1
Billion
cores

Flash Memory Summit 2011

Session 302: Nonvolatile Design Challenges and Methodologies

The Processor’s role in maximizing performance and reducing energy consumption

Neil Robinson

Tensilica At a Glance

1 Billionth core shipped in Apr 2011, 2B end 2012

1
Billion
cores

Market

- **Application-specific processor solutions, currently in over 20 market areas**
  - Storage, Audio, Baseband, Printers, Cameras, Network infrastructure/access & more
- **1 billionth core shipped in Apr 2011, expecting 2B in 2012**

Business – Semiconductor IP licensing

- **180+ Licensees worldwide**
- **Licensed by 8 of the top 12 semiconductor manufacturers**
- **Over 30 SSD manufacturers are using Tensilica-based Controllers today**

Technology

- **Customers generate processors with selected options and custom instructions**
  - Adding to a base instruction set, backwards compatible to 1999
- **Software Dev. tools and Physical components created automatically**
- **Processor is automatically verified (800+ different processors in silicon today)**

SSD Controller

IOPS: Data Management + Computational Throughput

Data Management

Getting data to the processor…

Latencies to where data is stored.

How often the same data is fetched again (inefficiency).

Available bus bandwidth.

Computational Throughput

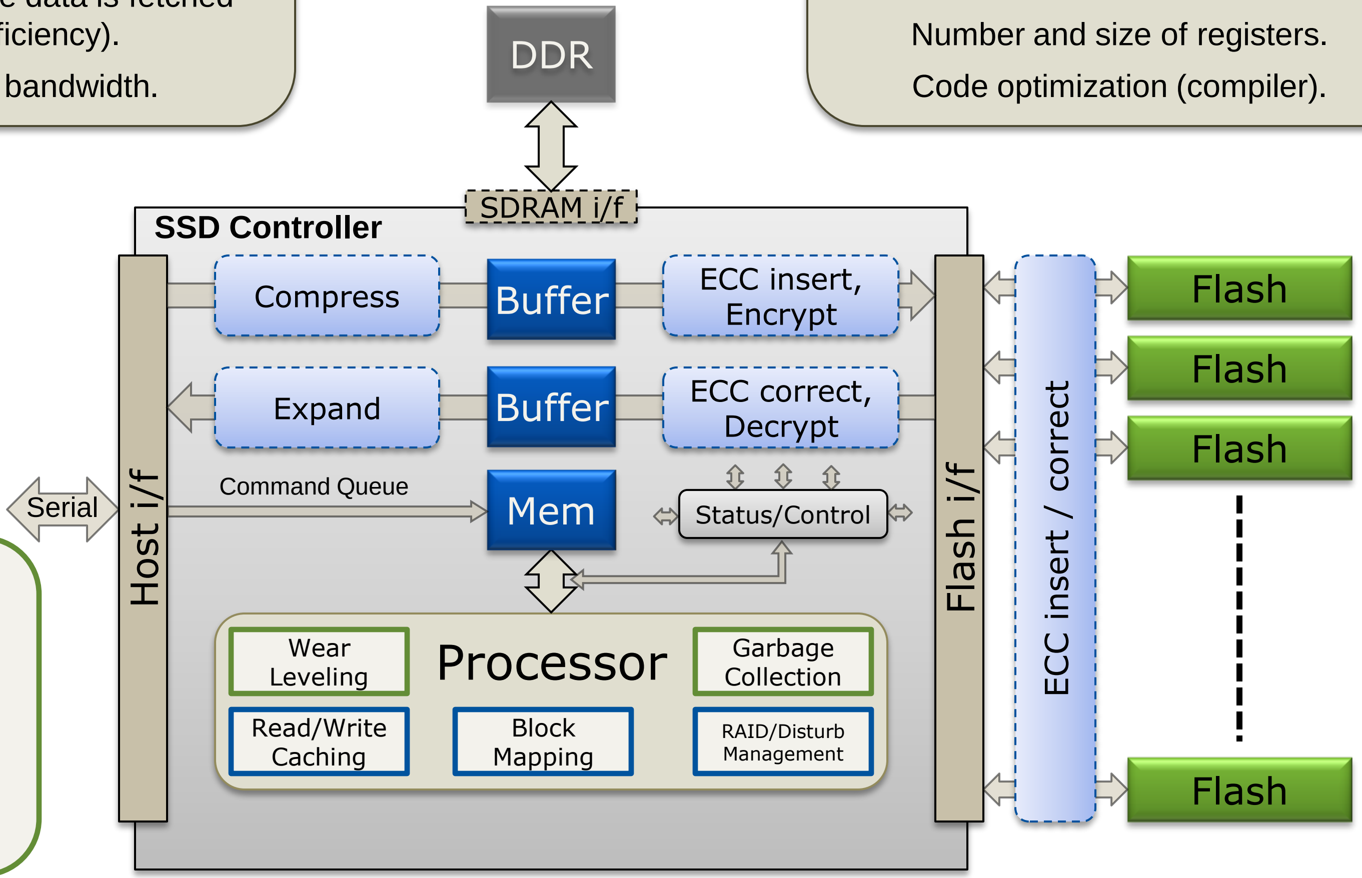
How many cycles it takes to execute each task…

Suitability of instruction set.

Number and size of registers.

Code optimization (compiler).

External RAM on some controllers



Every SSD Controller is different…

Processor choice affects how well each designer can solve the issues that their particular design encounters…

Improving Data Management

Conventional processors vs Xtensa processors

Conventional Processors:
Everything connects via the system bus.

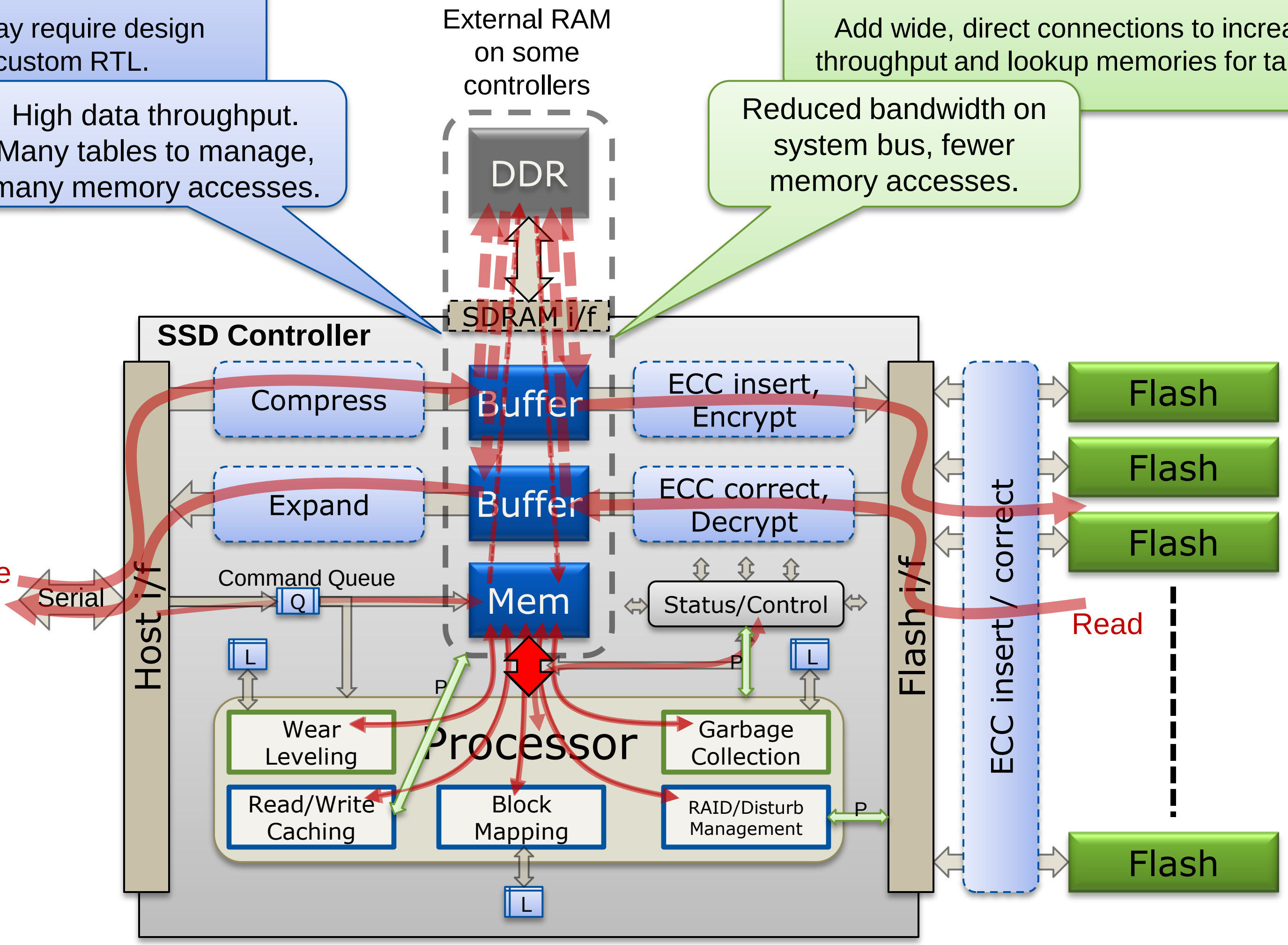
Limited bandwidth. May require design compromises or custom RTL.

High data throughput. Many tables to manage, many memory accesses.

Tensilica Processors:
Alternative connections avoid system bus.

Add wide, direct connections to increase throughput and lookup memories for tables.

Reduced bandwidth on system bus, fewer memory accesses.



- **Q**ueue Interfaces
  - Up to 1024 bits
  - Multiple Interfaces
- **L**ookup Memories
  - Up to 1023bits of address+data
  - Multiple interfaces
- **P**orts (GPIO)
  - Up to 1024bits
  - Multiple Interfaces

Improving Data Management

Xtensa processors provide system flexibility

Conventional Processors:
Everything connects via the system bus.

Limited bandwidth. May require design compromises or custom RTL.

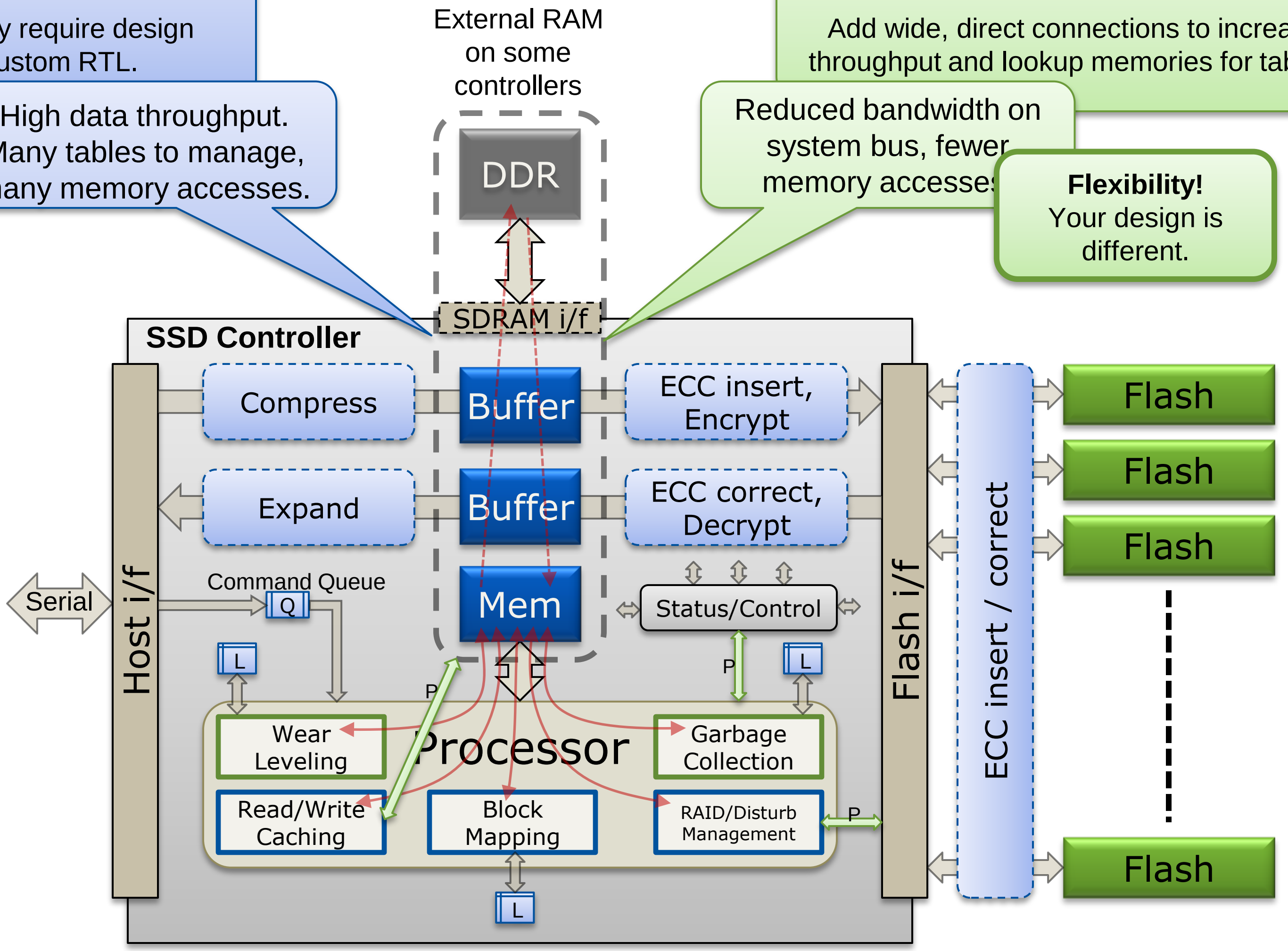
High data throughput. Many tables to manage, many memory accesses.

Tensilica Processors:
Alternative connections avoid system bus.

Add wide, direct connections to increase throughput and lookup memories for tables.

Reduced bandwidth on system bus, fewer memory accesses.

Flexibility!
Your design is different.



- **Q**ueue Interfaces
  - Up to 1024 bits
  - Multiple Interfaces
- **L**ookup Memories
  - Up to 1000bits of address
  - Multiple interfaces
- **P**orts (GPIO)
  - Up to 1024bits
  - Multiple Interfaces

Ex: Improving Data Management

By reducing the number of cycles required to fetch data

Table Lookup

Table lookups can be done locally - **no need to fetch over the system bus…**

Software Before

C Code:

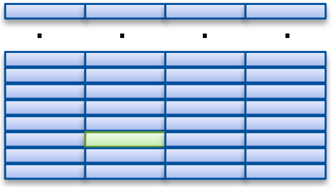
```
struct {
  unsigned data: 7;
  unsigned count: 9;
} table_a[4096];

unsigned data, el;
// Set up el
data = table_a[el].data;
```

Assembly:

6 cycles (**local** memory)

```
.
l32r a3, 60000978 <_stext+0x44> ;Base addr
addx4 a2, a2, a3     ;Offset (1 cyc. bubble)
l32i.n a2, a2, 0     ;Load data in table
extui a2, a2, 0, 7   ;Extract (1 cyc. bubble)
.
```

Table of 4096 entries
(eg. packed 7+9 bits)



Typically done in Software:
Small tables in local memory.
Large tables in system memory.

Often multiple, structured, data per entry.

SW After

C Code:

```
unsigned data, el;
// Set up el
data = LOOKUP_A(el);
```

Assembly:

1 cycle

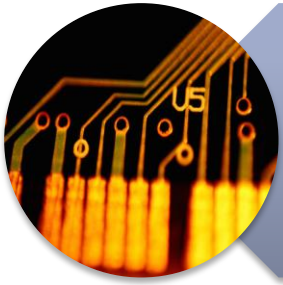
```
.
LOOKUP_A a2, a2
.
```

Instruction Added

```
TIE: (Instruction-mapped lookup memory)
lookup table {29, Wstage}{16, Wstage+1}
property lookup_memory table

operation LOOKUP_A {out AR data, in AR el}
                   {out table_Out, in table_In}
{
  assign table_Out = {1'b0, el[11:0], 16'b0}; //Fetch
  assign data = { table_In[15:9] }; //Top 7bits data
}
```

Increasing Computational Throughput

In Fixed Instruction-Set-Architecture (ISA) Processors



Run the processor faster

- May not be possible
- May have a system/board design impact

Optimize the SW

- If you have the time and the resources
- More maintenance if at assembly level

Choose a faster processor

- Will increase power/energy consumption
- Will cost more (in area & perhaps licensing)

Increasing Computational Throughput

In Xtensa Processors – more choices

Add new instructions

- Just to accelerate critical areas
- Often over 10x performance improvement

Improve register availability

- Add more registers for the compiler to use
- Set custom widths, up to 1024bits, to keep area as low as possible

Execute multiple instructions at once

- Add VLIW-style instructions without the code bloat
- A general purpose performance improvement

Increase local Memory bandwidth

- Second Load/Store unit
- Up to 512bits per cycle for each unit, 1024bits total per cycle

DMA to local memory

- For Command Queues/Code overlays etc.
- No processor cycles required to move data

Ex: Increasing Computational Throughput

By reducing the number of cycles required to process data

Byteswap

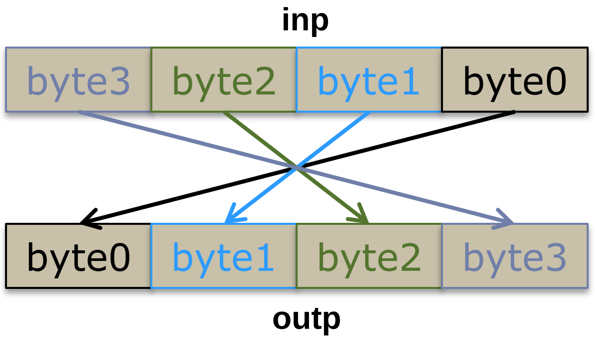
Customers improve performance by **more than 10x in real application code…**

Software Before

```
C Code:
unsigned bswap(unsigned inp) {
  unsigned outp = (inp<<24)
  | ((inp<<8)&0xff0000)
  | ((inp>>8)&0xff00)
  | (inp>>24);
  return outp;
}
```

```
Assembly:
.
.
l32r a6, 60000454 <_stext+0x44>
extui a3, a2, 24, 8
slli a4, a2, 8
slli a5, a2, 24
and a4, a4, a6
or a4, a4, a5
l32r a5, 60000458 <_stext+0x48>
srli a2, a2, 8
and a2, a2, a5
or a2, a2, a4
or a2, a2, a3
.
.
```

11 cycles



HW: 1 cycle

Software After

```
C Code:
unsigned bswap(unsigned inp) {
  unsigned outp = BYTESWAP(inp);
    return outp;
}
```

```
Assembly:
BYTESWAP a8, a8
```

1 cycle

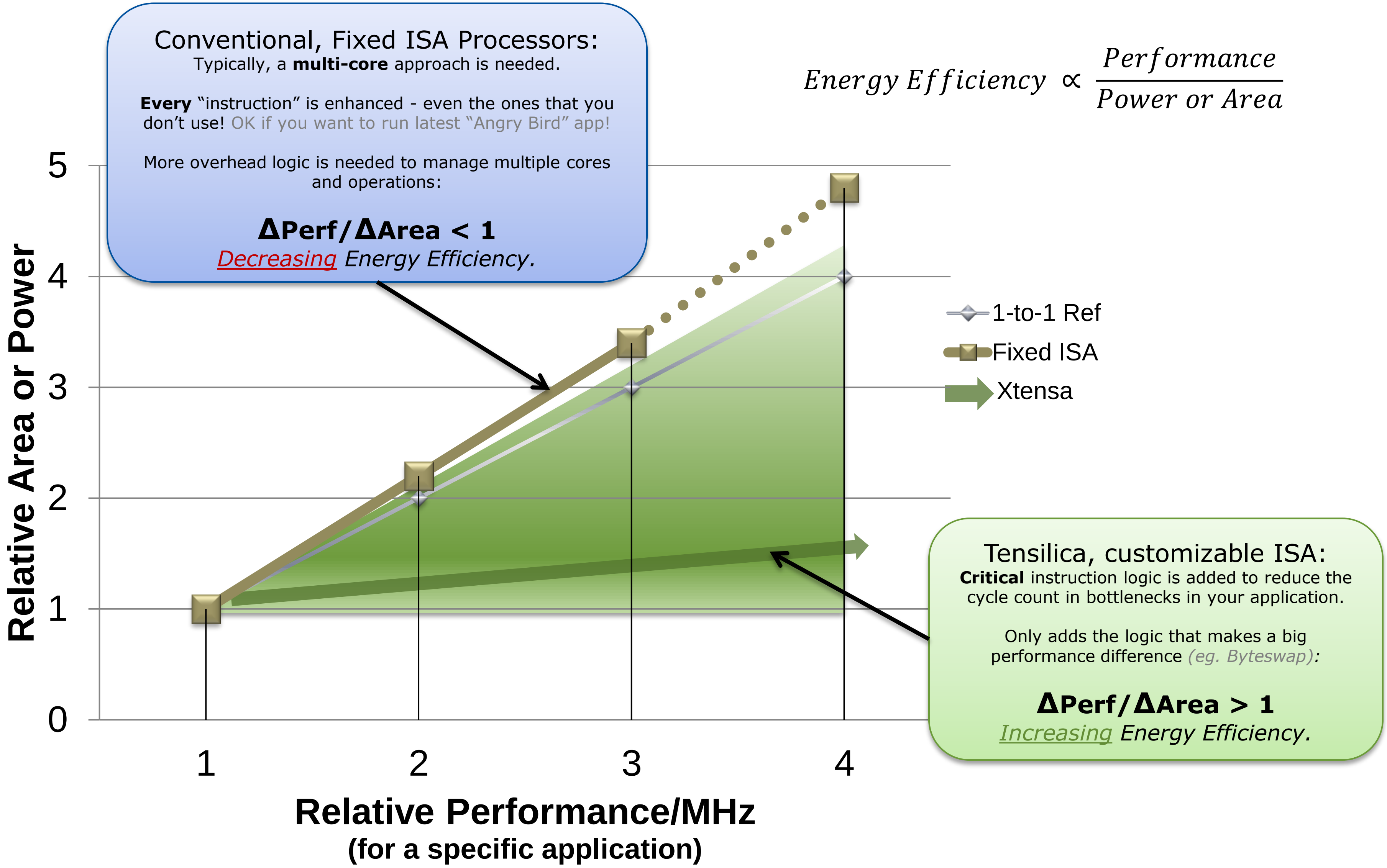
Instruction Added

```
TIE: (150 gates)
operation BYTESWAP
 {out AR outp,
  in AR inp}{} {
  assign outp =
  {
   inp[7:0],
   inp[15:8],
   inp[23:16],
   inp[31:24]
  };
}
```

Increasing Computational Throughput

Looking at Energy and Area Efficiency

$$Energy\ Efficiency \propto \frac{Performance}{Power\ or\ Area}$$



Conventional, Fixed ISA Processors:
Typically, a **multi-core** approach is needed.

Every "instruction" is enhanced - even the ones that you don't use! OK if you want to run latest "Angry Bird" app!

More overhead logic is needed to manage multiple cores and operations:

ΔPerf/ΔArea < 1
Decreasing Energy Efficiency.

Tensilica, customizable ISA:
Critical instruction logic is added to reduce the cycle count in bottlenecks in your application.

Only adds the logic that makes a big performance difference *(eg. Byteswap):*

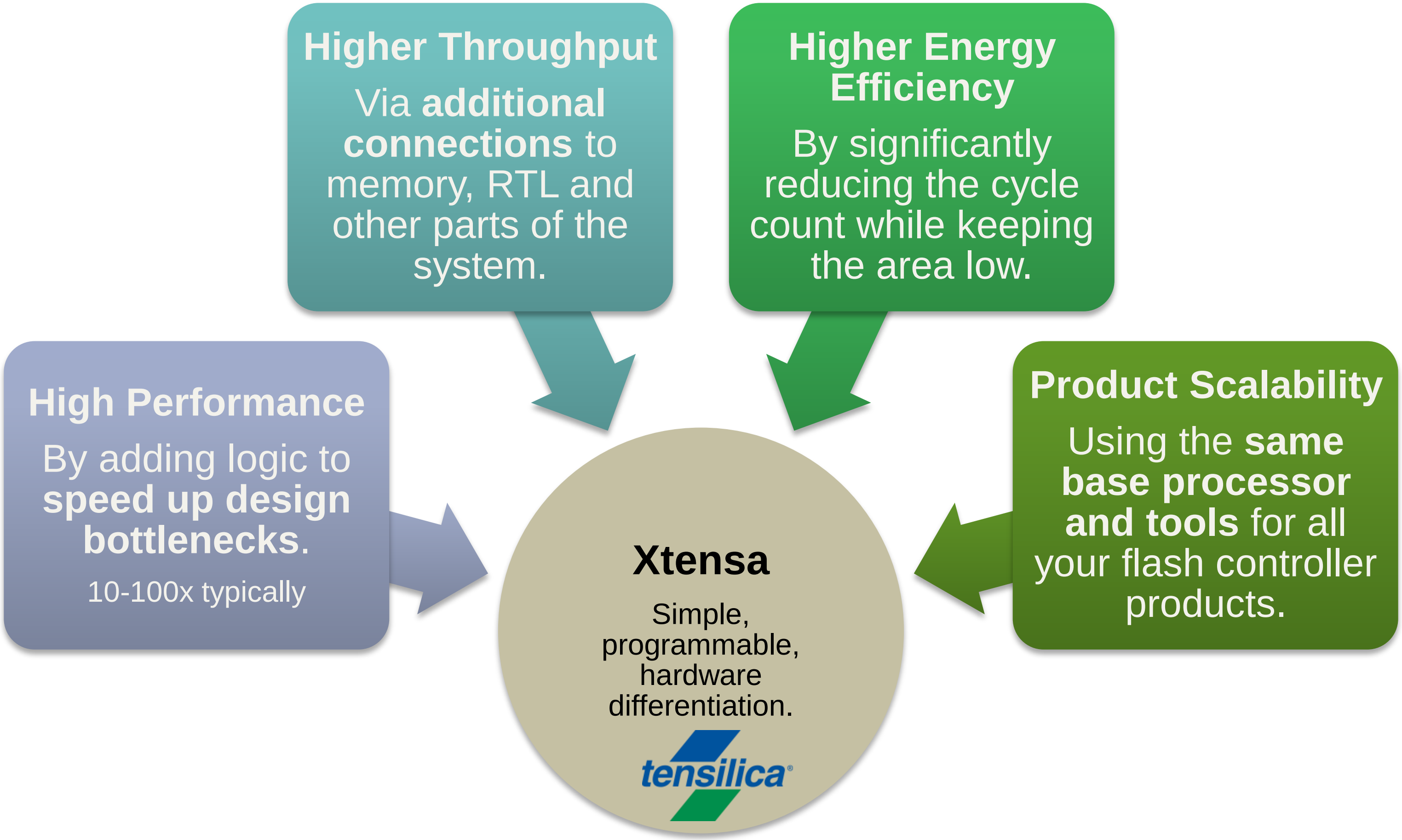
ΔPerf/ΔArea > 1
Increasing Energy Efficiency.

Processor choice affects design flexibility

Fix your design bottlenecks more efficiently with Xtensa

Increasing IOPS requires consideration of both:

Data Management and **Computational Throughput**



Higher Throughput

Via **additional connections** to memory, RTL and other parts of the system.

Higher Energy Efficiency

By significantly reducing the cycle count while keeping the area low.

High Performance

By adding logic to **speed up design bottlenecks**.

10-100x typically

Product Scalability

Using the **same base processor and tools** for all your flash controller products.

Xtensa

Simple, programmable, hardware differentiation.

tensilica

Thank you!