Instruction Set Extensions for Efficient AES Implementation on 32-bit Processors

Workshop on Cryptographic Hardware and Embedded Systems (CHES) 2006

Stefan Tillich and Johann Großschädl

IAIK – Graz University of Technology
Stefan.Tillich@iaik.tugraz.at
www.iaik.tugraz.at
Outline

• Motivation
• Design principles
• Proposed AES instruction set extensions
• Practical results
  • Hardware cost
  • Performance
  • Code size
• Note on side-channel attacks
• Conclusions
Motivation

• Mid-range to high-end embedded systems incorporate 32-bit processors
• Efficient implementation of cryptographic algorithms under different aspects
  • Performance
  • Code size, memory footprint
  • Limited energy budget
  • Flexibility, extensibility
• ISE design approach
  • Custom instructions added to general-purpose processor
  • Unites features of pure-software and dedicated hardware solutions
• Extensive study for symmetric cryptography by means of AES
AES Algorithm

- 128, 192, or 256-bit key
- 10, 12, or 14 rounds
- 4 x 4 byte State
- 4 transformations
  - SubBytes
  - ShiftRows
  - MixColumns
  - AddRoundKey

SubBytes ( )
ShiftRows ( )
MixColumns ( )
AddRoundKey ( )
Design Principles for AES Extensions

- Custom instructions to support round transformations and key expansion
- For all key sizes, modes of operation, etc.
- Easy integration in 32-bit embedded processors
- RISC-like instruction format (2 input operands, 1 output operand)
- No non-standard architectural features (e.g. dedicated lookup tables)
- Short critical path
Byte-Oriented AES Extensions

- Low-cost instructions
- 1st operand is register, 2nd operand is immediate value
- Immediate value determines operation
- Produce 1-byte result
- Result is written to a specified byte of destination register
Byte-Oriented AES Extensions

- **sbox rs1, imm, rd**
- Supports SubBytes, ShiftRows, key expansion

- **mixcol rs1, imm, rd**
- Supports MixColumns
Plain Word-Oriented AES Extensions

- How can performance be improved?
- Simple idea: Functionality of byte-oriented extensions quadrupled

- `sbox` -> `sbox4`, `mixcol` -> `mixcol4`
- Problem: `sbox4` & `mixcol4` cannot be combined efficiently
  - ShiftRows requires rows, MixColumns requires columns
  - ShiftRows becomes bottleneck!
Solving the "ShiftRows Problem"

- Pack AES State columns into 32-bit registers
- Work on 2 State columns simultaneously
- Split ShiftRows into 2 parts; performed implicitly
  - 1st part at end of SubBytes
  - 2nd part at beginning of MixColumns

- Implementation
  - Instructions have two register source operands (2 State columns)
  - Appropriate bytes selected for transformation
Advanced Word-Oriented AES Extensions

- `sbox4s rs1, rs2, rd`
- `isbox4s rs1, rs2, rd`
- `sbox4r rs1, rs2, rd`
- Supports SubBytes, ShiftRows, key expansion

- `mixcol4s rs1, rs2, rd`
- `imixcol4s rs1, rs2, rd`
- Supports ShiftRows, MixColumns
Implementation

- AES extensions integrated into SPARC V8-compatible Leon2 embedded processor
- Estimation of hardware overhead
  - Synthesis of complete 5-stage pipeline (integer unit) using UMC 0.13 µm standard-cell library
  - 4 ns critical path delay (i.e. 250 MHz)
- Performance evaluation
  - Prototyped modified processor on FPGA board
- Code size
  -Compilation with GNU toolchain
### Hardware Cost

<table>
<thead>
<tr>
<th>Integer unit variant</th>
<th>Gate equivalents</th>
<th>Area increase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline (no extensions)</td>
<td>13,349</td>
<td>-</td>
</tr>
<tr>
<td>Low cost (sbox &amp; mixcol)</td>
<td>13,853</td>
<td>+ 4%</td>
</tr>
<tr>
<td>Area/perf. trade-off (sbox &amp; mixcol4)</td>
<td>14,583</td>
<td>+ 9%</td>
</tr>
<tr>
<td>High performance (sbox4s &amp; mixcol4s)</td>
<td>16,370</td>
<td>+ 23%</td>
</tr>
</tbody>
</table>
AES-128 Encryption (Precomputed Key Schedule)  
Performance and Code Size

<table>
<thead>
<tr>
<th>AES impl.</th>
<th>Key exp.</th>
<th>Encryption</th>
<th>Code size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline (no extensions)</td>
<td>1.0 (739 cycles)</td>
<td>1.0 (1,637 cycles)</td>
<td>1.0 (2,168 byte)</td>
</tr>
<tr>
<td>Low cost (sbox &amp; mixcol)</td>
<td>2.1</td>
<td>2.8</td>
<td>- 72%</td>
</tr>
<tr>
<td>Area/perf. trade-off</td>
<td>2.1</td>
<td>4.8</td>
<td>- 78%</td>
</tr>
<tr>
<td>High performance</td>
<td>2.3</td>
<td>8.3</td>
<td>- 59%</td>
</tr>
<tr>
<td>T lookup (Gladman) 4 KB</td>
<td>1.7</td>
<td>1.5</td>
<td>+ 403%</td>
</tr>
</tbody>
</table>
# AES-128 Encryption (On-the-fly Key Expansion)

## Performance and Code Size

<table>
<thead>
<tr>
<th>AES impl.</th>
<th>Encryption</th>
<th>Code size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td></td>
<td>(2,239 cycles)</td>
<td>(1,636 byte)</td>
</tr>
<tr>
<td>Low cost</td>
<td>4.4</td>
<td>- 76%</td>
</tr>
<tr>
<td></td>
<td>(sbox &amp; mixcol)</td>
<td></td>
</tr>
<tr>
<td>Area/perf. trade-off</td>
<td>5.6</td>
<td>- 79%</td>
</tr>
<tr>
<td></td>
<td>(sbox &amp; mixcol4)</td>
<td></td>
</tr>
<tr>
<td>High performance</td>
<td>9.9</td>
<td>- 48%</td>
</tr>
<tr>
<td></td>
<td>(sbox4s &amp; mixcol4s)</td>
<td></td>
</tr>
<tr>
<td>T lookup</td>
<td>1.5</td>
<td>+ 231%</td>
</tr>
<tr>
<td>4 KB table</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Note on Side-Channel Attacks

• Not main focus of this work
• Instructions for S-box remove data-dependent table lookups
  • No cache-timing attacks possible
• Increased parallelism
  • Can increase power analysis resistance
• Masking countermeasures supported partly
  • MixColumns, masked S-box (re)computation
• Other countermeasures remain applicable
  • e.g. randomization, secure logic styles
Conclusions

• Instruction set extensions for AES for 32-bit processors
  • Highly flexible
  • Easily implementable
  • Short critical path

• Different area/performance trade-offs possible
  • Minimal to moderate increase in area (+ 4% to + 23% of IU)
  • Speedup to a factor of nearly 10 achievable
  • Significant reduction of code size (- 84% possible)

• Interesting design option towards "zero-overhead" symmetric cryptography on embedded systems