A high-level model of embedded flash energy consumption

James Pallister, PhD Student
Kerstin Eder, Primary PhD Supervisor
Simon Hollis, PhD Supervisor
Jeremy Bennett, Embecosm

To appear in CASES, ESWEEK 2014
Outline

Embedded flash memory

Effect on energy

Modelling

Modelling instruction streams

Optimisation
Outline

Hardware

Embedded flash memory

Effect on energy

Modelling

Modelling instruction streams

Optimisation

Software
What is embedded flash memory?

Not what we typically think of

- Similar to flash drives/SSDs

Flash memory that is on the same die as the processor
Deeply embedded SoCs
Deeply embedded SoCs

- Small, cache-less, SoCs
Deeply embedded SoCs

- Small, cache-less, SoCs
- Simple processors
Deeply embedded SoCs

- Small, cache-less, SoCs
- Simple processors
- Typically flash and RAM
Deeply embedded SoCs

- Small, cache-less, SoCs
- Simple processors
- Typically flash and RAM
  - Single cycle access to both
Deeply embedded SoCs

- Small, cache-less, SoCs
- Simple processors
- Typically flash and RAM
  - Single cycle access to both

- Code executed directly out of flash
Deeply embedded SoCs

- Small, cache-less, SoCs
- Simple processors
- Typically flash and RAM
  - Single cycle access to both
- Code executed directly out of flash
- Low energy requirements
Deeply embedded SoCs

- Small, cache-less, SoCs
- Simple processors
- Typically flash and RAM
  - Single cycle access to both

- Code executed directly out of flash
- Low energy requirements
- Position of code in flash affects the energy?
Flash memory
Flash memory

Address is fed into the decoder

Address bus

Bus to CPU
Flash memory

Decode the address, select the block
Flash memory

Assert the control gate on the desired word-line
Flash memory

Connect the flash cell to the bit-line with the select gates
Flash memory

The precharged bit-lines are pulled up or down, depending on the charge in the flash cell.
Flash memory

Sense amplifiers
Detect the change and buffer the output onto the data bus
Expected energy effect

Pages, blocks, word-lines, bit-lines

- Changing each of these has an energy cost
Expected energy effect
Expected energy effect

\[ \text{n-bit length instructions} \]

```
loop: nop
nop
subs r0, #1
bne loop
```
Expected energy effect

```
loop: nop
nop
subs r0, #1
bne loop
```
Expected energy effect

n-bit length instructions

Straddling two blocks.

loop: nop
nop
subs r0, #1

bne loop
Expected energy effect

n-bit length instructions

Straddling two blocks.

Each iteration must repeatedly power up one, then the other.
Expected energy effect

n-bit length instructions

Straddling two blocks.
Each iteration must repeatedly power up one, then the other
Higher energy cost when an instruction jumps from one 'region' to another.
Expected energy effect

n-bit length instructions

Straddling two blocks.

Each iteration must repeatedly power up one, then the other

Higher energy cost when an instruction jumps from one 'region' to another.
Actual results

Loop offset, \( o \), (bytes)

Bottom line (blue) → 8-byte loop

Top line (green) → 10-byte loop
Actual results

Bottom line (blue) → 8-byte loop    Top line (green) → 10-byte loop
Actual results

Loop offset, $o$, (bytes)

Bottom line (blue) → 8-byte loop

Top line (green) → 10-byte loop
Actual results

Bottom line (blue) → 8-byte loop       Top line (green) → 10-byte loop
Modelling

Each consecutive memory access has an address dependent energy consumption.

E.g. 0 → 2
Modelling

Each consecutive memory access has an address dependent energy consumption.

E.g. 0 → 2
Modelling

Each consecutive memory access has an address dependent energy consumption.

E.g. 0 → 2
Modelling

Each consecutive memory access has an address dependent energy consumption.

E.g. $0 \rightarrow 2$

![Diagram showing memory access energy consumption](image)
Modelling

Each consecutive memory access has an address dependent energy consumption.

E.g. $0 \rightarrow 2$

$E_0$

2-byte
Modelling

Each consecutive memory access has an address dependent energy consumption.

E.g. 0 → 2

\[ E_0 + E_1 \]
Modelling

Each consecutive memory access has an address dependent energy consumption.

E.g. $0 \rightarrow 2$

$E_0 + E_1$

4-byte
Modelling

Each consecutive memory access has an address dependent energy consumption.

E.g. $0 \rightarrow 2 \quad E_0 + E_1$
Modelling

Each consecutive memory access has an address dependent energy consumption.

E.g. 2 → 4
Modelling

Each consecutive memory access has an address dependent energy consumption.

E.g. $2 \rightarrow 4$
Modelling

Each consecutive memory access has an address dependent energy consumption.

E.g. 2 → 4
Modelling

Each consecutive memory access has an address dependent energy consumption.

E.g. $2 \rightarrow 4$
Modelling

Each consecutive memory access has an address dependent energy consumption.

E.g. $2 \rightarrow 4$
Modelling

Each consecutive memory access has an address dependent energy consumption.

E.g. 2 → 4

\[ E_0 + E_1 \]
Modelling

Each consecutive memory access has an address dependent energy consumption.

E.g. $2 \rightarrow 4$  

$E_0 + E_1$
Modelling

Each consecutive memory access has an address dependent energy consumption.

E.g. $2 \rightarrow 4$

$E_0 + E_1 + E_2$
Modelling

Each consecutive memory access has an address dependent energy consumption.

E.g. $2 \rightarrow 4$

$E_0 + E_1 + E_2$
Modelling - formula

If a $2^k$-byte region is changed, all smaller regions $(2^{k-1}, \ldots)$ will also have changed
Modelling - formula

\[ i \rightarrow j = \sum_{k=0}^{N(i,j)} E_k \]

\[ N(i, j) = \left\lfloor \log_2 \left( i \oplus j \right) \right\rfloor \]
Modelling - formula

\[ 2 \rightarrow 4 = \sum_{k=0}^{N(2,4)} E_k \]

\[ N(i, j) = \left\lfloor \log_2 \left( i \oplus j \right) \right\rfloor \]
Modelling - formula

\[ 2 \rightarrow 4 = \sum_{k=0}^{N(2,4)} E_k \]

\[ N(2, 4) = 2 \]
Modelling - formula

\[ 2 \rightarrow 4 = E_0 + E_1 + E_2 \]
# Modelling – multiple instructions

<table>
<thead>
<tr>
<th>Addr</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>loop:</td>
</tr>
<tr>
<td>0</td>
<td>add r1, #1</td>
</tr>
<tr>
<td>2</td>
<td>ldr r0, [r3, #4]</td>
</tr>
<tr>
<td>4</td>
<td>cmp r1, r2</td>
</tr>
<tr>
<td>6</td>
<td>bne loop</td>
</tr>
</tbody>
</table>

Each instruction is 2 byte  
Each memory access is two bytes
## Modelling – multiple instructions

<table>
<thead>
<tr>
<th>Addr</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>loop:</td>
</tr>
<tr>
<td>0</td>
<td>add r1, #1</td>
</tr>
<tr>
<td>2</td>
<td>ldr r0, [r3, #4]</td>
</tr>
<tr>
<td>4</td>
<td>cmp r1, r2</td>
</tr>
<tr>
<td>6</td>
<td>bne loop</td>
</tr>
</tbody>
</table>

0 → 2
Modelling – multiple instructions

<table>
<thead>
<tr>
<th>Addr</th>
<th>Code</th>
<th>Addr</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>loop:</td>
<td>0</td>
<td>→ 2</td>
</tr>
<tr>
<td>0</td>
<td>add r1, #1</td>
<td>2</td>
<td>→ ?</td>
</tr>
<tr>
<td>2</td>
<td>ldr r0, [r3, #4]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>cmp r1, r2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>bne loop</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>.word 0x1234</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Modelling – multiple instructions

<table>
<thead>
<tr>
<th>Addr</th>
<th>Code</th>
<th>Address Mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>loop:</td>
<td>0 → 2</td>
</tr>
<tr>
<td>0</td>
<td>add r1, #1</td>
<td>2 → ?</td>
</tr>
<tr>
<td>2</td>
<td>ldr r0, [r3, #4]</td>
<td>? → 4</td>
</tr>
<tr>
<td>4</td>
<td>cmp r1, r2</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>bne loop</td>
<td></td>
</tr>
<tr>
<td>?</td>
<td>.word 0x1234</td>
<td></td>
</tr>
</tbody>
</table>
## Modelling – multiple instructions

<table>
<thead>
<tr>
<th>Addr</th>
<th>Code</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>loop:</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>add r1, #1</td>
<td>0 → 2</td>
</tr>
<tr>
<td>2</td>
<td>ldr r0, [r3, #4]</td>
<td>2 → ?</td>
</tr>
<tr>
<td>4</td>
<td>cmp r1, r2</td>
<td>? → 4</td>
</tr>
<tr>
<td>6</td>
<td>bne loop</td>
<td>4 → 6</td>
</tr>
</tbody>
</table>
### Modelling – multiple instructions

<table>
<thead>
<tr>
<th>Addr</th>
<th>Code</th>
<th>0 → 2</th>
<th>2 → ?</th>
<th>? → 4</th>
<th>4 → 6</th>
<th>6 → 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td><code>loop:</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td><code>add r1, #1</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td><code>ldr r0, [r3, #4]</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td><code>cmp r1, r2</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td><code>bne loop</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td><code>prefetched</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Modelling – multiple instructions

<table>
<thead>
<tr>
<th>Addr</th>
<th>Code</th>
<th>Addr</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>loop:</td>
<td>0</td>
<td>→ 2</td>
</tr>
<tr>
<td>0</td>
<td>add r1, #1</td>
<td>2</td>
<td>2 → ?</td>
</tr>
<tr>
<td>2</td>
<td>ldr r0, [r3, #4]</td>
<td>?</td>
<td>? → 4</td>
</tr>
<tr>
<td>4</td>
<td>cmp r1, r2</td>
<td>4</td>
<td>4 → 6</td>
</tr>
<tr>
<td>6</td>
<td>bne loop</td>
<td>6</td>
<td>6 → 8</td>
</tr>
<tr>
<td>8</td>
<td>prefetched</td>
<td>8</td>
<td>8 → 0</td>
</tr>
</tbody>
</table>
Modelling – multiple instructions

<table>
<thead>
<tr>
<th>Addr</th>
<th>Code</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>loop:</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>add r1, #1</td>
<td>0 → 2</td>
</tr>
<tr>
<td>2</td>
<td>ldr r0, [r3, #4]</td>
<td>2 → ?</td>
</tr>
<tr>
<td>4</td>
<td>cmp r1, r2</td>
<td>? → 4</td>
</tr>
<tr>
<td>6</td>
<td>bne loop</td>
<td>4 → 6</td>
</tr>
<tr>
<td>8</td>
<td>prefetched</td>
<td>6 → 8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8 → 0</td>
</tr>
</tbody>
</table>

\[4E_0 + 4E_1 + 2E_2 + 2E_3 + ?\]
Approximating

Ignore the data accesses
- Mostly to RAM
- Infrequently to flash (if so, usually one time loads)

Almost allows us to statically analyse
code for flash energy consumption

Conditional branches still unknown

\[ 5E_0 + 5E_1 + 3E_2 + 2E_3 \]
## Parameters

<table>
<thead>
<tr>
<th>SoC</th>
<th>$E_2$ (A)</th>
<th>$E_3$</th>
<th>$E_4$ (B)</th>
<th>$E_5$</th>
<th>$E_6$</th>
<th>$E_7$ (C)</th>
<th>$E_8$ (C)</th>
</tr>
</thead>
<tbody>
<tr>
<td>STM32F0</td>
<td>300</td>
<td>27</td>
<td>6</td>
<td>0</td>
<td>9</td>
<td>100</td>
<td>6</td>
</tr>
<tr>
<td>STM32F1</td>
<td>500</td>
<td>0</td>
<td>6</td>
<td>34</td>
<td>4</td>
<td>10</td>
<td>190</td>
</tr>
<tr>
<td>ATMEGA328P</td>
<td>0</td>
<td>22</td>
<td>36</td>
<td>27</td>
<td>9</td>
<td>107</td>
<td>24</td>
</tr>
<tr>
<td>PIC32MX250F128B</td>
<td>225</td>
<td>0</td>
<td>10</td>
<td>18</td>
<td>8</td>
<td>13</td>
<td>113</td>
</tr>
<tr>
<td>MSP430F5529</td>
<td>408</td>
<td>0</td>
<td>34</td>
<td>26</td>
<td>15</td>
<td>13</td>
<td>13</td>
</tr>
</tbody>
</table>
Cross validation
A potential optimisation

For code which is executed frequently, ensure it crosses as few costly boundaries as possible.

Use the model to make these decisions.
A potential optimisation

For code which is executed frequently, ensure it crosses as few costly boundaries as possible.

Use the model to make these decisions.
A potential optimisation

For code which is executed frequently, ensure it crosses as few costly boundaries as possible.

Use the model to make these decisions.
Conclusion

5-15% energy reduction

Flexible model
- Validated on 5 SoCs
- Average error: 11%

Compiler optimisation to be implemented
- Analysis indicates 30-40% of all loops can be better aligned
- 4% increase in executable size
Thanks!

james.pallister@bristol.ac.uk
http://mageec.org