



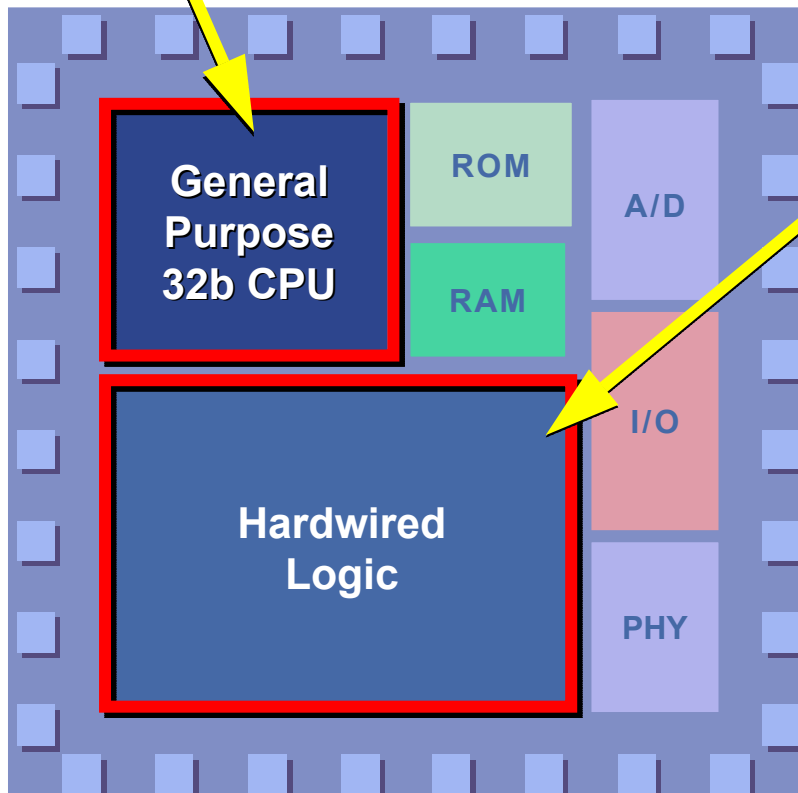
The Configurable Processor Company

Introducing **The Xtensa LX** **Processor Generator**

MAY 19, 2004

Ashish Dixit, VP Hardware Engineering
adixit@tensilica.com

General-purpose processors are not fast enough for data-intensive applications, don't have enough I/O or compute bandwidth



RTL - is the choice:

- High performance due to parallelism
- Large number of wires in/out of the block
- Languages/Tools familiar to many

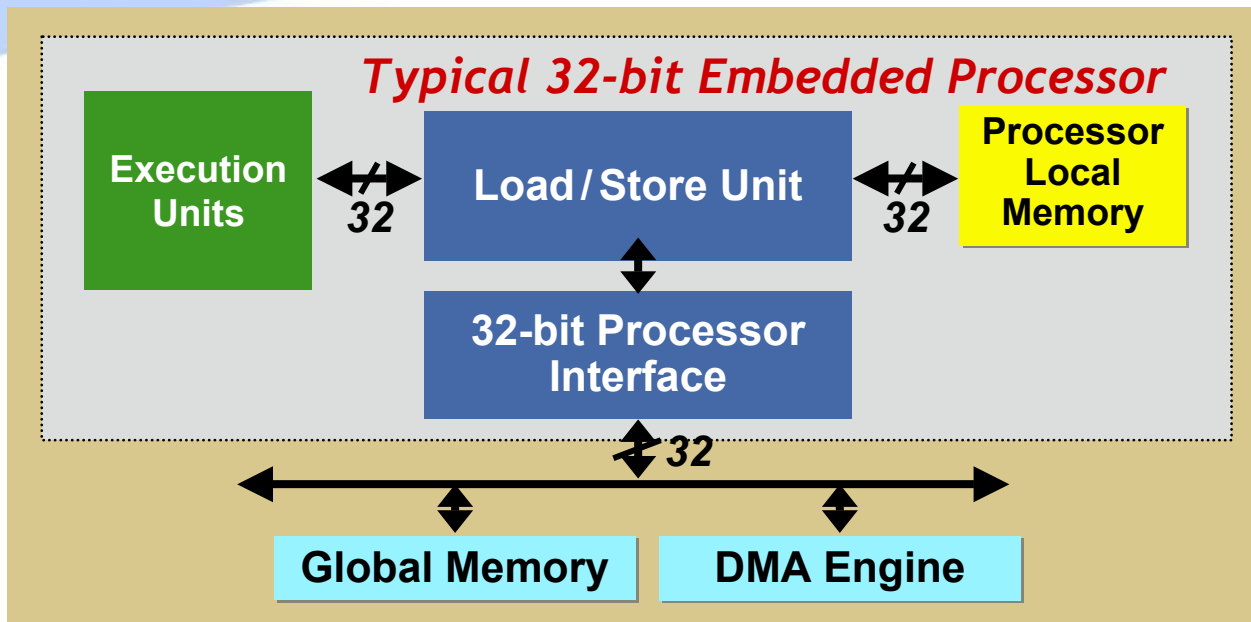
But ...

- Slow to design and verify
- Inflexible after tapeout
- High re-spin risk and cost
- Slows time to market

- **Higher performance and lower power for SOCs is becoming even more crucial**
 - Example: QCIF → CIF → D1 drives *I/O throughput* 16X
 - Example: H.264 motion estimation requires 5x to 10x *more computation* than H.263 – in a handheld device!
 - Even line-power devices struggle with *power dissipation* as transistor counts in standard cell designs move above 100M
- **Algorithm complexity drives engineers towards programmable solutions (processors)**
 - However, low power and higher performance conflict
 - Higher processor frequency alone is insufficient

Limitations of Conventional Embedded Cores

Sample Pseudo-code



```

inst1:    lw r2, (r1)
<memory latency>
inst2:    RISCop1 r3,r2
inst3:    RISCop2 r4,r3
inst4:    RISCop3 r5,r4
...
instn:    sw r15, (r14)
  
```

Limitations:

- 1 operation / cycle
- Load/Store overhead for every 32b data element processed

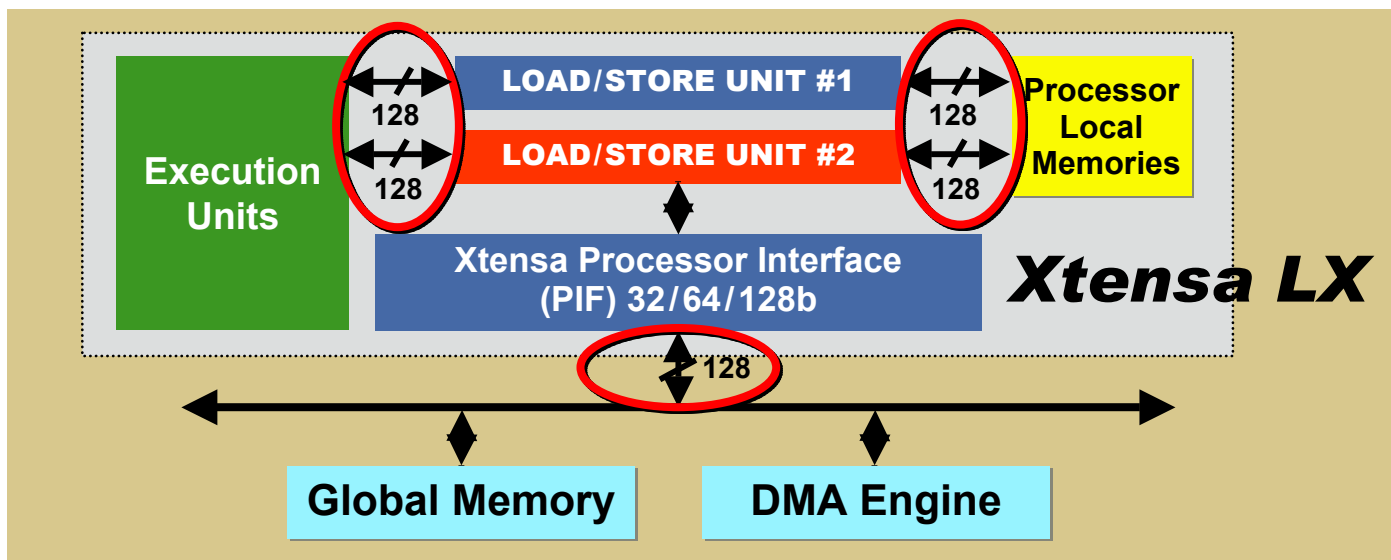
Xtensa LX Goals

- High data bandwidth
- High computation performance
- Low Power
- Programmability

Xtensa LX: Programmable building blocks for SoCs

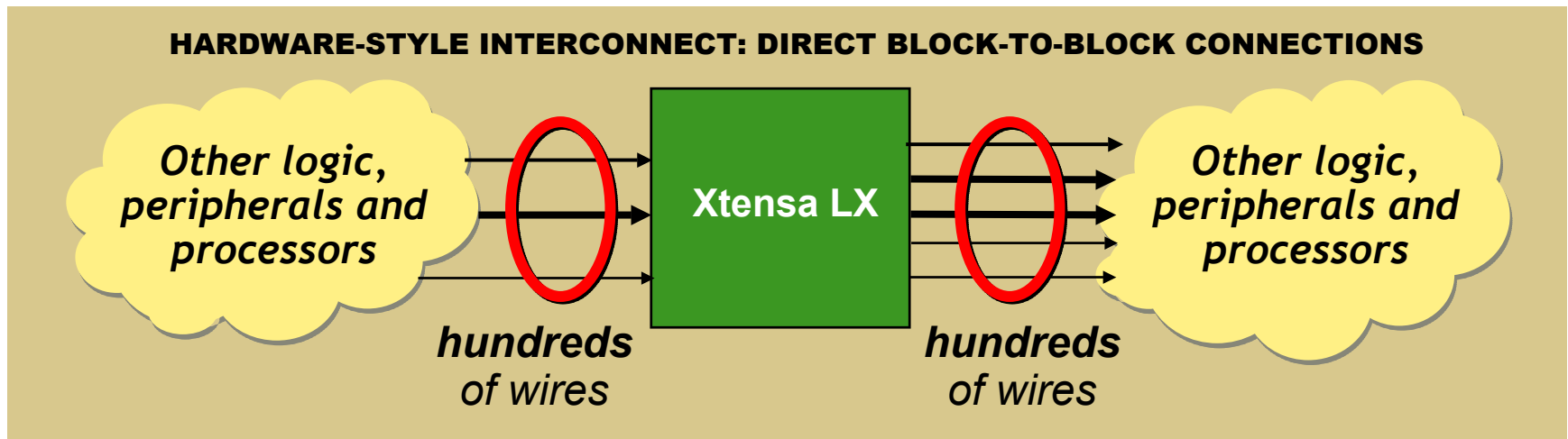
Xtensa LX: Higher *Conventional* Data Bandwidth

- Optional second load /store unit
- Optional wide interfaces – up to 128 bits
- Compound instructions allow simultaneous data movement and computation for higher sustained data throughput
- Master/slave PIF hides remote access latency
 - External device can read/write Xtensa's local memory thru PIF
 - Example: Transfer to one RAM, while processing out of others
- Up to 19 GB/sec of data throughput @ 400 MHz



Designer defined Ports

- Wires registered directly to/from execution units
- RTL-style interblock connectivity, but with instruction semantics
- Processor generator automatically generates hardware RTL, ISS and system models, and programming primitives



Input Wires

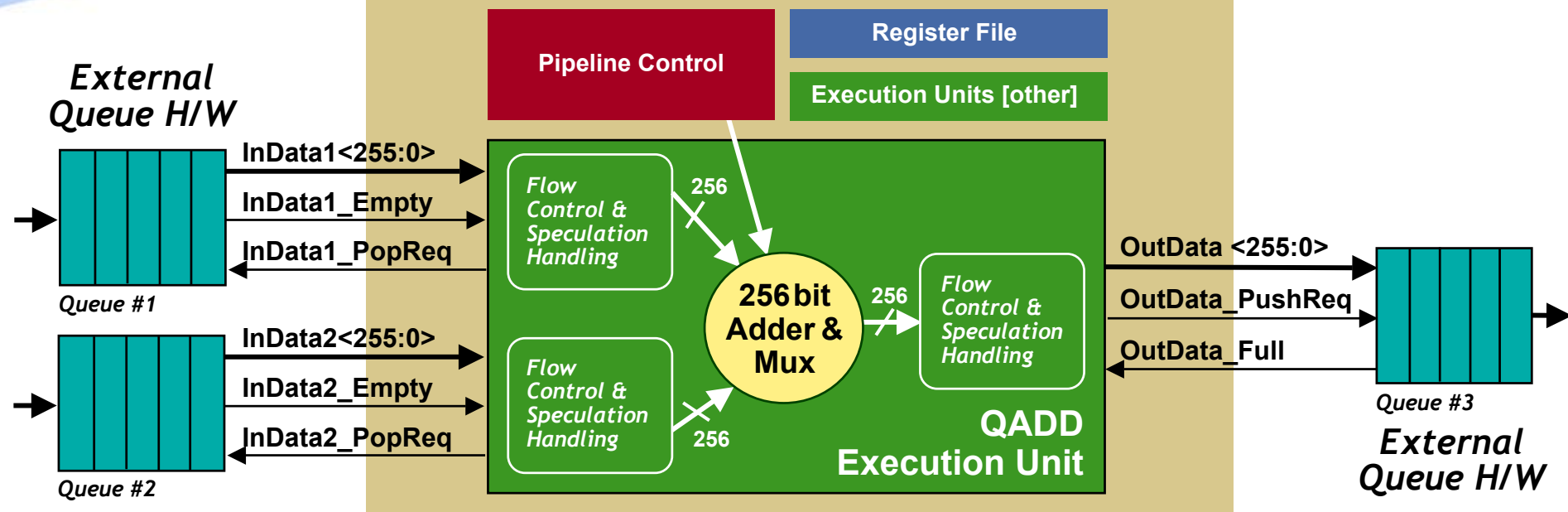
- TIE statement defines input wires [ports] of arbitrary width [1 to 1024]
- Input wire is registered and available for computation

Export State

- TIE statement defines output wires with committed value of a state

Xtensa LX: Designer Defined Interfaces – Queues

Xtensa LX



Example Tensilica Instruction Extension (TIE) Code:

```

queue InData1 256 in
queue InData2 256 in
queue OutData 256 out
operation QADD {} { in InData1, in InData2, in SumCtrl, out OutData}
{
assign OutData = SumCtrl ? (InData1 + InData2) : InData1;
}

```

Traditional Alternatives for Increasing Processor Operations / Clock

■ **Superscalar, out of order, multi-threading, etc.**

- Acceptable for general-purpose architectures
- Increases complexity – large gate count for control
- Large gate count not acceptable for SOC cores
 - Compared to RTL as the alternative implementation vehicle

■ **VLIW**

- Acceptable for general-purpose architectures
- Large increase in code size
- Parallel hardware acceptable for SOC – if tailored to application
- Large core size & large on-chip RAM/ROM not acceptable for embedded SOC cores

■ **SOC design demands**

- High performance (parallel execution)
- Minimal [or no] increase in gate count or code size

Xtensa LX: Higher Computation FLIX: Wider Instructions

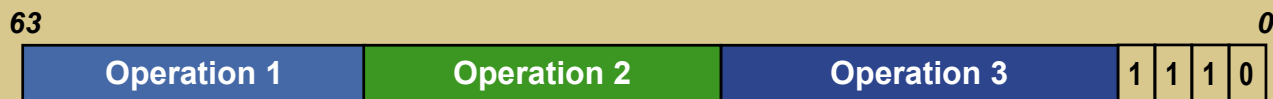
Additional instruction widths allow parallel operation

- 32 or 64 bit instruction widths added
- Fully compatible with existing 16/24 bit Xtensa ISA
- Freely, modelessly intermixed with 16/24 bit instructions
- High compute performance, without code bloat

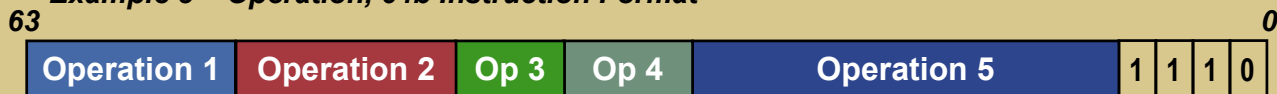
Multiple operations defined per wide instruction

- 1 to “N” operations per FLIX instruction format
- Mix and match different formats in a single core
- Only ~ 2000 gates of added instruction decode/control HW

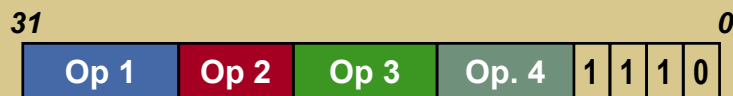
Designer-Defined FLIX Instruction Formats with Designer-Defined Number of Operations



Example 3 – Operation, 64b Instruction Format



Example 5 – Operation, 64b Instruction Format



Example 4 – Operation, 32b Instruction Format



FLIX: Software Automation

■ TIE language enables concise description of FLIX instructions

■ TIE Compiler automatically generates

- All operation encodings
- RTL implementation
- Complete toolchain and models
 - Pipeline accurate ISS; XTMP system simulation API; Co-simulation BFM
 - XCC: C compiler that schedules and packs operations into wide instructions

TIE Code Example FLIX Instruction Definition

```
format x64 64 {slotMAC, slotALU}

slot_opcodes slotMAC {MUL16U, MAC16S}
slot_opcodes slotALU {ADD, SUB, MIN, MAX}

state ACC 40
operation MAC16S {in AR m0, in AR m1}
                {inout ACC}
{ assign ACC = ACC + m0[15:0] * m1[15:0] }
```

Resulting Compiler-Generated Assembly Code

```
l16ui a15, a3, 0 // 24-bit instruction
addi.n a3, 2 // 16-bit
{mac16s a15, a14; sub a9, a15, a13} // 64-bit FLIX
s16i a9, a4, 0 // 24-bit instruction
```



Simple FLIX Example: “Static Superscalar”

- **3 line TIE description generates “static superscalar” machine**
 - Many variations possible
- **2 slots of Xtensa base ISA Operations**
 - No new designer-defined functional units
- **XCC compiler statically pairs base ISA operations**

ACTUAL TIE CODE FOR STATIC SUPERSCALAR MACHINE

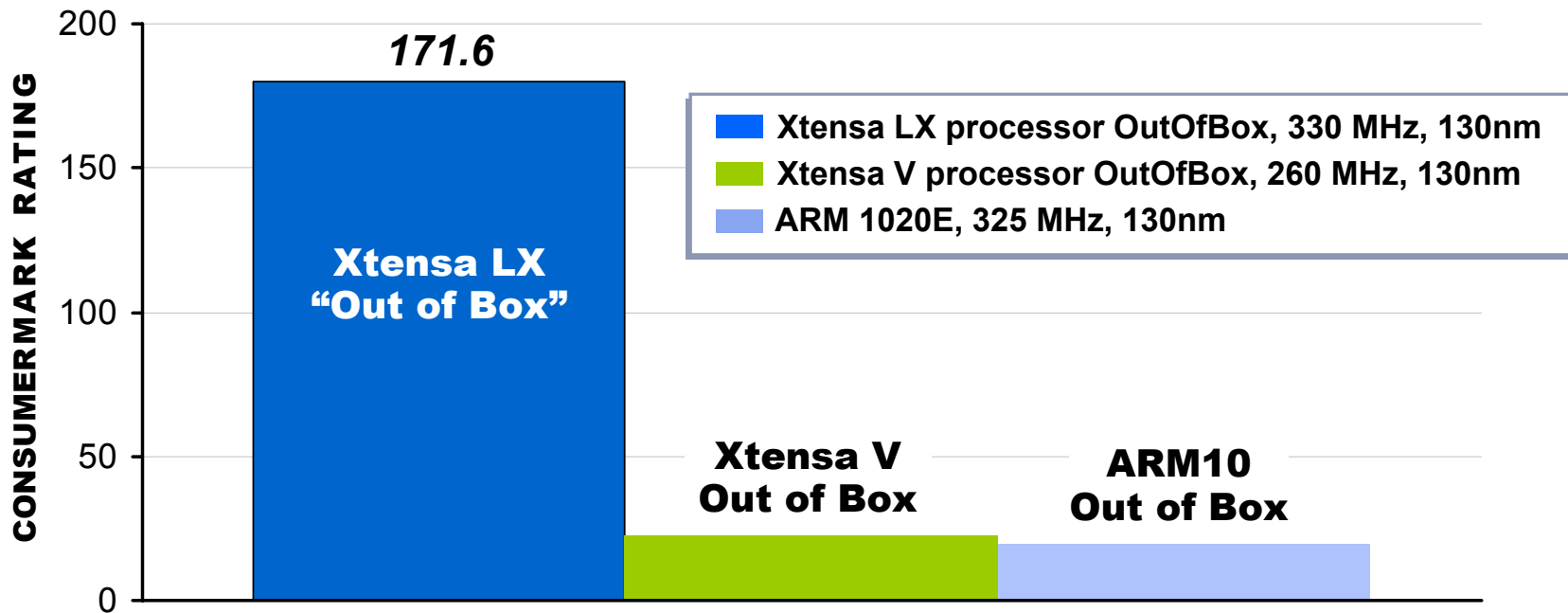
```
format f64 64 { slot0, slot1 }  
  
slot_opcodes slot0 { L32I, L32R, L16SI, L16UI, L8UI, S32I, S16I,  
S8I, MUL16S, MUL16U, MULL ABS, MAX, MAXU, MIN, MINU,  
NEG, EXTUI, SEXT, AND, OR, XOR, ADD, ADDI, ADDMI,  
ADDX2, ADDX4, ADDX8, SUB, SUBX2, SUBX4, SUBX8, SLL,  
SLLI, SRA, SRAI, SRC, SRL, SRLI, J, JX, BNEZ, BEQZ, BNE,  
BEQ, BLT, BGE, BGEZ, BLTZ, BGEU, BLTU, MOVI, MOV.N }  
  
slot_opcodes slot1 { ABS, NEG, EXTUI, SEXT, ADD, ADDI, ADDMI,  
ADDX2, ADDX4, ADDX8, SUB, SUBX2, SUBX4, SUBX8, SLL,  
SLLI, SRA, SRAI, SRC, SRL, SRLI, MOVI, MOV.N }
```

RESULTS

- **46% average performance increase compared to base-case (scalar) Xtensa LX on EEMBC Consumer Suite**
- **12,500 gates more than base Xtensa LX**
 - Approx 40K total gates
 - Includes 6-port base register file
- **325 MHz in 0.13 TSMC LV (worst case)**
- **Power: 160 μ W/Mhz**

Xtensa LX FLIX Configuration – Out of the Box

No manually generated extensions, no C code tuning



Tensilica Xtensa LX processor: simulator score, April 2004, using TSMC 130nm LVLK-OD process

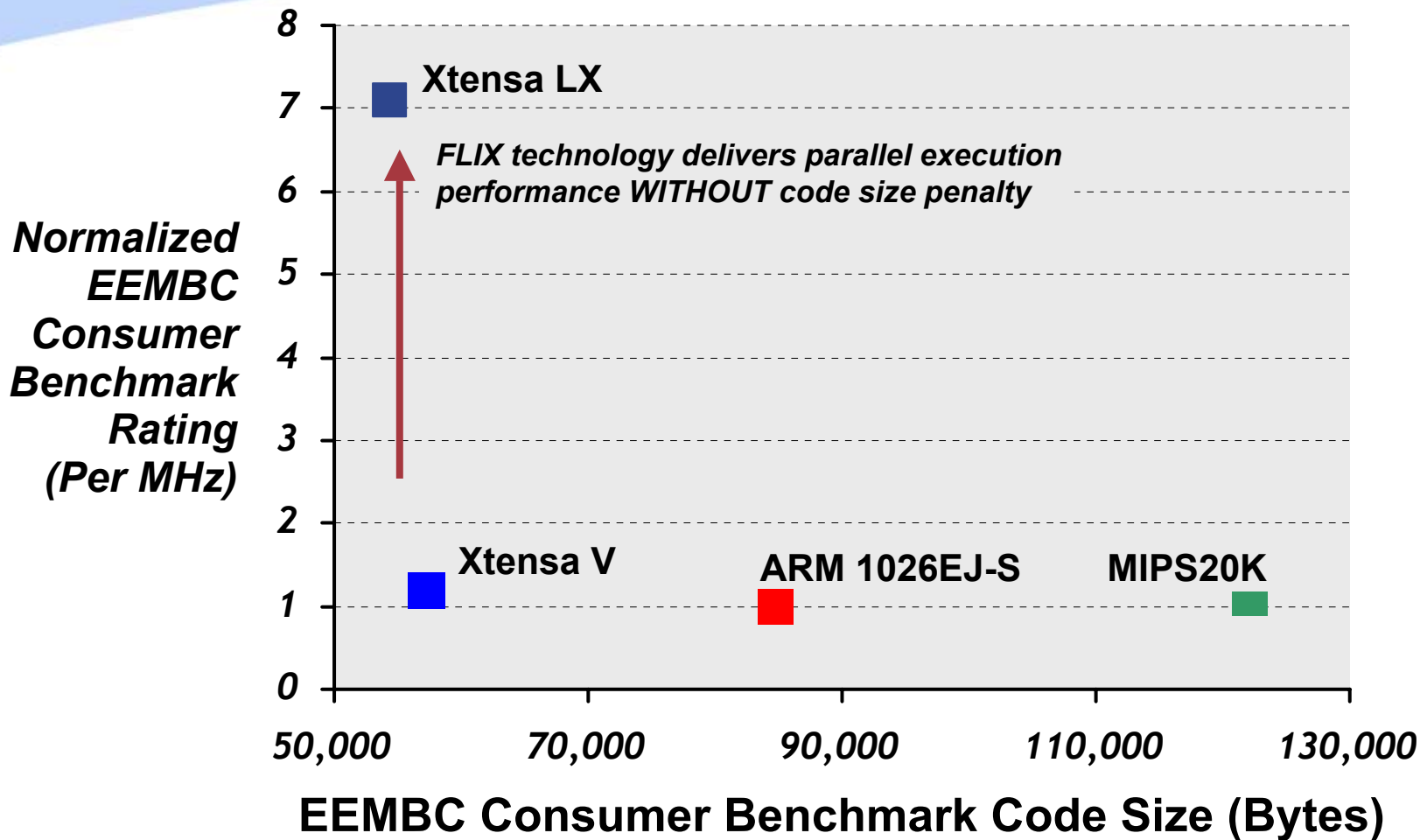
Tensilica Xtensa V processor: simulator score, July 2002, using TSMC 130nm LV process

ARM: ARM1020E architectural core, simulator

Competitive Data as of March 24, 2004 Source: www.eembc.org



FLIX Technology: High Performance, No “Code Bloat”



All score simulated and “Out-of-the-Box.”

■ Use fewer cycles to complete the task

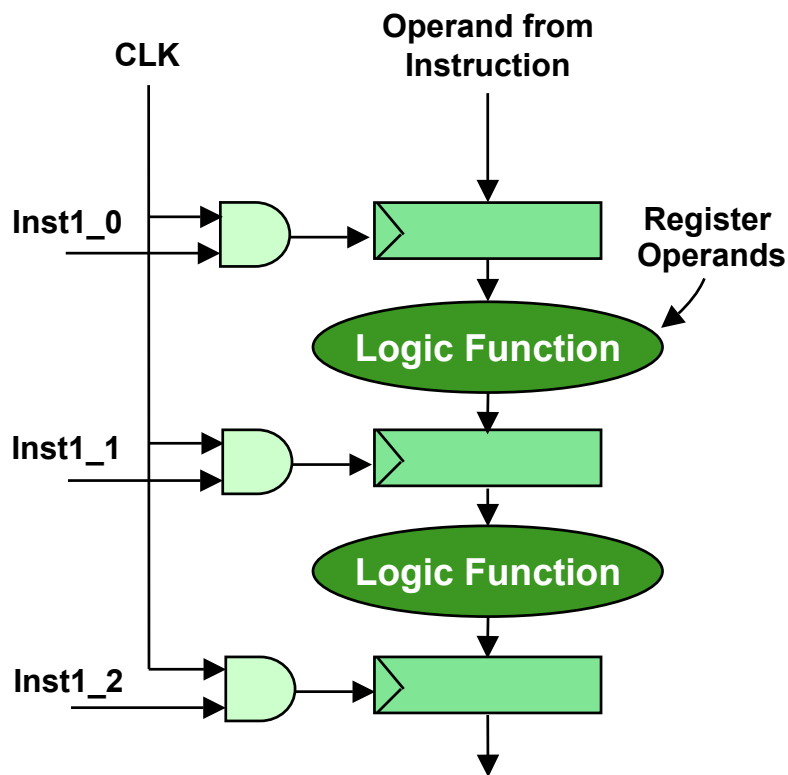
- Architectural power exploration and efficiency using extensions
- FLIX enables tasks to complete in fewer cycles
- **Example:** Viterbi code consumes 2 μJ with FLIX, compared to 66 μJ without it

■ Reduce signal switching

- Leave out functions that are not needed
- Software power management: Wait instruction
- System power management: RunStall input signal
- Automatic clock-gating for each instruction semantic

Automatic Instruction Semantic Fine-Grained Clock Gating

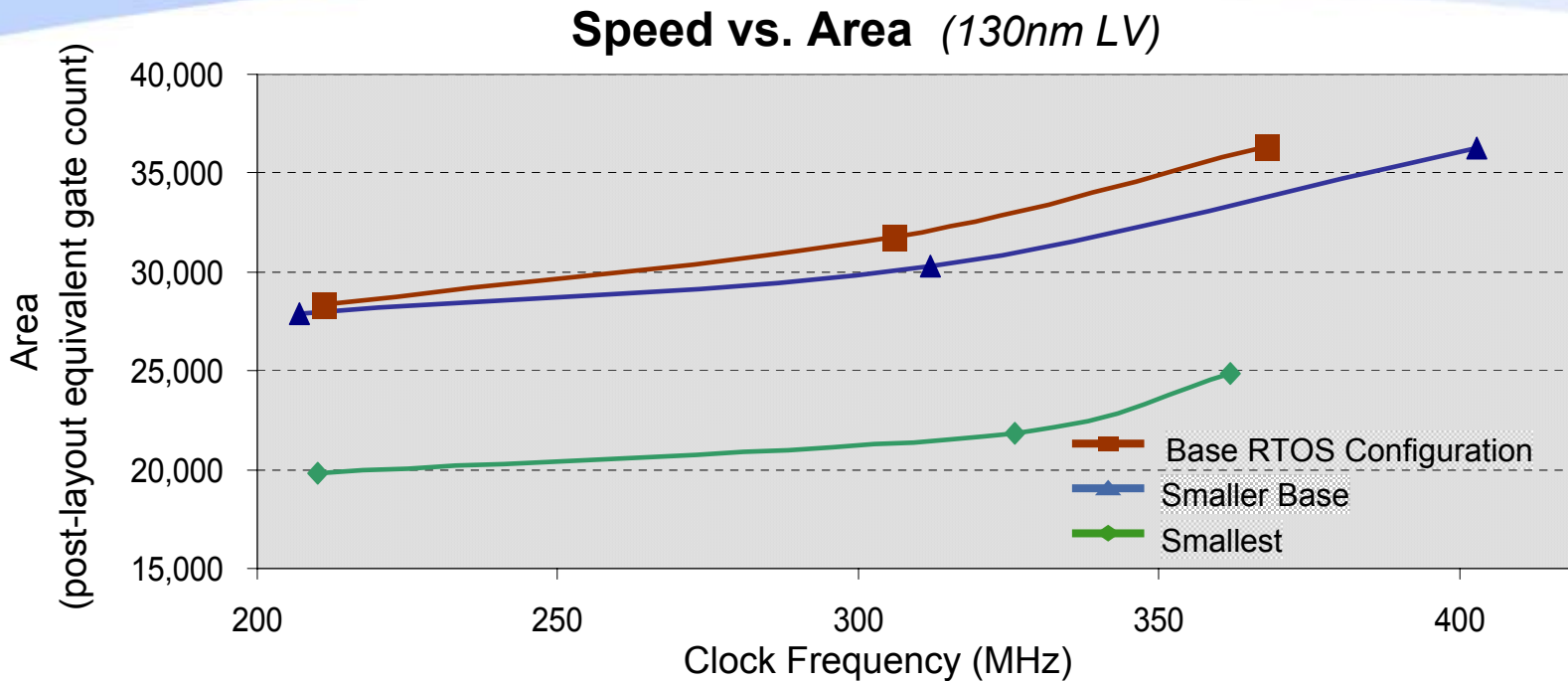
All instruction semantics automatically clock-gated by the TIE compiler






Power Dissipation @ 100Mhz	With Coarse Clock Gating	With Fine-Grained Clock Gating
Design #1 <i>(Floating Pt. block)</i>	8.37 mW	6.32 mW
Design #2 <i>(FLIX extensions)</i>	21.4 mW	16.36 mW

- Automatic clock gating provides 20 to 25% saving in power
- Xtensa LX minimum configuration has power dissipation of 38 μ W/MHz (130 nm LV, 1.0 V)

Xtensa LX: Power/Speed/Area Data



*Example
Xtensa LX
Configurations*

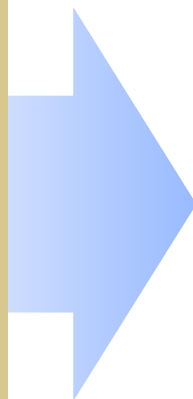
Configuration Description	Power (130nm LV, 1.0V)
 Base RTOS Configuration (RTOS ready – with Caches)	76 μW/MHz
 Smaller Base (no zero-overhead LOOPS)	47 μW/MHz
 Smallest (no Caches, no PIF, no zero-overhead LOOPS)	38 μW/MHz



HW & SW automatically generated

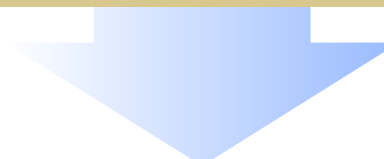
Xtensa Xplorer

- Integrated Development Environment based on Eclipse
- TIE Development tools
- C Development tools
- Profiling & visualization tools



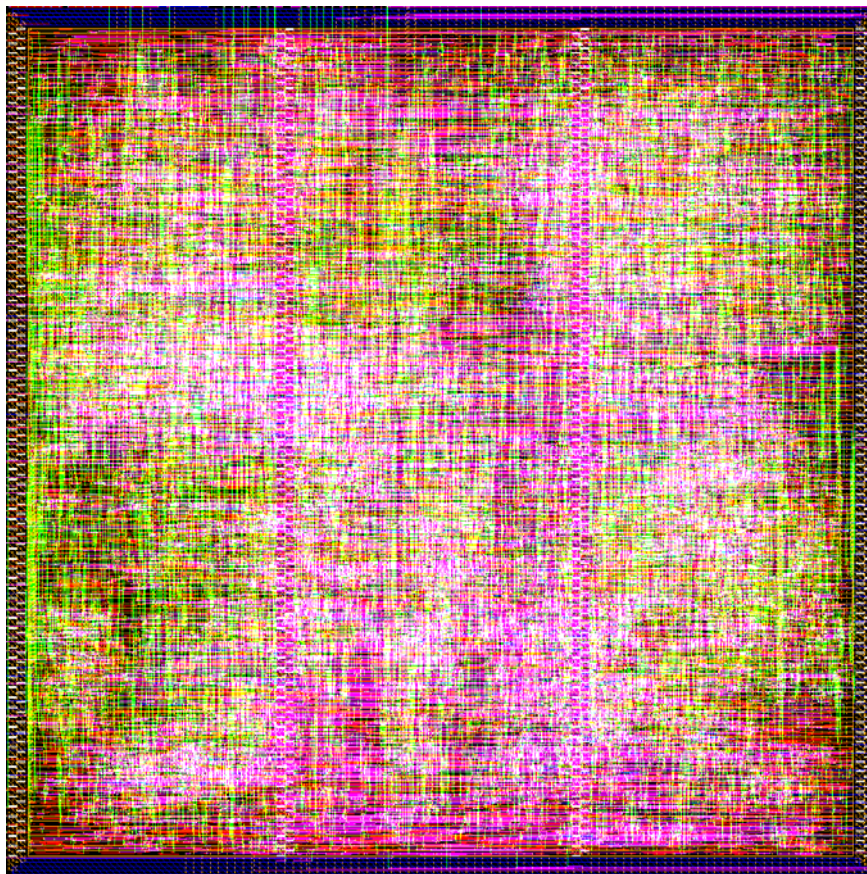
Software

- Scheduling assembler
- Xtensa C/C++ Compiler: vectorizing C/C++ compiler
- Xtensa Instruction Set Simulator – Pipeline accurate
- Debuggers
- XTMP: System Modeling API
- Bus Functional Model for HW/SW co-design environment
- RTOS: VxWorks, ATI Nucleus, XTOS



Hardware

- Synthesizeable RTL
- DC, PC flows
- Apollo, Silicon_Ensemble flows



Xtensa LX processor core layout - 90nm

Xtensa LX processor for SOCs – *an alternative to RTL design for compute intensive functions*

- High I/O and memory bandwidth
- High compute throughput with parallel operations
- Excellent energy efficiency

In use today by selected customers

- General release in Q3-2004