New Results in Generating True Random Numbers on Simple Microcontrollers

Josef Hlaváč, Simona Buchovecká, Róbert Lórencz

Faculty of Information Technology
Czech Technical University in Prague

Introduction & motivation

- Å Situation
  - ï Simple microcontrollers lack dedicated TRNG
  - ï Many embedded applications use some crypto
  - ï Bad RNG can kill security
  - ï In-field hardware upgrades are difficult
- Å Problem
  - ï Can we implement a reasonable TRNG in firmware only, with no added hardware?

Recap of our previous work

Å Microcontrollers are not *entirely* deterministic

Å The method:

ï Internal master RC oscillator (~ 8 MHz)

Åclocks a timer/counter (TIMER1)

ï External XTAL oscillator for RTC (32768 Hz)

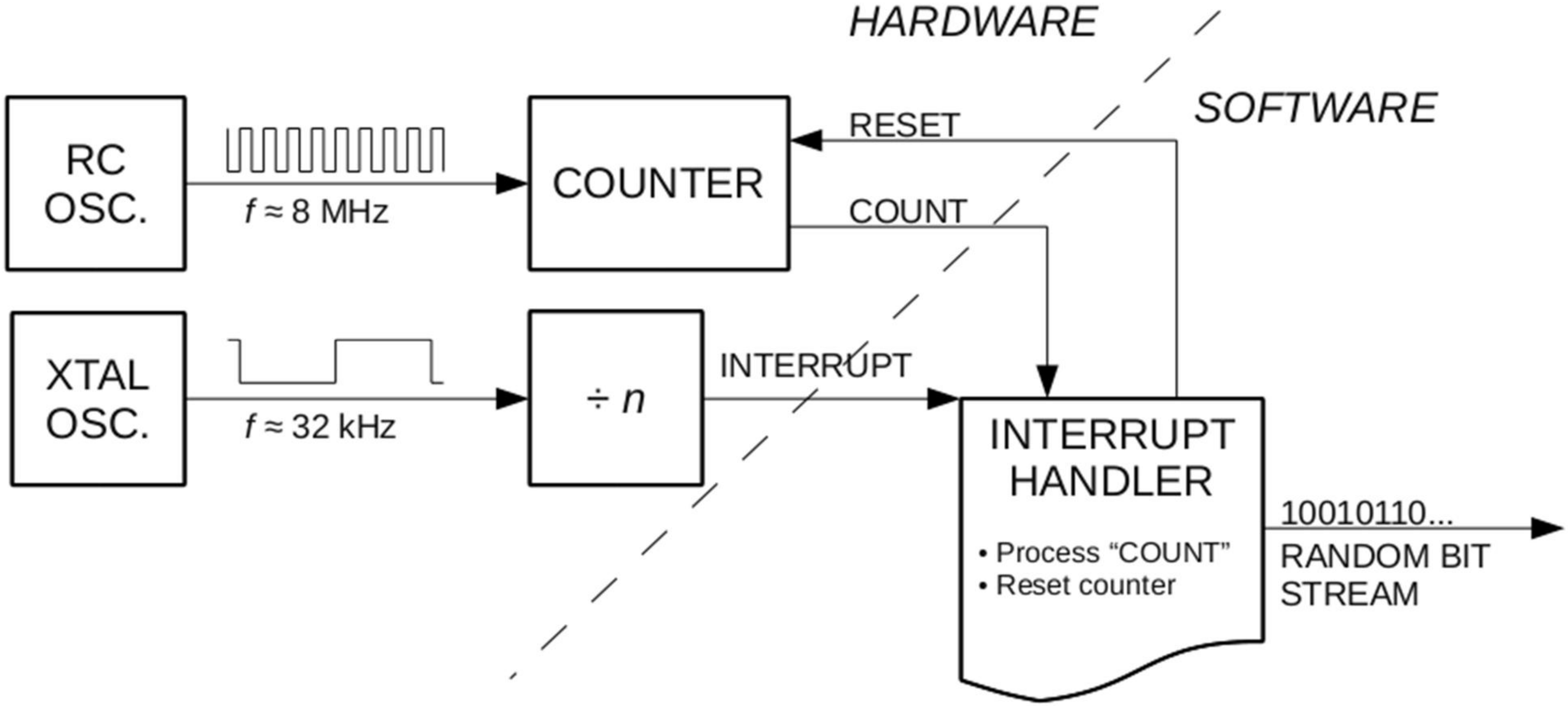
Åfeeds another counter (TIMER2)

Åalready included in many Atmel AVR applications

ï TIMER1 is sampled at intervals timed by TIMER2

Ålow-order bits are extracted into the random bitstream

Recap of our previous work



New tests

Å Parasitic frequencies

Å Usability with a simpler microcontroller

Å Influence of on-chip components

Parasitic frequencies

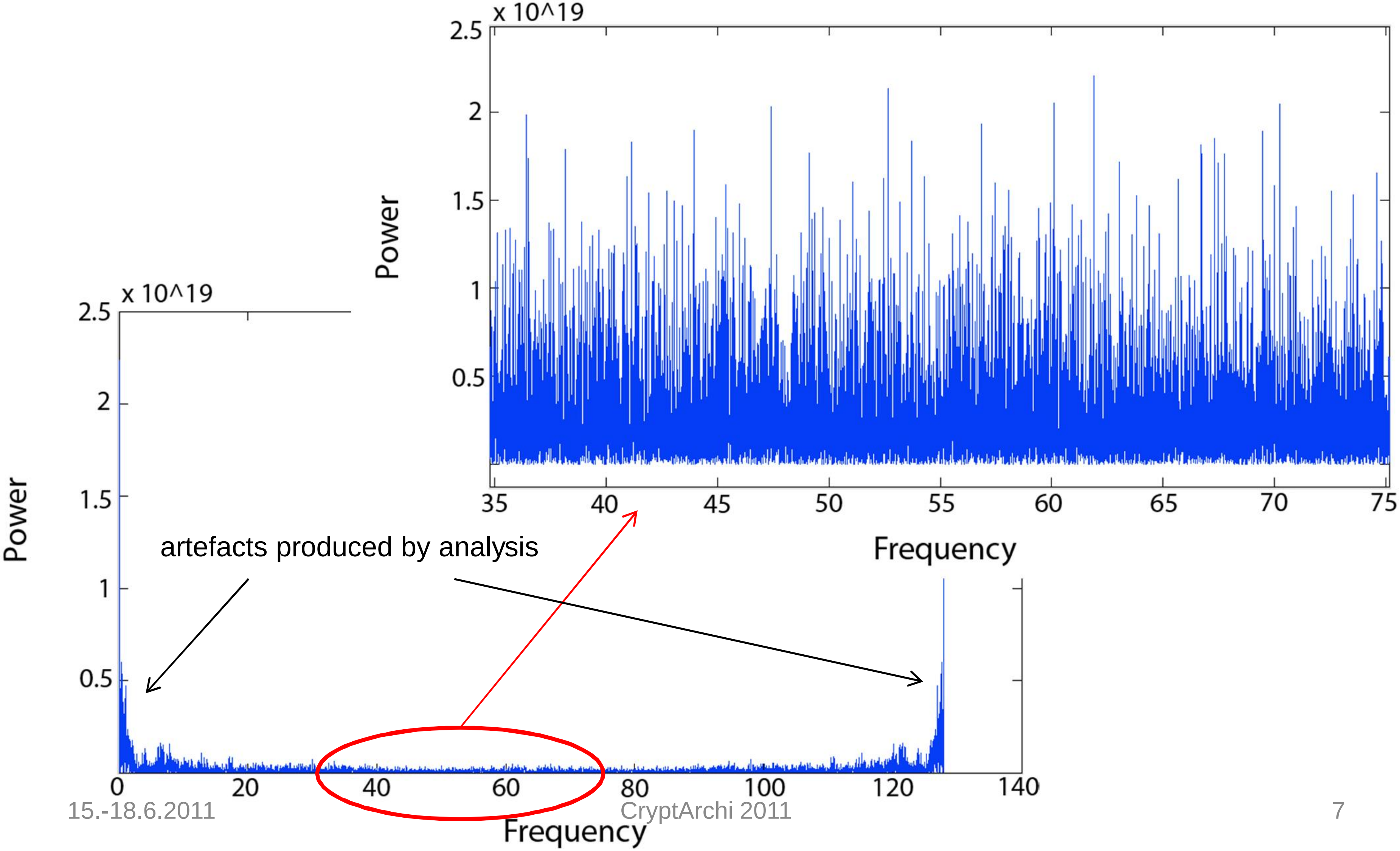
Å Reasoning:

ï The circuit might be picking up 50Hz (the strongest source of interference around), influencing the TRNG

Å Proposed test:

ï FFT: Let us look at the data as samples

Parasitic frequencies



artefacts produced by analysis

Parasitic frequencies

Å Result:

ï Parasitic frequency componets NOT found

Simpler microcontroller

Å All experiments so far with AVR Butterfly

ï demonstration board with ATmega169P

Å We created a simple module with ATmega8

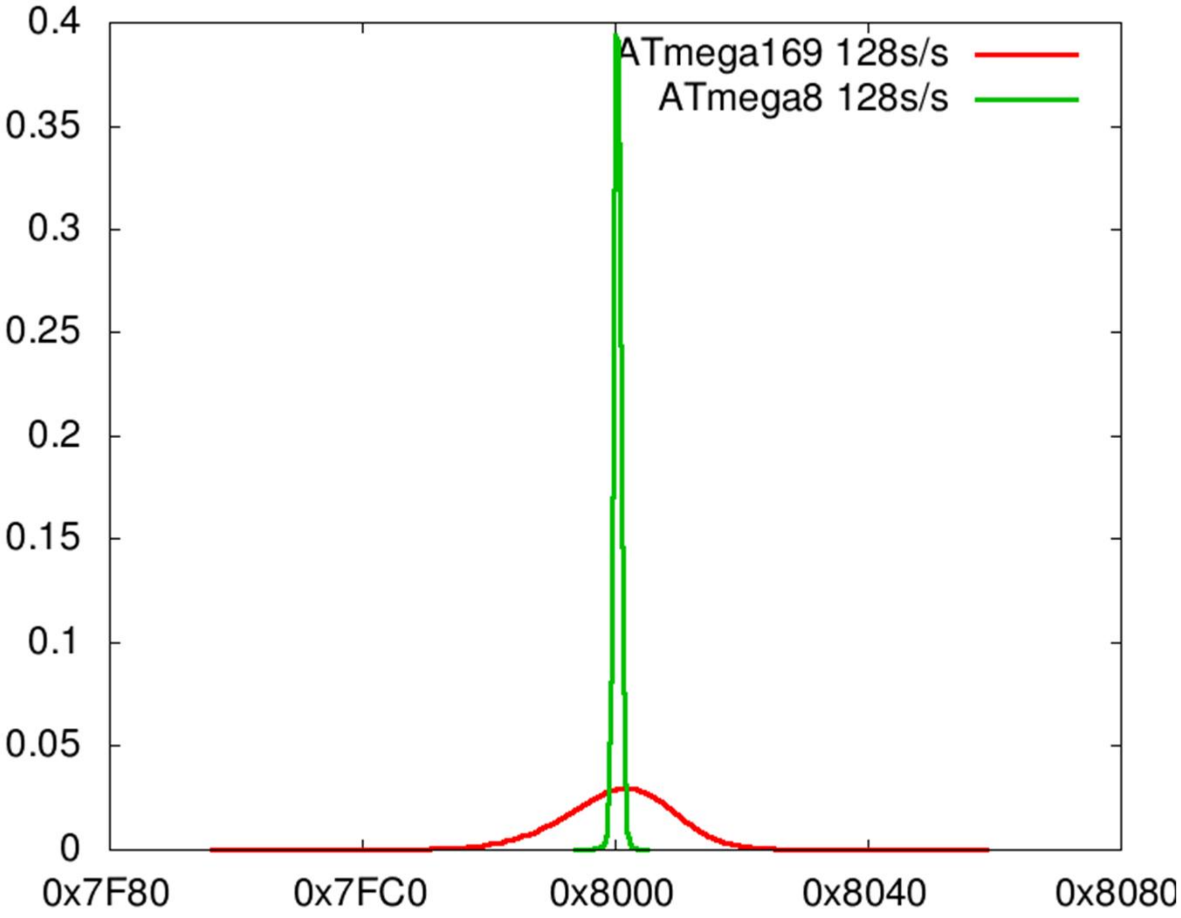
ï one of the simplest devices in the ATmega series

Å First results disappointing:

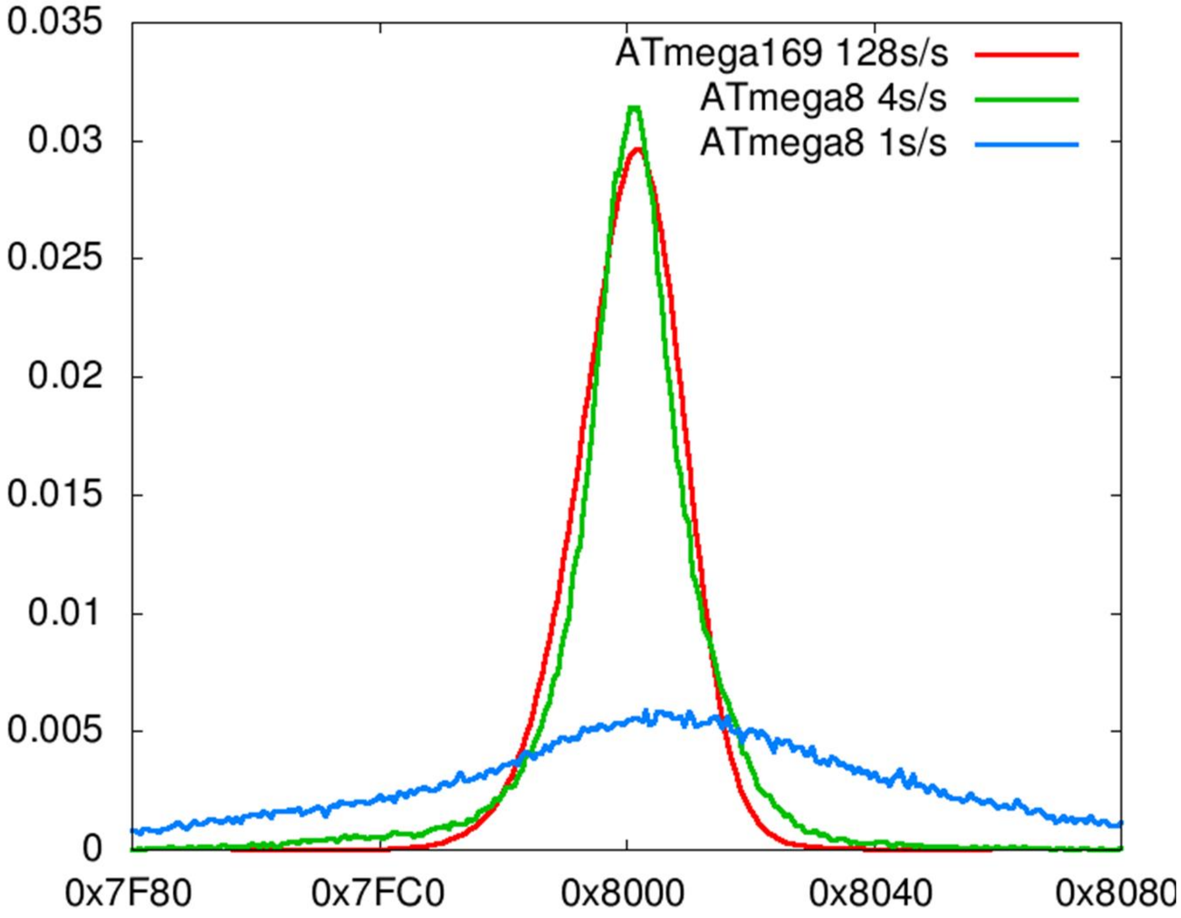
ï no usable randomness at 128 samples/s
(cf. ATmega169: at least 1 bit in each sample)

ï weird parts in some generated data
(xtal / communication problems?)

Simpler microcontroller

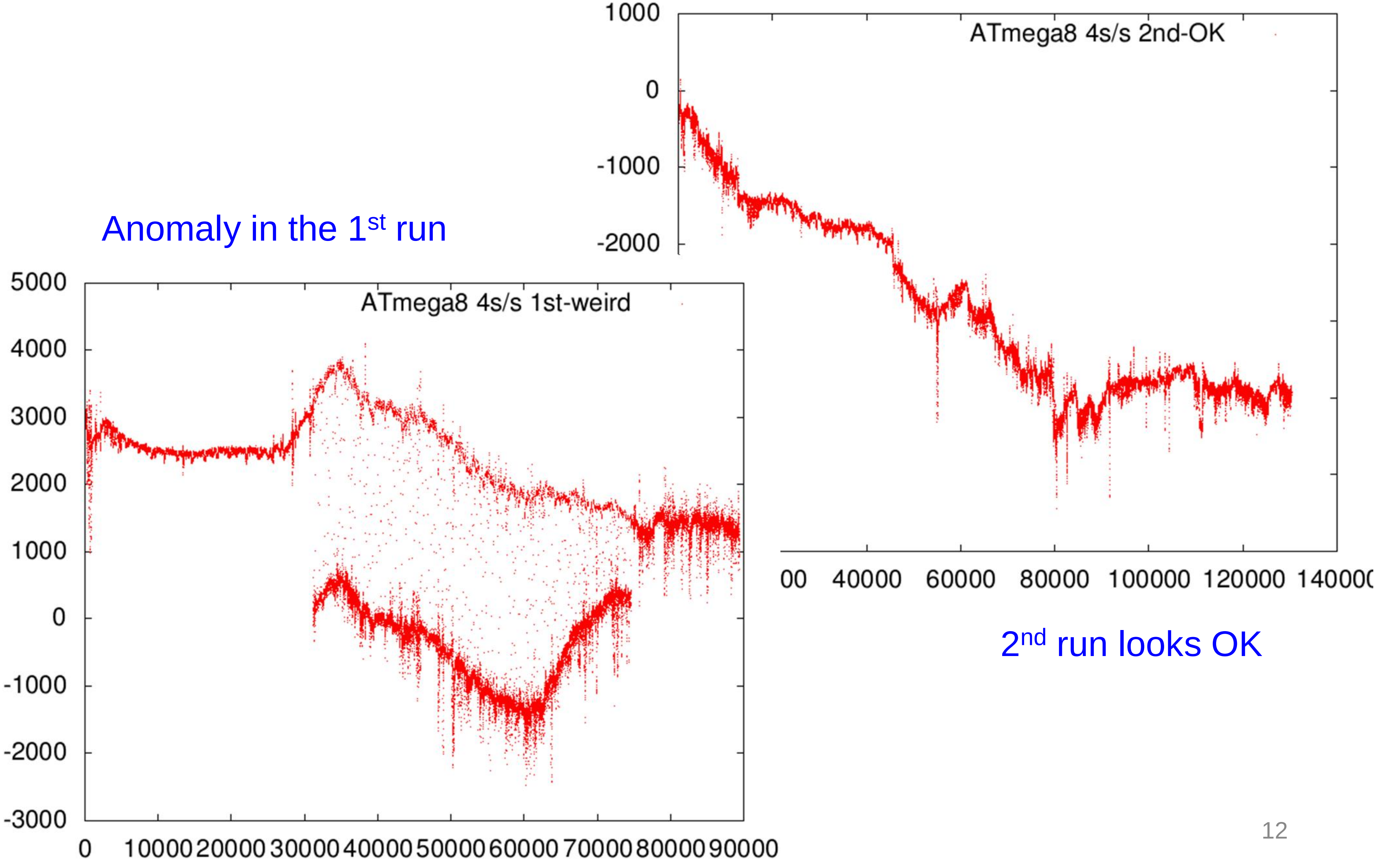


Simpler microcontroller



Simpler microcontroller

Anomaly in the 1st run



2nd run looks OK

Simpler microcontroller

Å Estimated entropy per sample (“ent”)

ï bits 1-8 from filtered samples

ATmega169 128 samp/s	ATmega8 128 samp/s	ATmega8 4 samples/s	ATmega8 1 sample/s
6.0	1.9	6.2	7.9

Å Summary

ï method still works; however,

ï significantly less entropy available

On-chip components

Å Q: How is the TRNG method influenced by (in)activity of various on-chip components?

Å We investigated (ATmega169P):

ï LCD driver

Å enabled / disabled / power down

ï SLEEP instruction in main program loop

Å idle mode / power down mode

ï USART

Å fast / slow transmit speed

On-chip components

Å In the following histograms:

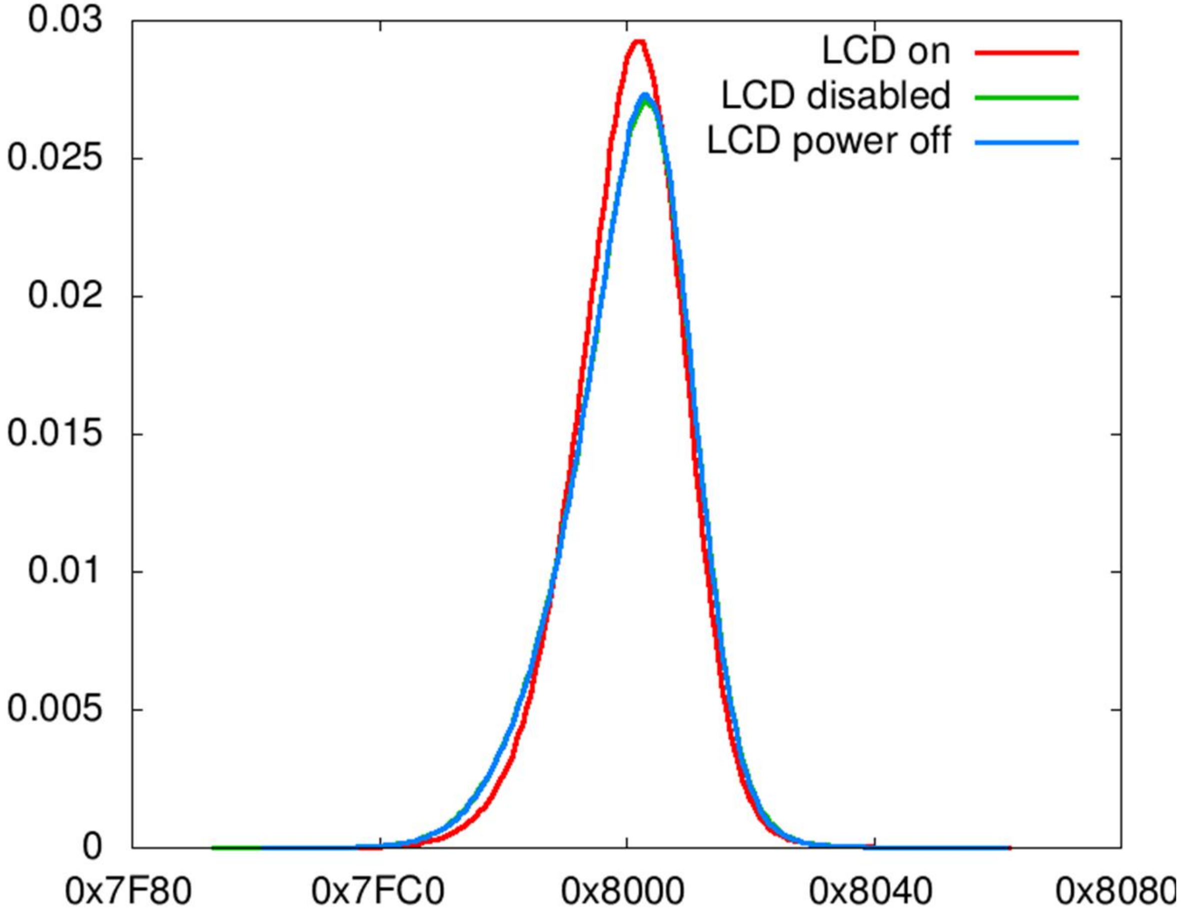
ï COUNTs plotted for various situations

ï All values are shifted >> 1
(to get rid of program-dependent LSB)

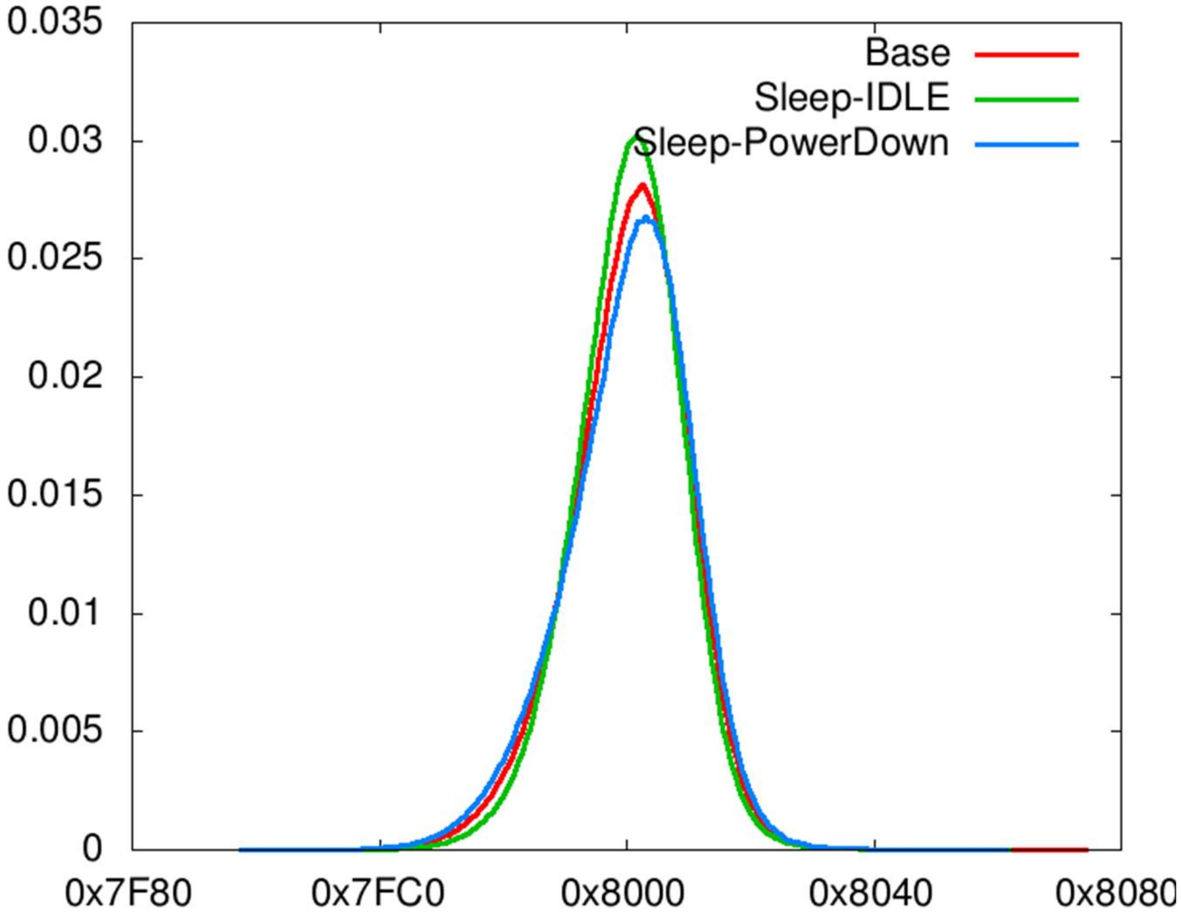
ï Values are filtered (subtracted moving average)
to eliminate the influence of slowly-changing
environment parameters, centered around 0x8000

ï Wider and lower curve is better – more random

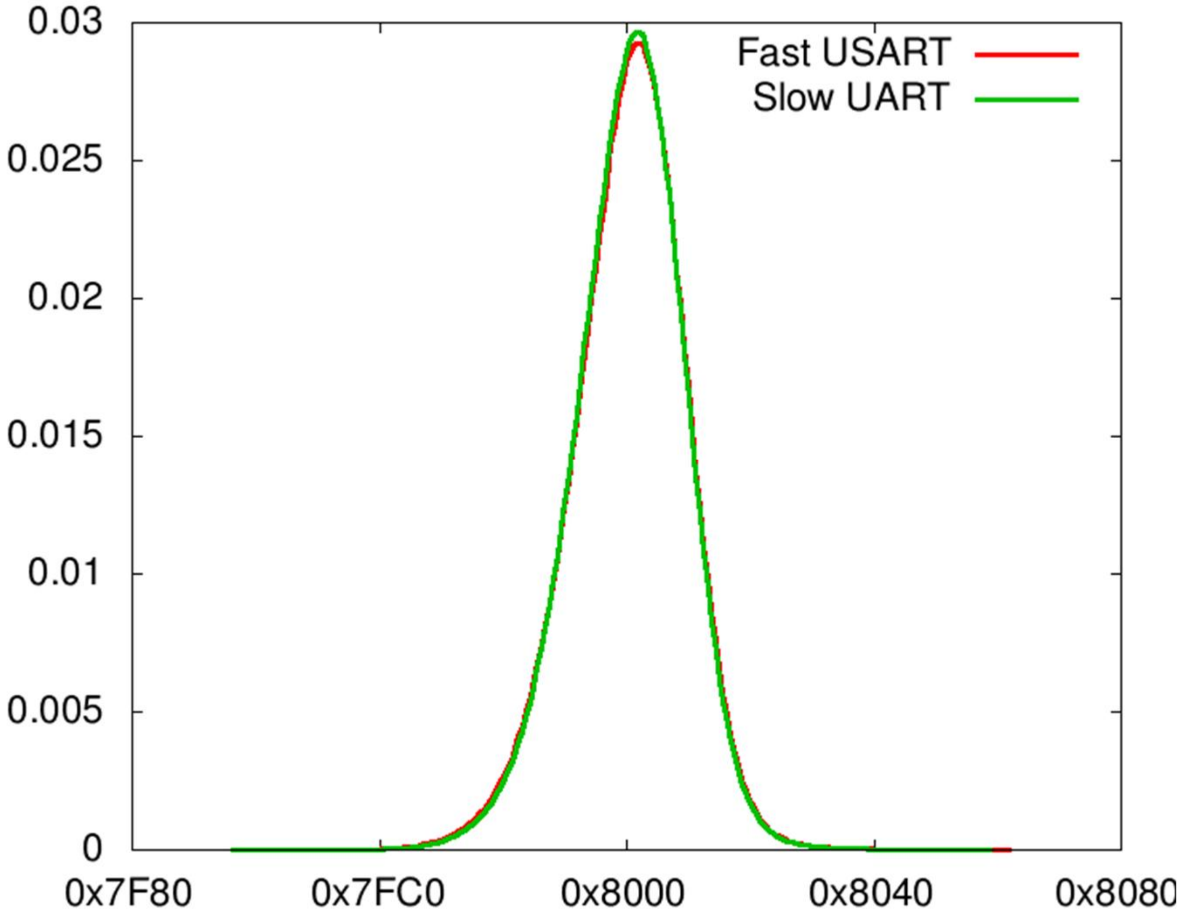
On-chip components: LCD driver



On-chip components: Sleep modes



On-chip components: USART speed



On-chip components: Summary

- LCD:
  - There is a measurable effect but it is not really significant.
- Sleep:
  - There is a measurable effect but it is not really significant.
- USART speed:
  - No measurable effect.

Conclusion

Å Influence of external 50Hz unconfirmed

- ï Future work: Try frequency injection attack

Å On-chip components have little effect

- ï LCD, SLEEP: effect measurable but insignificant
- ï USART speed: effect hardly measurable

Å Simpler microcontroller not as good TRNG

- ï TRNG possible at a much lower rate