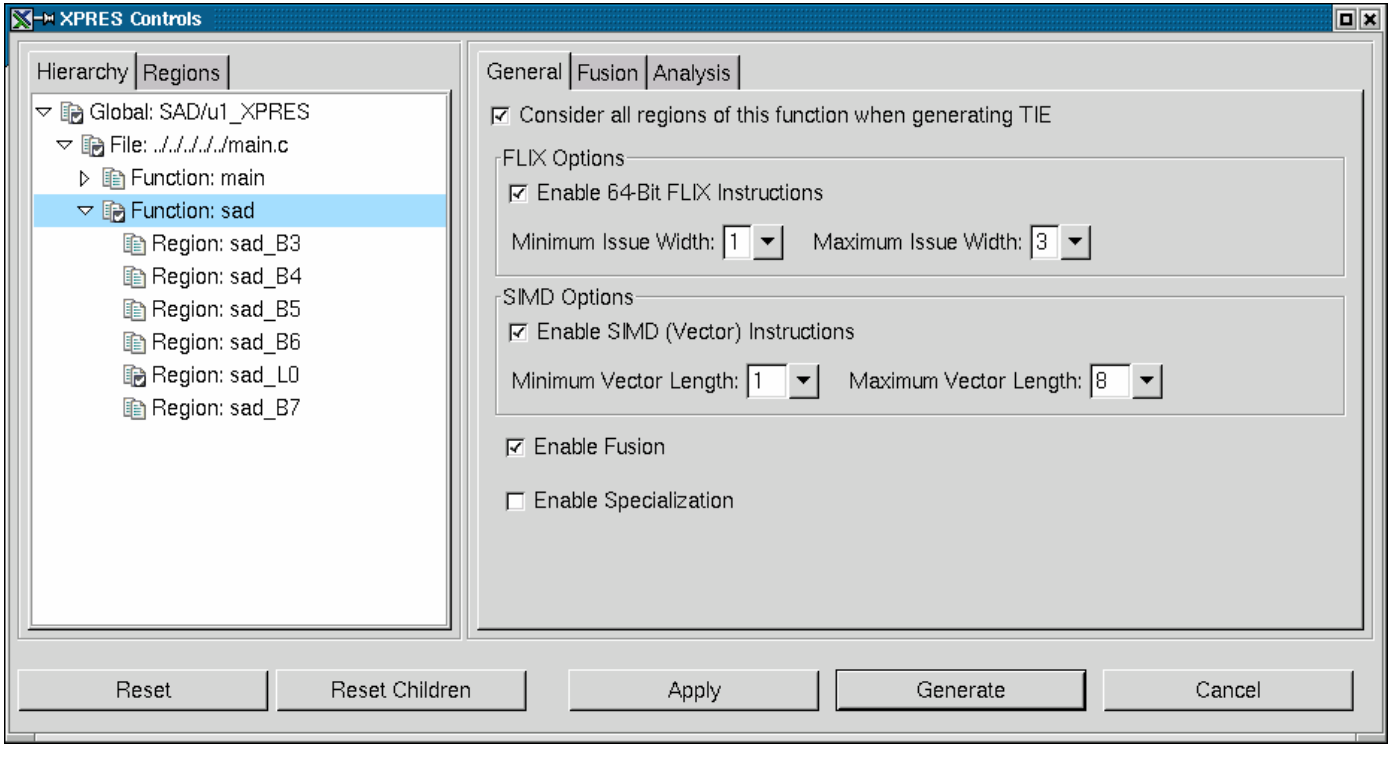
Writing Software to Run on XPRES Generated Hardware Blocks

USE ORIGINAL, UNMODIFIED C/C ++ APPLICATION
OR OTHER SIMILAR SOFTWARE APPLICATIONS - EVEN YEARS LATER

- **C/C++ code does not need to be modified**
- **Compiler infers the use of instruction extensions from standard C/C++**
- **Developer may tune, modify, adapt C/C++ code at any time**

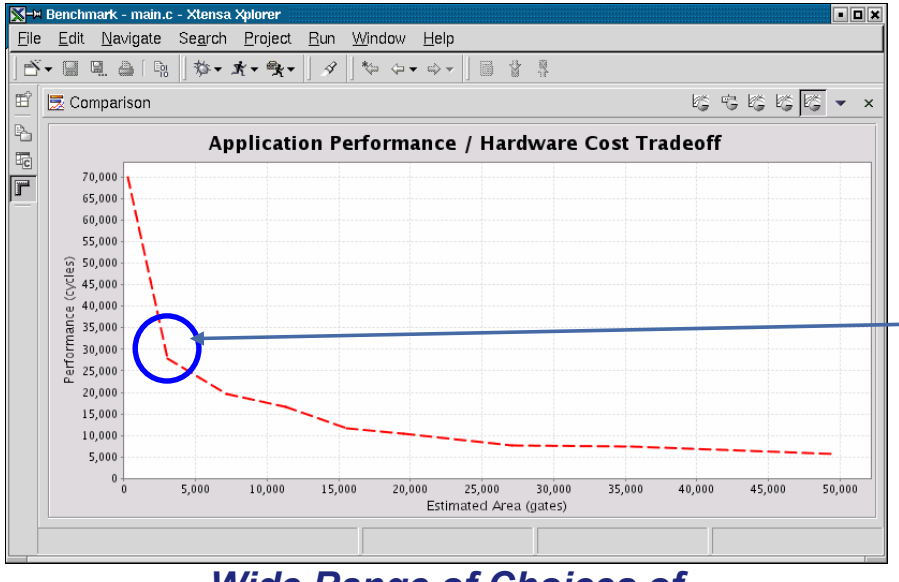
Controlling XPRES Compiler

Optional manual control over the type of processor extensions generated by XPRES and which code sections to optimize



Example: Sum of Absolute Differences for Video Processing

Visually Presented in Xtensa Xplorer



Wide Range of Choices of Performance Increase -v- Hardware Cost

Generated Xtensa LX Configurations

i	Speedup -v- Base processor	Gates Added
1	18.6x	49,681
2	14.1x	35,439
3	13.9x	27,082
4	10.1x	19,547
5	9.0x	15,527
6	6.8x	8994
7	5.3x	7158
8	1.5x	301

XPRES Compiler – Fast and Delivers High Quality of Results

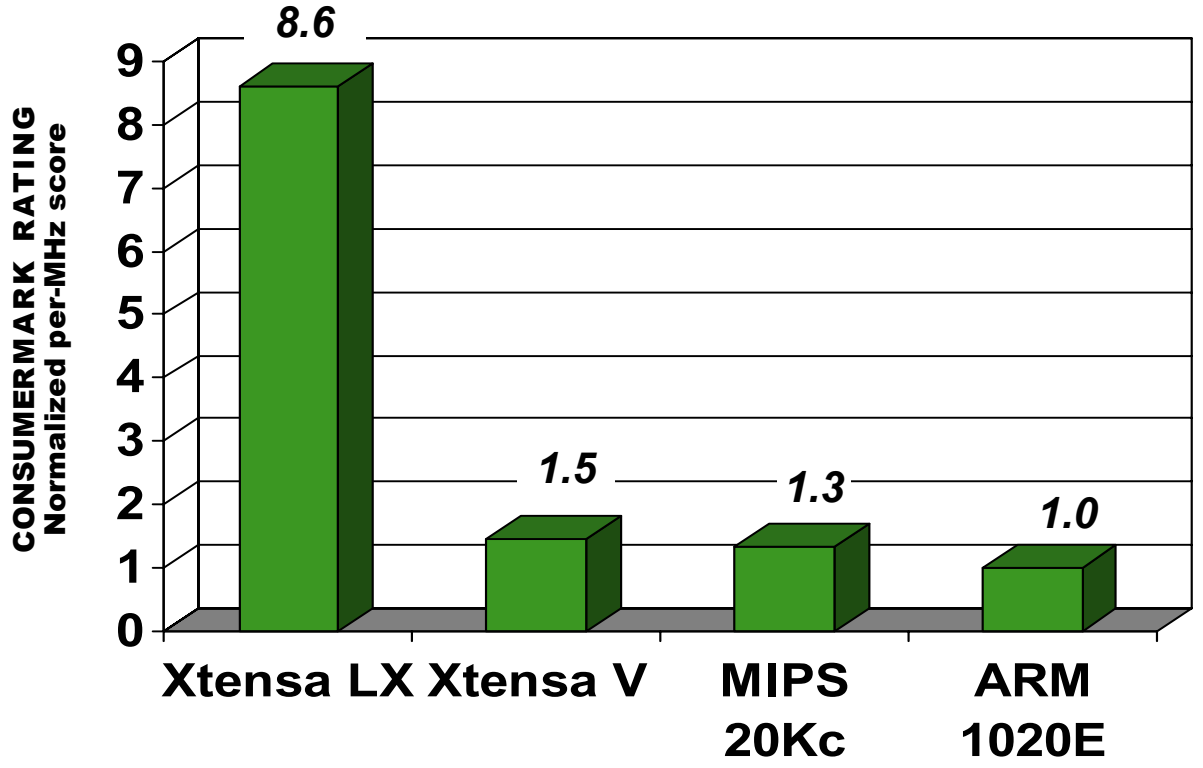
		Dramatic Productivity Enhancement	
Application	**Speedup**	**Configurations Visited**	**Run Time to Generate Configurations**
Radix-4 FFT	**10.5x**	**175,796**	**3 minutes**
GSM Encoder	**3.9x**	**576,722**	**15 minutes**
MPEG-4 Encoder	**3.0x**	**1,830,796**	**30 minutes**

Proof: EEMBC Consumer Benchmark

- **EEMBC Consumer Benchmark**
  - Thousands of lines of C code
  - JPEG compression / decompression, color conversion, image processing
- **“Out of the Box” benchmarks**
  - No C code modification allowed
- **The Process**
  - EEMBC C code analyzed by XPRES Compiler
    - XPRES Compiler created optimal set of extensions
    - Extensions used by Xtensa Processor Generator to create an optimal Xtensa LX processor
  - Elapsed design time: less than 1 day

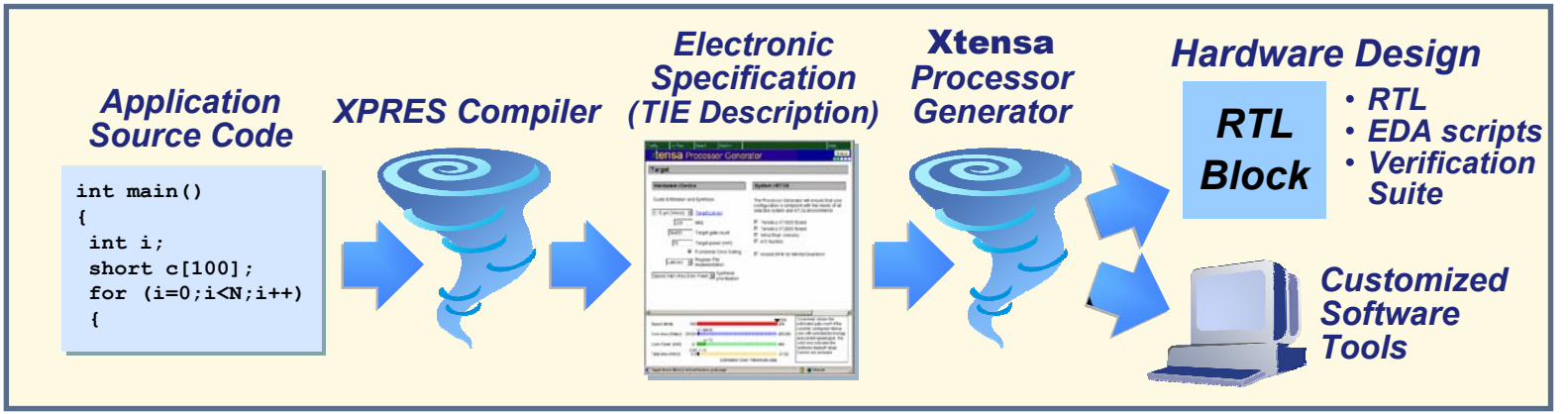
EEMBC Consumer Benchmark "Out of the Box" Scores

Xtensa LX processor: ~9X faster than other cores
As fast as dedicated RTL - Design time less than 1 day



XPRES Compiler: Automating Processor Extension

From an ANSI C/C++ application the XPRES Compiler generates an optimized set of processor extensions ……..



… that is reusable over a range of <u>similar</u> application software code.