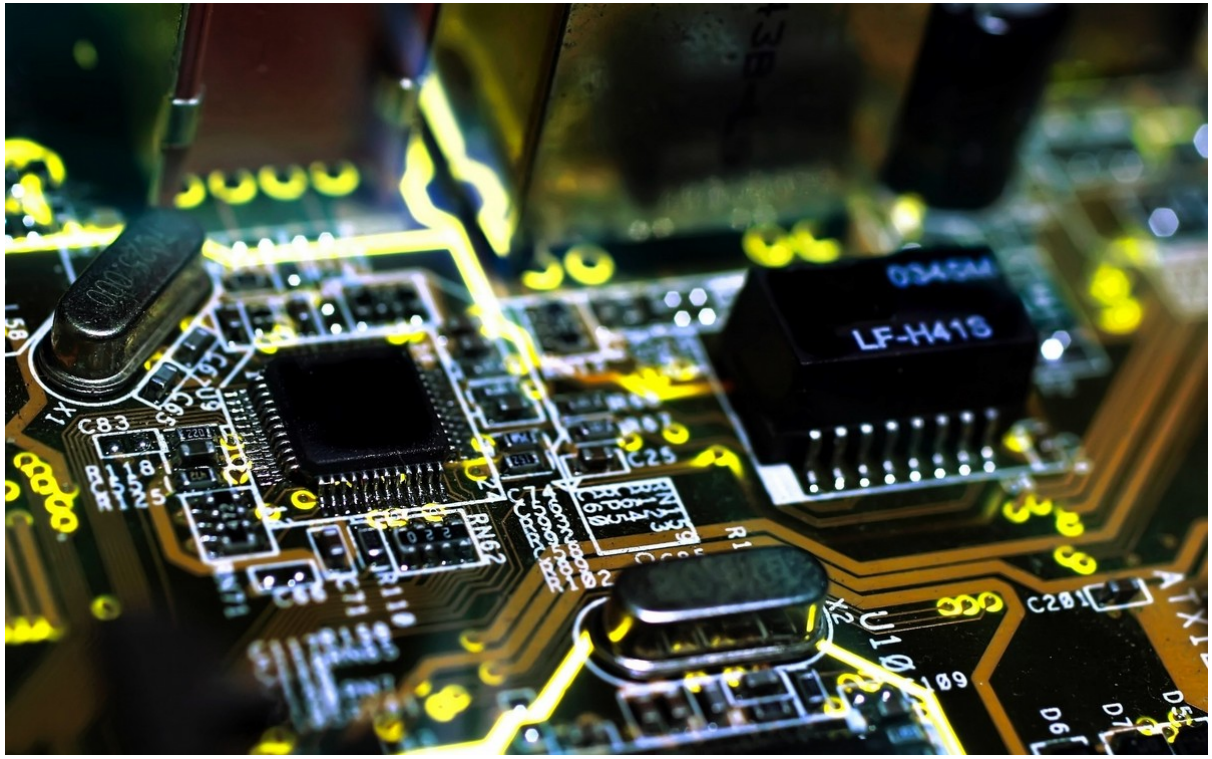
A high-level model of embedded flash energy consumption



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

Outline

Hardware

Software

Embedded flash memory

Effect on energy

Modelling

Modelling instruction streams

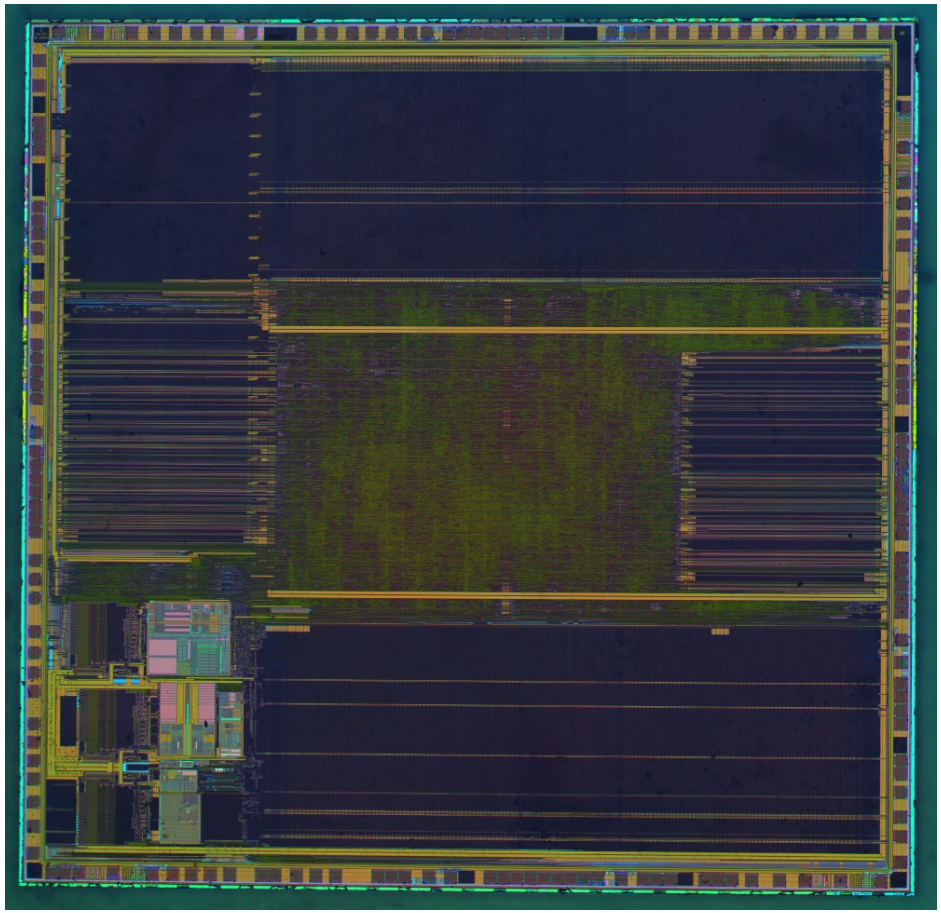
Optimisation

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

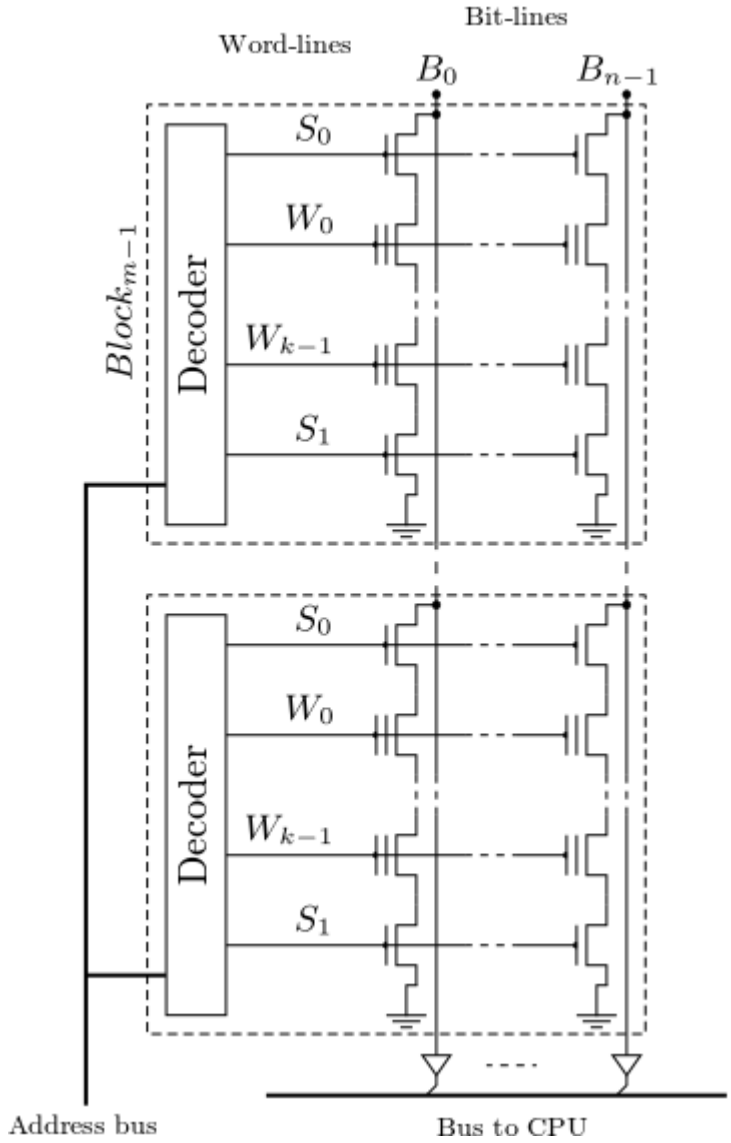


Deeply embedded SoCs

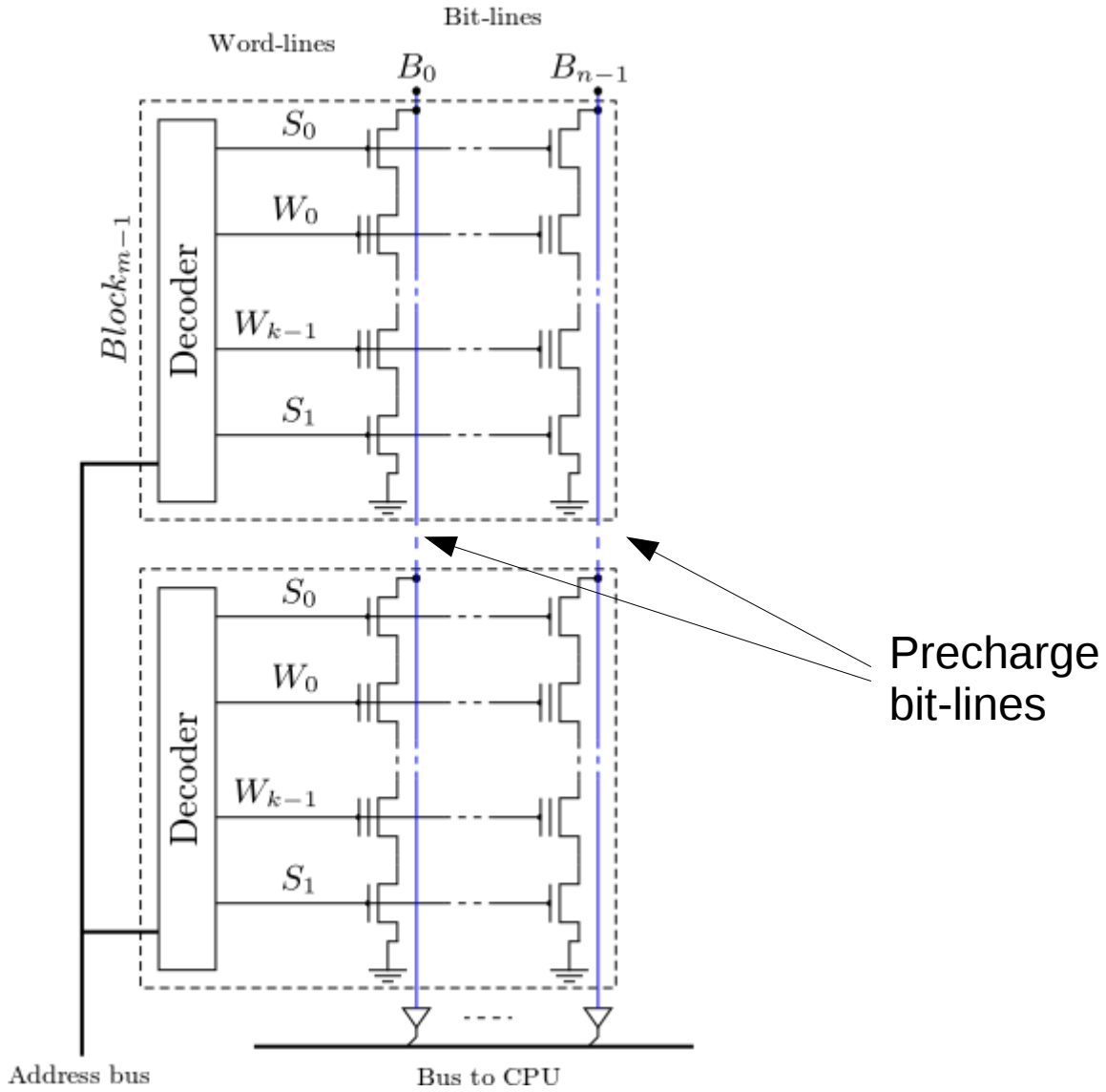
- Small, cache-less, SoCs
- Simple processors
- Typically flash and RAM
  - Single cycle access to both

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

Flash memory

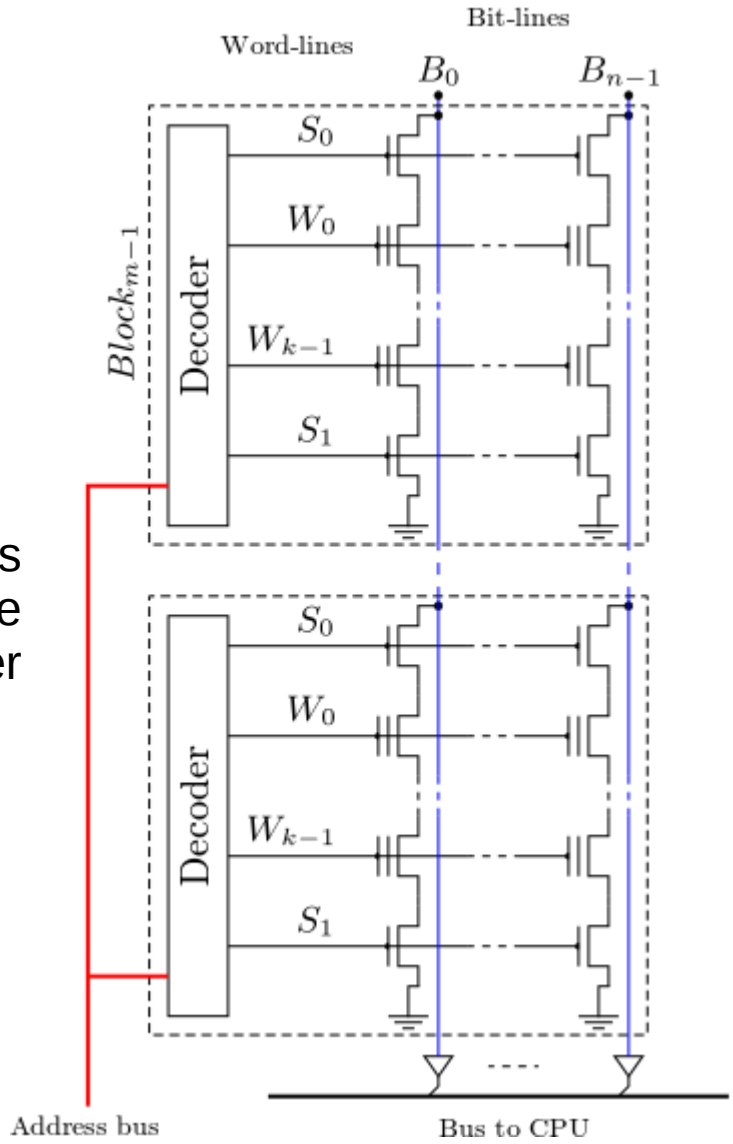


Flash memory



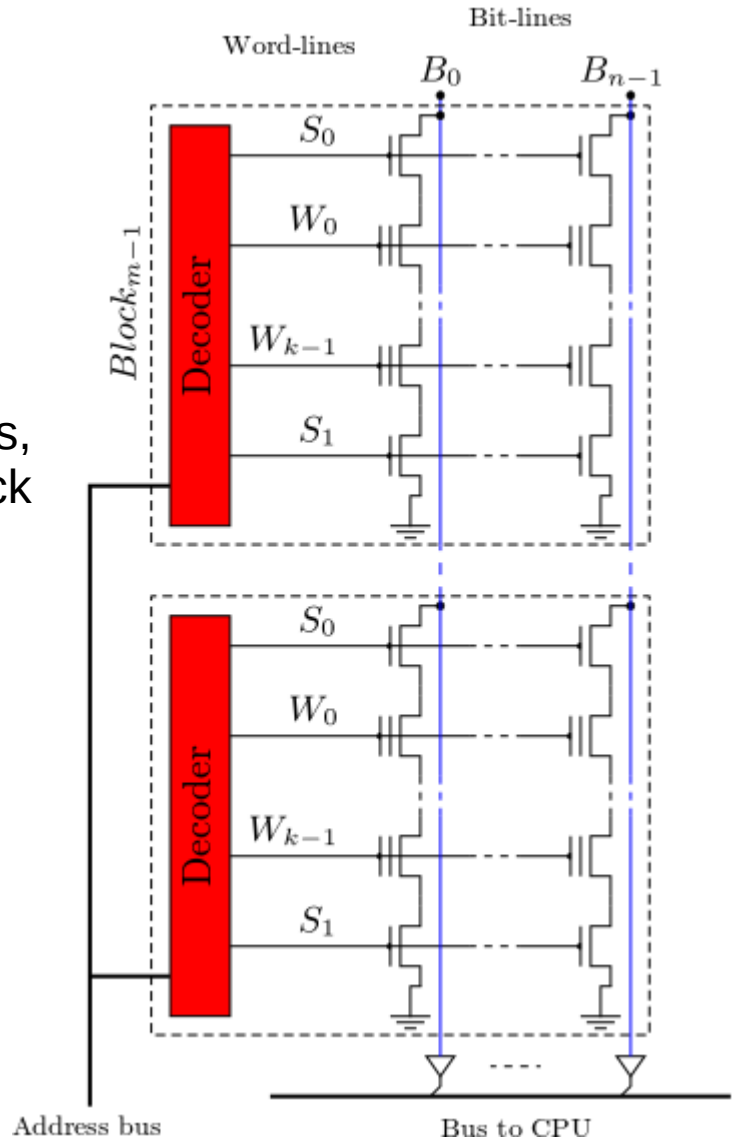
Flash memory

Address is fed into the decoder

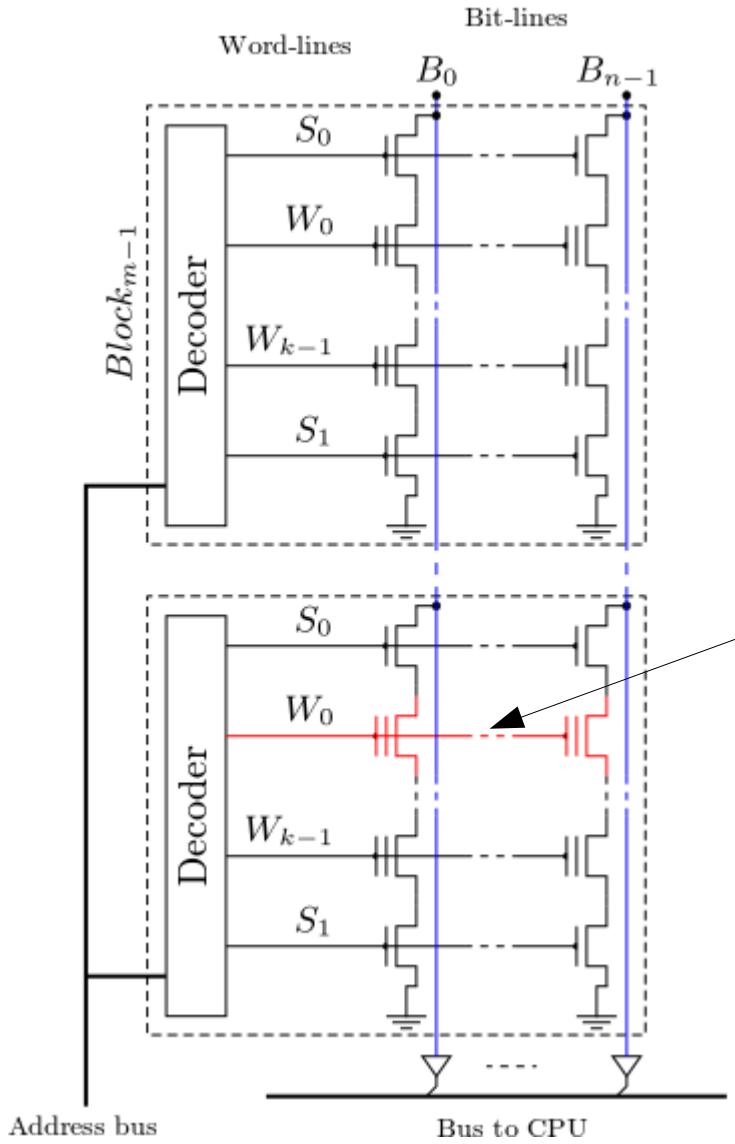


Flash memory

Decode the address,
select the block

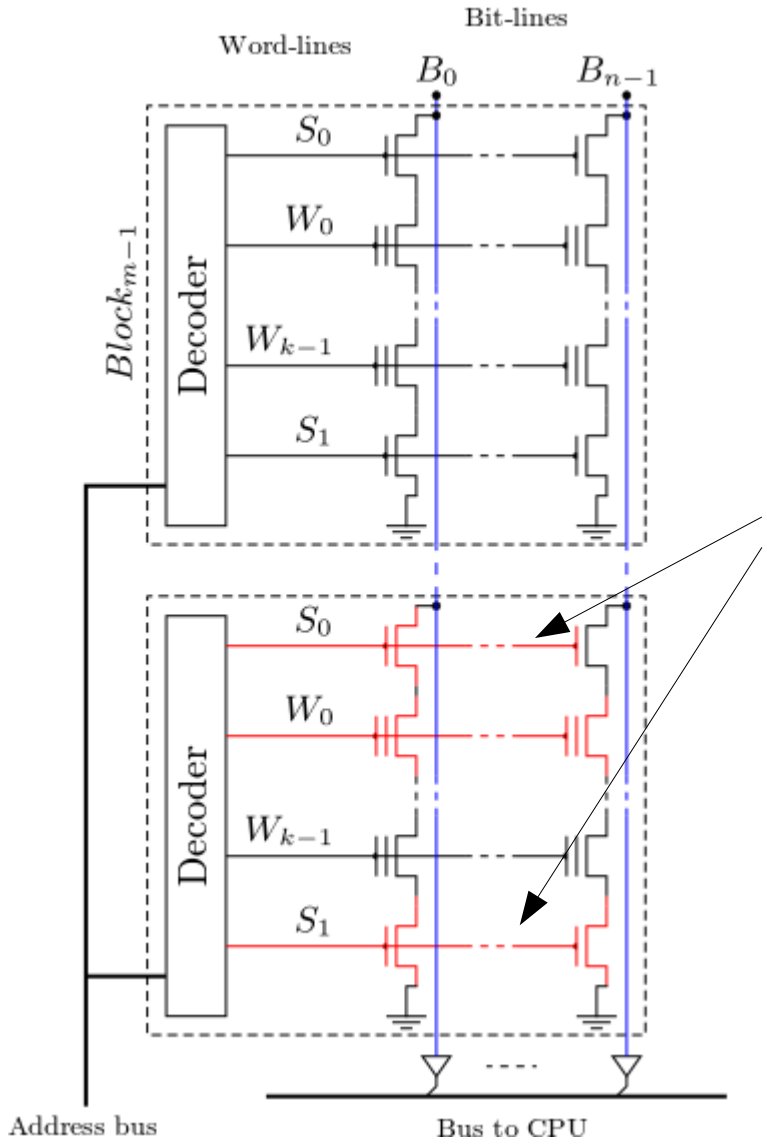


Flash memory



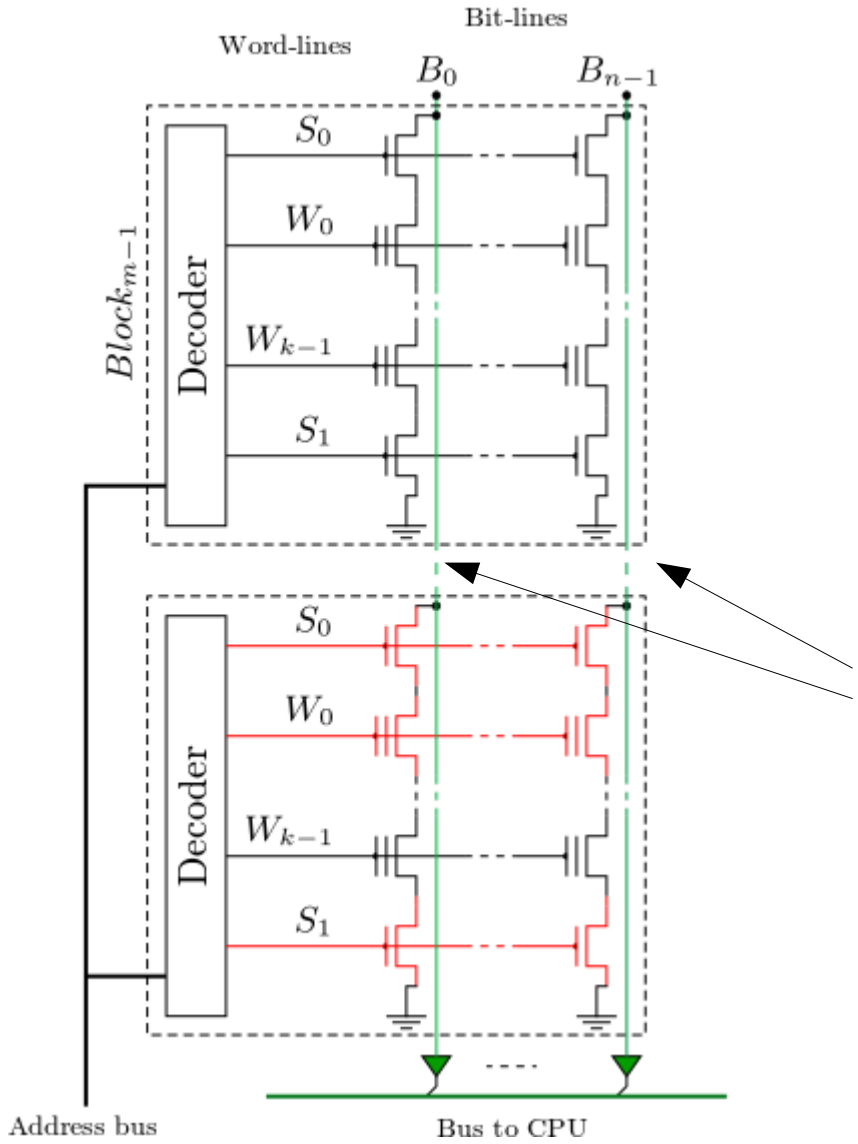
Assert the control gate on the desired word-line

Flash memory



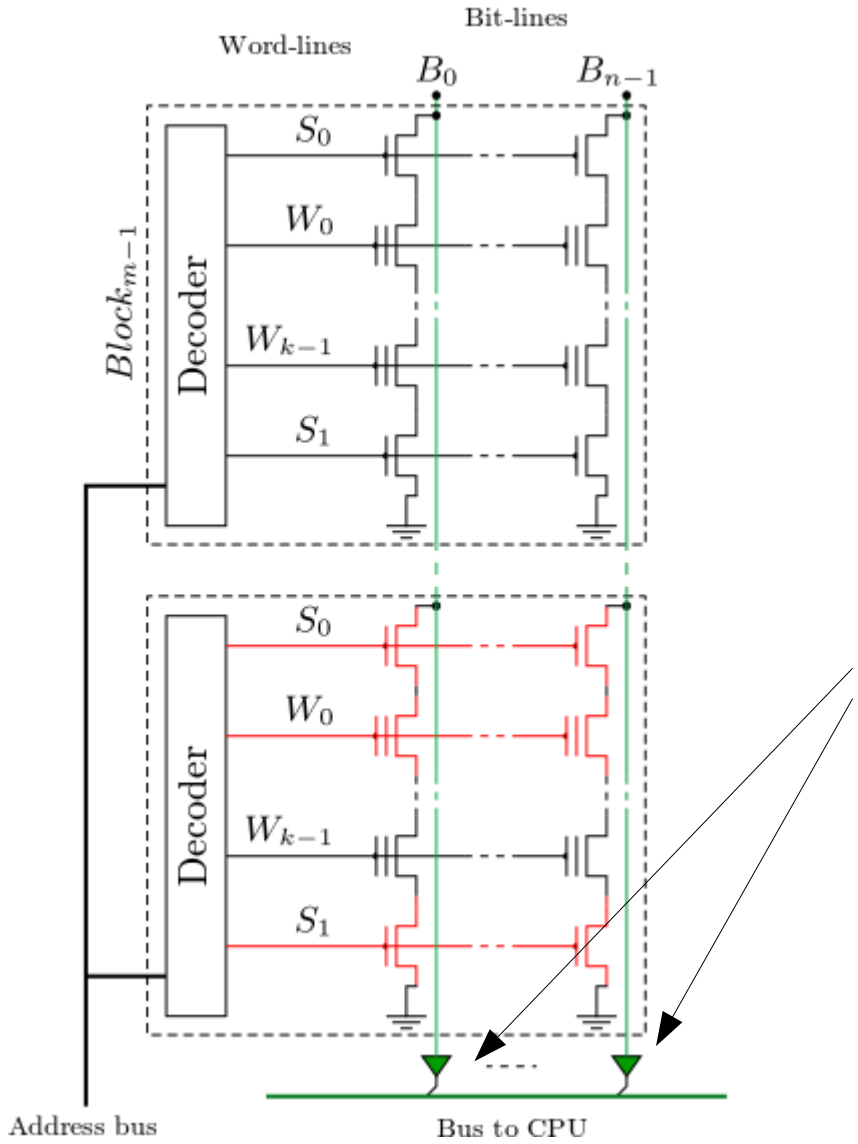
Connect the flash cell to the bit-line with the select gates

Flash memory



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

Flash memory



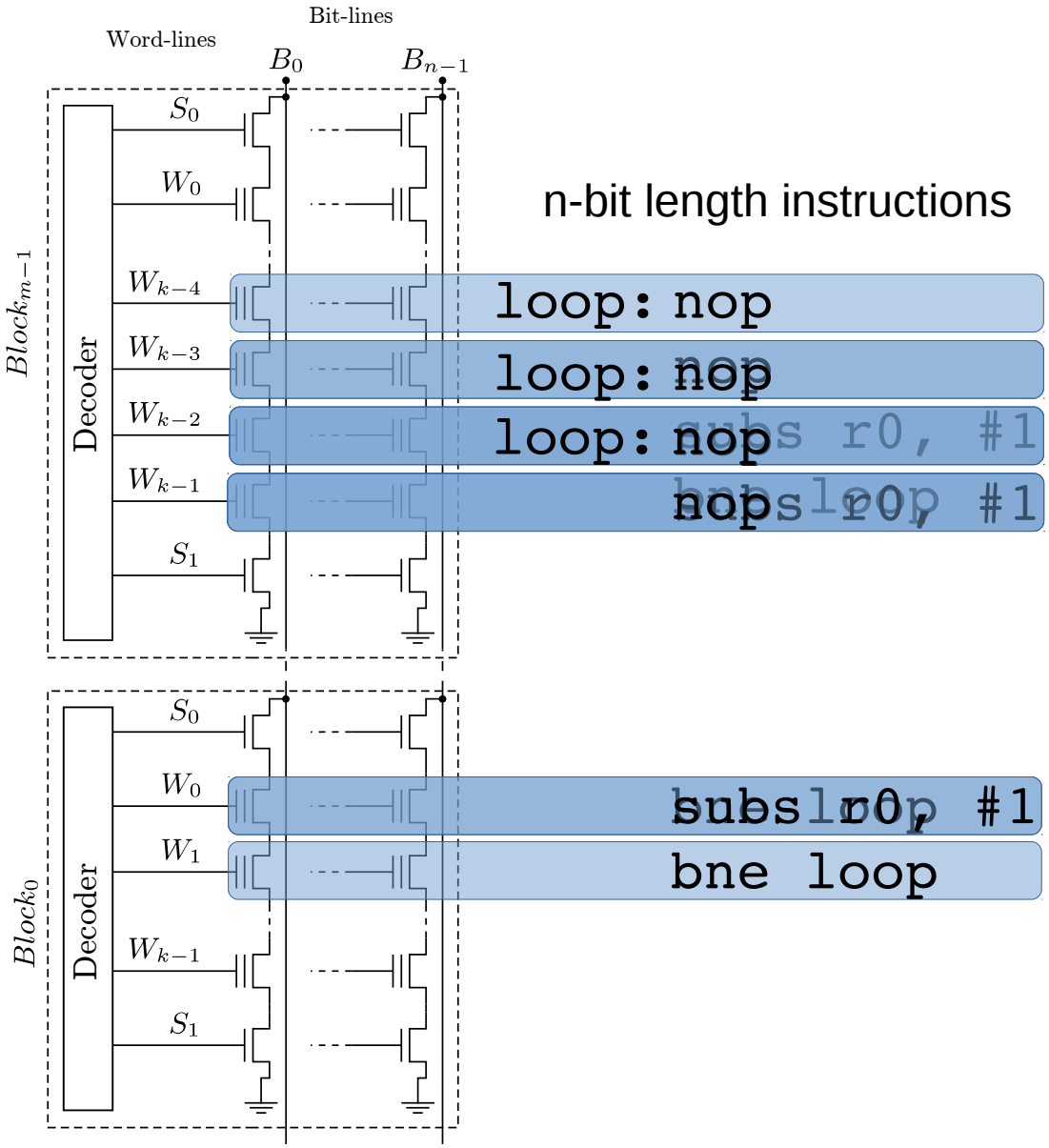
Sense amplifiers
Detect the change
and buffer the output
onto the data bus

Expected energy effect

Pages, blocks, word-lines, bit-lines

- Changing each of these has an energy cost

Expected energy effect



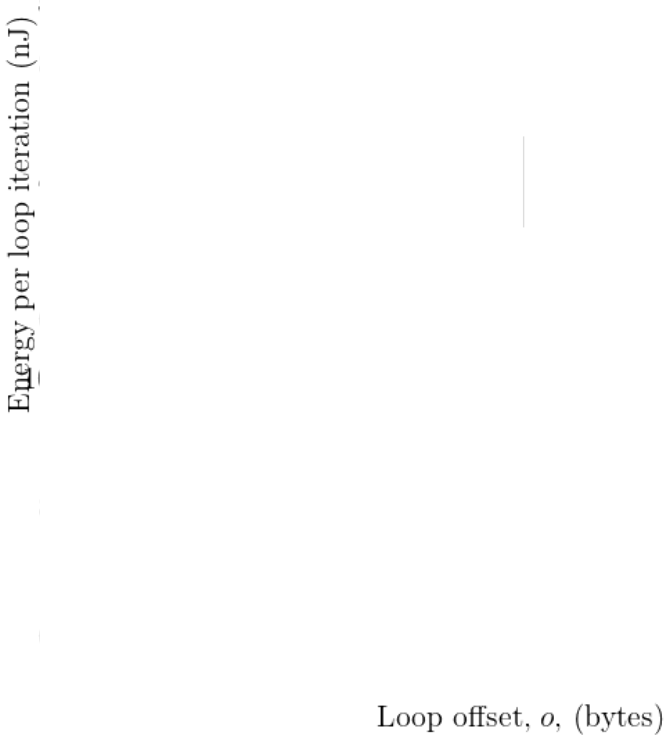
n-bit length instructions

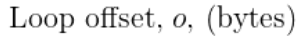
Straddling two blocks.

Each iteration must repeatedly power up one, then the other

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

Actual results





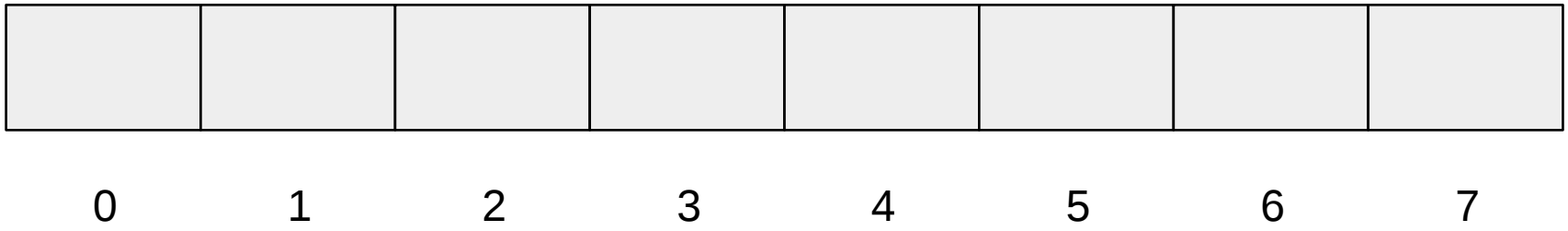
Bottom line (blue) → 8-byte loop

Top line (green) → 10-byte loop

Modelling

Each consecutive memory access has an address dependent energy consumption.

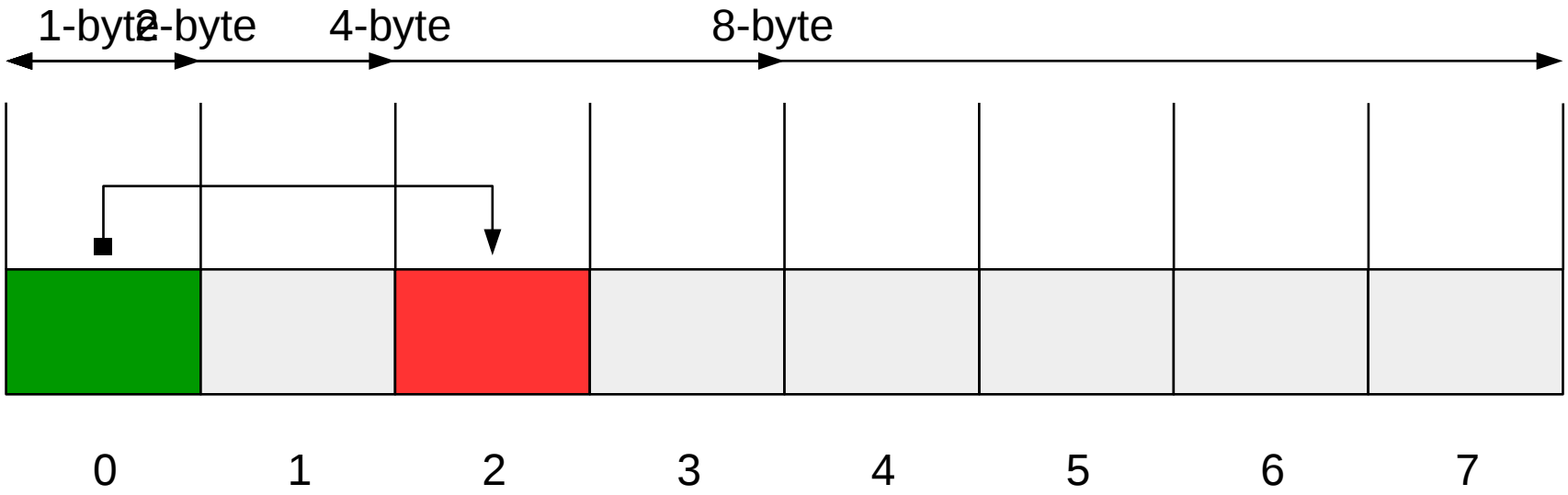
E.g. 0 → 2



Modelling

Each consecutive memory access has an address dependent energy consumption.

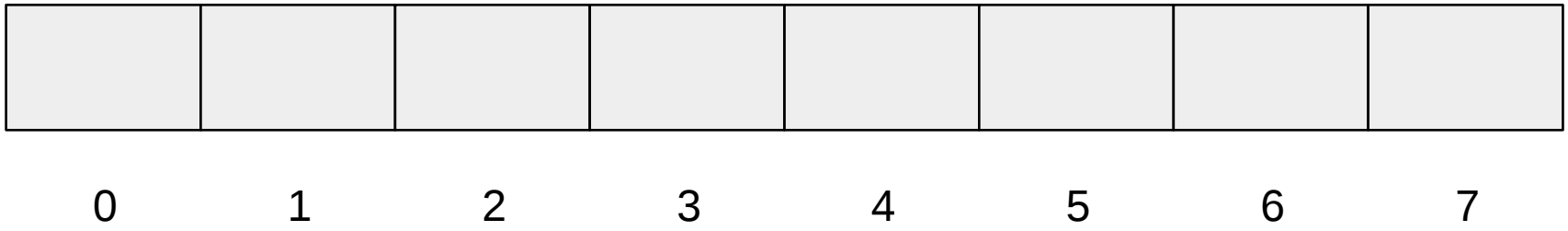
E.g. 0 → 2 $E_0 + E_1$



Modelling

Each consecutive memory access has an address dependent energy consumption.

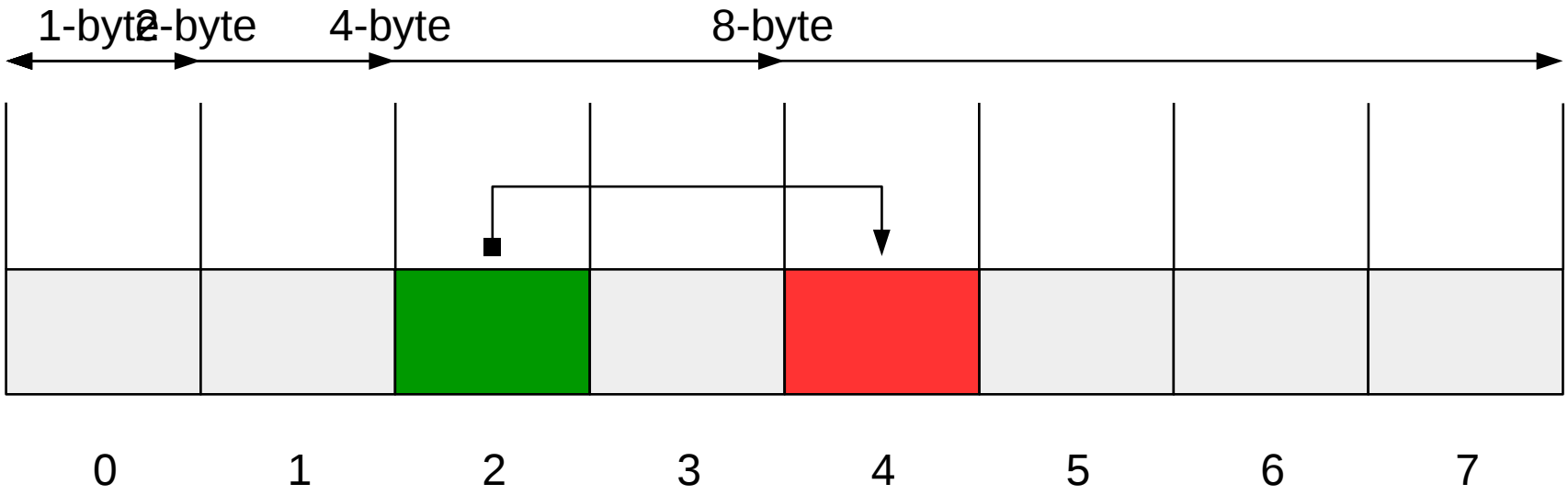
E.g. $2 \rightarrow 4$



Modelling

Each consecutive memory access has an address dependent energy consumption.

E.g. 2 → 4 $E_0 + E_1 + E_2$



Modelling - formula

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

Modelling - formula

$$i \to j = \sum_{k=0}^{N(i,j)} E_k$$

$$N(i,j) = \left\lfloor \log_2 \left(i \oplus j \right) \right\rfloor$$

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

$$2 \rightarrow 4 = E_0 + E_1 + E_2$$

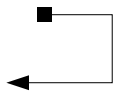
Modelling – multiple instructions

Addr	*Code*
0	`loop:`
0	`    add r1, #1`
2	`    ldr r0, [r3, #4]`
4	`    cmp r1, r2`
6	`    bne loop`

Each instruction is 2 byte
Each memory access is two bytes

Modelling – multiple instructions

Addr	*Code*
0	loop:
0	add r1, #1
2	ldr r0, [r3, #4]
4	cmp r1, r2
6	bne loop



0 → 2

Modelling – multiple instructions

Addr	*Code*
0	loop:
0	add r1, #1
2	ldr r0, [r3, #4]
4	cmp r1, r2
6	bne loop
?	.word 0x1234

0 → 2
2 → ?

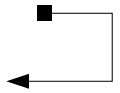
Modelling – multiple instructions

Addr	*Code*
0	loop:
0	add r1, #1
2	ldr r0, [r3, #4]
4	cmp r1, r2
6	bne loop
?	.word 0x1234

0 → 2
2 → ?
? → 4

Modelling – multiple instructions

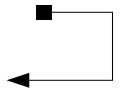
Addr	*Code*
0	loop:
0	add r1, #1
2	ldr r0, [r3, #4]
4	cmp r1, r2
6	bne loop



0 → 2
2 → ?
? → 4
4 → 6

Modelling – multiple instructions

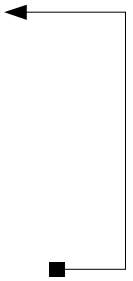
Addr	*Code*
0	loop:
0	add r1, #1
2	ldr r0, [r3, #4]
4	cmp r1, r2
6	bne loop
8	*prefetched*



0 → 2
2 → ?
? → 4
4 → 6
6 → 8

Modelling – multiple instructions

Addr	*Code*
0	loop:
0	add r1, #1
2	ldr r0, [r3, #4]
4	cmp r1, r2
6	bne loop
8	*prefetched*



0 → 2
2 → ?
? → 4
4 → 6
6 → 8
8 → 0

Modelling – multiple instructions

Addr	*Code*
0	loop:
0	add r1, #1
2	ldr r0, [r3, #4]
4	cmp r1, r2
6	bne loop
8	*prefetched*

0 → 2
2 → ?
? → 4
4 → 6
6 → 8
8 → 0

$$4E_0 + 4E_1 + 2E_2 + 2E_3 + ?$$

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

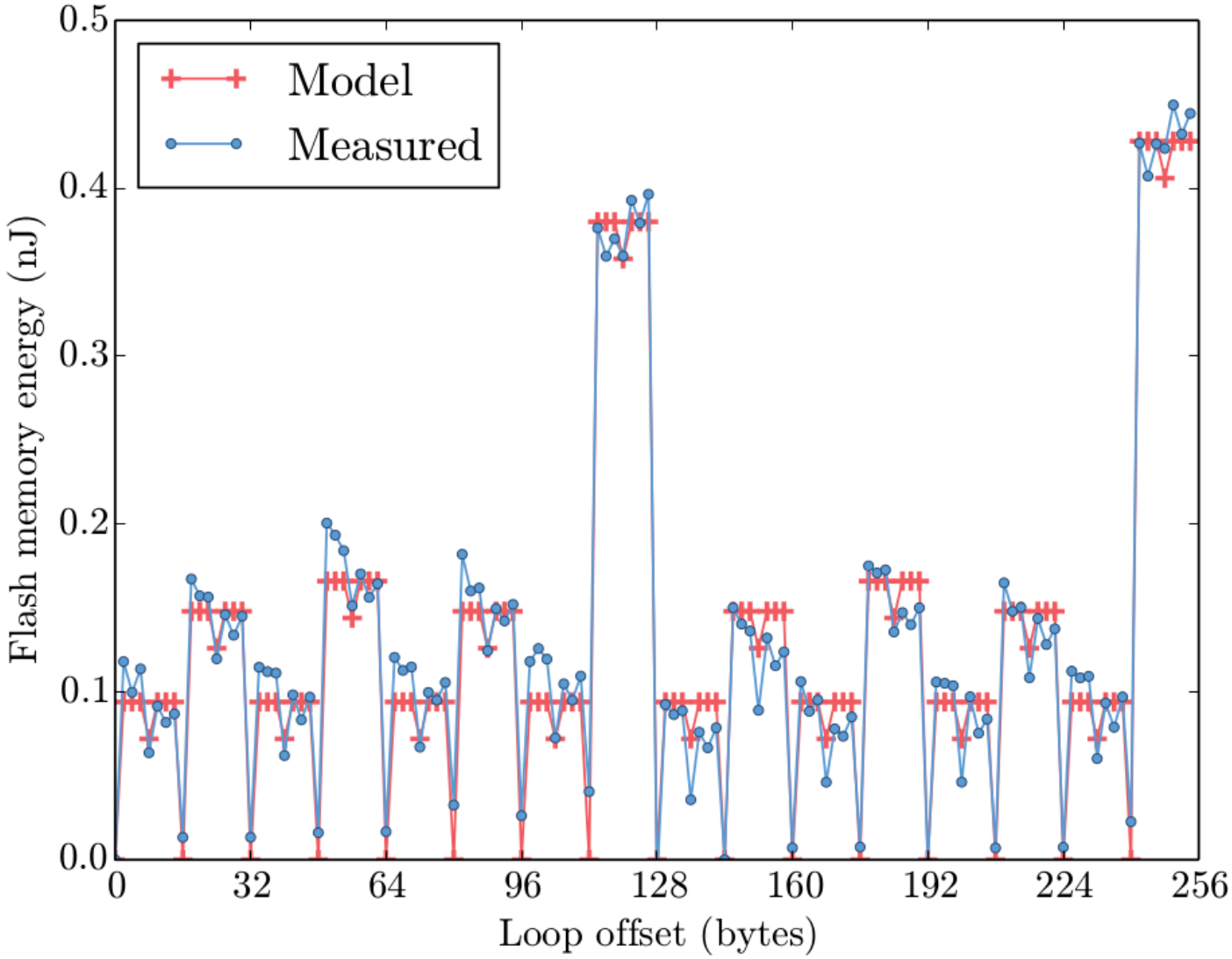
0 → 2
2 → 4
4 → 6
6 → 8
8 → 0

$$5E_0 + 5E_1 + 3E_2 + 2E_3$$

Parameters

SoC	Model parameters (pJ)						
	E_2 (**A**)	E_3	E_4 (**B**)	E_5	E_6	E_7 (**C**)	E_8 (**C**)
STM32F0	300	27	6	0	9	100	6
STM32F1	500	0	6	34	4	10	190
ATMEGA328P	0	22	36	27	9	107	24
PIC32MX250F128B	225	0	10	18	8	13	113
MSP430F5529	408	0	34	26	15	13	13

Cross validation

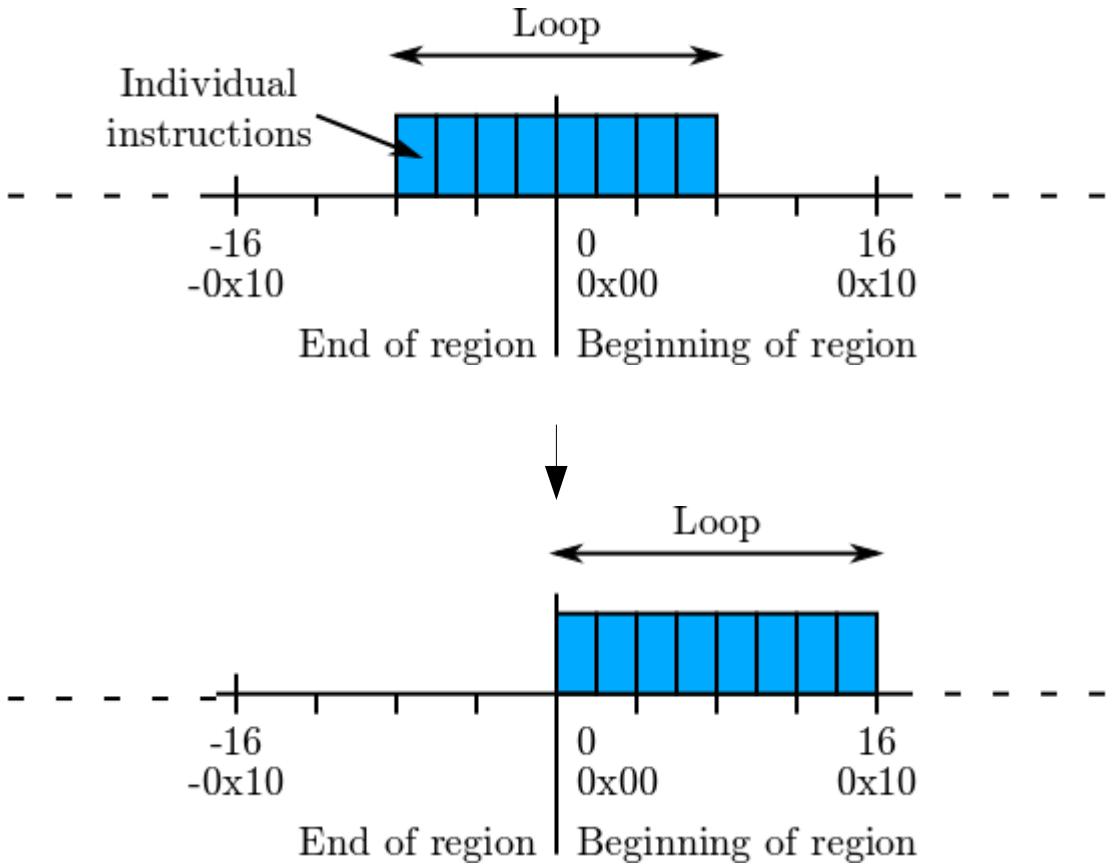


STM32F0

A potential optimisation

For code which is executed frequenty, 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://arxiv.org/abs/1404.1602

http://mageec.org