
*The Configurable Processor Company*

# 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**

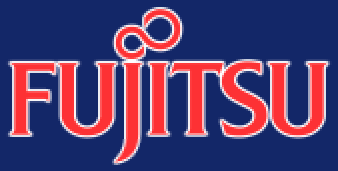
# Tensilica:
## Configurable Processor Leader

- **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

# Customers Lead in Growth Markets







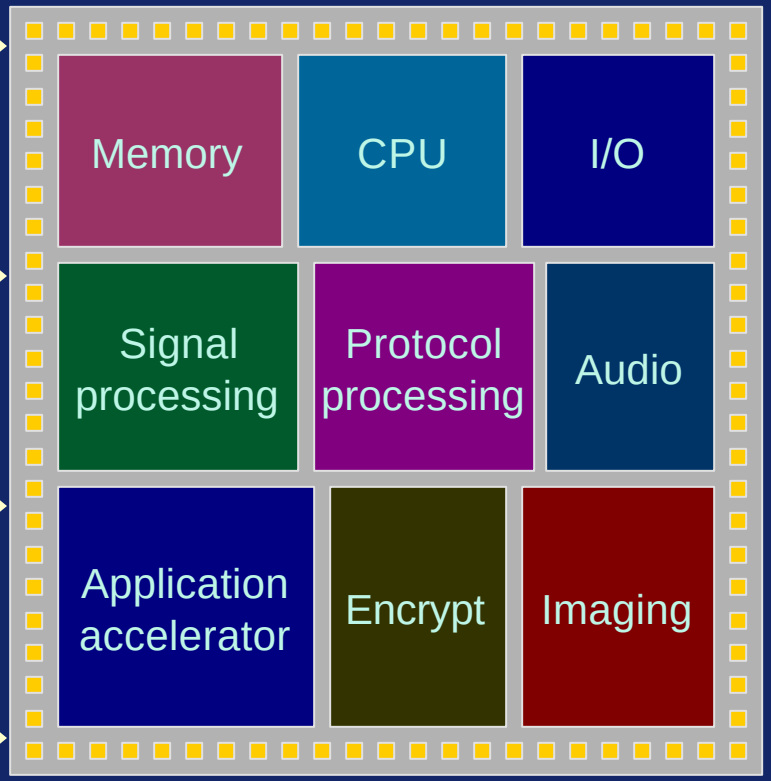





# Why System On Chip?

## Needs

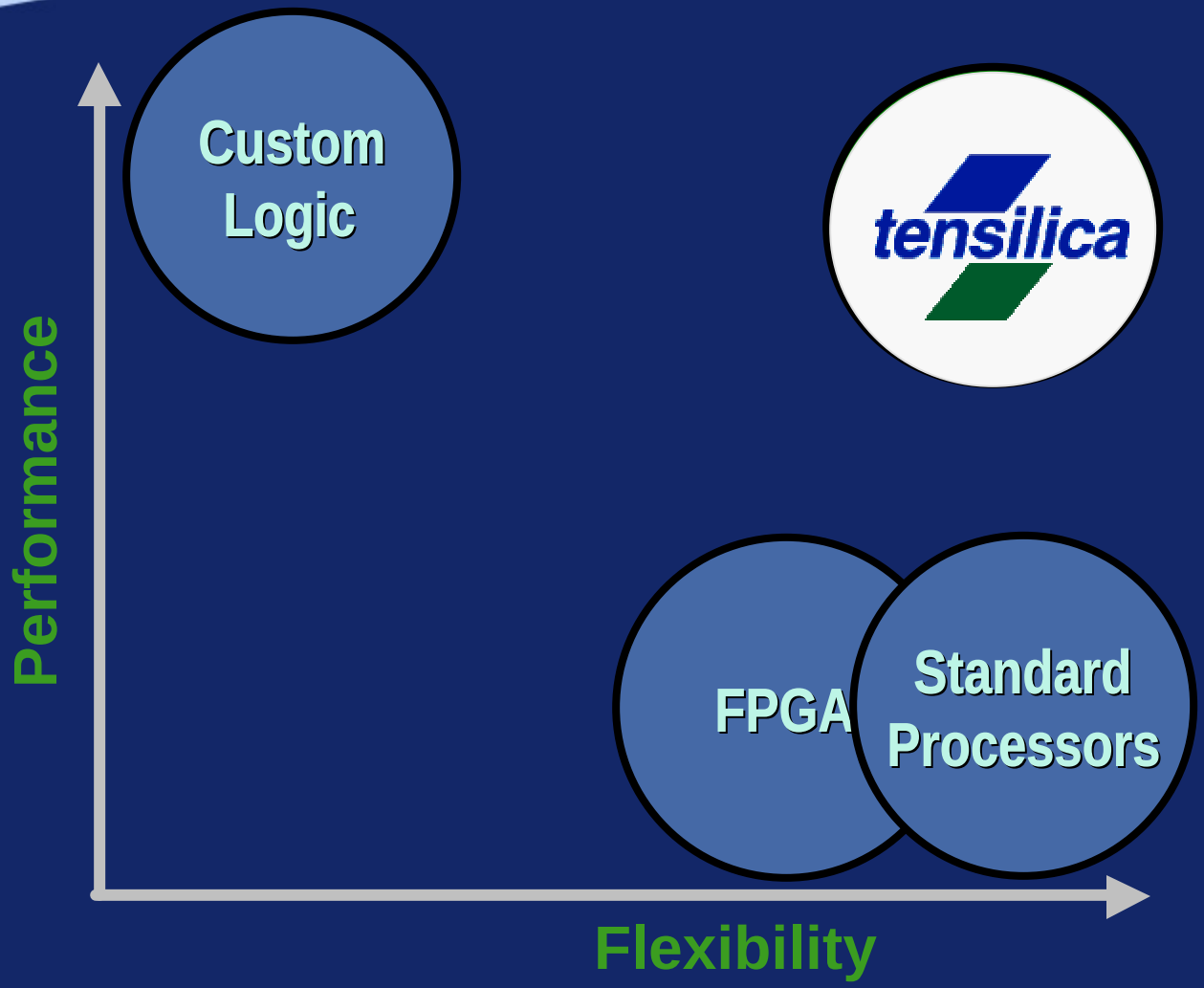
- Lower Cost/Size/Power
- Higher Performance
- Time to Market
- More Functionality




## Triggers

- 0.18µm - 0.13µm Silicon (speed, density)
- Foundry Proliferation (affordable silicon)
- Advanced EDA Tools (portable design)

# System On Chip Designer's Dilemma



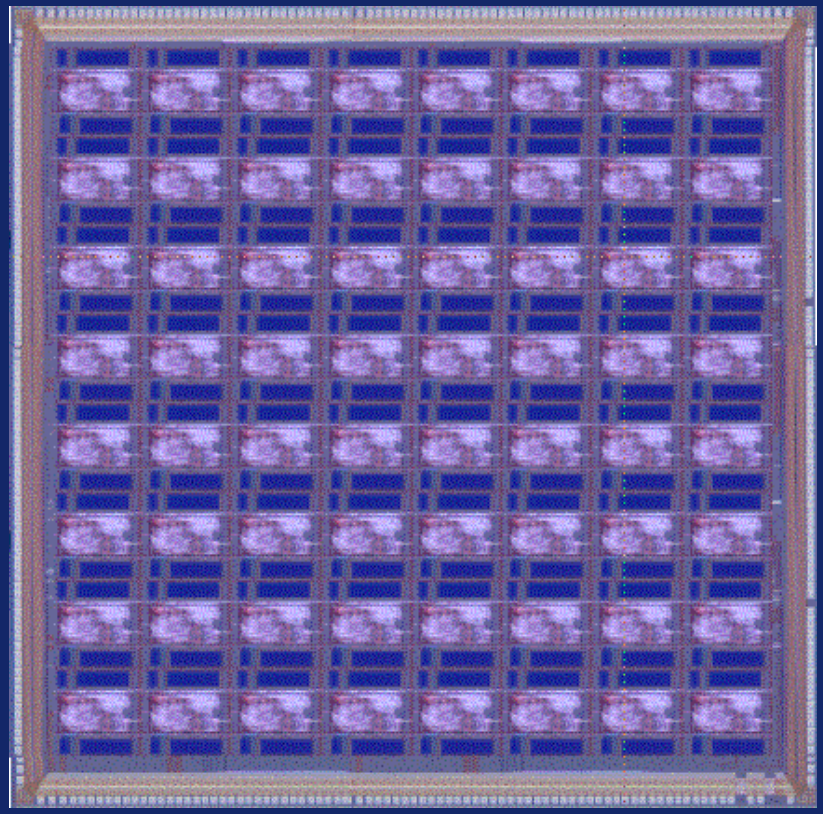
# Automated, Optimized Processor Design

**Cost**
- small size (<0.25mm$^2$ in 0.13µ)

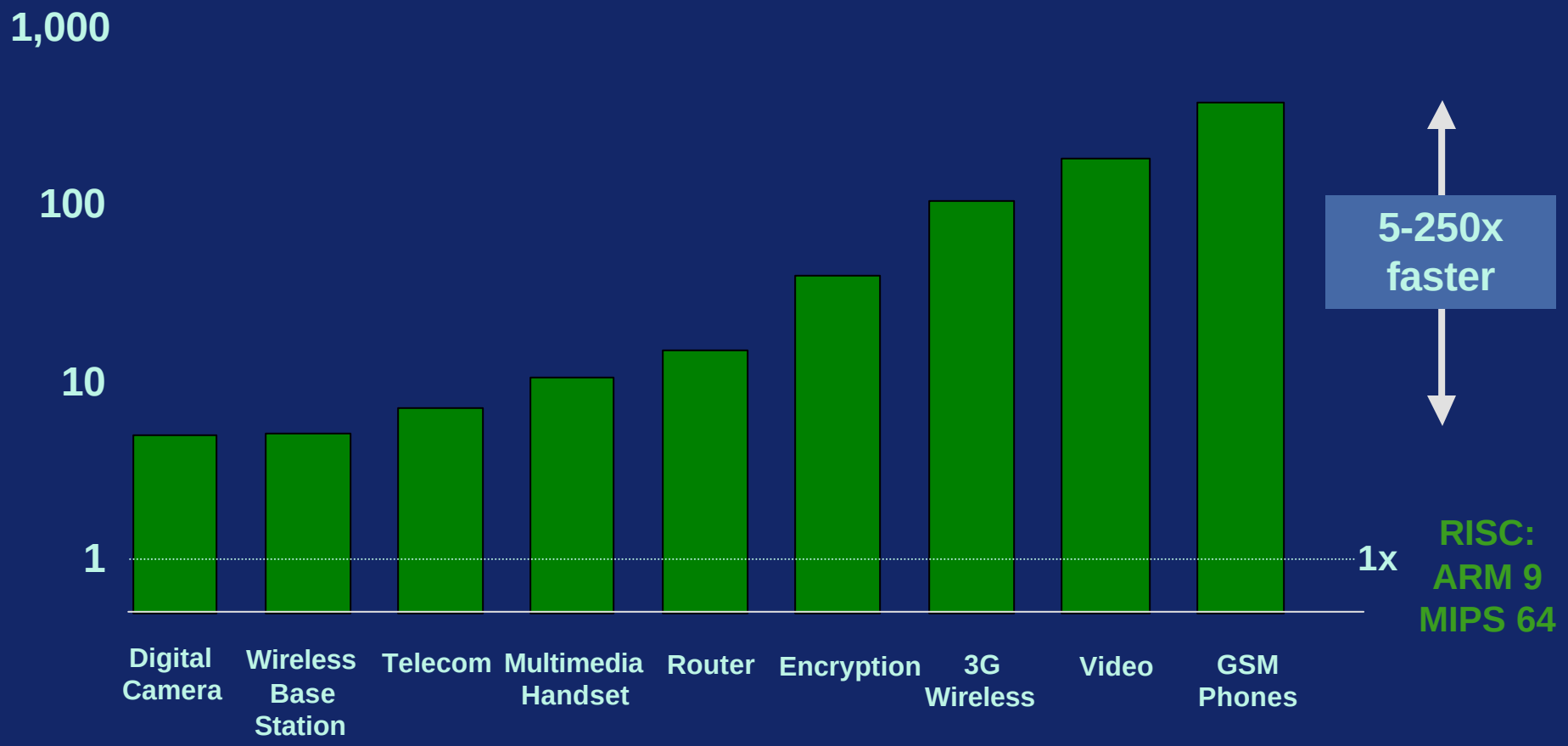
**Performance**
- application extensions

**Productivity**
- rapid hardware and software

**Building Block Use**
- processors everywhere

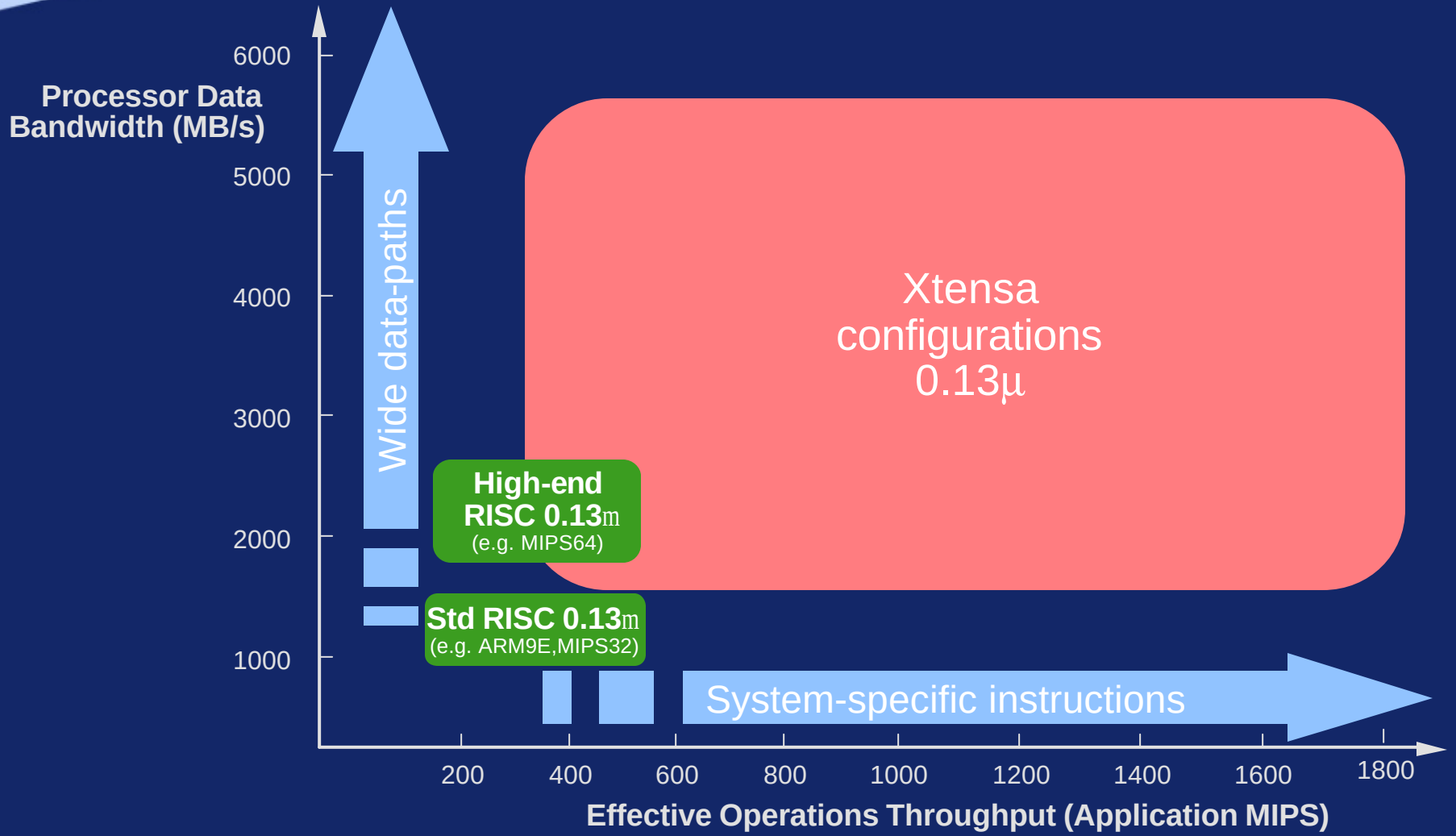


# Direct Impact:
## Performance




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

# Direct Impact:
## Bandwidth

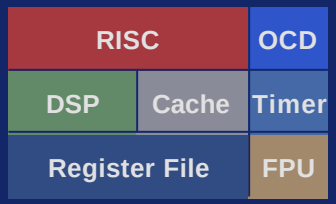


# Direct Impact:
## Complete Hardware and Software in One Hour

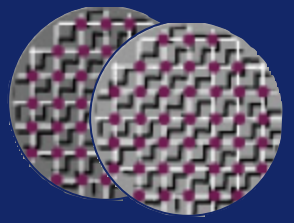

Electronic Specification

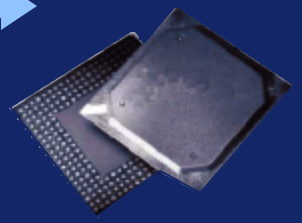
Tensilica Processor Generator

Hardware Design




Customized Software

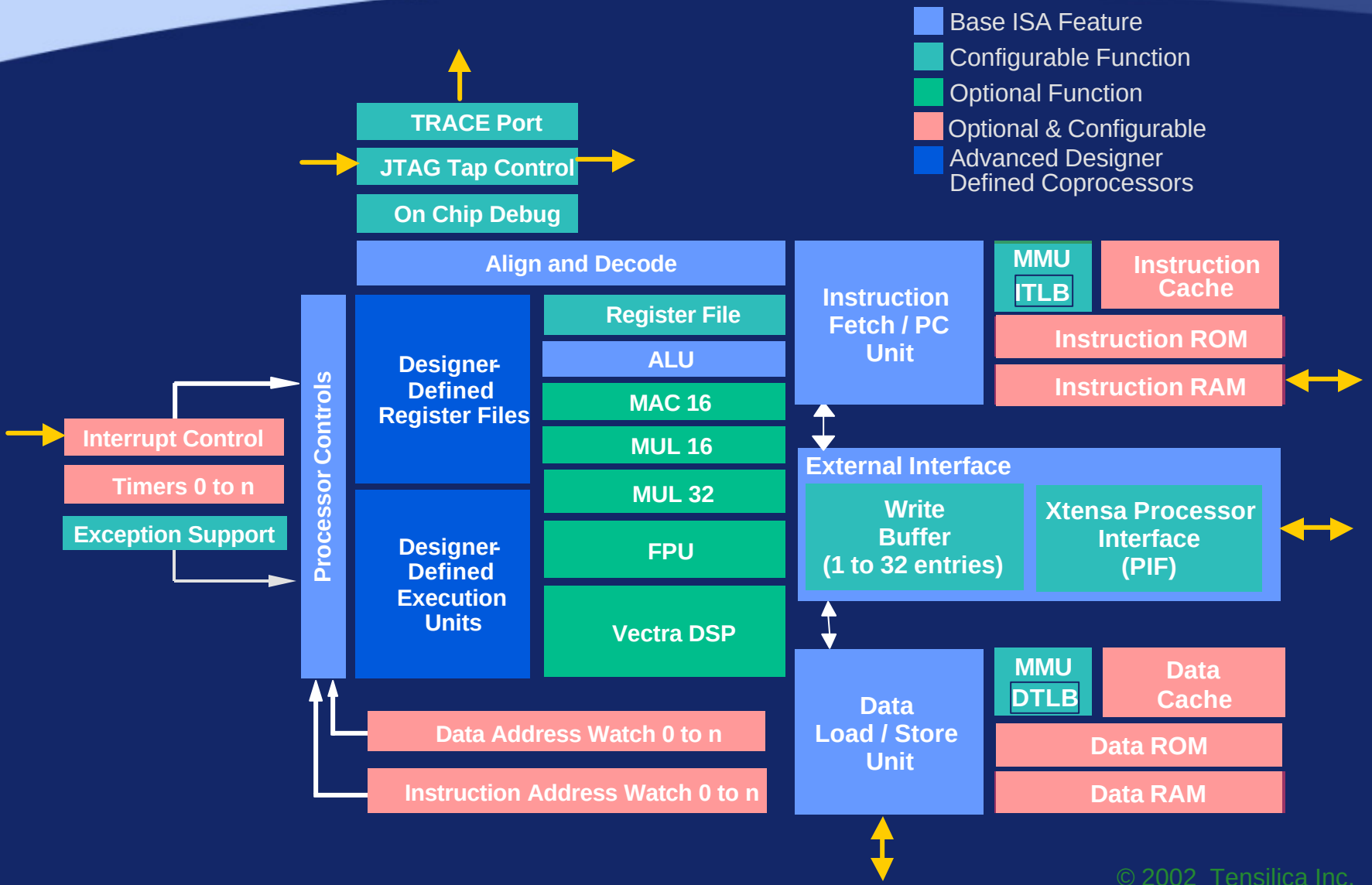




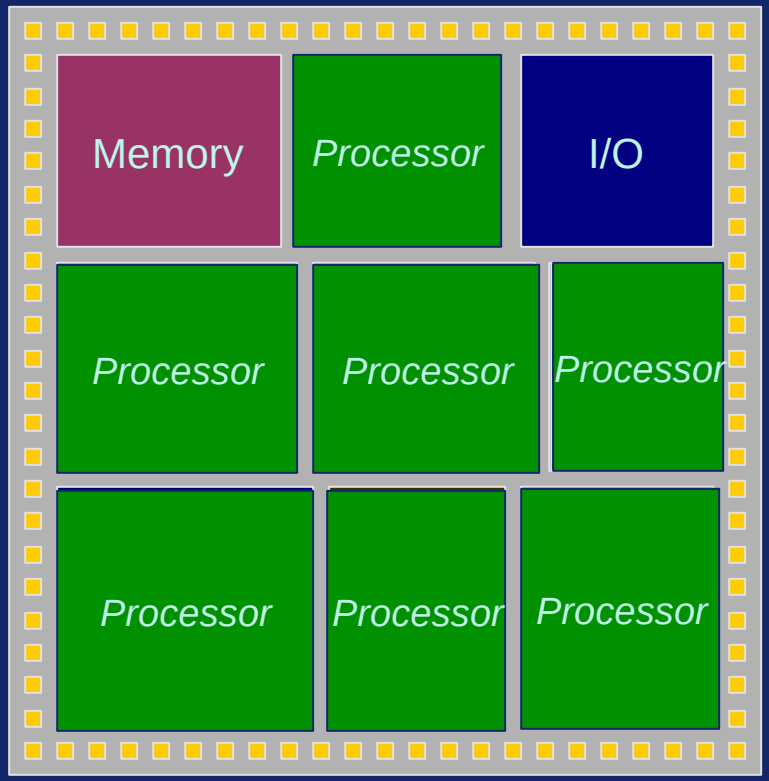
Build using any IC process

Design processor in one hour

# Direct Impact:
## Configurability and Extensibility



# Implication:
## Processor as Universal Building Block



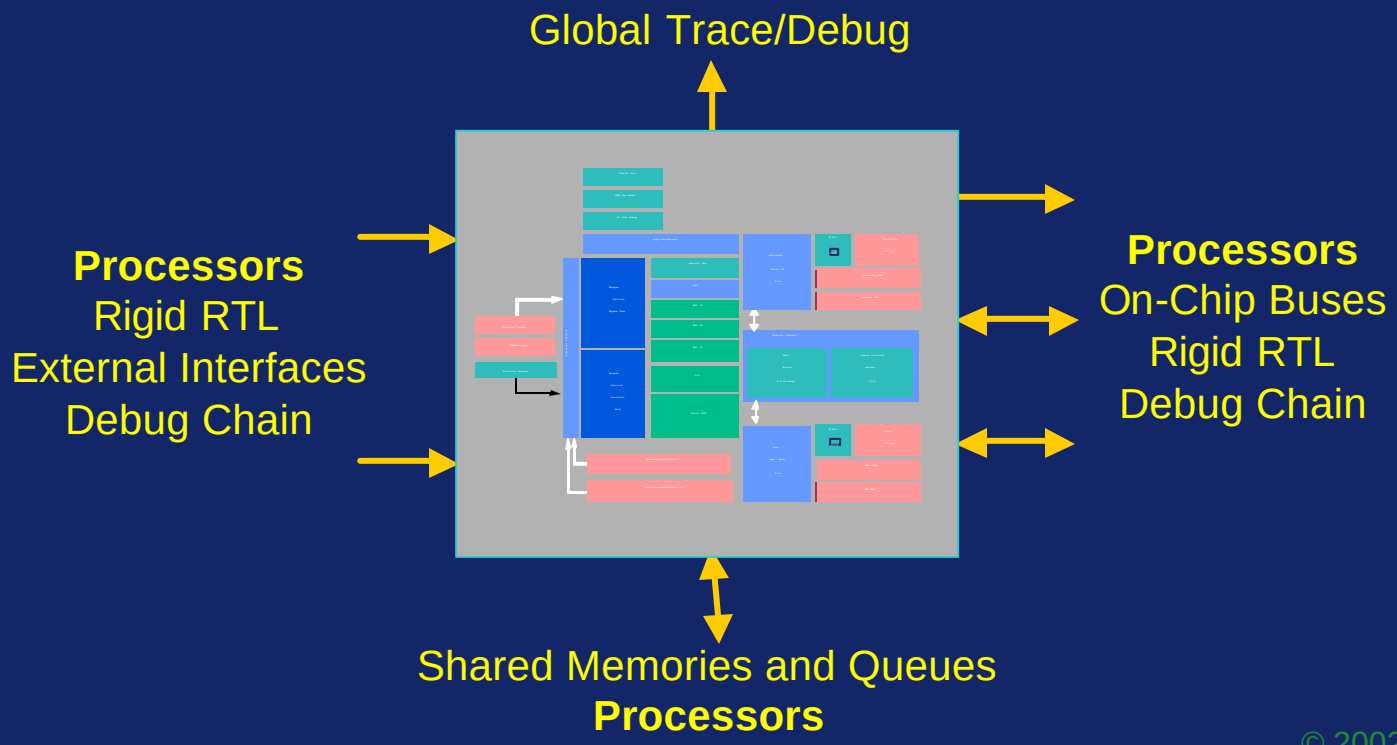
# Implication:
## Interface Flexibility Makes MP Natural

Direct interface among processors and to logic at >5GB/s

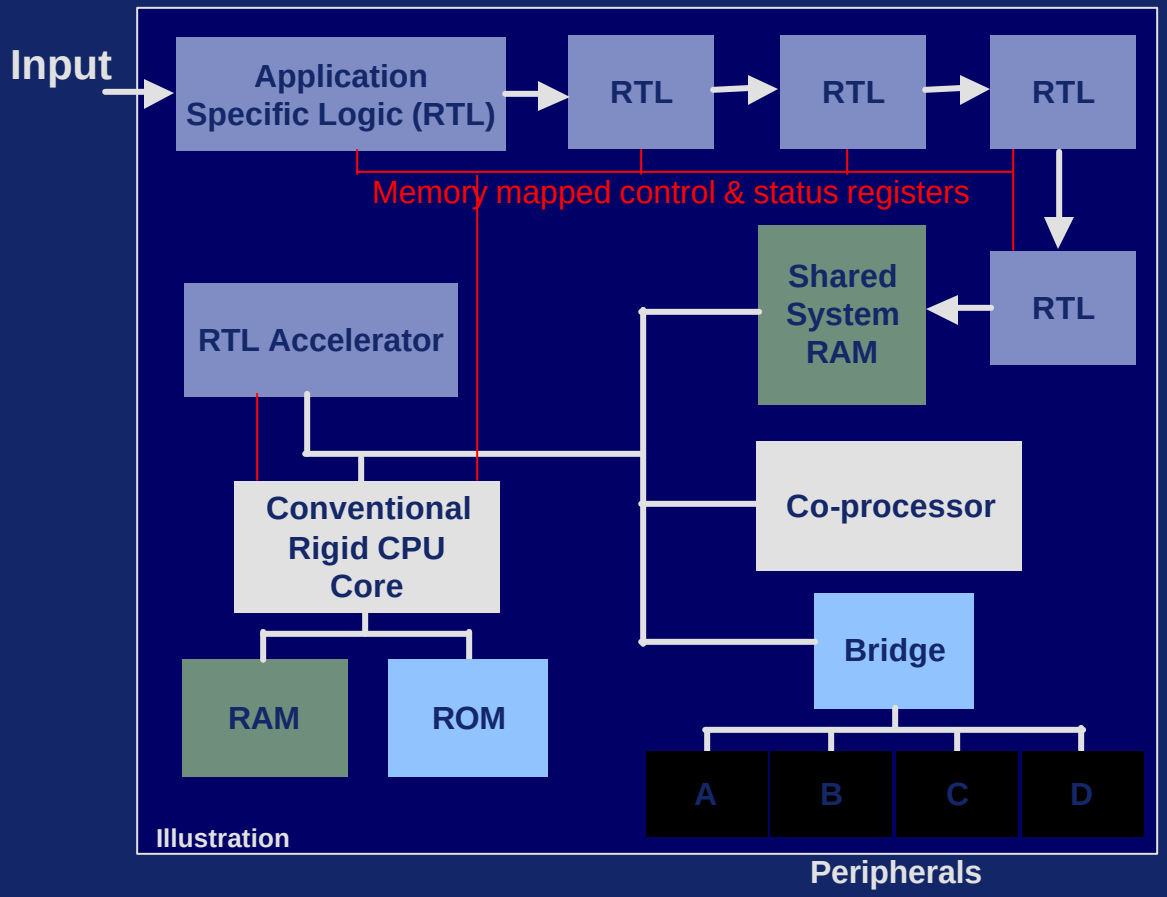
Flexible memory sharing

Comprehensive global debug and trace

Bridges to standard on-chip buses



# High Level View:
## Problems of Rigid Processors + RTL



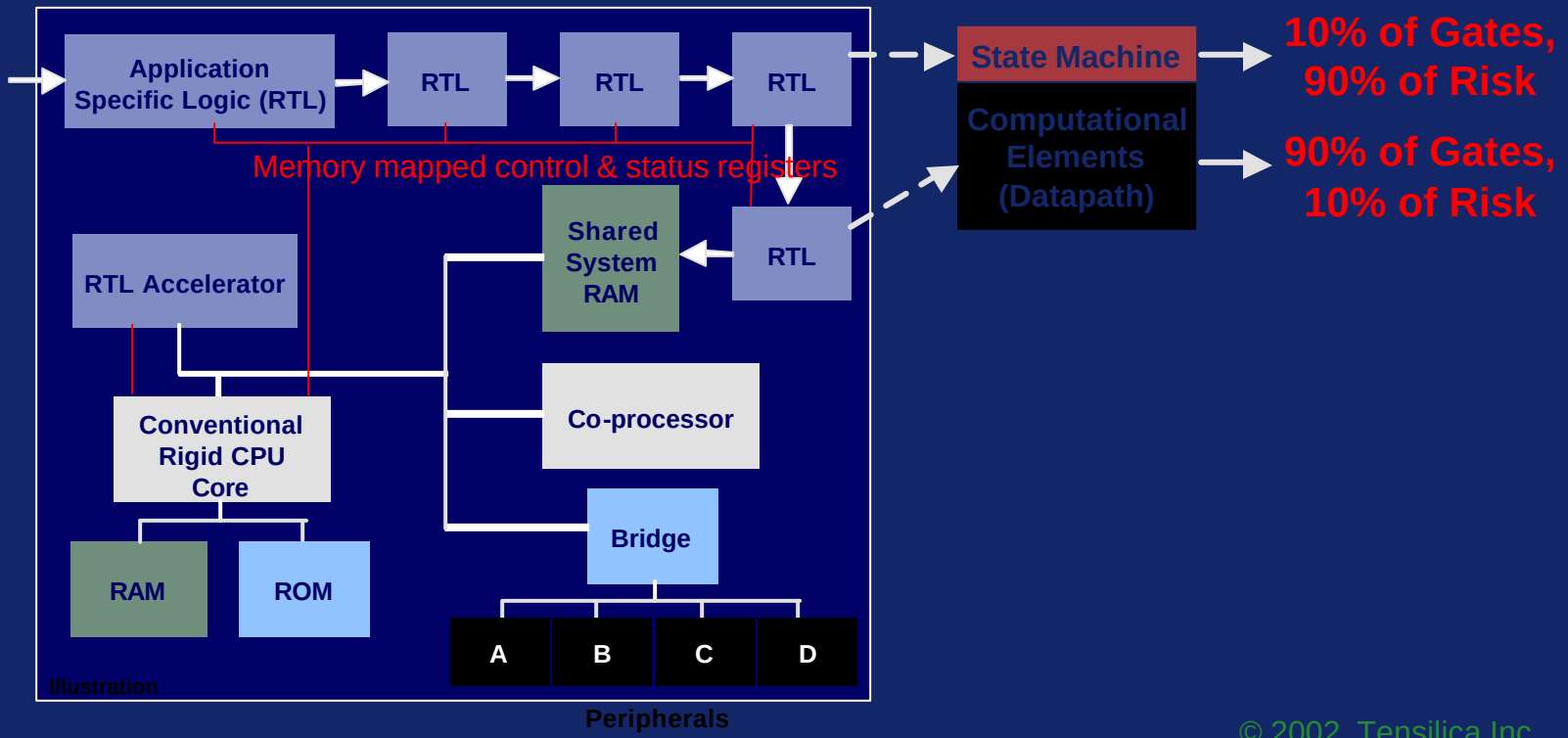

Rigid CPU cores cannot be modified to fit the application

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

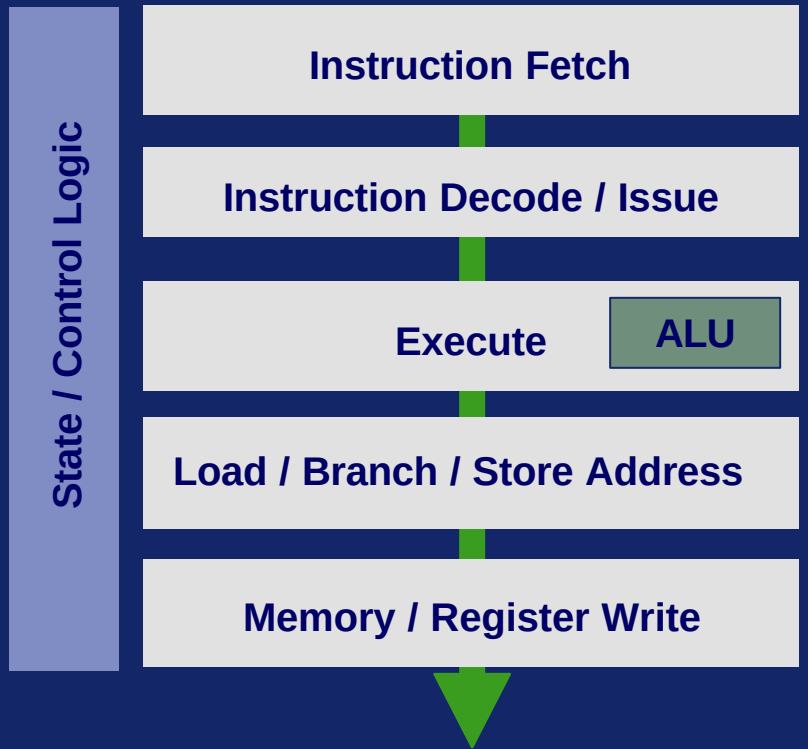
- Verification of complex state machines
- Costly silicon respins to make changes
- Obsolescence as markets or standards change



# Classic RISC Pipeline

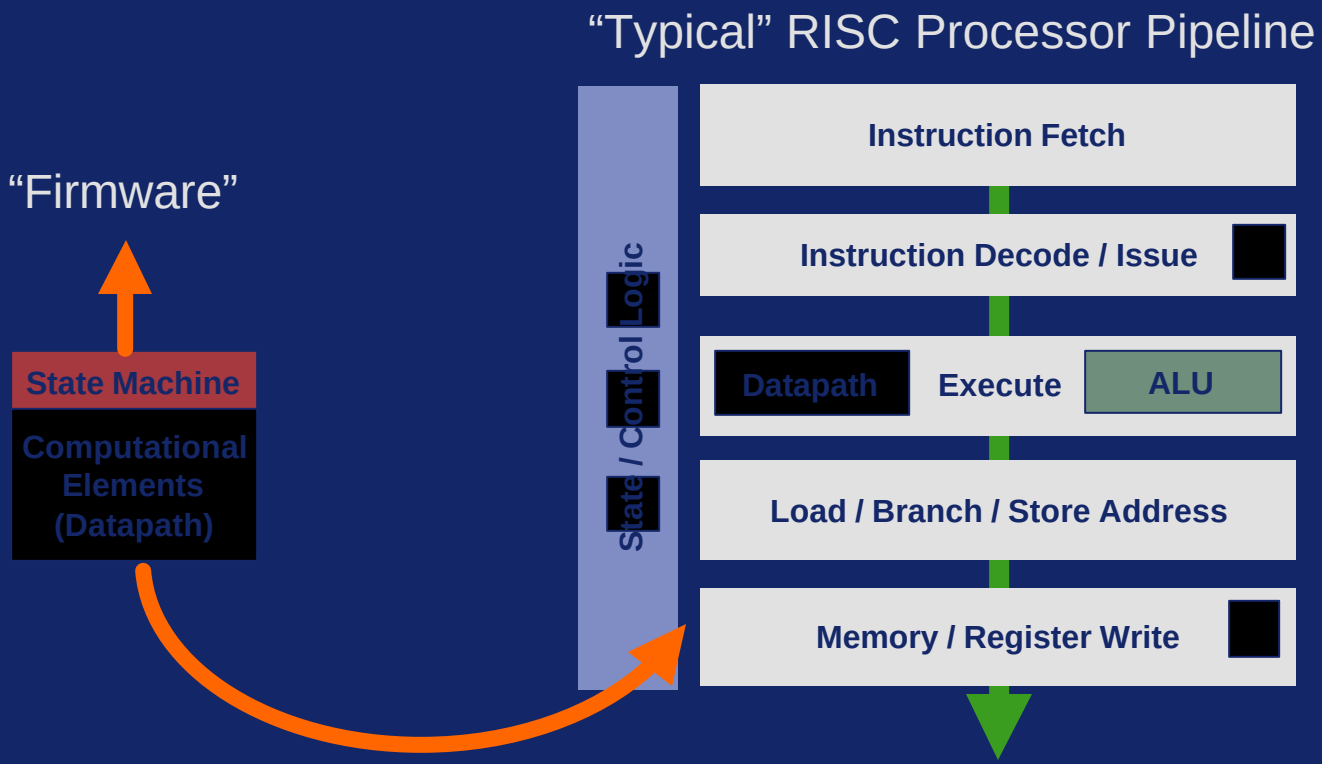
- Good integer performance for multi-purpose computing
- Real-life SOC applications do not crunch 32b integer data on a general purpose CPU

"Typical" RISC Processor Pipeline



# Custom Datapath in Hardware, Control Logic in Firmware

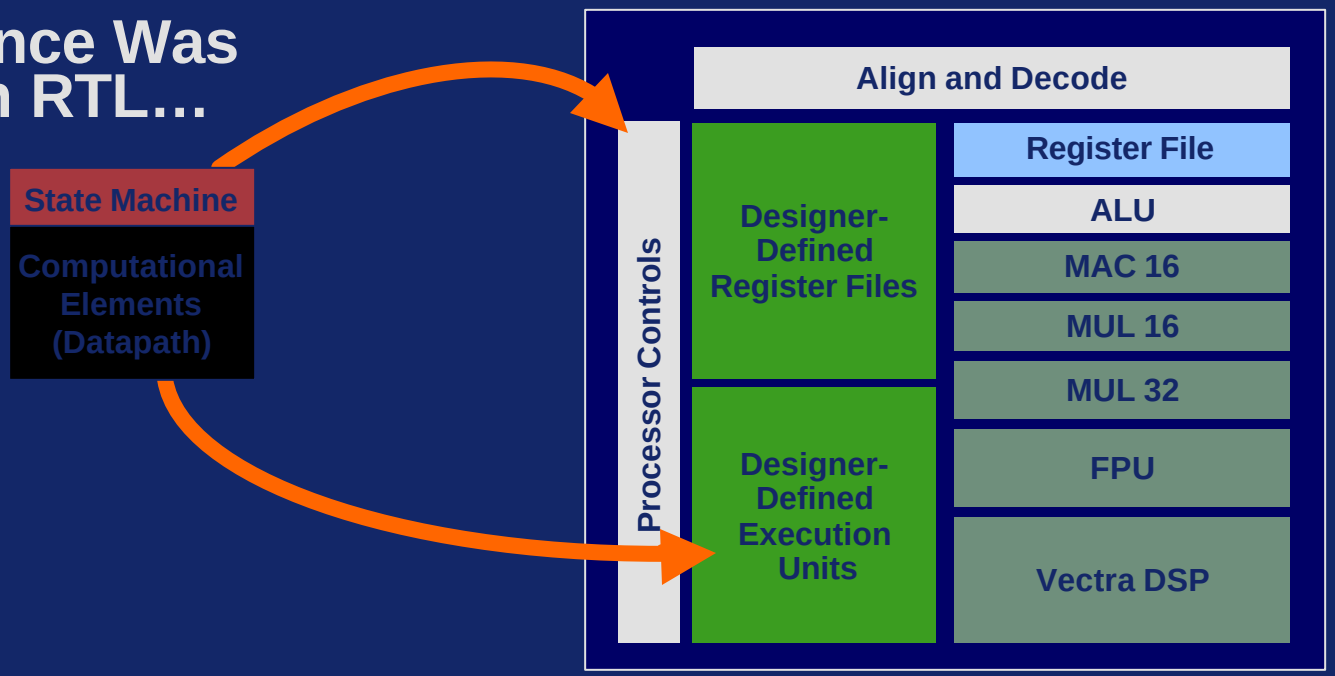
- What if you could transform the processor, with custom instructions, custom execution units, custom register files and state variables?



# Xtensa's Designer Defined Execution Units

**… is implemented as designer-defined execution units; designer-defined instructions, and processor registers & state**

**What Once Was Built In RTL…**




Xtensa Processor Detailed block diagram (partial view)

# Xtensa Reshapes SOC Design

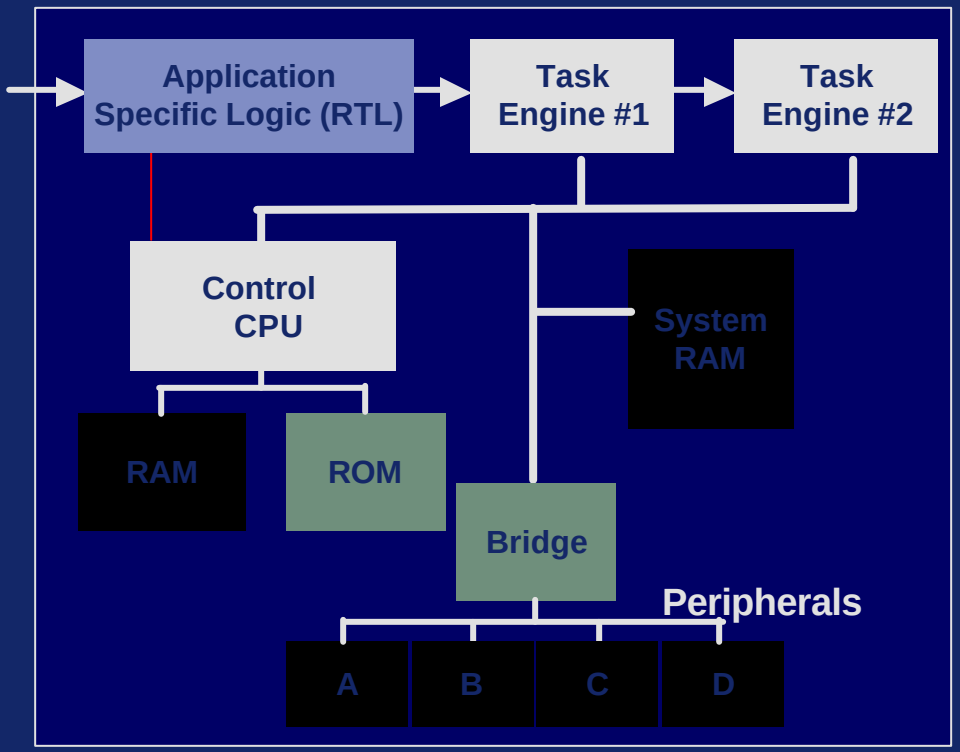
**Optimized Processors as Building Blocks**

Task-specific, optimized processors with RTL-like performance

**Lower Risk**

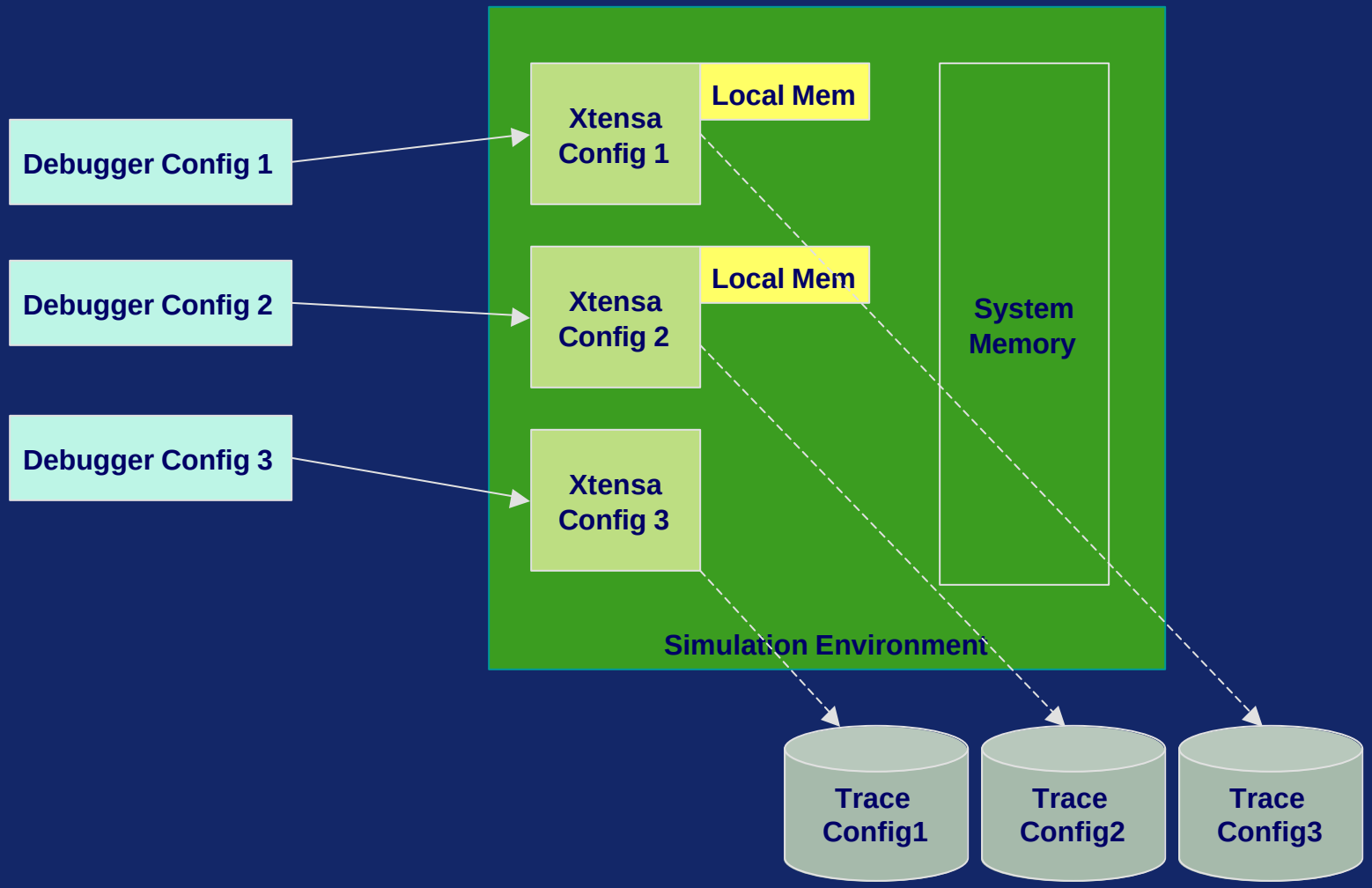
Verify Algorithm in Software within hours

**Higher productivity**

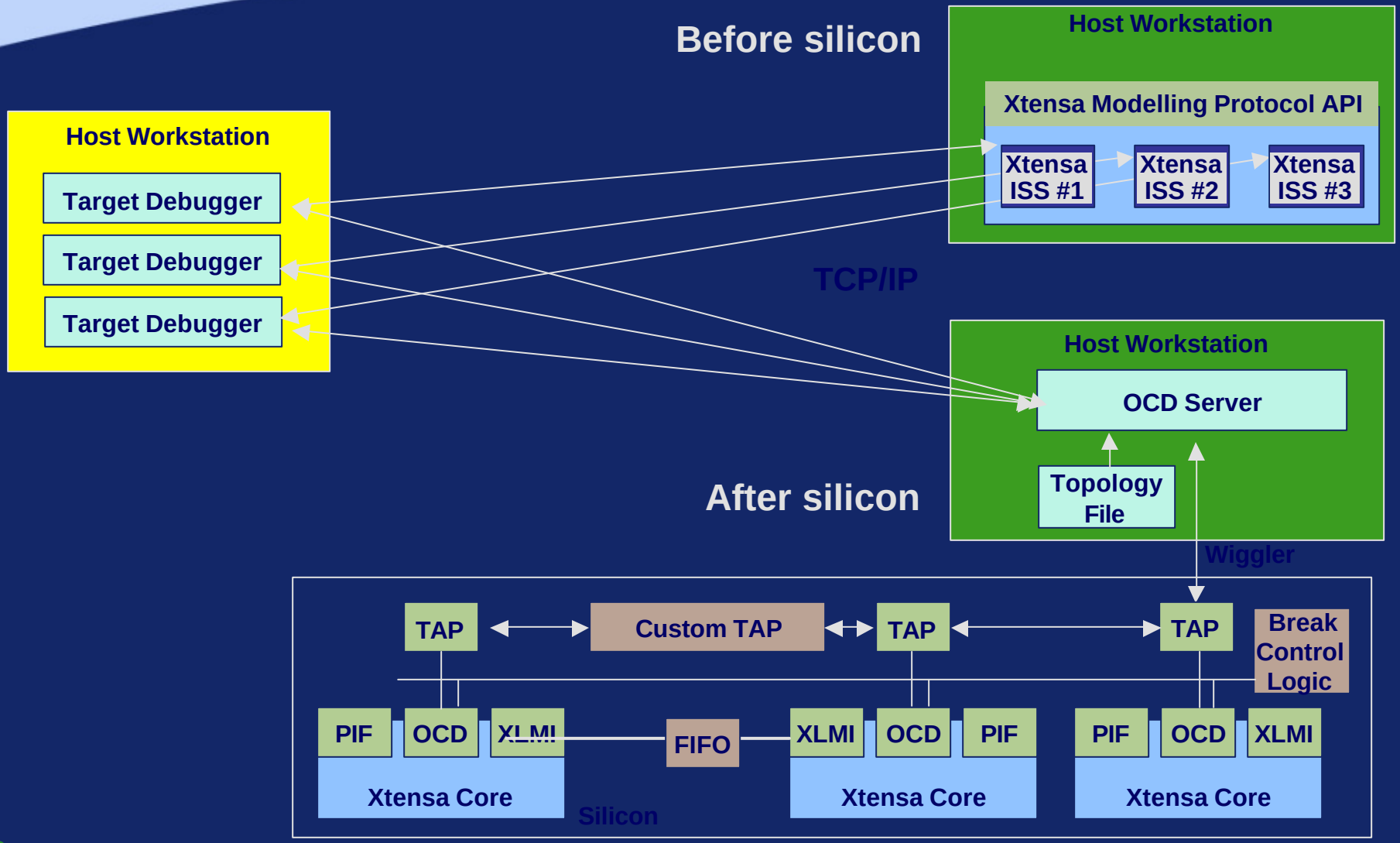



Bus topology shown for illustration purposes only
Numerous topologies and configurations possible

# Detailed View:
## System Modeling



# Multiple Processor Debug:
## Network Modeling, SW Dev, Bring-up



# XTMP: Xtensa Modeling Protocol

**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 Components

## 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**

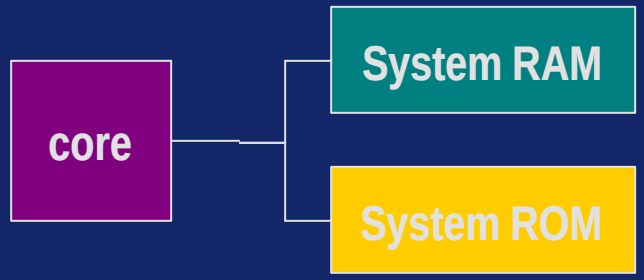
# Controlling the Simulation

## 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

```
main(){
:
    process packets;
:
}
```

```
XTMP_main(){
    declare objects;

    instantiate core;
    instantiate memories;
    instantiate connector;

    connect component;

    load program;
    start simulation;
}
```

# Uniprocessor Example Code

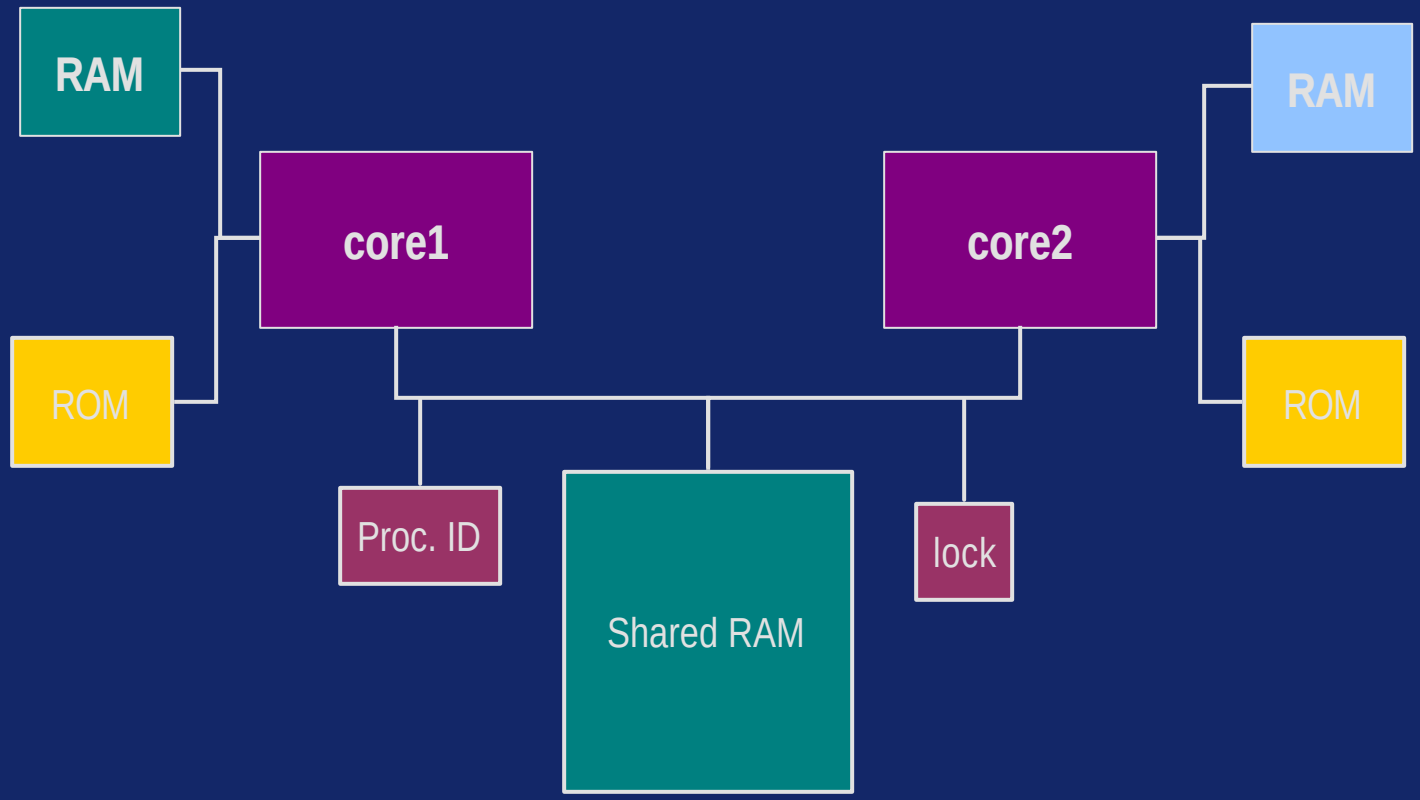
```
#include "iss/mp.h“

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);
}
```

Instantiate devices

Connect devices to form system

Start system simulation

# Multiprocessor Example



# Multiprocessor Example Code

```
#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 );
```

# Multiprocessor Example Code (2)

```
pid = XTMP_procIdNew( "Processor ID Device“,   XTMP_byteWidth(core1),
  XTMP_isBigEndianFromParams(p), dontcare);
lock = XTMP_lockNew( “lock",TMP_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);
```

# Multiprocessor Example Code (3)

```
XTMP_loadProgram( core1, "proc1.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

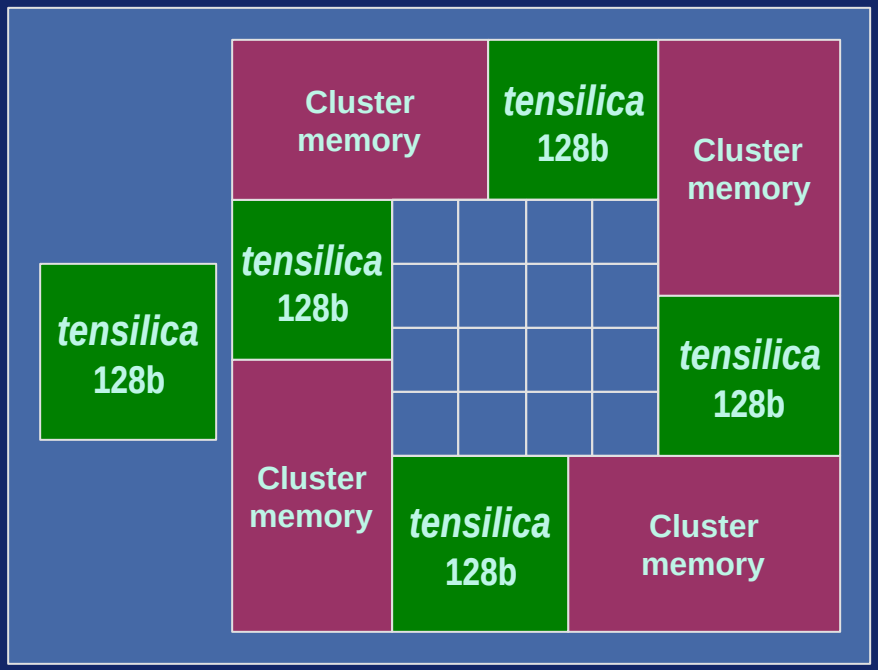
| | Standard CPU | tensilica | RTL Logic |
|---|---|---|---|
| Application-tuned data-paths | No: one-size-fits-all instruction set | Yes: specifiy in high-level TIE | Yes: specify in low-level RTL |
| Task control and sequencing | Programmable in C/C++ | Programmable in C/C++ | Hardwired RTL state machines only |
| Simulation and prototyping | Fast simulators or development boards | Fast simulators or development boards | RTL simulation: >100x slower |
| Multiple engines working together | Little interface, debug or modeling support | Simple direct interface, debug and simulation | Possible, but hard to design and model |

- 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***

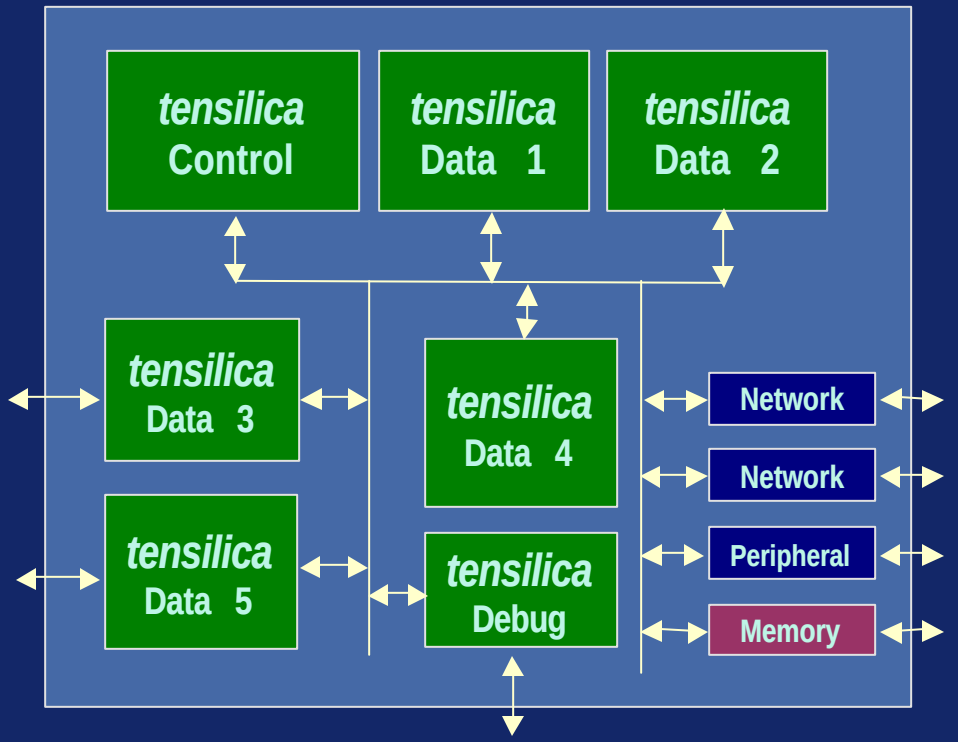



# Example:
# Time to Market, Long Product Life

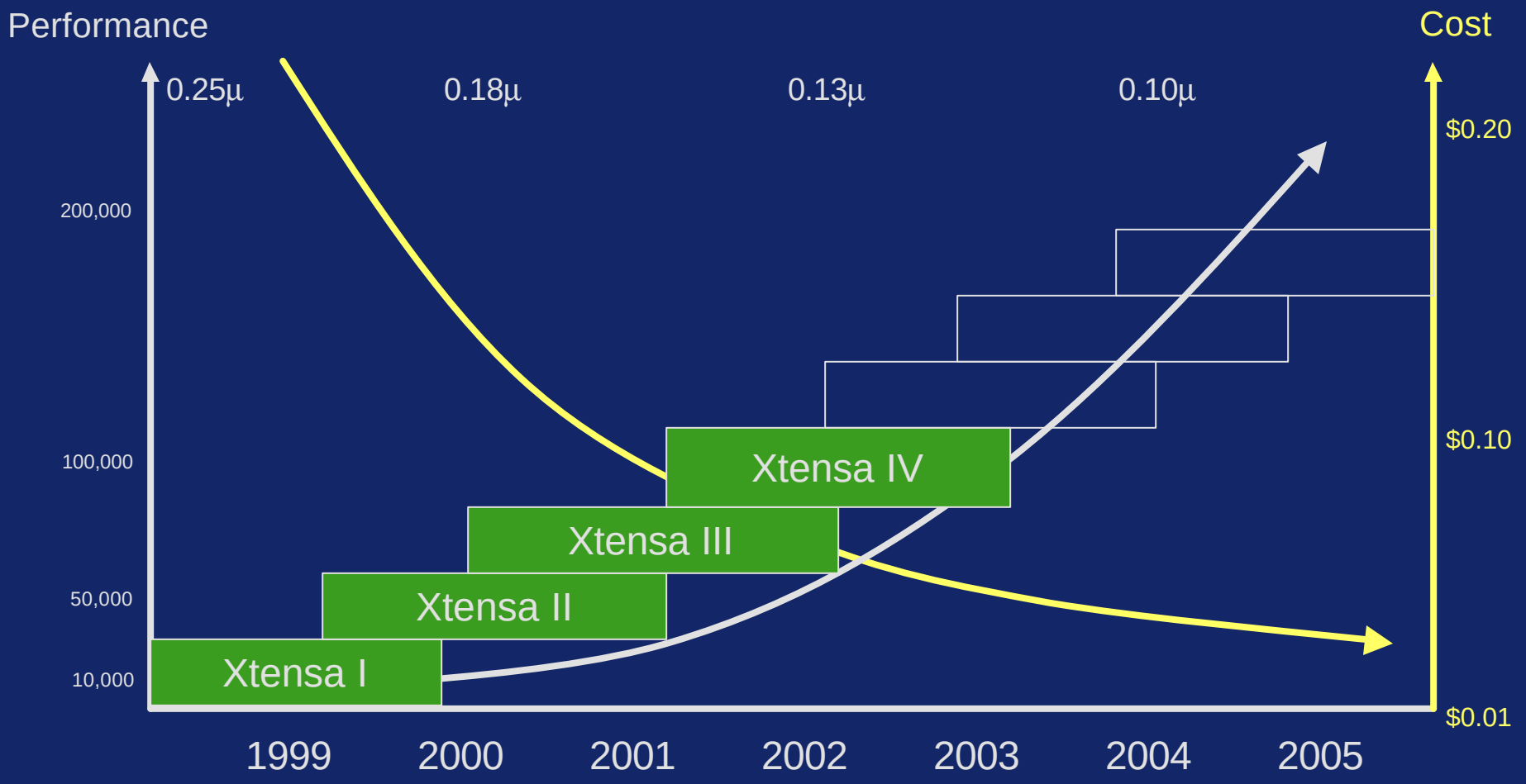
## Broadband Satellite Gateway

***All real-time processing in seven Tensilica elements***

HUGHES NETWORK SYSTEMS

# 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"*