



*The Configurable Processor Company*

# Designing with Configurable Processors Instead of RTL

**1<sup>st</sup> International SOC  
Conference**

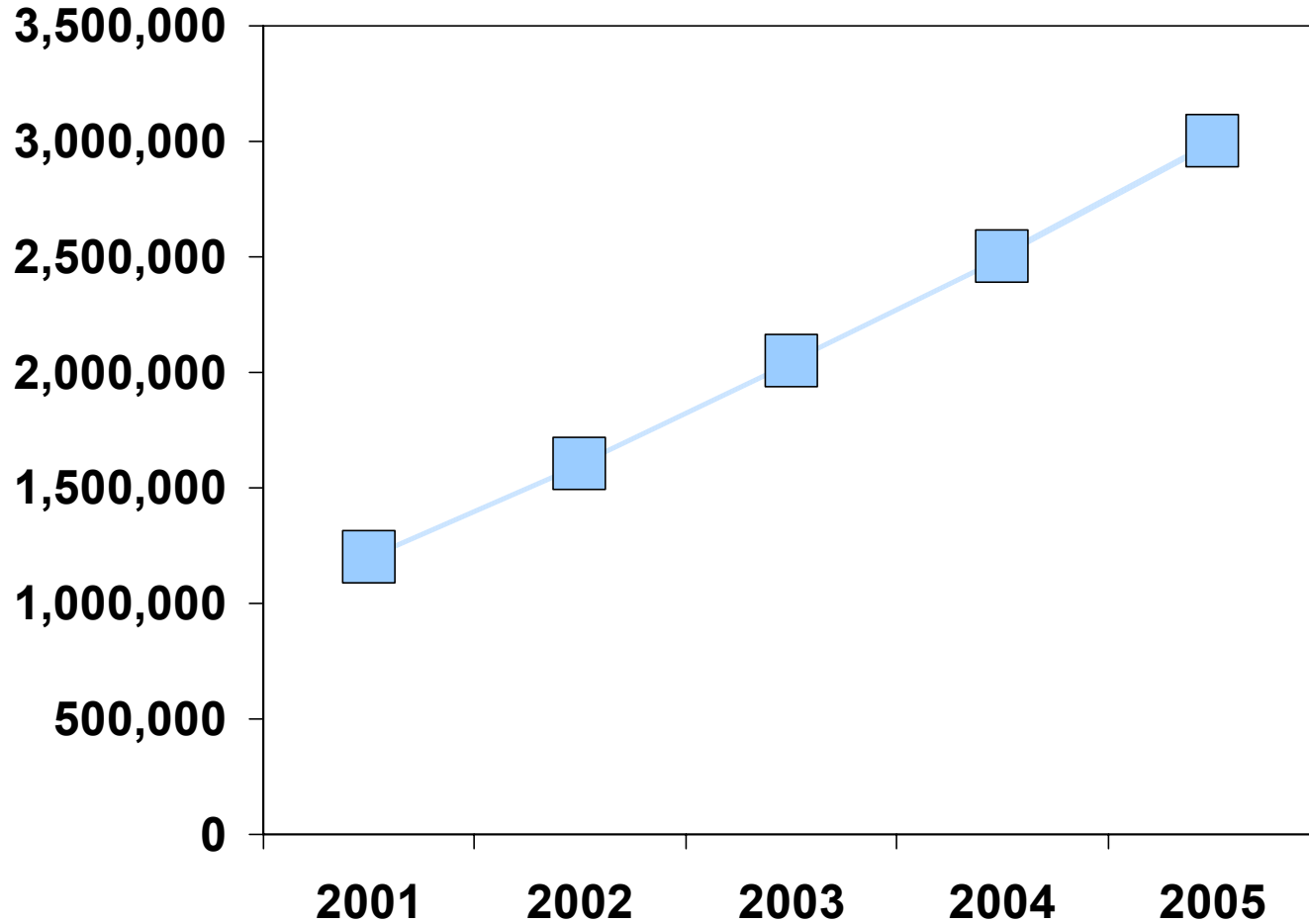
**Newport Beach, CA  
April 19, 2003**

**Presented by:**

Steve Leibson  
Technology Evangelist  
Tensilica, Inc  
[sleibson@tensilica.com](mailto:sleibson@tensilica.com)

# ICs hold more gates each year

Average Number of Gates Per New SoC

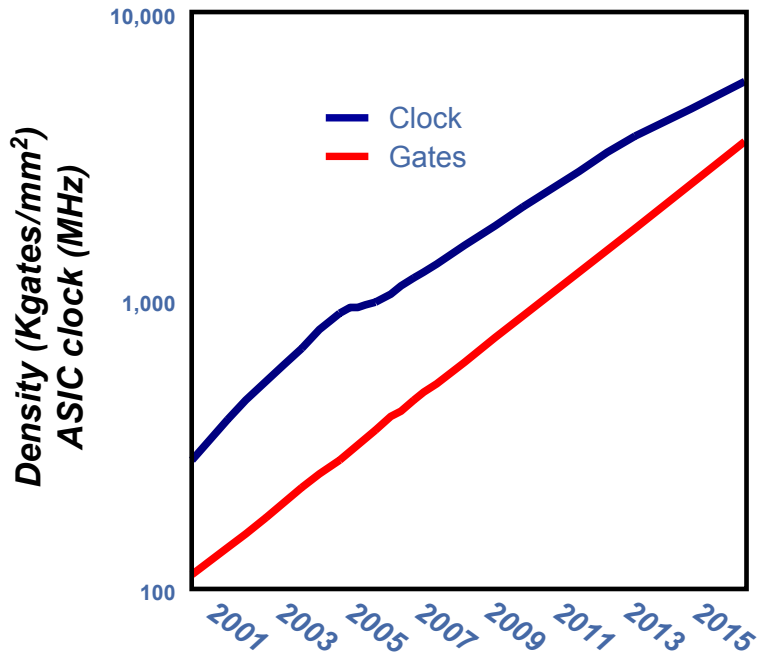


Source: Gartner Dataquest 2001

# ROI issue #1: The growing design-productivity gap

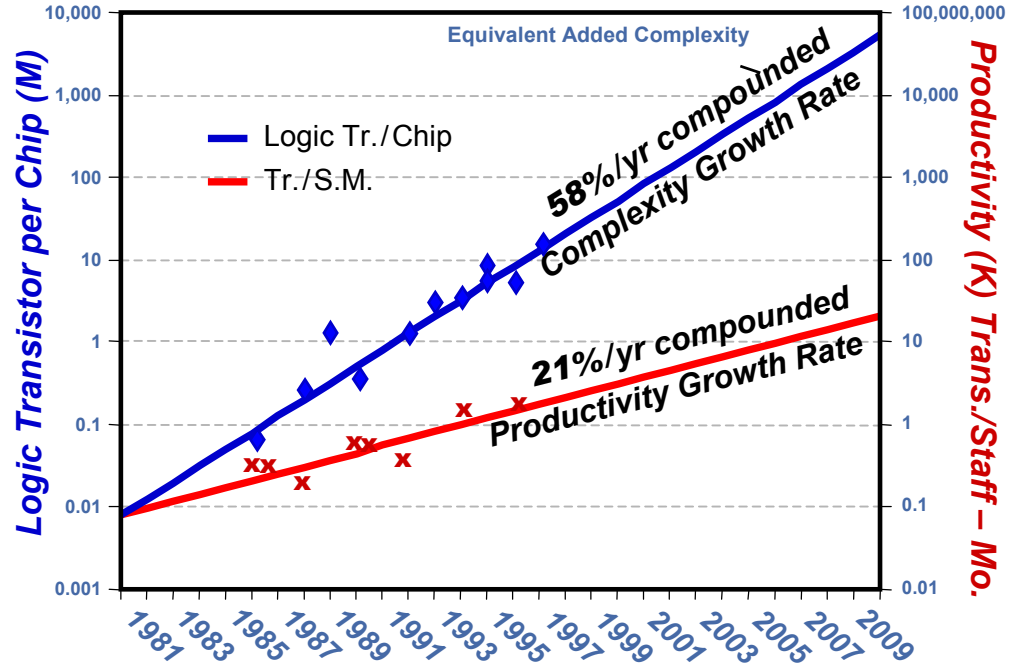
## Moore's Law:

Standard cell density and speed



## Design Productivity Crisis (SRC 1997)

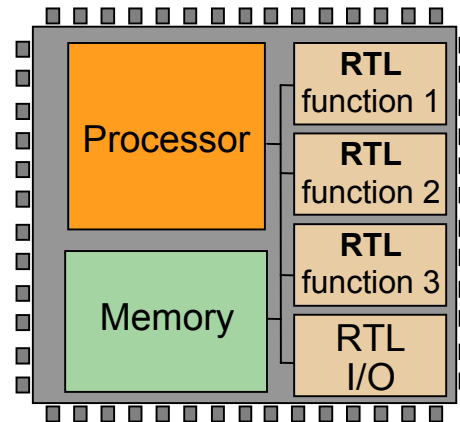
Potential Design Complexity and Designer Productivity



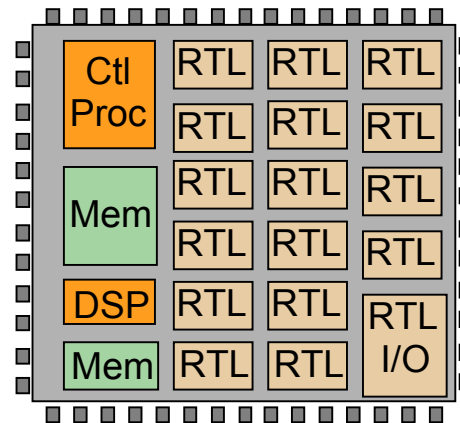
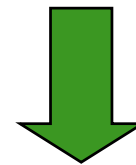
$$ROI = \frac{\text{Return}}{\text{Investment}} = \frac{\text{volume} * (\text{chip ASP} - \text{chip unit cost})}{\text{chip development cost}}$$

# The SOC meets Moore's Law

The growing number of transistors on an SOC drives the trend towards more RTL blocks on the chip



**Yesterday's SOC**



**Today's SOC**

# The problem with so many RTL blocks

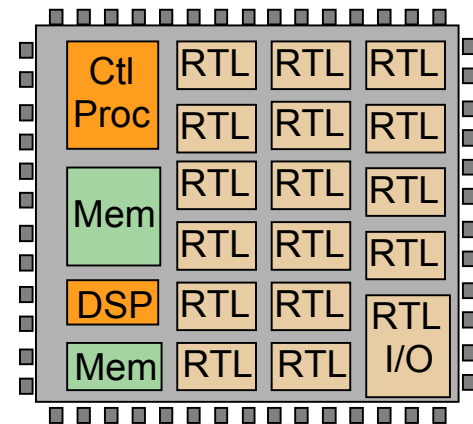
## Every RTL block must be:

Designed for function and reuse

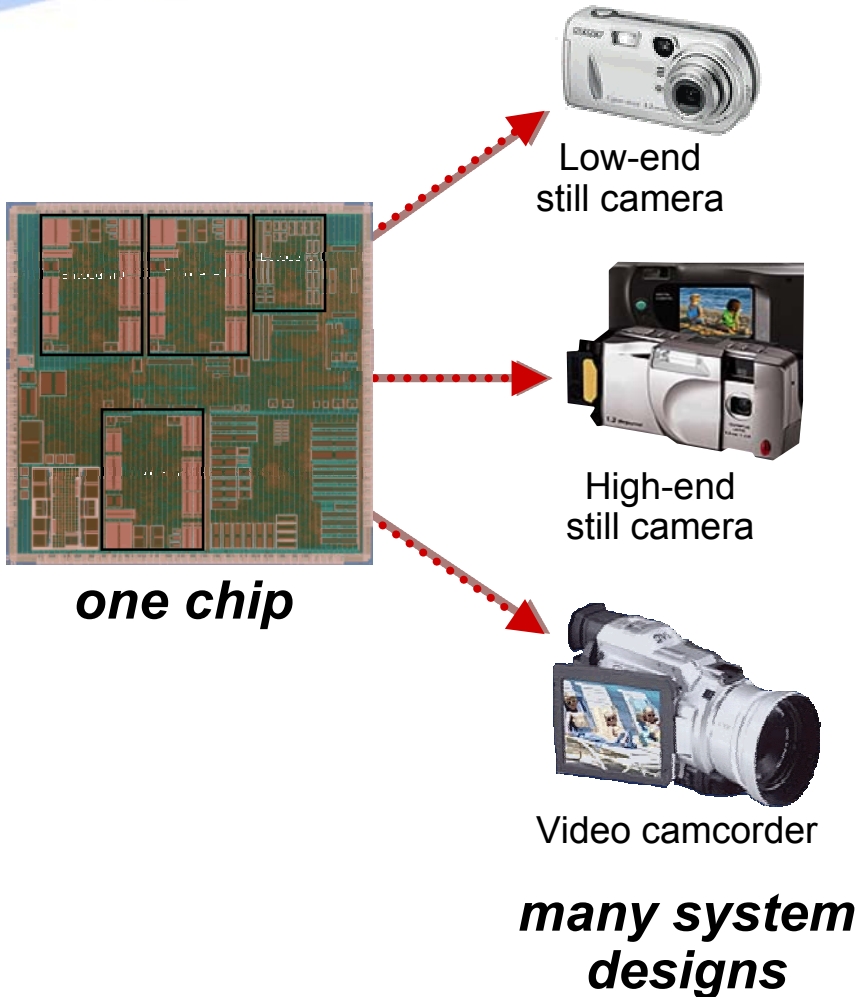
Verified (80% of the job)

Integrated

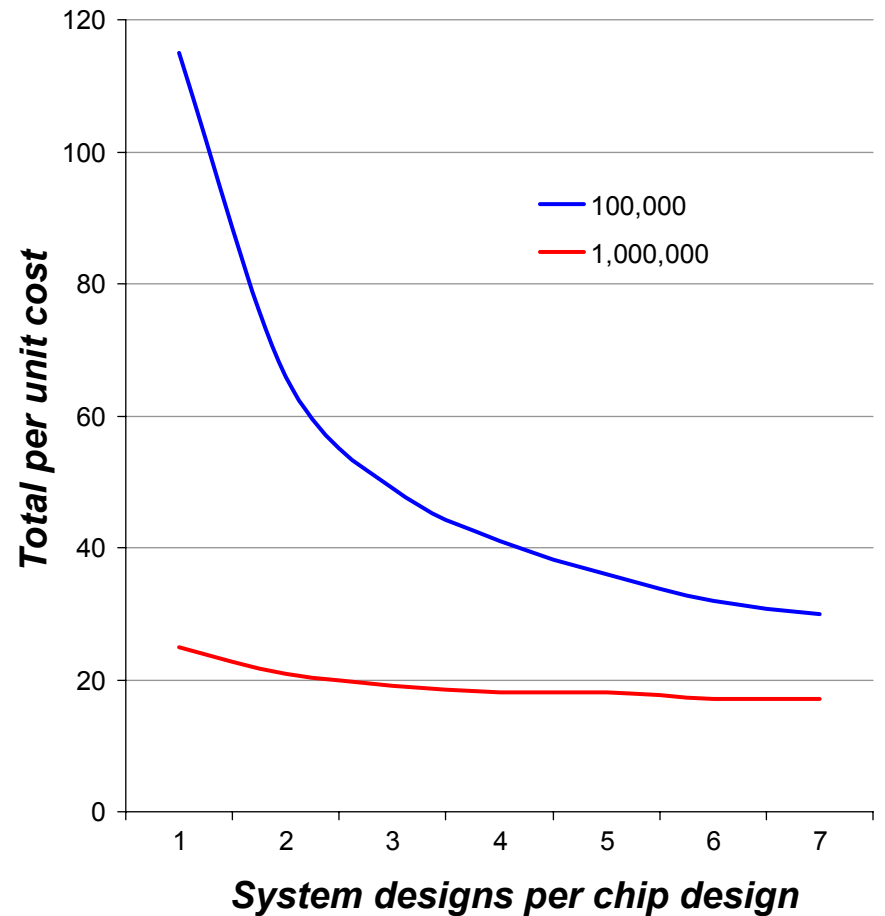
## Hard-coded RTL blocks are not flexible



# ROI Issue #2: One flexible SOC design, many system designs



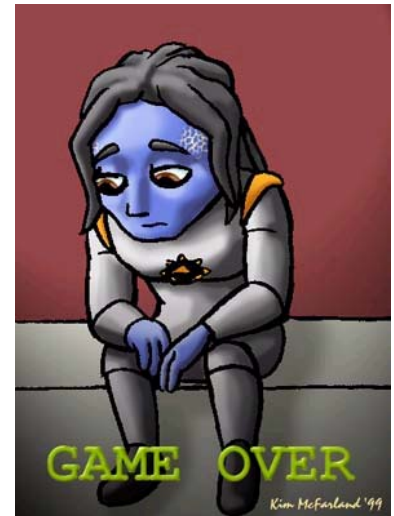
## SOC Flexibility = Per-Unit Cost Reduction (Model: 100K and 1M system volumes)



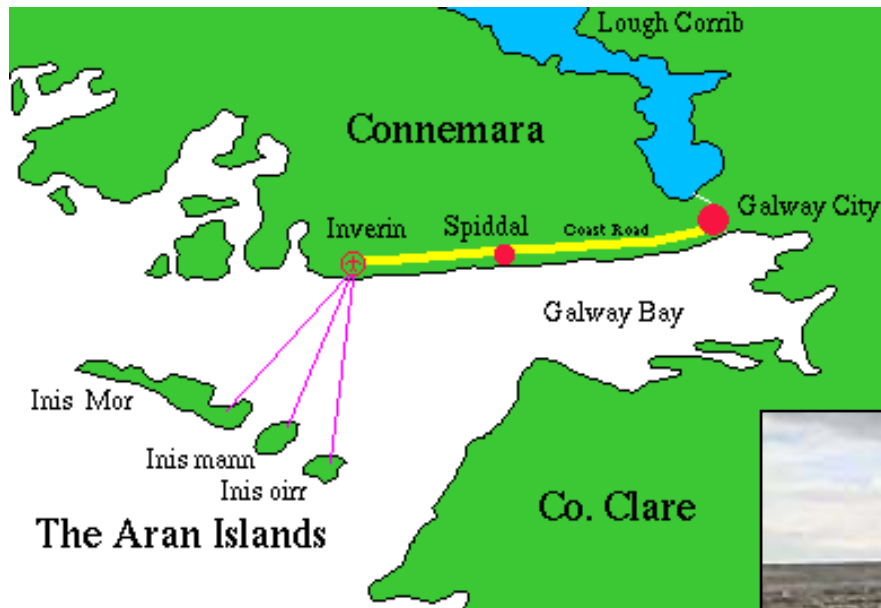
\$10M design cost, \$15 manf. cost, 5% premium for programmability

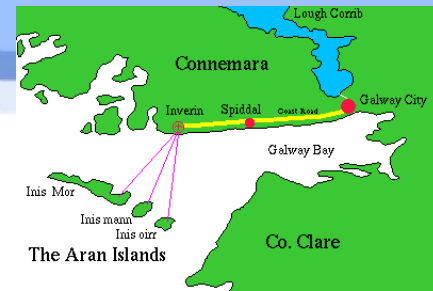
# How do you tackle the productivity gap while adding SOC flexibility?

- Wishing
- Hoping
- Praying
- IP Reuse
- None of these are working very well
  - GAME OVER?



# Fertilizer production in the Aran islands





## Each year, the locals gathered seaweed for fertilizer

Some used locally, some exported

## Once gathered, the seaweed was reduced to a mash

The transformation wasn't easy

The seaweed had to be boiled for days to produce mash

The Islanders had to maintain a fire for days

They had to cut peat to keep the fires burning because firewood is scarce

Peat is not very efficient, so it must be gathered in very large quantities

## Then, agricultural theory changed

Customers now preferred raw seaweed over mash

Raw seaweed is just as nourishing to the soil

Market value of the mash collapsed. Market value of raw product increased.

## The need for the labor-intensive production of mash vaporized

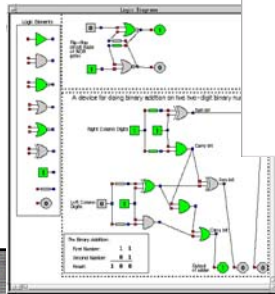
## It was years before the islanders changed their practices

Didn't want to lose a tradition

Didn't want to accept that previous exerted effort was not really needed

# Stages of IC design: Old IC-design habits die hard

Evolving IC-design methods chasing the growing design gap



**Increasing abstraction level**

**2000s - ???**

How will we design SOCs?

How to bridge the design gap?

**1990s - Verilog/VHDL/Synopsys Dynasty**

HDLs and logic synthesis – ASICs

**1980s - DMV Period (Daisy/Mentor/Valid)**

Gate-level schematics, logic simulation - LSI

**1970s – Applicon, Calma, and CV Era**

Transistor-level design - MSI

**1960s - Age of Rubylith**

Drawing polygons - SSI





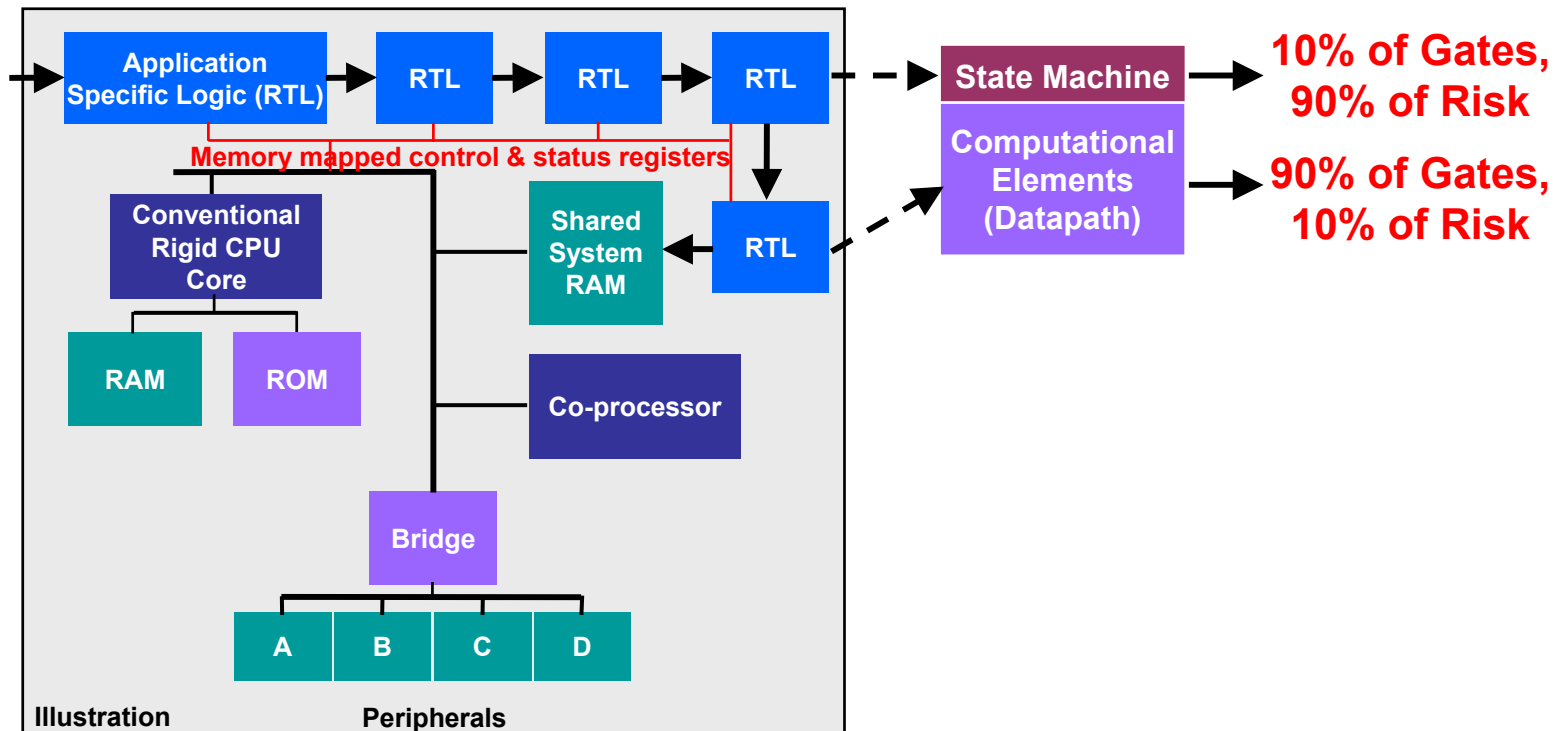
# Solving The SOC Design and ROI Issues:

- **Move to a higher level of abstraction**
- **Design with processors rather than RTL**

# Fixed RTL logic: The risk factors

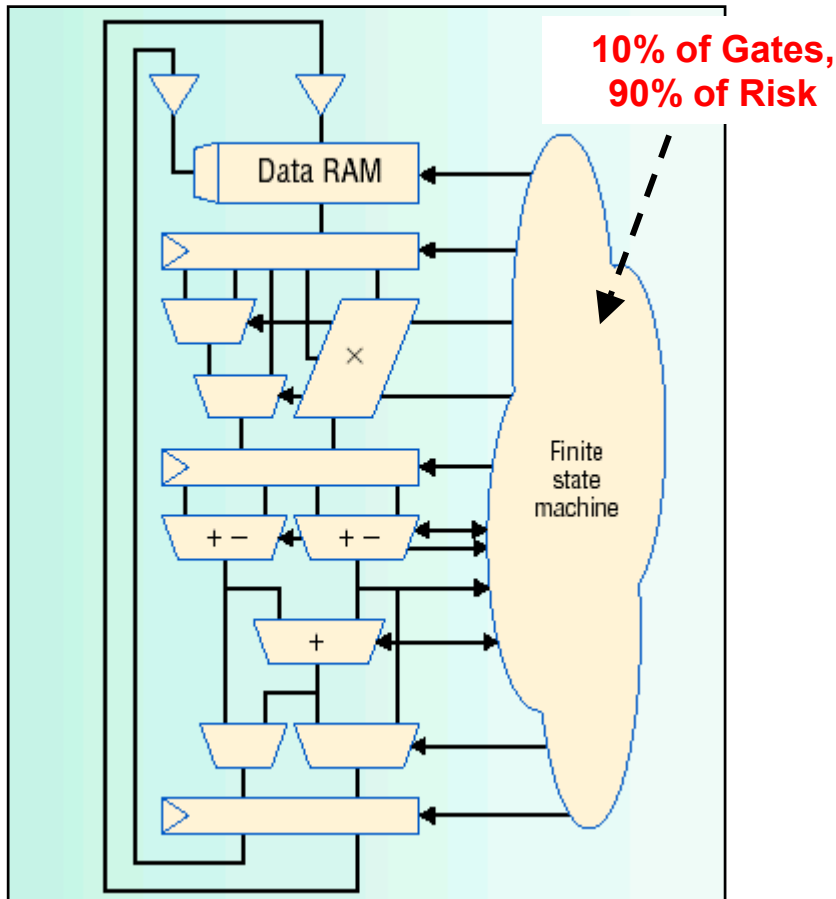
## RTL Logic Increases Risk, Slows Time To Market

- Verification of complex state machines
- Costly silicon respins to make changes
- Low flexibility: obsolescence as markets or standards change

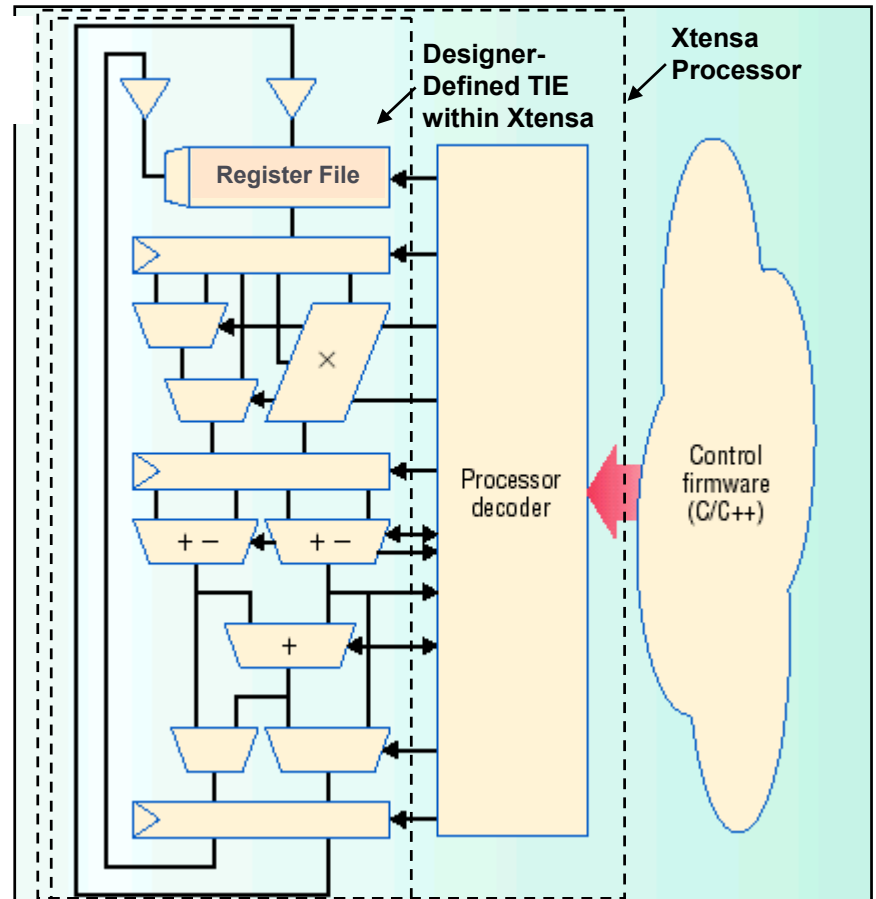


# An alternative to RTL design

## RTL-based SOC Design



## Xtensa-based SOC Design

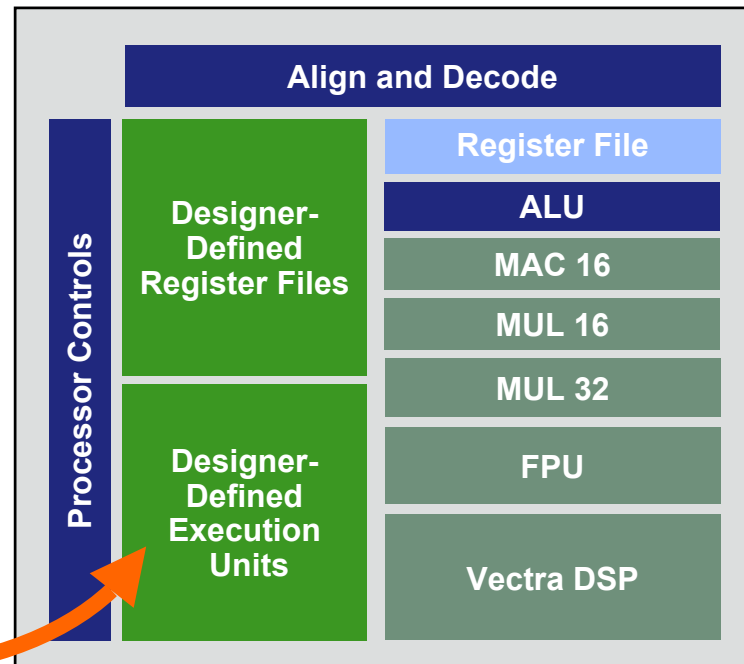
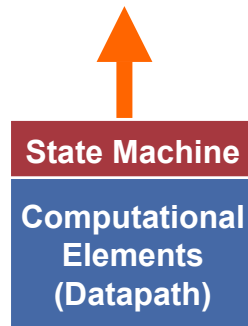


# Xtensa technology: Designer-defined execution units

What once was built  
in RTL...

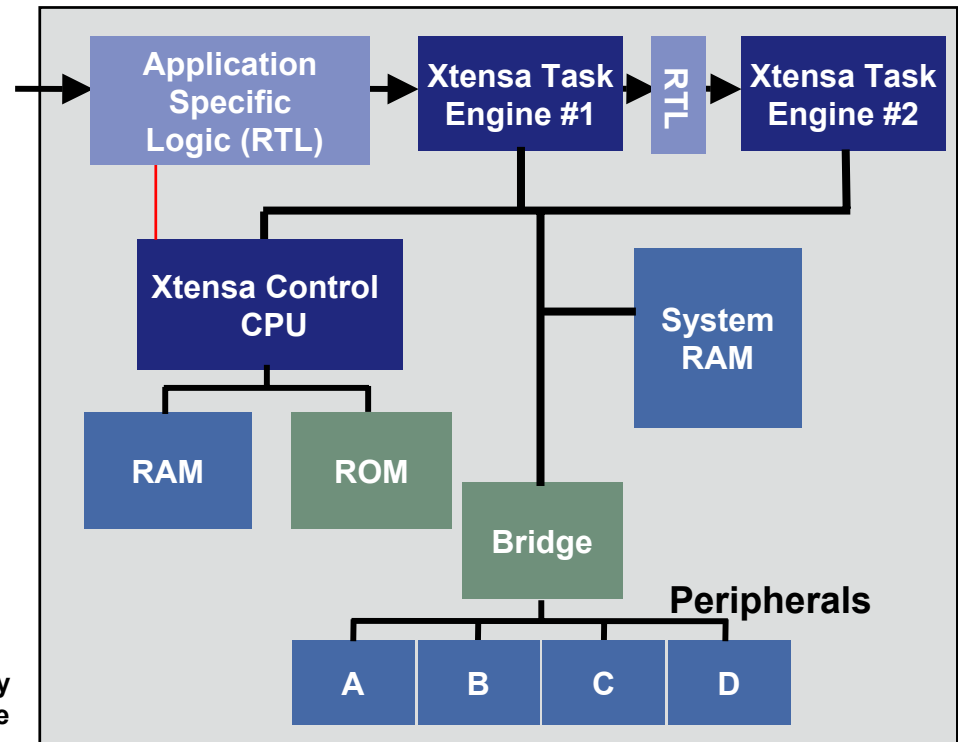
... is implemented in firmware plus  
designer-defined execution units,  
instructions, and registers added to a  
pre-verified processor core

“Firmware”



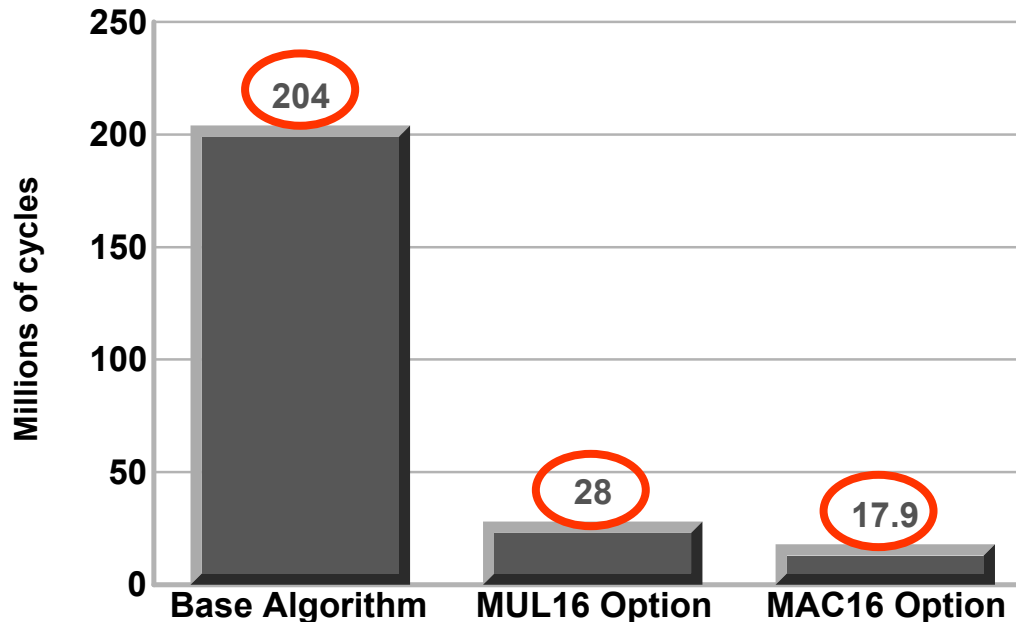
# Xtensa technology reshapes SOC design

- **Use optimized processors as building blocks**
  - Task-specific, optimized processors
  - Optimal performance / power / cost; similar to RTL performance
  - Hardware & Software Design productivity
  
- **Lower risk**
  - Verify Algorithm in Software within hours
  - Change in Software
  
- **Higher design productivity**



Bus topology shown for illustration purposes only  
 Numerous topologies and configurations possible

# Proof of concept: GSM audio codec



- **Code profiling shows that ~80% of the time is spent in multiply**
  - Select the MUL16 option
- **More profiling shows further optimization possible using the MAC16 option**

# Adding a MAC16 to an Xtensa processor

Config << Prev Select Next >> Help

**xtensa Processor Generator** Status

**Optional Instructions**

MAC16 >>  CLAMPS (req. by MAC16)  
 MUL32 <<  Add MUL32 options: MULUH and MULSH  
 MUL16  NSA/NSAU  
 Floating Point (coprocessor ID 0)  MIN/MAX MINU/MAXU  
 Enable boolean registers  SEXT

None Number of Coprocessors (NCP)

**Exceptions & Interrupts**

None Interrupt Count None High Priority Interrupt levels Enable Debugging  
 None Timers

**Endianness & Call Windows**

Byte Ordering  Little Endian  Big Endian

---

Speed (MHz) 100 209 209

Core Area (Gates) 29120 36430 282200

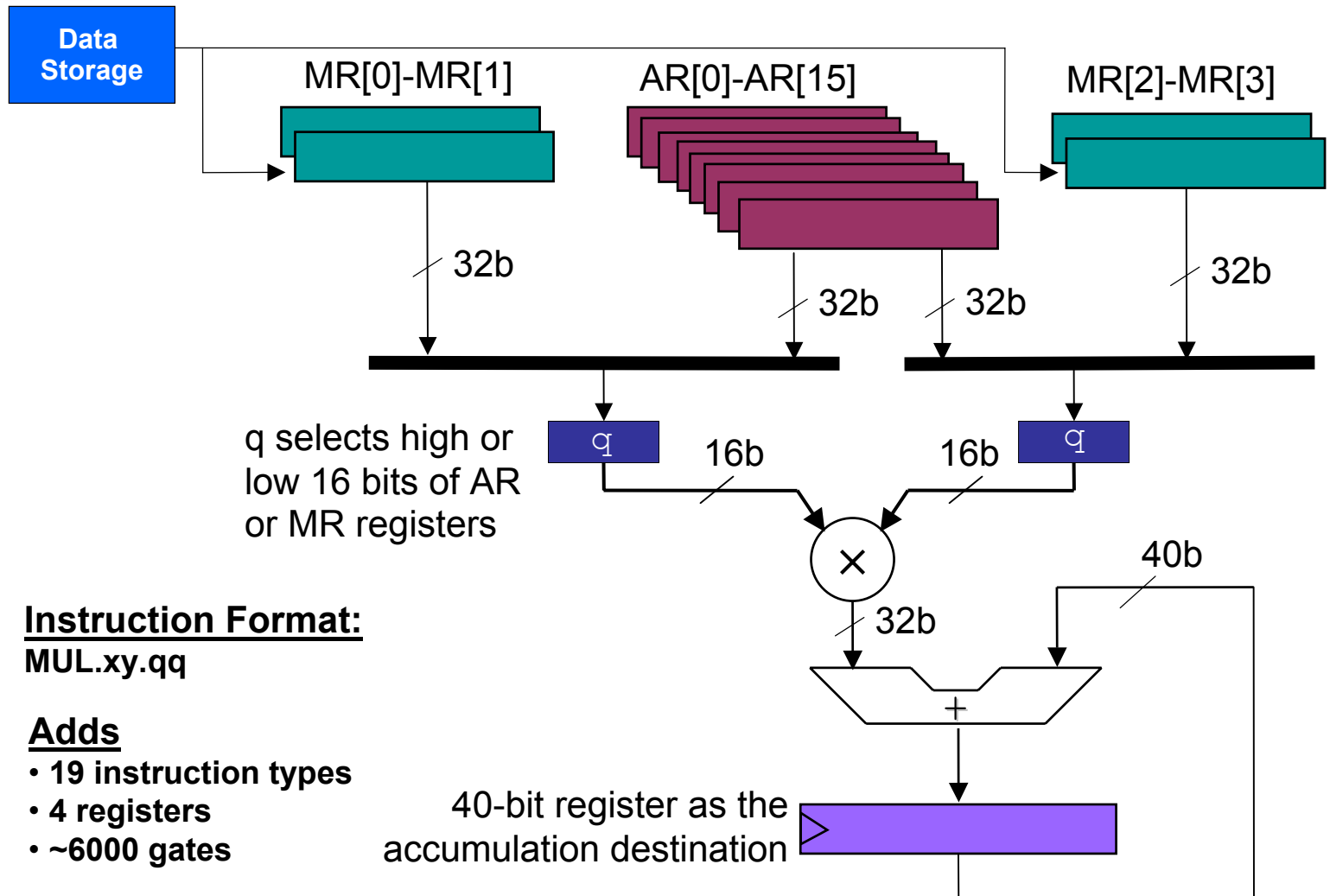
Core Power (mW) 31 75 446

Total Area (mm2) 0.83 1.13 47.92

Estimation Chart: Preliminary data

Try it yourself at  
[www.tensilica.com](http://www.tensilica.com)

# Xtensa MAC16 DSP unit



**Instruction Format:**  
**MUL.xy.qq**

**Adds**

- 19 instruction types
- 4 registers
- ~6000 gates

40-bit register as the accumulation destination

# Proof of concept: GSM Viterbi decode

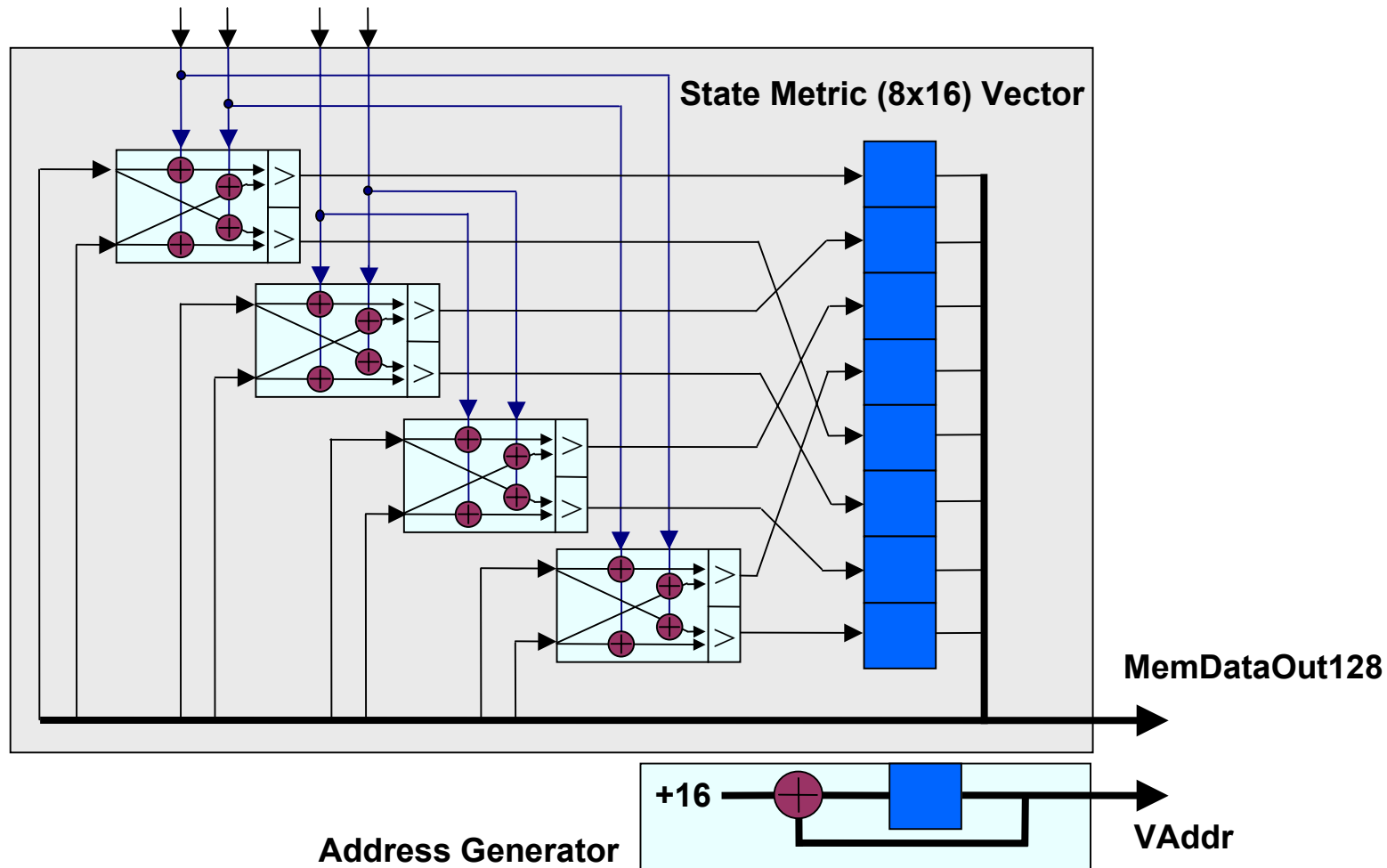
## GSM Viterbi decode

- 8 “butterfly” operations required per bit for decode
- Each butterfly requires 8 logical operations:  
4 Additions, 2 Comparisons & 2 Selects

| Processor          | Clock cycles / butterfly operation | Note:  |
|--------------------|------------------------------------|--|
| Base Xtensa core   | 42                                 | “out of box”                                 |
| “Typical” RISC     | ~50 - 80                           |  |
| TI DSP TMS320C64xx | 1.75                               | 600 MHz DSP,<br>hand optimized assembly code |

# Viterbi Butterfly datapath hardware

**Path Metrics**  
(256 bits of state, loaded in 2 instructions)





# Adding two TIE\* instructions accelerates Viterbi by 250x

**VBIN:** Loads 8 GSM coded symbols (16 bytes) into a TIE state register in 1 instruction

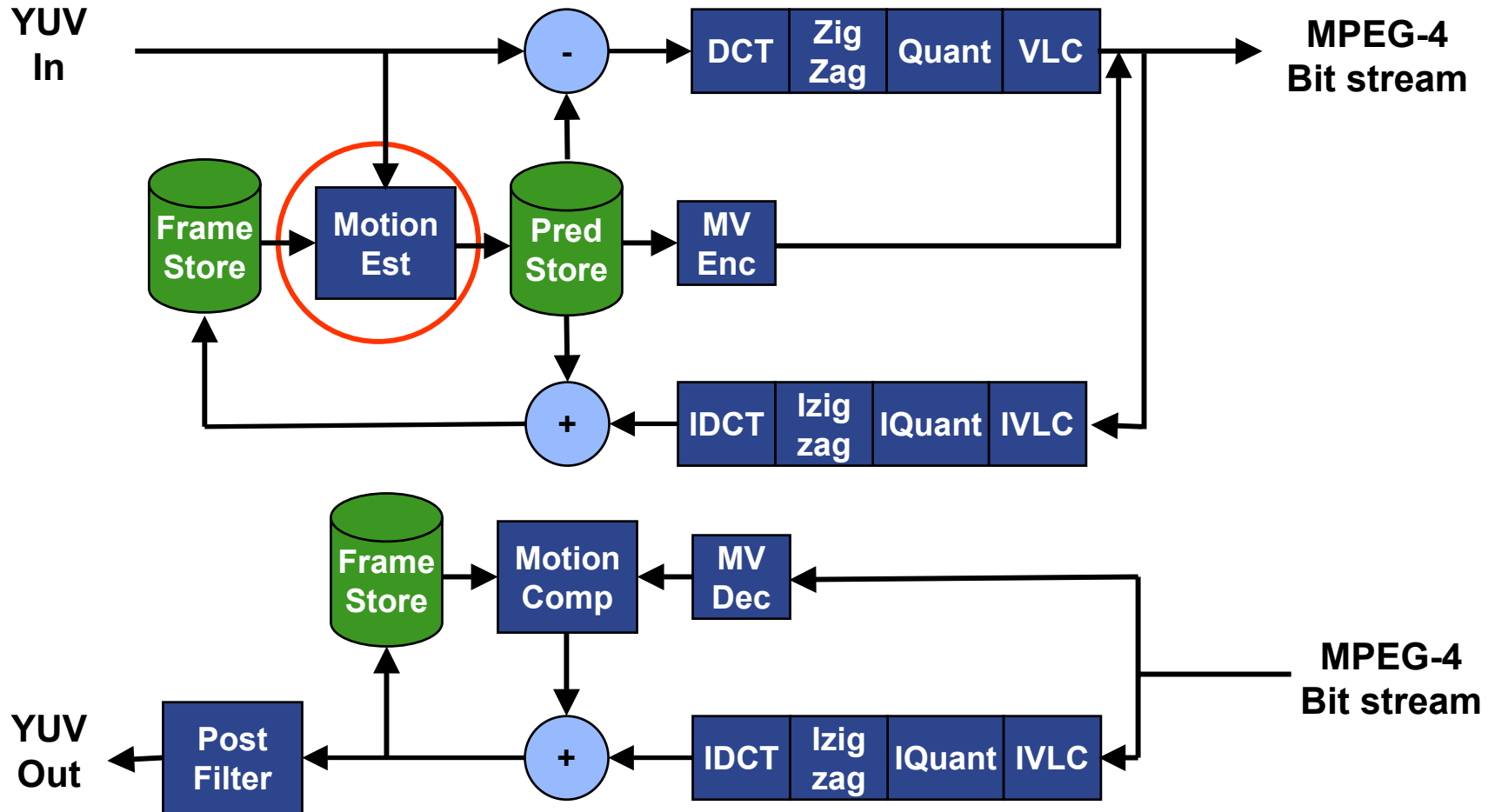
**VBOUT:** Calculates all path metrics of a trellis column for a single pair of GSM coded data and stores the result

| Processor          | Clock cycles / butterfly operation | Note:   |
|--------------------|------------------------------------|---|
| Base Xtensa core   | 42                                 | “out of box”                                    |
| “Typical” RISC     | ~50 - 80                           |   |
| TI DSP TMS320C64xx | 1.75                               | 600 MHz DSP,<br>hand optimized assembly code    |
| Xtensa plus TIE    | 0.16                               | Adds 11K gates;<br>compiler generated code only |

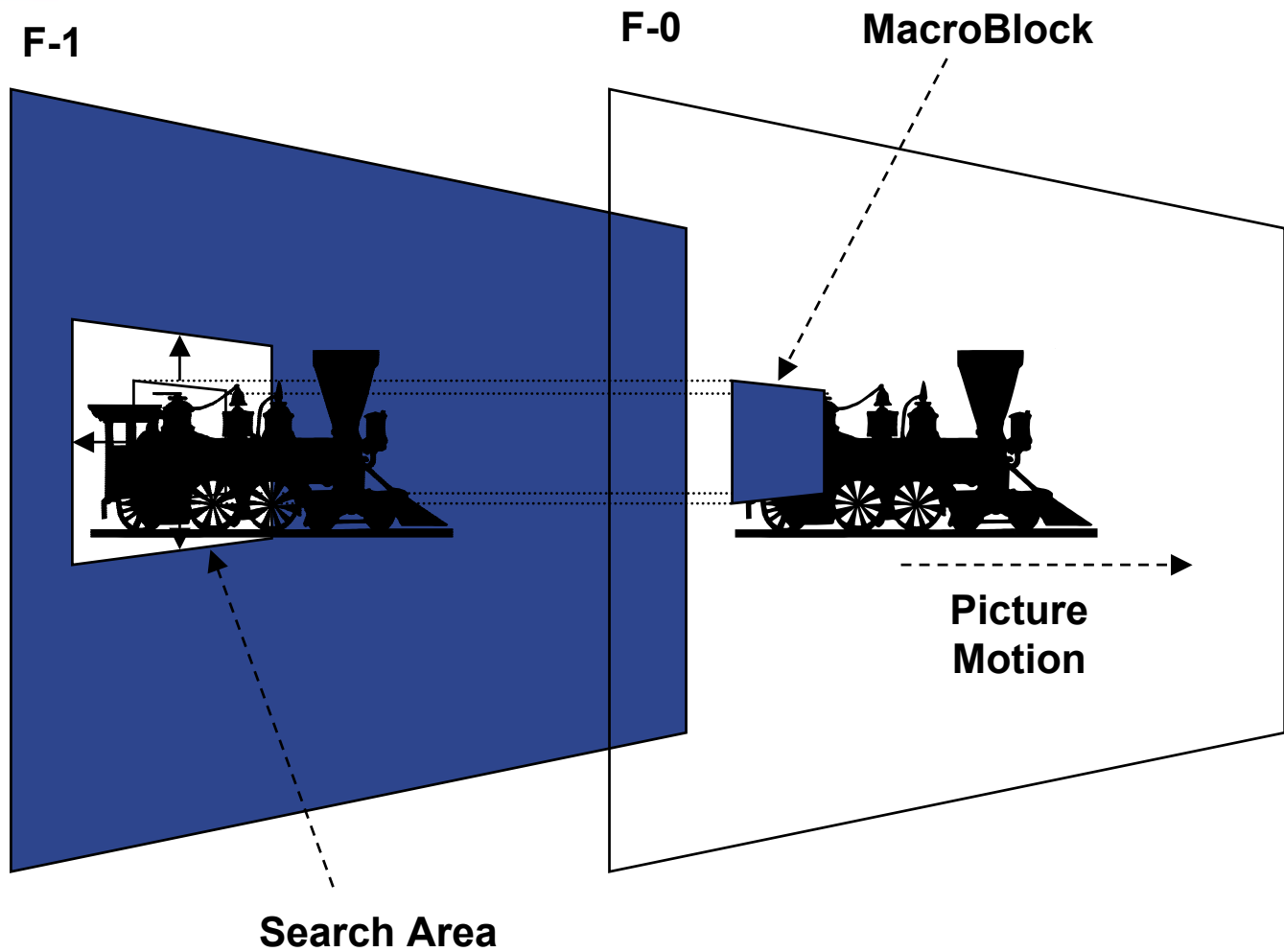
250X speed up using TIE  
11X faster than “state-of-art” DSP

\* TIE is Tensilica’s Instruction Extension language for its Xtensa processor

# Proof of concept: MPEG-4 video codec



# Motion estimation



## Exhaustive motion search – QCIF

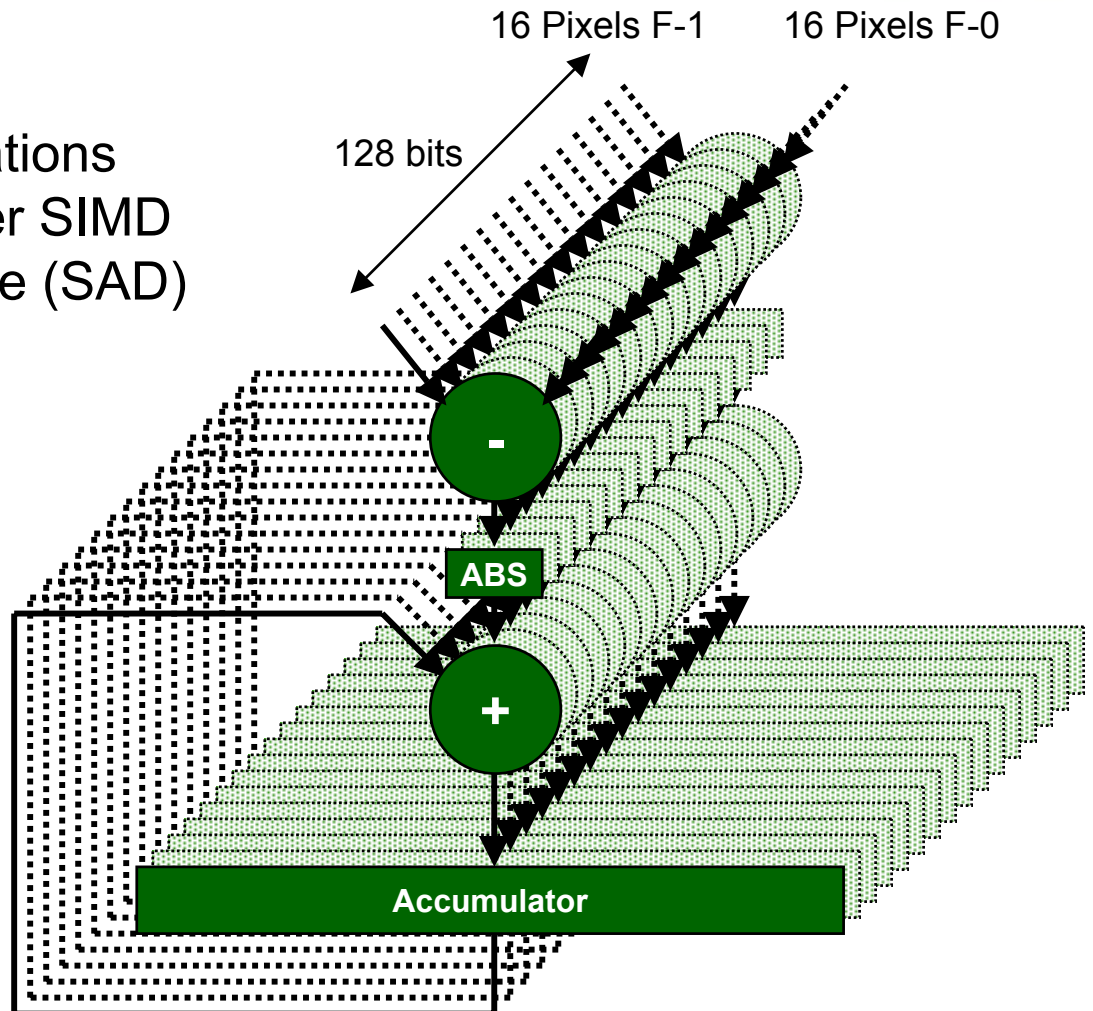
- Sum of absolute difference (SAD) – 3 operations  
subtract, absolute, accumulate
- Operands – 2 loads + 2 address increments
- 16x16 pixels per Macroblock
- 16x16 search area
- 99 Macroblocks per QCIF picture
- 15 frames per second
- Total + 681,246,720 operations per second



How do you do 682 MHz of work in just 35 MHz?

16-pixel SIMD SAD operations  
 3 arithmetic operations per SIMD  
 Sum of absolute difference (SAD)

3 operations on 16  
 pixels every clock  
 equals 48 RISC  
 operations/clock



## MPEG-4 performance: Xtensa + MPEG-4 video instructions

|                               |                     |                                      |
|-------------------------------|---------------------|--------------------------------------|
| <b>Post Filtering</b>         | <b>12.4 Mcycles</b> |                                      |
| <b>Decode</b>                 | <b>17.3 Mcycles</b> |                                      |
| <b>Complete MPEG-4 Decode</b> | <b>29.7 Mcycles</b> | <b>112K gates extensions in .18u</b> |
| <b>Complete MPEG-4 Decode</b> | <b>32.7 Mcycles</b> | <b>92K gates extensions in .18u</b>  |

### Flexibility

- Software control of power great algorithmic flexibility.

### New Instructions

- Color Space conversion
- Luma/Chroma filtering
- Sum of absolute difference



# Why not just add all these instructions to the base processor?

- **Because**
  - Every embedded application doesn't need a multiplier – but some do
  - Few embedded applications need a Viterbi butterfly – but some do
  - Only MPEG-4 applications need a 16-pixel SAD unit for motion estimation
- **The list of special-case embedded hardware needed for SOC design is endless**
  - But gate budgets are finite, now and for the foreseeable future
  - Verification time is limited and finite, now and for the foreseeable future

# Summary: A Plan for SOC Success

- 1. Accept the new SOC reality – hardware must be flexible to improve ROI**
  - We will not design future SOC's as we did ASIC's
  - Neither will our competitors
- 2. Plan to build the required features while keeping the design productivity gap in mind**
  - All hand-designed RTL must be verified
  - Verification is now 80% of RTL design
- 3. Companies that bridge the gaps between desired features, flexibility, and design productivity will create the most successful SOC's**