Reduce Power and Energy Consumption through ISA Extension

Steve Leibson
Technology Evangelist
Tensilica, Inc.
sleibson@tensilica.com
Agenda

• Power and energy consumption and the need for task acceleration
• Hardware acceleration versus ISA extension
• Three application examples
  • AES
  • Viterbi
  • FFT
• Conclusions
On-Chip Energy Consumption is Rising

- Dynamic and static energy consumption are rising with each new IC fabrication node (that’s very bad!)
Dynamic and static energy consumption are rising with each new IC fabrication node (that’s very bad!)

- Trend to higher clock rates drives dynamic power up
On-Chip Energy Consumption is Rising

Dynamic and static energy consumption are rising with each new IC fabrication node (that’s very bad!)

- Trend to higher clock rates drives dynamic power up
- Core voltages drop to compensate for higher dynamic power levels
On-Chip Energy Consumption is Rising

- Dynamic and static energy consumption are rising with each new IC fabrication node (that’s very bad!)
  - Trend to higher clock rates drives dynamic power up
  - Core voltages drop to compensate for higher dynamic power levels
  - Transistor threshold voltages drop to allow lower core voltages
On-Chip Energy Consumption is Rising

- Dynamic and static energy consumption are rising with each new IC fabrication node (that’s very bad!)
  - Trend to higher clock rates drives dynamic power up
  - Core voltages drop to compensate for higher dynamic power levels
  - Transistor threshold voltages drop to allow lower core voltages
  - Leakage and static energy consumption rise due to lower transistor threshold voltages
Why Reduce On-Chip Energy Consumption?

- Higher energy consumption hurts specs, costs money
Why Reduce On-Chip Energy Consumption?

- Higher energy consumption hurts specs, costs money
  - Less battery life
Why Reduce On-Chip Energy Consumption?

• Higher energy consumption hurts specs, costs money
  • Less battery life
    • Less talk, play, record time
    • Less standby time
Why Reduce On-Chip Energy Consumption?

- Higher energy consumption hurts specs, costs money
  - Less battery life
    - Less talk, play, record time
    - Less standby time
  - More costly power supply
Why Reduce On-Chip Energy Consumption?

• Higher energy consumption hurts specs, costs money
  • Less battery life
    • Less talk, play, record time
    • Less standby time
  • More costly power supply
    • Bigger supply costs more money, takes more space in package
    • Higher power supply heat hurts reliability, raises warranty costs
Why Reduce On-Chip Energy Consumption?

• Higher energy consumption hurts specs, costs money
  • Less battery life
    • Less talk, play, record time
    • Less standby time
  • More costly power supply
    • Bigger supply costs more money, takes more space in package
    • Higher power supply heat hurts reliability, raises warranty costs
  • More costly package cost, heat sinking, and fans
Why Reduce On-Chip Energy Consumption?

- Higher energy consumption hurts specs, costs money
  - Less battery life
    - Less talk, play, record time
    - Less standby time
  - More costly power supply
    - Bigger supply costs more money, takes more space in package
    - Higher power supply heat hurts reliability, raises warranty costs
  - More costly package cost, heat sinking, and fans
    - Cheap plastic IC packages cannot dissipate a lot of power
    - Bigger heat sinks cost more, take more space in enclosure
    - Fans increase audible noise and need even more space
So How Much Could it Cost?

**MacDailyNews**

Friday, July 06, 2007 - 02:07 PM EDT — Apple Stock Quote: 132.1699 (-0.5601, -0.414%)  
Serious flaws in Xbox 360 hardware to cost Microsoft at least $1 billion  
Friday, July 06, 2007 - 01:41 PM EDT

**Telegraph.co.uk**

Microsoft takes $1bn hit on Xbox  
By Emma Thurwell, Online City Reporter  
Last Updated: 9:51am BST 06/07/2007

SEATTLE, Washington (AP) — In another setback for Microsoft Corp.’s unprofitable entertainment devices division, the company says it is planning to spend at least $1 billion to repair serious problems with its Xbox 360 video game console.

Microsoft declined to detail the problem, which caused an onslaught of “general hardware failures” in recent months but said the company expects the repairs to cost $1 billion.

The glitches, and the bad publicity, cost Microsoft $1 billion over the last two years. In the last three months, the company has sold 10 million Xbox 360s, but said it was planning to spend $1 billion to repair the problem.

The software giant said that an “unsatisfactory number of repairs” to its Xbox 360 has forced it to take action.

From now on, any Xbox 360 customer who experiences a “general hardware failure” — which is indicated by three flashing red lights — will be covered by a three-year warranty from the date of purchase.

The Xbox 360 has been slow sales in Japan.

“We don’t think we’ve been doing the job done,” said Robbie Bach, president of Microsoft’s entertainment and devices division, which also makes the Zune digital music player, a distant competitor to Apple Inc.’s powermusic iPod. “In the past few months, we have been having to make Xbox 360 console repairs at a rate too high for our liking.”
It All Starts with the Chip Design
It All Starts with the Chip Design

- Find ways to cut on-chip power and energy consumption
It All Starts with the Chip Design

- Find ways to cut on-chip power and energy consumption
  - Drive clock rates down
    - (Very heretical)
It All Starts with the Chip Design

- Find ways to cut on-chip power and energy consumption
  - Drive clock rates down
    - (Very heretical)
  - To cut dynamic power dissipation and
It All Starts with the Chip Design

- Find ways to cut on-chip power and energy consumption
  - Drive clock rates down
    - (Very heretical)
  - To cut dynamic power dissipation and
  - To reduce the need for low-threshold transistors
It All Starts with the Chip Design

• Find ways to cut on-chip power and energy consumption
  • Drive clock rates down
    • (Very heretical)
  • To cut dynamic power dissipation and
  • To reduce the need for low-threshold transistors
  • To reduce static power dissipation
It All Starts with the Chip Design

- Find ways to cut on-chip power and energy consumption
  - Drive clock rates down
    - (Very heretical)
  - To cut dynamic power dissipation and
  - To reduce the need for low-threshold transistors
  - To reduce static power dissipation
- Avoid the use of buses when possible
It All Starts with the Chip Design

• Find ways to cut on-chip power and energy consumption
  • Drive clock rates down
    • (Very heretical)
  • To cut dynamic power dissipation and
  • To reduce the need for low-threshold transistors
  • To reduce static power dissipation

• Avoid the use of buses when possible
  • Find alternative communication methods that don’t require blocks to drive wide, shared, highly capacitive buses
Conventional Hardware Acceleration
Conventional Hardware Acceleration
Conventional Hardware Acceleration

Illustration showing a system with a processor, RAM, ROM, and an accelerator connected through a bus. The bus is labeled as a bottleneck with a note indicating that bus bandwidth must accommodate memory and accelerator traffic.
Conventional Hardware Acceleration

- Manually split tasks between processor and accelerator
- Hardware-accelerated tasks live outside of the software environment
- Bus bandwidth must accommodate memory and accelerator traffic
Reduce Energy Use With ISA Extension

ISA = instruction-set architecture (registers + instructions)
Reduce Energy Use With ISA Extension

- Use processor ISA extension to improve task-execution speed, energy consumption, or both

ISA = instruction-set architecture (registers + instructions)
Reduce Energy Use With ISA Extension

- Use processor ISA extension to improve task-execution speed, energy consumption, or both
- Algorithm-specific registers and operations reduce the number of cycles needed to execute the algorithm

ISA = instruction-set architecture (registers + instructions)
Reduce Energy Use With ISA Extension

- Use processor ISA extension to improve task-execution speed, energy consumption, or both.
- Algorithm-specific registers and operations reduce the number of cycles needed to execute the algorithm.
- Reduces energy consumption by executing same number of task iterations in many fewer clock cycles.

ISA = instruction-set architecture (registers + instructions)
Reduce Energy Use With ISA Extension

- Use processor ISA extension to improve task-execution speed, energy consumption, or both
- Algorithm-specific registers and operations reduce the number of cycles needed to execute the algorithm
- Reduces energy consumption by executing same number of task iterations in many fewer clock cycles
  - Fewer clock cycles allow the processor to sleep more at the same clock rate (low standby power)
  - or...
  - Allows the processor to run at a lower clock rate, reducing power dissipation and energy consumption in a superlinear ($1/2 CV^2F$) fashion due to lower operating frequency combined with lower core operating voltage

ISA = instruction-set architecture (registers + instructions)
Translating from Processor Speak

When processor vendors talk about “adding instructions” or “extending the ISA”

they mean:

“adding hardware to the processor”

• generally including (but not limited to):
  • new registers
  • new register files
  • execution-unit additions.
Acceleration Through ISA Extension
Acceleration Through ISA Extension
Acceleration Through ISA Extension

- No bottleneck!
- No Task Partitioning

ISA Extensions (Accelerator)
Acceleration Through ISA Extension

No bottleneck! No Task Partitioning

ISA Extensions (Accelerator)
Processor
RAM
ROM
BUS

Task-specific data types can be directly mapped to extended registers of any width and remain part of the software environment unlike bus-attached hardware accelerators
How to Add Hardware to a Processor

1. Analyze the algorithm or task
2. Identify key operations for acceleration
3. Identify appropriate data types (as opposed to “wedge” and “cut up” data types to fit existing resources)
4. Define and design resources to accommodate special data types and accelerate operations
Example: AES Cryptography

- Block-oriented crypto based on Rijndael cipher
  - FIPS 197, adopted by the US government in 2002
  - Obsoletes DES but not Triple DES
  - Current technology needs 147 trillion years to crack an AES cipher using exhaustive search at $2^{55}$ keys/sec

- Some AES Applications
  - SATA disk interface (1.2-4.8 Gbits/sec)
  - Wireless LAN (to 10 Mbits/sec to 1 Gbit/sec)
AES Cryptography Operations

- Four basic transformations in the AES algorithm
  - **SubBytes** – Table-based, byte-wise substitution in state array
  - **ShiftRow** – Rearrange bytes within rows of the state array
  - **MixColumn** – Galois matrix multiplication on state array columns
  - **AddRoundKey** – Byte-wise Galois field addition (128-bit XOR) with bytes from appropriate subkey in key schedule
AES Encryption Flowchart

Plain Text

First SubKey → AddRoundKey

SubBytes → ShiftRows → MixColumns → AddRoundKey

Subkey

No → Last Round?

Last Round?

AddRoundKey

SubBytes → ShiftRows

AddRoundKey

Last Subkey

Cypher Text
128-Bit AES State Array (Data Type)

Plaintext Input Stream

<table>
<thead>
<tr>
<th>Input Byte 0</th>
<th>Input Byte 4</th>
<th>Input Byte 8</th>
<th>Input Byte 12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Byte 1</td>
<td>Input Byte 5</td>
<td>Input Byte 9</td>
<td>Input Byte 13</td>
</tr>
<tr>
<td>Input Byte 2</td>
<td>Input Byte 6</td>
<td>Input Byte 10</td>
<td>Input Byte 14</td>
</tr>
<tr>
<td>Input Byte 3</td>
<td>Input Byte 7</td>
<td>Input Byte 11</td>
<td>Input Byte 15</td>
</tr>
</tbody>
</table>

32 bits
Processor-Based Thinking *Immediately* Kicks In

<table>
<thead>
<tr>
<th>Plaintext Input Stream</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Byte 0</td>
</tr>
<tr>
<td>Input Byte 1</td>
</tr>
<tr>
<td>Input Byte 2</td>
</tr>
<tr>
<td>Input Byte 3</td>
</tr>
<tr>
<td>Input Byte 4</td>
</tr>
<tr>
<td>Input Byte 5</td>
</tr>
<tr>
<td>Input Byte 6</td>
</tr>
<tr>
<td>Input Byte 7</td>
</tr>
<tr>
<td>Input Byte 8</td>
</tr>
<tr>
<td>Input Byte 9</td>
</tr>
<tr>
<td>Input Byte 10</td>
</tr>
<tr>
<td>Input Byte 11</td>
</tr>
<tr>
<td>Input Byte 12</td>
</tr>
<tr>
<td>Input Byte 13</td>
</tr>
<tr>
<td>Input Byte 14</td>
</tr>
<tr>
<td>Input Byte 15</td>
</tr>
</tbody>
</table>

32 bits

32-bit Register

32-bit Register

32-bit Register

32-bit Register
Please **Resist** the Temptation

Plaintext Input Stream

<table>
<thead>
<tr>
<th>Input Byte 0</th>
<th>Input Byte 4</th>
<th>Input Byte 8</th>
<th>Input Byte 12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Byte 1</td>
<td>Input Byte 5</td>
<td>Input Byte 9</td>
<td>Input Byte 13</td>
</tr>
<tr>
<td>Input Byte 2</td>
<td>Input Byte 6</td>
<td>Input Byte 10</td>
<td>Input Byte 14</td>
</tr>
<tr>
<td>Input Byte 3</td>
<td>Input Byte 7</td>
<td>Input Byte 11</td>
<td>Input Byte 15</td>
</tr>
</tbody>
</table>

32-bit Register

32-bit Register

32-bit Register

32-bit Register

32 bits
AES State Array Row and Column Operations

- **ShiftRow** transformation operates on rows.
- **MixColumns** transformation operates on columns.
AES ShiftRow Transformation

<table>
<thead>
<tr>
<th>Input Byte 0</th>
<th>Input Byte 4</th>
<th>Input Byte 8</th>
<th>Input Byte 12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Byte 1</td>
<td>Input Byte 5</td>
<td>Input Byte 9</td>
<td>Input Byte 13</td>
</tr>
<tr>
<td>Input Byte 2</td>
<td>Input Byte 6</td>
<td>Input Byte 10</td>
<td>Input Byte 14</td>
</tr>
<tr>
<td>Input Byte 3</td>
<td>Input Byte 7</td>
<td>Input Byte 11</td>
<td>Input Byte 15</td>
</tr>
</tbody>
</table>

- No Shift
- Rotate Left 1 Byte
- Rotate Left 2 Bytes
- Rotate Left 3 Bytes
AES MixColumn Transformation

AES Galois Matrix Multiplication on a State-Array Column

Notes:
Galois multiplication is much simpler than binary multiplication.
The multiplicand is always 1, 2, or 3.
**Baseline AES Results Coded in C for RISC processor**

<table>
<thead>
<tr>
<th></th>
<th>Cycle Count (per 10 blocks)</th>
<th>Estimated Energy (μJ)</th>
<th>Estimated Instantaneous Power (mW @ 100 MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES Encryption</td>
<td>353,493</td>
<td>57.03</td>
<td>16.13</td>
</tr>
<tr>
<td>AES Encryption and Decryption</td>
<td>679,517</td>
<td>108.18</td>
<td>15.92*</td>
</tr>
</tbody>
</table>

Estimates generated by Xenergy energy estimator and the Xtensa ISS, assuming TSMC 130 LV process @ 100 MHz.

* Lower power for encryption and decryption is due to more stalled (lower-power) processor cycles while waiting for memory.
Add Three 128-Bit AES Encoding Instructions and a 128-Bit AES Register

ENSTEP 0

First SubKey

AddRoundKey

ENSTEP 1

Nine Iterations

SubBytes

ShiftRows

MixColumns

Subkey

AddRoundKey

Inner Loop

No

Last Round?

ENSTEP 2

SubBytes

ShiftRows

AddRoundKey

128-Bit AES Register

128-Bit AES Register

ENSTEP = AES Encoding Step (new instructions)
Code Encryption Rounds in TIE

The three new ENSTEP instructions use combinations of one or more AES encoding steps:

1. Load 128-bit subkey from the key schedule with automatic subkey increment (forces a 2-cycle instruction to wait for the load)
2. AddRoundKey transformation (ENSTEP 0,1,2)
3. SubBytes transformation (ENSTEP 1 & 2)
4. ShiftRow transformation (ENSTEP 1 & 2)
5. MixColumn transformation (ENSTEP 1 only)
128-Bit AddRoundKey Transformation

AES Register = Data XOR Subkey
128-Bit SubBytes Transformation

Use lookup tables to perform 16 simultaneous byte substitutions for the 16 data bytes
128-Bit SubBytes Transformation

Use lookup tables to perform 16 simultaneous byte substitutions for the 16 data bytes.

Saving System Power: Avoid memory and bus traffic associated with retrieving fixed substitution values by implementing the lookup tables as logic within the function unit.
128-Bit SubBytes Transformation

Use lookup tables to perform 16 simultaneous byte substitutions for the 16 data bytes.

Saving System Power: Avoid memory and bus traffic associated with retrieving fixed substitution values by implementing the lookup tables as logic within the function unit.
128-Bit ShiftRow Transformation

Byte-lane scrambling: No logic. Nothing but wires.
128-Bit MixColumn Transformation

- Use 64 byte-wide Galois multipliers (which use logical operations) to perform the entire matrix-multiplication function in one operation.
- Simplify each byte-wide Galois multiplier by exploiting the fact that the multiplicand is always 1, 2, or 3.
  - Multiplicand = 1: use the identity function.
  - Multiplicand = 2: shift the input value left by one bit. If MSB = 1 after the shift, XOR the intermediate result with x01.
  - Multiplicand = 3: XOR the (multiplicand = 1) value with the (multiplicand = 2) value.
What have we built? An Optimized 128-bit AES Encryption Function Unit and Register

Schedule these functions over two execution cycles, maximizing the processor’s clock-rate ceiling while exploiting subkey load latency.
One More Efficiency Trick: Eliminate Execution Pipeline Bubbles

2-Cycle ENSTEP 1 instruction iteration (AES register dependency causes pipeline bubbles)
One More Efficiency Trick: Eliminate Execution Pipeline Bubbles

2-Cycle ENSTEP 1 instruction iteration (AES register dependency causes pipeline bubbles)

2-Cycle ENSTEP 1 instruction interleaving removes dependency and pipeline bubbles
Resulting Processor Block Diagram

- Instruction-Fetch and Instruction-Decode Unit
- Base ISA Execution Pipeline
- 32-Entry General-Purpose Register File
- Base ALU
- Branch Unit
- Load/Store Unit
- Instruction Memory Region Protection Unit
- Instruction RAM Interface
- Processor Interface Bus Control
- Write Buffer
- Data Memory Region Protection Unit
- Data RAM Interface #1
- Data RAM Interface #2
- Data RAM #1
- Data RAM #2
- External Bus Interfaces
- Instruction RAM
- 32-Bit Main Bus

* Caches optional
Resulting Processor Block Diagram

128-Bit AES ISA Extensions

AES Execution Pipeline

- 128-Bit AES Register
- 16 Byte-Wide Lookup Tables
- 128-Bit XOR
- AddRoundKey

Instruction-Fetch and Instruction-Decode Unit

Base ISA Execution Pipeline

- Instruction Memory Region Protection Unit

32-Entry General-Purpose Register File

Base ALU

Branch Unit

Load/Store Unit

Instruction RAM Interface

External Bus Interfaces

- Instruction RAM
- 32-Bit Main Bus

Data Memory Region Protection Unit

Data RAM Interface #1

Data RAM Interface #2

Write Buffer

Processor Interface Bus Control

Data RAM #1

Data RAM #2

* Caches optional
Resulting Processor Block Diagram

128-Bit AES ISA Extensions

- AES Execution Pipeline
  - 128-Bit AES Register
  - 16 Byte-Wide Lookup Tables

- Extension hardware automatically generated from instruction descriptions

- Instruction-Fetch and Instruction-Decode Unit
- Base ISA Execution Pipeline
- 32-Entry General-Purpose Register File
- Base ALU
- Branch Unit
- Load/Store Unit
- Instruction Memory Region Protection Unit
- Instruction RAM Interface
- Process Interface Bus Control
- Write Buffer
- Data Memory Region Protection Unit
- Data RAM Interface #1
- Data RAM Interface #2
- Data RAM #1
- Data RAM #2

External Bus Interfaces

- Instruction RAM
- 32-Bit Main Bus
- Data RAM #1
- Data RAM #2

* Caches optional
128-Bit AES ISA Extensions

Extension hardware only clocked when executing AES instructions

Extension hardware automatically generated from instruction descriptions

* Caches optional
for (k=0; k<BLOCKS; k=k+2) 
{
    blockA = *p_block++; //load plaintext block to aes regfile
    blockB = *p_block++;
    ENSTEP0(p_keyA,blockA); //encode round 0
    ENSTEP0(p_keyB,blockB);
    ENSTEP1(p_keyA,blockA); //encrypt rounds 1-9
    ENSTEP1(p_keyB,blockB);
    ENSTEP1(p_keyA,blockA);
    ENSTEP1(p_keyB,blockB);
    ENSTEP1(p_keyA,blockA);
    ENSTEP1(p_keyB,blockB);
    ENSTEP1(p_keyA,blockA);
    ENSTEP1(p_keyB,blockB);
    ENSTEP1(p_keyA,blockA);
    ENSTEP1(p_keyB,blockB);
    ENSTEP1(p_keyA,blockA);
    ENSTEP1(p_keyB,blockB);
    ENSTEP1(p_keyA,blockA);
    ENSTEP1(p_keyB,blockB);
    ENSTEP1(p_keyA,blockA);
    ENSTEP1(p_keyB,blockB);
    ENSTEP1(p_keyA,blockA);
    ENSTEP1(p_keyB,blockB);
    ENSTEP1(p_keyA,blockA);
    ENSTEP1(p_keyB,blockB);
    ENSTEP1(p_keyA,blockA);
    ENSTEP1(p_keyB,blockB);
    ENSTEP2(p_keyA,blockA); //encrypt round 10
    ENSTEP2(p_keyB,blockB);
    *p_result++ = blockA; //store ciphertext blocks to memory
    *p_result++ = blockB;
    p_keyA=p_keyB=&keyschedule[0]; //reset pointer to keyschedule
}
## Results of ISA Optimization

<table>
<thead>
<tr>
<th></th>
<th>Cycle Count (per 10 blocks)</th>
<th>Estimated Energy (μJ)</th>
<th>Estimated Instantaneous Power (mW @ 100 MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Straight C</td>
<td>ISA Optimized</td>
<td>Straight C</td>
</tr>
<tr>
<td>AES Encryption</td>
<td>353,493</td>
<td>10,713</td>
<td>57.03</td>
</tr>
<tr>
<td>AES Encryption and Decryption</td>
<td>679,517</td>
<td>16,966</td>
<td>108.18</td>
</tr>
</tbody>
</table>

Estimates generated by Xenergy energy estimator with the Xtensa ISS.

**Notes:**

1. Instantaneous power *(power is instantaneous)* increases slightly due to more gates but energy consumption decreases due to fewer cycles.

2. Estimates do not account for lower core operating voltage due to lower clock rate.
## Results of ISA Optimization

<table>
<thead>
<tr>
<th></th>
<th>Cycle Count (per 10 blocks)</th>
<th>Estimated Energy (μJ)</th>
<th>Estimated Instantaneous Power (mW @ 100 MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Straight C</td>
<td>ISA Optimized</td>
<td>Straight C</td>
</tr>
<tr>
<td>AES Encryption</td>
<td>353,493</td>
<td><strong>10,713</strong></td>
<td>57.03</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>30-40x</strong></td>
<td></td>
</tr>
<tr>
<td>AES Encryption and Decryption</td>
<td>679,517</td>
<td><strong>16,966</strong></td>
<td>108.18</td>
</tr>
</tbody>
</table>

Estimates generated by Xenergy energy estimator with the Xtensa ISS.

Notes:

1. Instantaneous power (power is instantaneous) increases slightly due to more gates but energy consumption decreases due to fewer cycles.

2. Estimates do not account for lower core operating voltage due to lower clock rate.
Further 8x Cycle-Count Reduction: 1024-bit AES
Registers Fed by External FIFO Queues

Throughput: Unrolled inner loop encrypts ~ 8.5 bytes/cycle
(versus 1.07 bytes/cycle for the bus-based version)
The Outer Limits: Two more Orders of Magnitude

Throughput: Unrolled inner loop encrypts ~ 850 bytes/cycle
You still need only three new instructions.

Note: Practical gate-count and routing limit is 1 < X < 100X
Wide Interconnect and Wire Density: What’s Practical? Do the Math

8-10 Metal Layers, ITRS 2006 wire spacing

<table>
<thead>
<tr>
<th>Technology</th>
<th>Wire Density</th>
</tr>
</thead>
<tbody>
<tr>
<td>90nm</td>
<td>100,000+</td>
</tr>
<tr>
<td>65nm</td>
<td>Almost 200,000</td>
</tr>
</tbody>
</table>

90nm: 100,000+ wires/square mm
65nm: Almost 200,000 wires/square mm
Conclusions: Key Points for AES

- Two new 128-bit AES registers and three new instructions reduce energy consumption by 30-40x for the AES encryption application.
  - Scheduled one instruction over two cycles so as not to decrease processor’s maximum possible clock rate after synthesis.
  - Added second 128-bit AES data register to maintain throughput of one AES instruction per cycle by interleaving operations on two data blocks.
- More improvement possible if you bypass the processor’s bus using wide, direct-access ports to the 128-bit AES registers.
- The RISC processor retains ability to run other application code even with AES-specific ISA extensions.
Final Conclusions: Key Points

✓ Many, many algorithms lend themselves to these energy-saving design techniques
✓ Applying domain-specific expertise nets big reductions in energy used to execute a task
✓ Think different: the smallest core doesn’t necessarily use the least energy—cycle counts rule!
✓ Stay off the bus!!!
Bonus Example #1: Viterbi Decode

Viterbi Coding

• Determines the most likely path through a state sequence given the presence of noise
• Overcomes noise through spread-spectrum, redundancy, and convolutional coding
• Used for cellular phone reception

Hedy Lamarr

Dr. Andrew Viterbi
Co-Founded QUALCOMM in 1985
Viterbi Trellis: Traversing the States

Tracing the Most Likely Path Through States
Viterbi Butterfly: Two Sources, Two Destinations, Four Arcs per Group

Select next state using a computed distance metric.
Results of ISA Optimization

<table>
<thead>
<tr>
<th></th>
<th>Cycle Count</th>
<th>Estimated Energy (μJ)</th>
<th>Estimated Instantaneous Power (mW @ 100 MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Straight C</td>
<td>ISA Optimized</td>
<td>Straight C</td>
</tr>
<tr>
<td>Viterbi Decode</td>
<td>279,537</td>
<td>7632</td>
<td>65.69</td>
</tr>
</tbody>
</table>

Estimates generated by Xenergy energy estimator with the Xtensa ISS.

Notes:

1. Instantaneous power (*power is instantaneous*) increases due to more gates but energy consumption decreases due to fewer cycles.

2. Estimates do not account for lower core operating voltage due to lower clock rate.
## Results of ISA Optimization

<table>
<thead>
<tr>
<th></th>
<th>Cycle Count</th>
<th>Estimated Energy (μJ)</th>
<th>Estimated Instantaneous Power (mW @ 100 MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Straight C</strong></td>
<td>279,537</td>
<td>65.69</td>
<td>23.5</td>
</tr>
<tr>
<td><strong>ISA</strong></td>
<td>7632</td>
<td>2</td>
<td>~1.15x</td>
</tr>
<tr>
<td><strong>Notes:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1. Instantaneous power \textit{(power is instantaneous)} increases due to more gates but energy consumption decreases due to fewer cycles.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. Estimates do not account for lower core operating voltage due to lower clock rate.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Other Coding Algorithms are Equally Targetable

Related convolutional codes

- Turbo (3G cellular, deep-space communications)
- LDPC (Low-Density Parity Check, 3GPP cellular)
Bonus Example #2: FFT

Fourier Transform

- Decomposes signals into frequency components
- Very “mathy”
- Foundation of DSP

\[ X(f) = \int_{-\infty}^{\infty} x(t) \ e^{-i2\pi ft} \, dt, \]

Fast Fourier Transform

- Fast algorithm for computing discrete Fourier transform

\[ X_k = \sum_{n=0}^{N-1} e^{-2\pi i k \cdot (n/N)} x_n \]
FFTs Hurt My Brain

Pain Related Brain Activity is Reduced with No FFT

With FFT  No FFT
FFT for 802.11g Wireless PHY

- Each FFT to be completed in 3.2 usec
  - 64-point
  - decimation-in-frequency
  - 16-bit real/16-bit imaginary complex data
- Radix-4 FFT Butterfly requires:
  - Twelve 16x16-bit multipliers
  - More than twenty 16-bit adders
## Results of ISA Optimization

<table>
<thead>
<tr>
<th>64-point, 16-bit Decimation in Frequency Complex FFT</th>
<th>Cycle Count</th>
<th>Estimated Energy (μJ)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>One FFT in 3.2 usec</td>
<td>Required Clock Rate</td>
</tr>
<tr>
<td>Straight C</td>
<td>32187</td>
<td>10 GHz</td>
</tr>
<tr>
<td>Add 32-bit Multiplier</td>
<td>5071</td>
<td>1.6 GHz</td>
</tr>
<tr>
<td>Multiple Instruction Issue and 32-bit Multiplier</td>
<td>2975</td>
<td>930 MHz</td>
</tr>
<tr>
<td>Radix-4 FFT ISA Extension</td>
<td>146</td>
<td>46 MHz</td>
</tr>
</tbody>
</table>
## Results of ISA Optimization

<table>
<thead>
<tr>
<th>64-point, 16-bit Decimation in Frequency Complex FFT</th>
<th>Cycle Count</th>
<th>Estimated Energy (µJ)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>One FFT in 3.2 usec</td>
<td>Required Clock Rate</td>
</tr>
<tr>
<td>Straight C</td>
<td>32187</td>
<td>10 GHz</td>
</tr>
<tr>
<td>Add 32-bit Multiplier</td>
<td>5071</td>
<td>1.6 GHz</td>
</tr>
<tr>
<td>Multiple Instruction Issue and 32-bit Multiplier</td>
<td>2975</td>
<td>930 MHz</td>
</tr>
<tr>
<td>Radix-4 FFT ISA Extension</td>
<td>146</td>
<td>46 MHz</td>
</tr>
</tbody>
</table>

Impossible using synthesized logic even with 65nm ASIC fabrication technology.