Configurable and Extensible Processors Change System Design

Ricardo E. Gonzalez

Tensilica, Inc.

Presentation Overview

Yet Another Processor?

- No, a new way of building systems
- Puts system designers in the drivers seat

Configurable and Extensible

- Select and size only what you need
- Add system-specific instructions for 4-50× improvement

Portable

- Build your system ASICs in any foundry

Complete

- Hardware and software configured/extended together

Simple, Fast, and Robust

- Configure/extend in hours instead of months

Get the YAP stuff out of the way

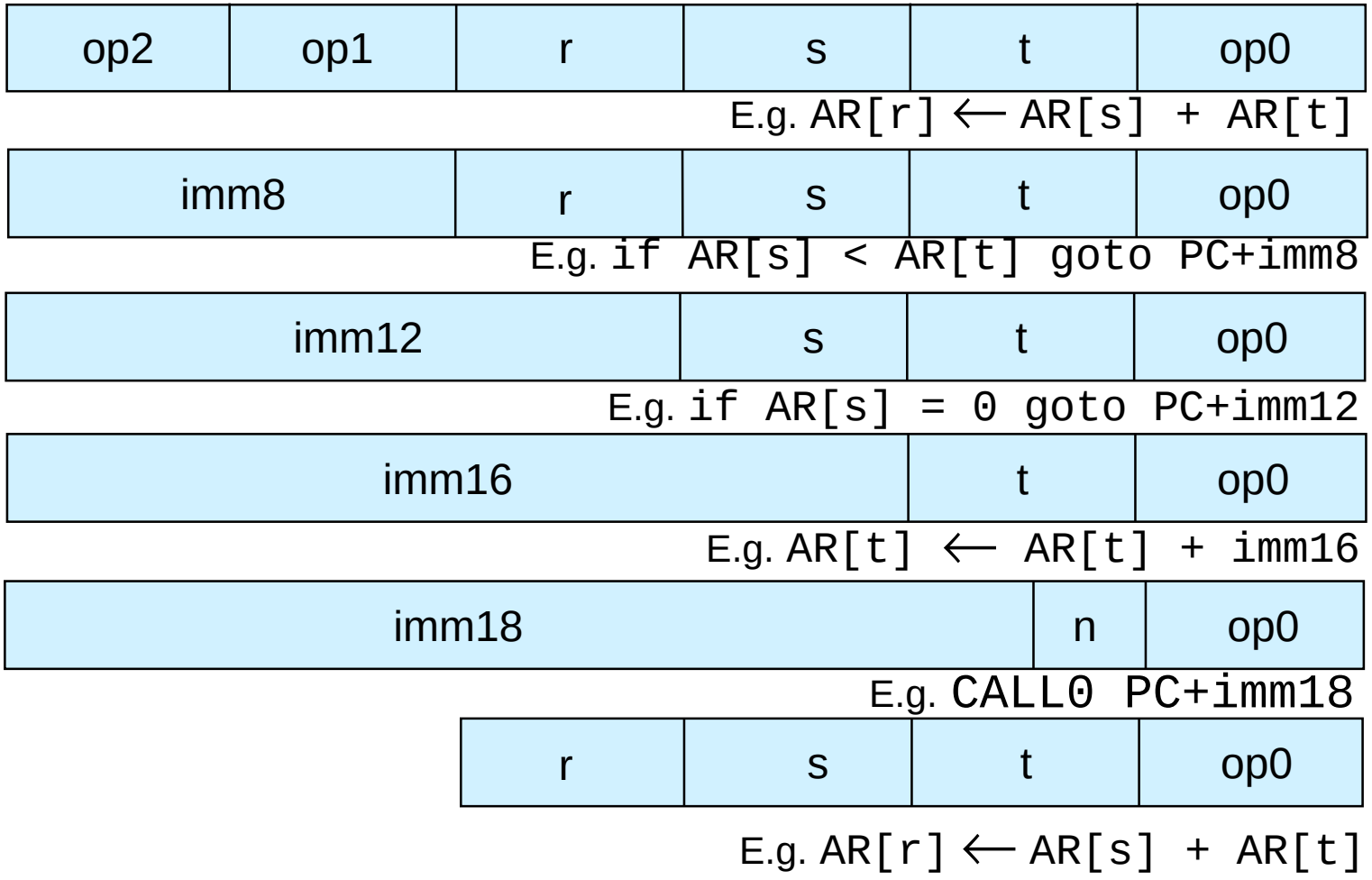
Instruction Set Architecture

Name	Xtensa
Instructions	24-bit and 16-bit formats
Code size	Half of MIPS or ARM
	Better than Thumb or MIPS16
Registers	32-64 × 32b, windowed
Address bits	32
Data bits	varies

Xtensa V1.5 Implementation

Pipeline	5-stage, single-issue
Inst Cache	direct-mapped, 1-16KB, 16 to 64B line
Data Cache	direct-mapped, 1-16KB, 16 to 64B line, write-thru
Write buffer	4-32 entries
MMU	none

Xtensa's unique 24/16 encoding



Xtensa's new ISA takes performance and code size to a new level

Huffman encoding (EPIC image compression)

```
for (i=0; i< NUM; i++)
    if (histogram[i] != NULL)
        insert (histogram[i], &tree);
```

Xtensa code

```
L16: addx4 a2, a3, a5
     l32i  a10, a2, 0
     beqz  a10, L15
     add   a11, a4, a7
     call8 insert
L15: addi  a3, a3, 1
     bge   a6, a3,L16
```

7 instructions
17 bytes

ARM code

```
J4:ADD   a1,sp,#4
   LDR   a1,[a1,a3,LSL#2]
   CMP   a1,#0
   MOVNE a2,sp
   BLNE  insert
   ADD   a3,a3,#1
   CMP   a3,#&3e8
   BLT   J4
```

8 instructions
36 bytes

Thumb code

```
L4:  LSL r1,r7,#2
     ADD r0,sp,#4
     LDR r0,[r0,r1]
     CMP r0,#0
     BEQ L13
     MOV r1,sp
     BL  insert
L13:ADD r7,#1
     CMP r7,r4
     BLT L4
```

10 instructions
20 bytes

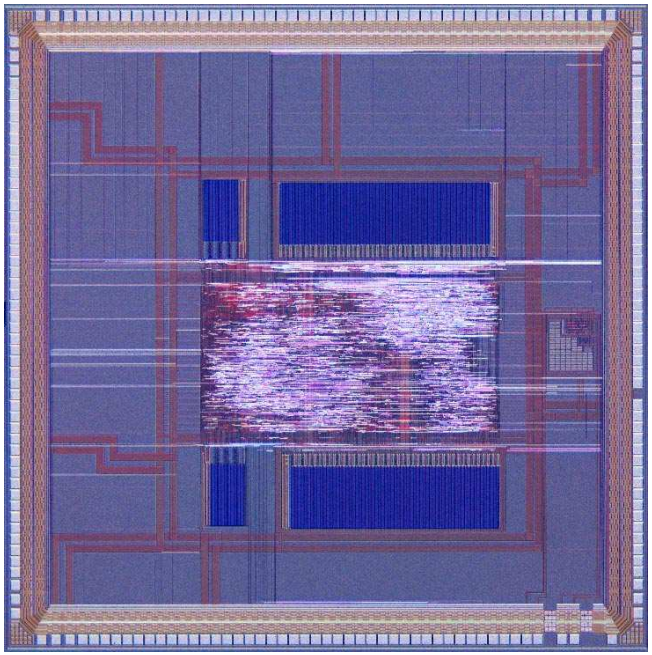
Is it real?

YES!

V1.5 released June
Customer designs underway

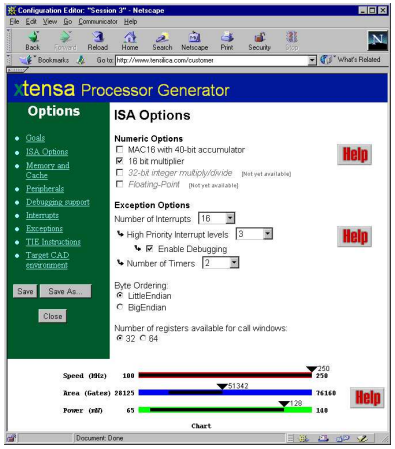
First silicon fully functional using TSMC 0.25u CMOS process

- > 200 MHz typical
- 140 MHz worst case (T,V,P)

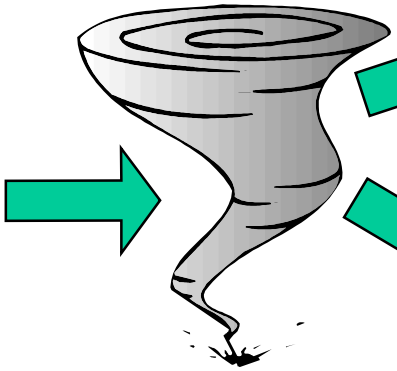


	Core Area	Gates	Core Power
Area Optimized	1.1mm^2	27K	0.8 mW/MHz
Speed Optimzed	1.5mm^2	35K	1.0 mW/MHz

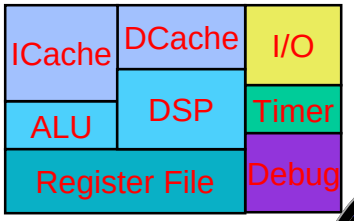
But that's not as neat as...



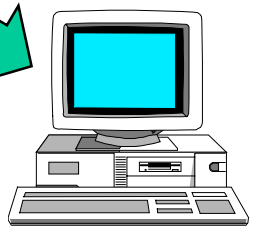
Describe the processor attributes from a browser



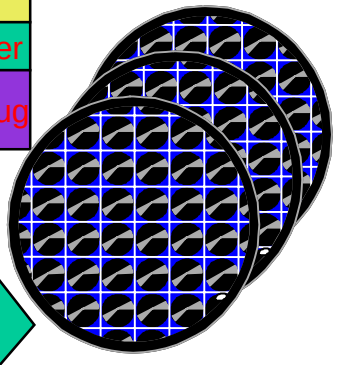
Using the Xtensa processor generator, create...

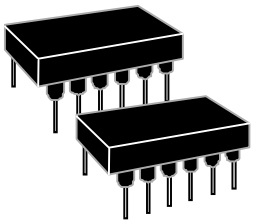


Tailored, HDL µP core



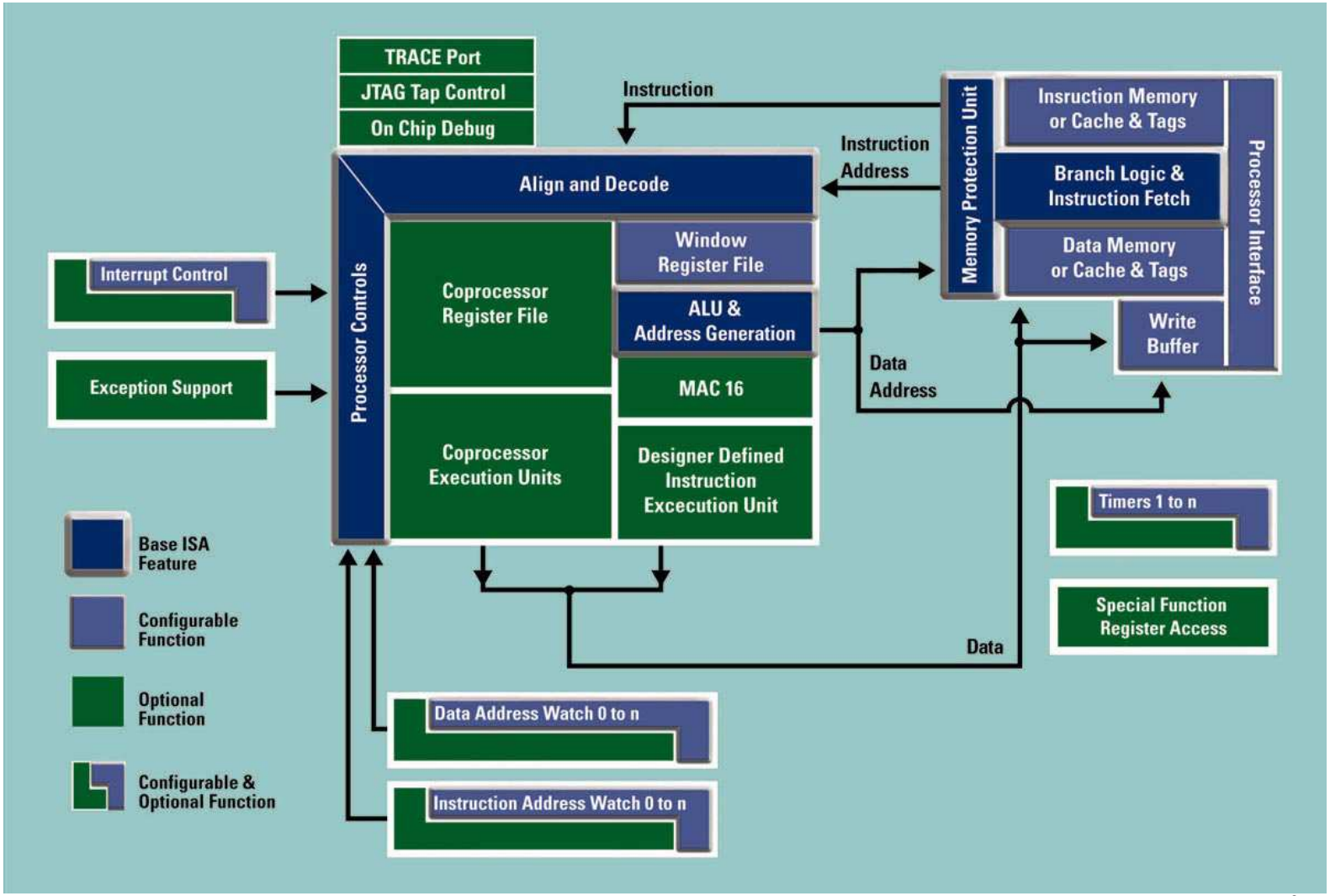
Customized Compiler, Assembler, Linker, Debugger, Simulator, Eval Board





Use a standard cell library to target to the silicon process

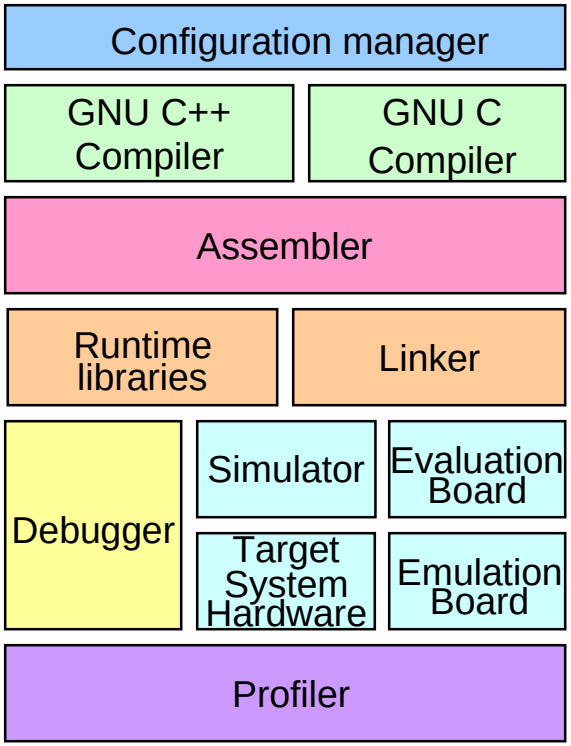
Xtensa is designed for configurability and extensibility



SW development tools are ‘in-sync’

Built on top of industry standard GNU tools

All tools automatically tuned and extended for application-specific instructions and configuration



- ✔ANSI C/C++ Compiler
- ✔Assembler
- ✔Debugger
- ✔Linker
- ✔Code Profiler
- ✔Instruction Set Simulator
- ✔Function Libraries

Extensibility via TIE Language

TIE (Tensilica Instruction Extension)

- Basis for Designer-Defined Instructions
- Describes instruction encoding and semantics
  - Semantics in Verilog subset
- Is independent of pipeline
  - Easy to write
  - Same description will work with future Tensilica designs
- Translated to Verilog, VHDL, and C
  - RTL, Compiler, Assembler, Simulator, Performance model, and Debugger support is automatic
  - Decode, interlock, bypass, and immediate logic generated from encoding descriptions

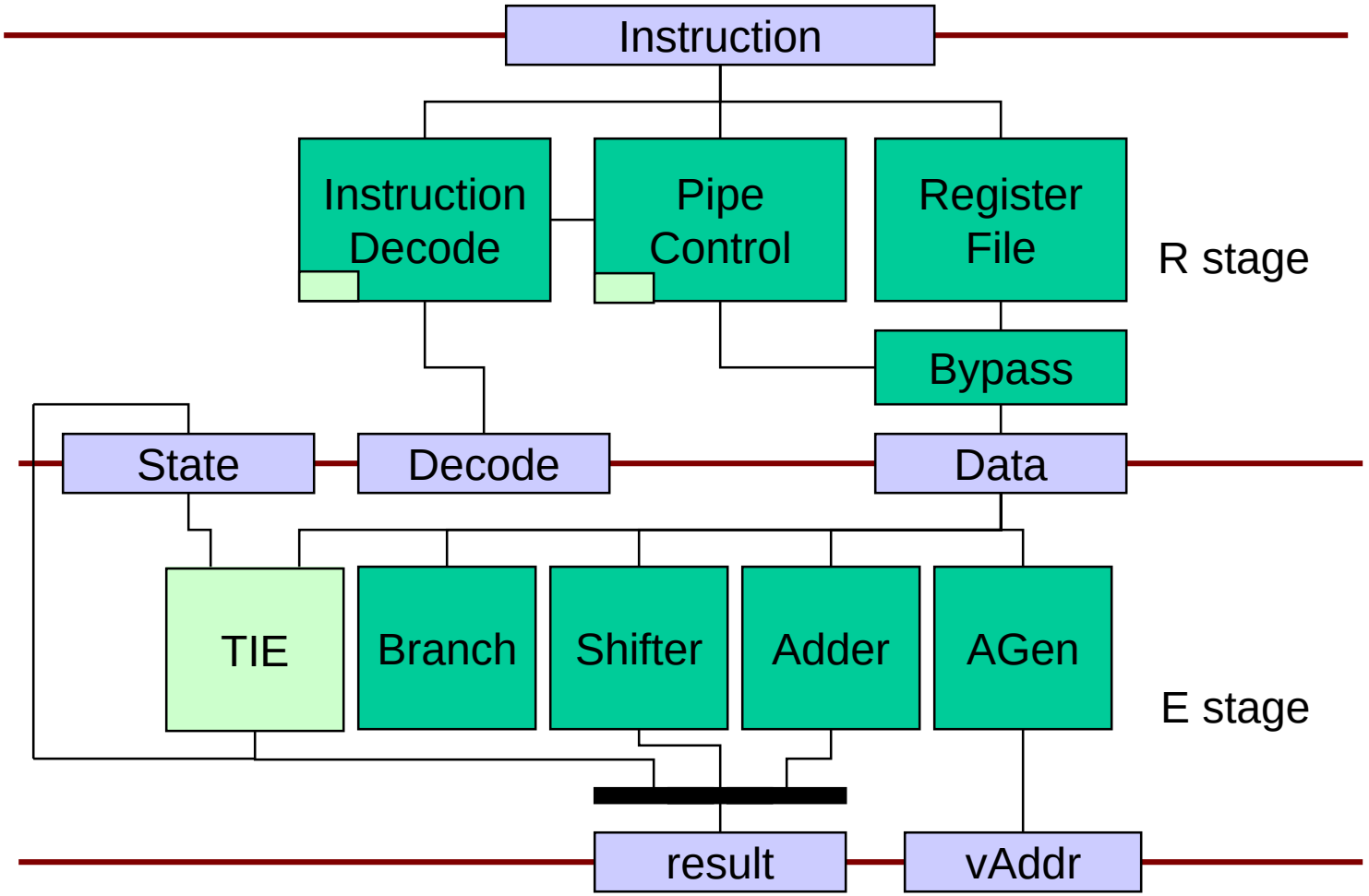
Simple TIE Example

```
// define a new opcode for byteswap
opcode  BYTESWAP  op2=4'b0000  CUST0

// define a new instruction class
iclass  bs  {BYTESWAP}   {out arr, in ars}

// semantic definition of byteswap
semantic bs {BYTESWAP} {
  assign arr =
   {ars[7:0],ars[15:8],ars[23:16],ars[31:24]};
}
```

Slightly More Complicated TIE

```
opcode  BYTESWAP  op2=4'b0000  CUST0

// declare state SWAP and ACCUM
state SWAP     1
state ACCUM    40

// map ACCUM and SWAP to user register file entries
user_register 0 ACCUM[31:0]
user_register 1 {SWAP, ACCUM[39:32]}

iclass  bs  {BYTESWAP}   {out arr, in ars} {in SWAP,
   inout ACCUM}

semantic bs {BYTESWAP} {
   wire [31:0] ars_swapped =
        {ars[7:0],ars[15:8],ars[23:16],ars[31:24]};
   assign arr = SWAP ? ars_swapped : ars;
   assign COUNT = COUNT + SWAP;
   assign ACCUM = {ACCUM[39:30] + ars_swapped[31:24],
                   ACCUM[29:20] + ars_swapped[23:16],
                   ACCUM[19:10] + ars_swapped[15:8],
                   ACCUM[ 9: 0] + ars_swapped[7:0]};
}
```

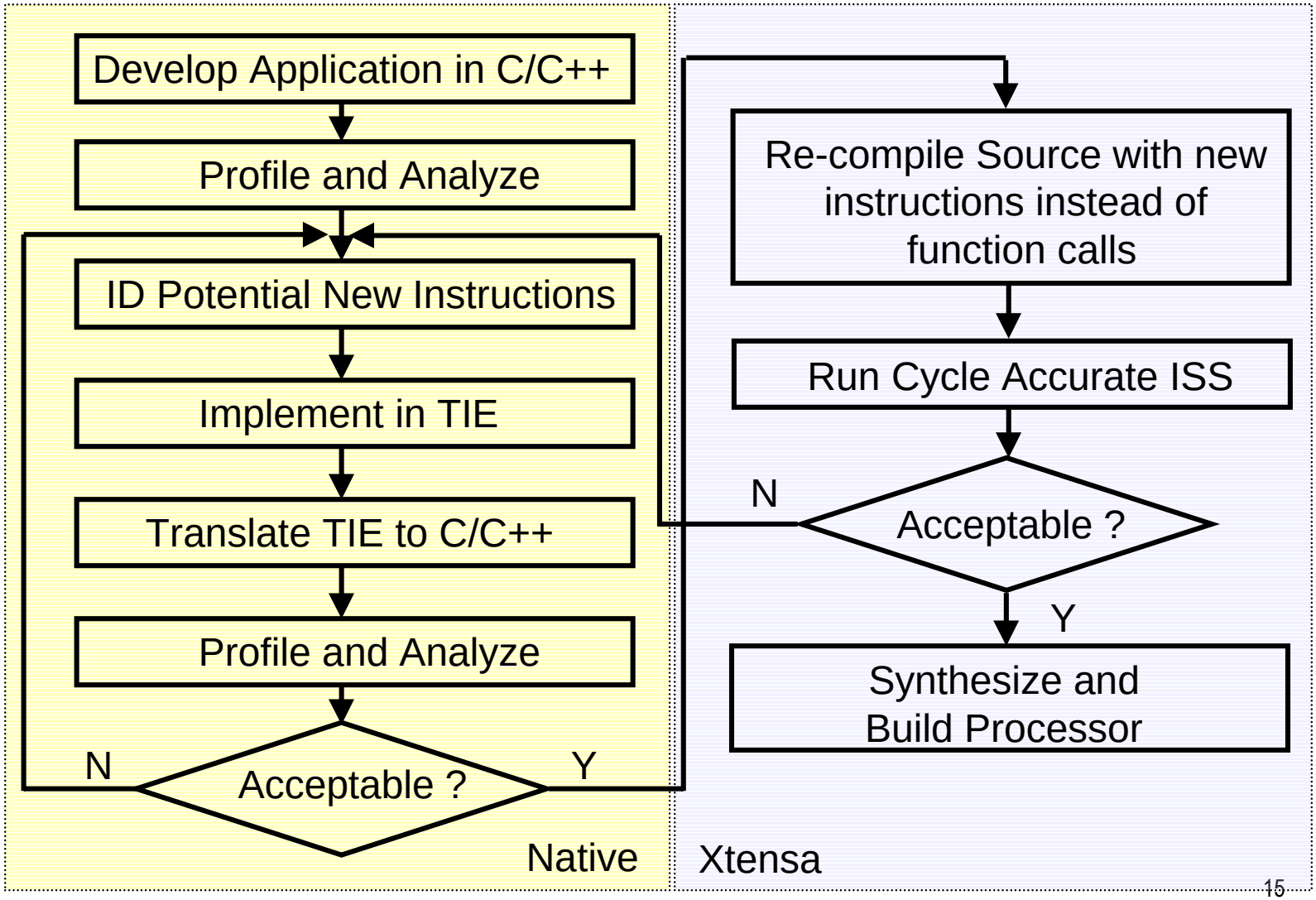
TIE Hardware Generation



TIE Software Support

- Translate from TIE to C for native development
- Add instructions to assembler, debugger, simulator, performance model
- Add intrinsics to the compiler
  - developer can code in C/C++
    e.g. write
    `out[I] = byteswap(in[I]);`

Typical TIE Development Cycle



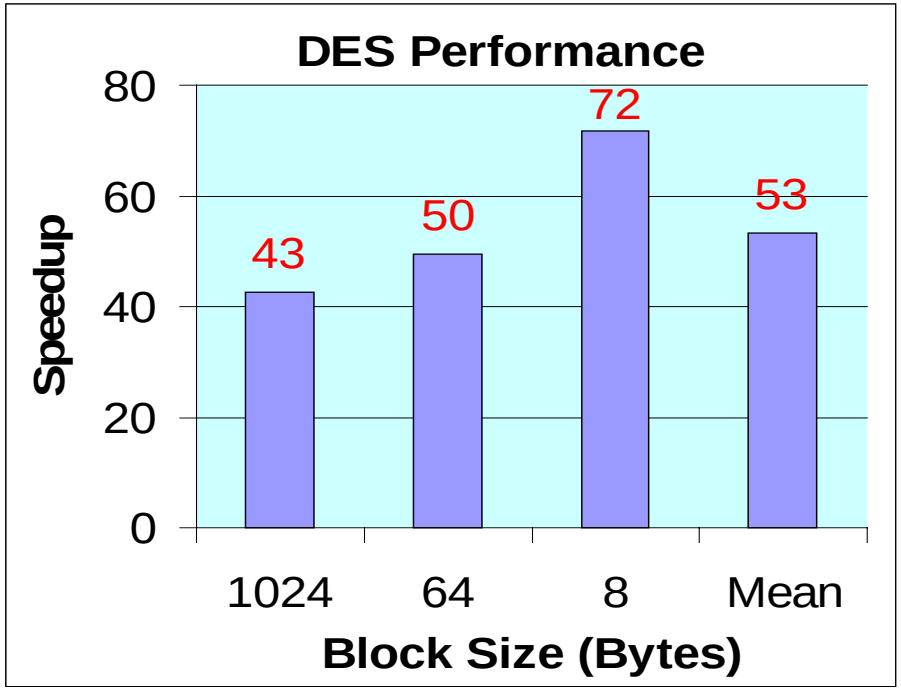
Triple DES Example

Triple DES used for

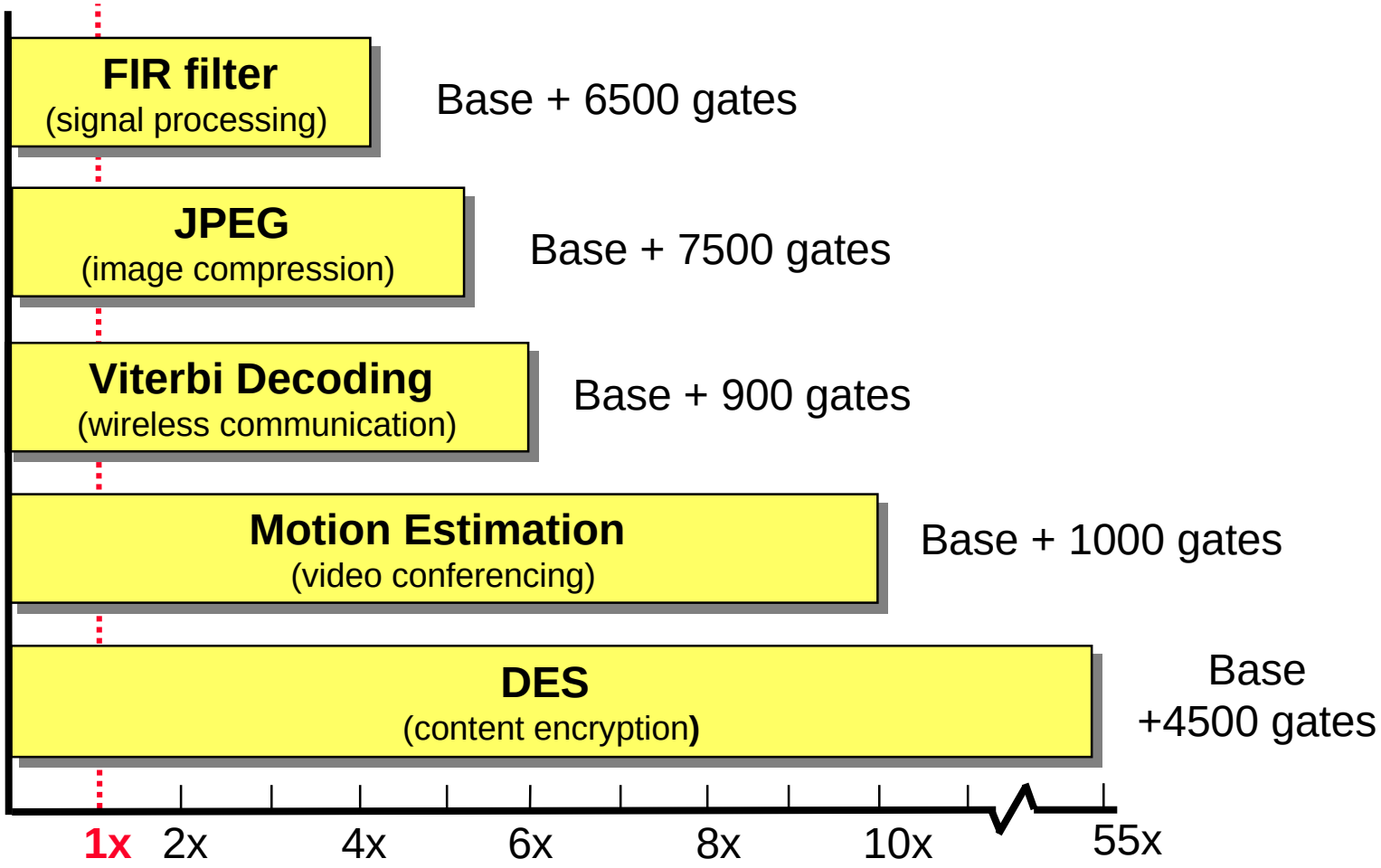
- Secure Shell Tools (SSH)
- Internet Protocol for Security (IPSEC)

Add 4 TIE instructions:

- Speed increased by 43X to 72X
- Code and data storage requirement reduced by 36X
- ~4500 additional gates
- No cycle time impact



Application-specific processors make a huge difference in performance



Improvement in MIPS, MIPS/Watt over general-purpose 32b RISC

Extension via TIE vs. special logic outside the CPU

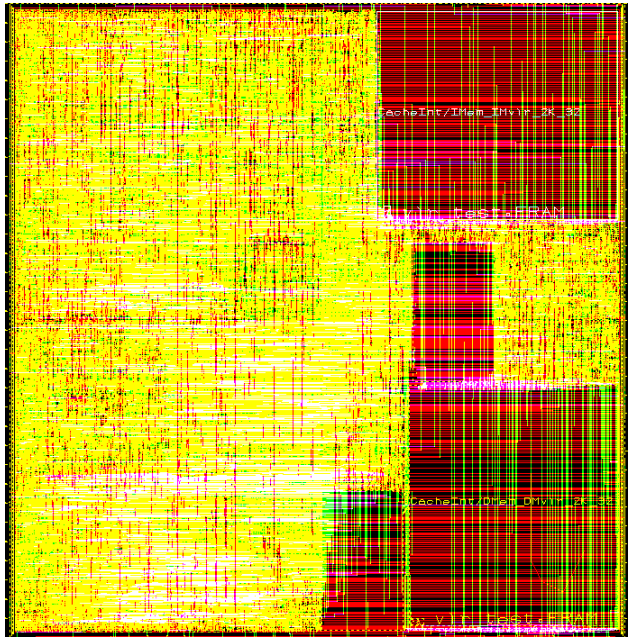
Advantages	Disadvantages
+ No latency introduced by communication between processor and special hardware	– Some Verilog constructs not supported
+ Easier to accomplish complicated control using the instruction stream	
+ Field upgrade or fix bugs by changing code not hardware	
+ Easier prototyping, verification and debug	

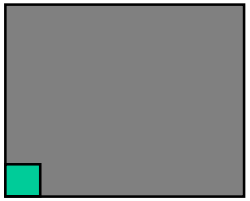
Extension via TIE vs. RTL Modification

TIE is a high-level specification:

- Software generated to match hardware
- Easier to write
  - don't need to understand pipeline
  - mixed native/cross development methodology
- Easier to verify instruction definition
  - verify in C, not by simulating RTL
  - correct by construction pipeline etc. logic
- Faster to market
- Faster design iteration, better feedback produces better final architecture

The tail shouldn't wag the dog





Custom processors tie the system to a foundry

But the processor may be a small part of the total design – why is it forcing the foundry choice?

Xtensa uses virtually any standard cell library with commercial memory generators

to scale on a typical $10 IC (3-6% of 60mm^2)

Synthesized vs. custom (1)

Custom Pros:

- Potentially highest MHz
  - dynamic circuits
  - controlled layout
- Good micro-architecture support
  - CAM structures for TLBs, address buffers, etc.
  - specialized RAMs (e.g. pipelined)

Custom Cons:

- Compromised by Porting
  - Designs rarely see volume in target process
  - Old designs fail to exploit new processes
- Hard to integrate
  - CAD tool differences
  - foreign layout
- Design+Debug longer and costlier
- Hard to modify
  - cannot configure
  - cannot extend

Synthesized vs. custom (2)

Synthesized Cons:

- Standard cell libraries
  - limited functionality
  - not designed for high-speed
- Portability implications
  - avoid tri-state
  - avoid 2-phase transparent latch clocking
- Large clocking overhead

Synthesized Pros:

- Portable
  - Moves to new foundries easily
  - Takes advantage of new process technology as it becomes available
- Allows configurability
  - Need only change netlist; layout is automatic
  - Extensibility delivers more than a few extra MHz
- Map RTL multiple ways
  - use low-power library
  - change synthesis goals

Conclusions

- Configurable/extensible processors are here
  - Key is integrated hardware/software solution
  - Boon for system designers
- Synthesizable processors are effective solutions
  - Competitive with traditional offerings in MHz, mm^2, mW
  - Portable and easier to work with
- Advantage of extensibility overwhelms the limited (and often theoretical) advantages of custom design