Ardrand: The feasibility of the Arduino as a random number generator

Benedikt Kristinsson
Advisor: Ýmir Vigfússon

December 14, 2011

Randomness

- Hard on CPU

Randomness

- Hard on CPU
- External sources needed

Randomness

- Hard on CPU
- External sources needed
- But why?

Randomness

- Hard on CPU
- External sources needed
- But why?

Cryptography

- Bad seeding methods have resulted in breaking of cryptosystems

Cryptography

- Bad seeding methods have resulted in breaking of cryptosystems
  - Netscape browser

Cryptography

- Bad seeding methods have resulted in breaking of cryptosystems
  - Netscape browser
  - Enigma

Cryptography

- Bad seeding methods have resulted in breaking of cryptosystems
  - Netscape browser
  - Enigma
- Linux router

Cryptography

- Bad seeding methods have resulted in breaking of cryptosystems
  - Netscape browser
  - Enigma
- Linux router

PRNG

- Deterministic

PRNG

- Deterministic
- Only as secure as its seed

PRNG

- Deterministic
- Only as secure as its seed
- Unpredictable sequences

PRNG

- Deterministic
- Only as secure as its seed
- Unpredictable sequences

Possible ways

- External hardware

Possible ways

- External hardware
- Obtain keys from outside

Possible ways

- External hardware
- Obtain keys from outside
- Need

Possible ways

- External hardware
- Obtain keys from outside
- Need
  - Available hardware

Possible ways

- External hardware
- Obtain keys from outside
- Need
  - Available hardware
  - Cheap

Possible ways

- External hardware
- Obtain keys from outside
- Need
  - Available hardware
  - Cheap
  - Statistically sound

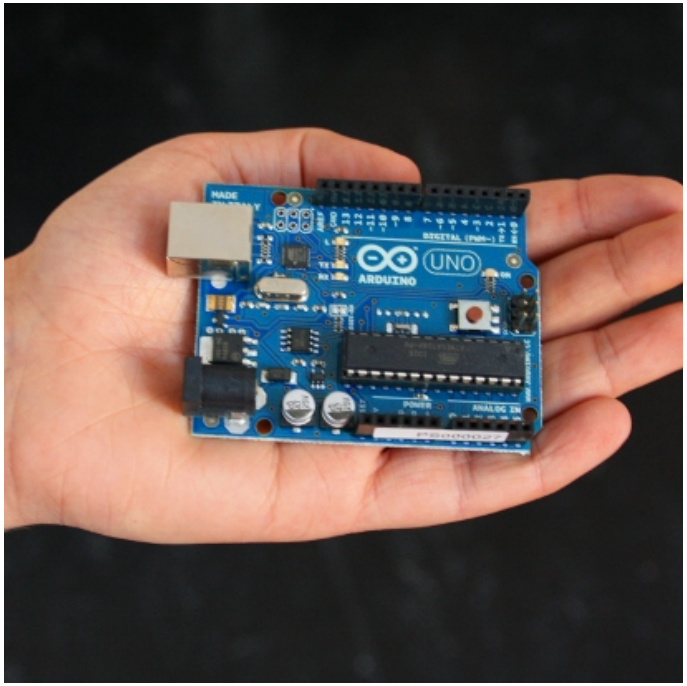
Possible ways

- External hardware
- Obtain keys from outside
- Need
  - Available hardware
  - Cheap
  - Statistically sound
  - Fast

Possible ways

- External hardware
- Obtain keys from outside
- Need
  - Available hardware
  - Cheap
  - Statistically sound
  - Fast

Today: Arduino



Arduino

- Available

Arduino

- Available
- Cheap ($30)

Arduino

- Available
- Cheap ($30)
- Analog noise from `analogRead`

Arduino

- Available
- Cheap ($30)
- Analog noise from `analogRead`
- Does it work ?

Arduino

- Available
- Cheap ($30)
- Analog noise from `analogRead`
- Does it work ?
- Is it fast enough?

Arduino

- Available
- Cheap ($30)
- Analog noise from `analogRead`
- Does it work ?
- Is it fast enough?

If it is important for a sequence of [random] values generated to differ [...] initialize the random number generator with a fairly random input, such as analogRead() on an unconnected pin.

Hypothesis

Hypothesis: Values returned from `analogRead` are random

Hypothesis

Hypothesis: Values returned from `analogRead` are random

- Need stats!

Hypothesis

Hypothesis: Values returned from `analogRead` are random

- Need stats!
- Need an controlled environment (Iceland vs. Azerbaijan)

Hypothesis

Hypothesis: Values returned from `analogRead` are random

- Need stats!
- Need an controlled environment (Iceland vs. Azerbaijan)

Analysis

- Obtain sequences

Analysis

- Obtain sequences
- Algorithms used

Analysis

- Obtain sequences
- Algorithms used
- Statistical tests

Analysis

- Obtain sequences
- Algorithms used
- Statistical tests

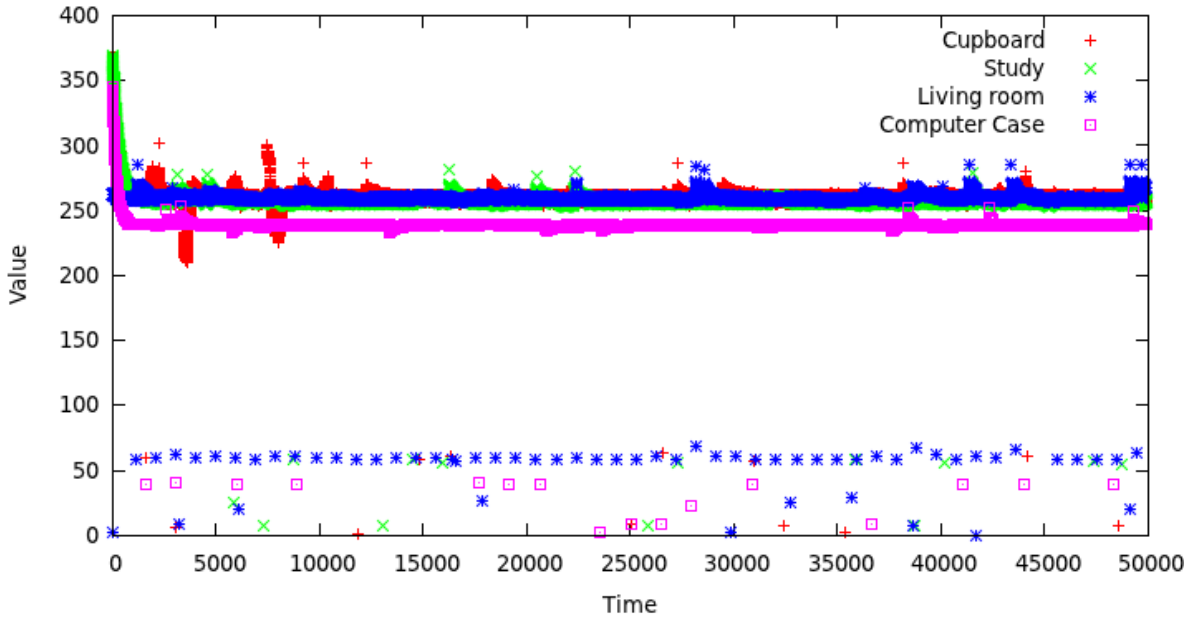
Picture?

Does the environment matter?

Q: Does the environment matter?

Does the environment matter?

Q: Does the environment matter?



Yes!

Obtained numbers

- Odd locations

Obtained numbers

- Odd locations
  - Freezer

Obtained numbers

- Odd locations
  - Freezer
  - Fridge

Obtained numbers

- Odd locations
  - Freezer
  - Fridge
  - On top of heating element

Obtained numbers

- Odd locations
  - Freezer
  - Fridge
  - On top of heating element
  - Bathtub

Obtained numbers

- Odd locations
  - Freezer
  - Fridge
  - On top of heating element
  - Bathtub
- Normal locations

Obtained numbers

- Odd locations
  - Freezer
  - Fridge
  - On top of heating element
  - Bathtub
- Normal locations
  - Rooms in flat

Obtained numbers

- Odd locations
  - Freezer
  - Fridge
  - On top of heating element
  - Bathtub
- Normal locations
  - Rooms in flat
  - CS lab

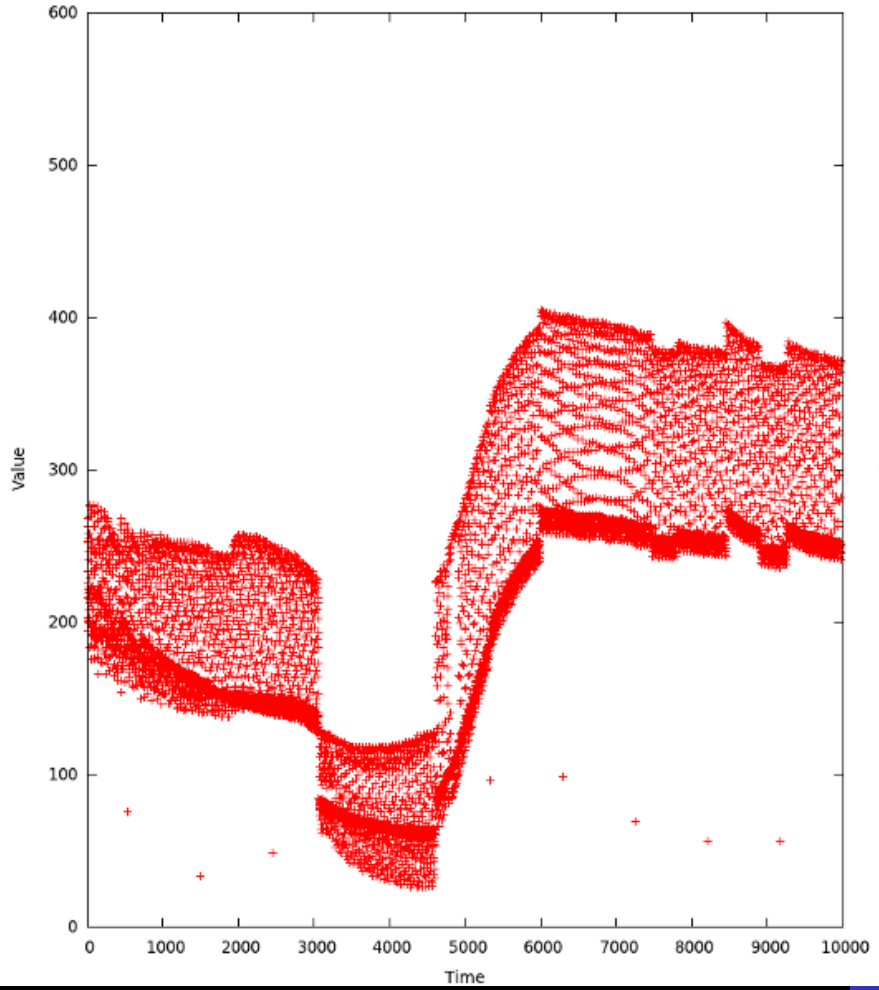
Obtained numbers

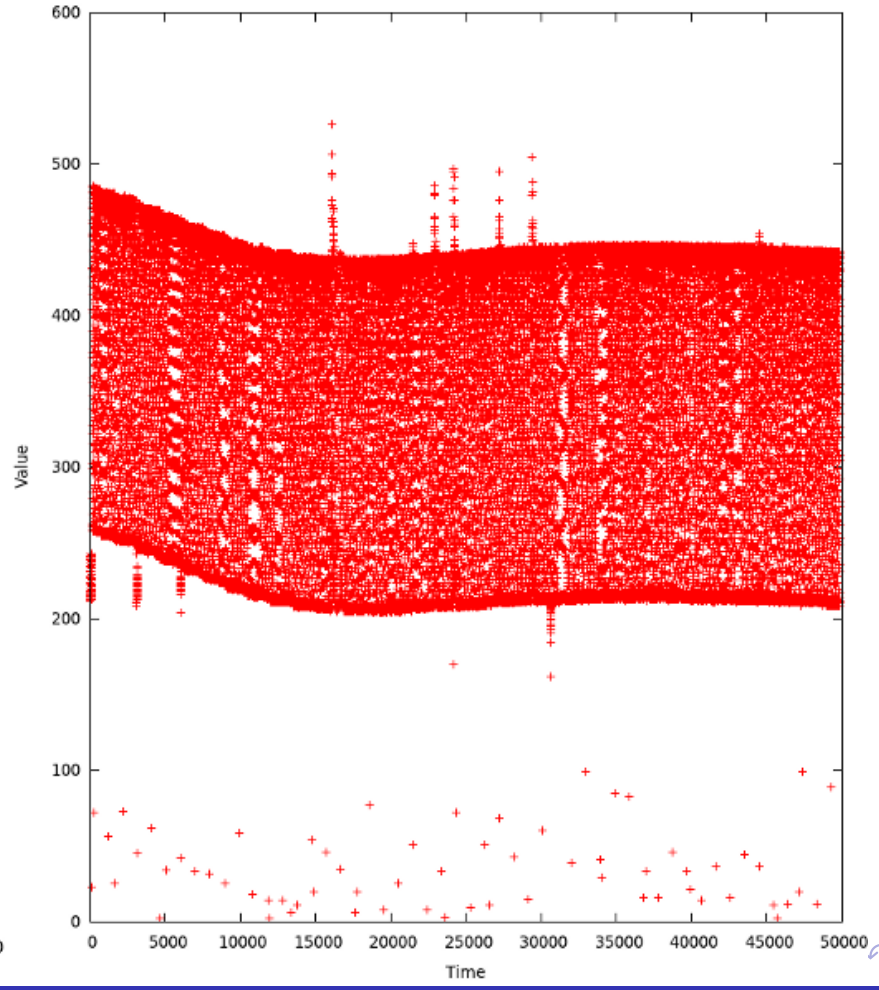
- Odd locations
  - Freezer
  - Fridge
  - On top of heating element
  - Bathtub
- Normal locations
  - Rooms in flat
  - CS lab
  - Garage

Obtained numbers

- Odd locations
  - Freezer
  - Fridge
  - On top of heating element
  - Bathtub
- Normal locations
  - Rooms in flat
  - CS lab
  - Garage

Temperature is important





Meanrand

Idea

Keep track of the mean of the values read, generate a 0 if below and a 1 otherwise.

- Observed bitrate: 25-85 bps

Meanrand

Idea

Keep track of the mean of the values read, generate a 0 if below and a 1 otherwise.

- Observed bitrate: 25-85 bps
- Slow and not very random

Meanrand

Idea

Keep track of the mean of the values read, generate a 0 if below and a 1 otherwise.

- Observed bitrate: 25-85 bps
- Slow and not very random

Updownrand

Idea

Read one value. Generate a 1 bit if the next value is higher and a 0 bit otherwise.

Updownrand

Idea

Read one value. Generate a 1 bit if the next value is higher and a 0 bit otherwise.

- Observed bitrate: 4 bps

Updownrand

Idea

Read one value. Generate a 1 bit if the next value is higher and a 0 bit otherwise.

- Observed bitrate: 4 bps
- Rejected: too slow

Updownrand

Idea

Read one value. Generate a 1 bit if the next value is higher and a 0 bit otherwise.

- Observed bitrate: 4 bps
- Rejected: too slow
- Not very random

Mixmeanupdownrand

Idea

See what happens if we mix `Mean-RAND` and `Updown-RAND`.
Generate one bit from either and XOR them together.

Mixmeanupdownrand

Idea

See what happens if we mix `Mean-RAND` and `Updown-RAND`. Generate one bit from either and XOR them together.

- Observed bitrate: 2 bps

Mixmeanupdownrand

Idea

See what happens if we mix `Mean-RAND` and `Updown-RAND`. Generate one bit from either and XOR them together.

- Observed bitrate: 2 bps
- Rejected: too slow

Mixmeanupdownrand

Idea

See what happens if we mix `Mean-RAND` and `Updown-RAND`. Generate one bit from either and XOR them together.

- Observed bitrate: 2 bps
- Rejected: too slow
- Not very random either

Leastsignrand

Idea

Return the least significant (rightmost) bit for each value from `analogRead`

Leastsignrand

Idea

Return the least significant (rightmost) bit for each value from `analogRead`

Math

Let $b = b_9, \ldots, b_1, b_0$ be a 10-bit integer generated by `analogRead`. Return b_0.

Leastsignrand

Idea

Return the least significant (rightmost) bit for each value from `analogRead`

Math

Let $b = b_9, \ldots, b_1, b_0$ be a 10-bit integer generated by `analogRead`. Return b_0.

- Observed bitrate: 290 bps

Leastsignrand

Idea

Return the least significant (rightmost) bit for each value from `analogRead`

Math

Let $b = b_9, \ldots, b_1, b_0$ be a 10-bit integer generated by `analogRead`. Return b_0.

- Observed bitrate: 290 bps
- Fastest

Leastsignrand

Idea

Return the least significant (rightmost) bit for each value from `analogRead`

Math

Let $b = b_9, \ldots, b_1, b_0$ be a 10-bit integer generated by `analogRead`. Return b_0.

- Observed bitrate: 290 bps
- Fastest
- Passes most tests in some settings

Twoleastsignrand

Idea

Return the XOR of the two least significant (rightmost) bits for each value from `analogRead`

Twoleastsignrand

Idea

Return the XOR of the two least significant (rightmost) bits for each value from `analogRead`

Math

Let $b = b_9, \ldots, b_1, b_0$ be a 10-bit integer generated by `analogRead`. Return $b_0 \oplus b_1$.

Twoleastsignrand

Idea

Return the XOR of the two least significant (rightmost) bits for each value from `analogRead`

Math

Let $b = b_9, \ldots, b_1, b_0$ be a 10-bit integer generated by `analogRead`. Return $b_0 \oplus b_1$.

- Observed bitrate: ≈ 170 bps

Twoleastsignrand

Idea

Return the XOR of the two least significant (rightmost) bits for each value from `analogRead`

Math

Let $b = b_9, \ldots, b_1, b_0$ be a 10-bit integer generated by `analogRead`. Return $b_0 \oplus b_1$.

- Observed bitrate: ≈ 170 bps
- Second fastest, but not fast enough

Twoleastsignrand

Idea

Return the XOR of the two least significant (rightmost) bits for each value from `analogRead`

Math

Let $b = b_9, \ldots, b_1, b_0$ be a 10-bit integer generated by `analogRead`. Return $b_0 \oplus b_1$.

- Observed bitrate: ≈ 170 bps
- Second fastest, but not fast enough
- Passes all tests in some settings

The von Neumann box

Used to remove bias from a generator

The von Neumann box

Used to remove bias from a generator

Idea

Input two bits and discard them if they are the same. A 1,0-pair becomes a 1-bit and 0,1 pair becomes a 0-bit.

The von Neumann box

Used to remove bias from a generator

Idea

Input two bits and discard them if they are the same. A 1,0-pair becomes a 1-bit and 0,1 pair becomes a 0-bit.

Math

Let p be the probability that the generator yields a 1-bit and q that it yields a 0-bit. This relies on the fact that 01 and 10 are equiprobable since $p \cdot q = q \cdot p$.

The von Neumann box

Used to remove bias from a generator

Idea

Input two bits and discard them if they are the same. A 1,0-pair becomes a 1-bit and 0,1 pair becomes a 0-bit.

Math

Let p be the probability that the generator yields a 1-bit and q that it yields a 0-bit. This relies on the fact that 01 and 10 are equiprobable since $p \cdot q = q \cdot p$.

Applied in all our algorithms.

Statistical testing

- Impossible to prove that a generator is random [AJM, PO, SA, 1996]

Statistical testing

- Impossible to prove that a generator is random [AJM, PO, SA, 1996]
- Not rejected rather than accepted as random

Statistical testing

- Impossible to prove that a generator is random [AJM, PO, SA, 1996]
- Not rejected rather than accepted as random
- FIPS boundaries

Statistical testing

- Impossible to prove that a generator is random [AJM, PO, SA, 1996]
- Not rejected rather than accepted as random
- FIPS boundaries

Monobit

Idea

A random sequences should contain roughly the same number of 1's and 0's. This gives a statistic on this ratio.

Monobit

Idea

A random sequences should contain roughly the same number of 1's and 0's. This gives a statistic on this ratio.

Math

Let n_0 denote the number of 0's and n_1 the number of 1's. We then find

$$X_1 = \frac{(n_0 - n_1)^2}{2}$$

Monobit

Idea

Based on the idea of five-card hands in poker. In a random sequence we would expect each hand to show up about the same amount of time.

Monobit

Idea

Based on the idea of five-card hands in poker. In a random sequence we would expect each hand to show up about the same amount of time.

Math

Let m be the size of the poker hand and $k = \lfloor \frac{n}{m} \rfloor$, where n is the length of the sequence. Find

$$X_3 = \frac{2^m}{k} \left(\sum_{i=1}^{2^m} n_i^2 \right) - k$$

Runs

Runs examples

100011

Runs

Runs examples

100011

- Has one run (gap) of length 3 (three zeroes)

Runs

Runs examples

100011

- Has one run (gap) of length 3 (three zeroes)
- One run (block) of length 2

Runs

Runs examples

100011

- Has one run (gap) of length 3 (three zeroes)
- One run (block) of length 2
- One run of length 1

Runs

Runs examples

100011

- Has one run (gap) of length 3 (three zeroes)
- One run (block) of length 2
- One run of length 1

Runs

Runs examples

100011

- Has one run (gap) of length 3 (three zeroes)
- One run (block) of length 2
- One run of length 1

Idea

Find the number of runs of each length. The longer the run, the unlikelier it is. The FIPS publication has a nice table listing how many sequences of each length should appear.

Runs

Runs examples

100011

- Has one run (gap) of length 3 (three zeroes)
- One run (block) of length 2
- One run of length 1

Idea

Find the number of runs of each length. The longer the run, the unlikelier it is. The FIPS publication has a nice table listing how many sequences of each length should appear.

Math

Let G_i and B_i be the number of gaps and blocks of length i and e_i denote the expected number of blocks of length i. Find

$$X_4 = \sum_{i=1}^{k} \frac{(B_i - e_i)^2}{e_i} + \sum_{i=1}^{k} \frac{(G_i - e_i)^2}{e_i}$$

Results

Algorithm	Monobit	Poker	Runs	Long runs	Bandwidth
`Leastsign`	ACC	ACC	(REJ)	ACC	290.55 bps
`Twoleastsign`	ACC	ACC	ACC	ACC	172.0 bps
`Mean`	ACC	REJ	REJ	REJ	25.32 bps

Results

Algorithm	Monobit	Poker	Runs	Long runs	Bandwidth
`Leastsign`	ACC	ACC	(REJ)	ACC	290.55 bps
`Twoleastsign`	ACC	ACC	ACC	ACC	172.0 bps
`Mean`	ACC	REJ	REJ	REJ	25.32 bps

- `Twoleastsign` passes NIST tests as well when it passes our tests

What does this mean?

- Arduino not a feasible target using our methods

What does this mean?

- Arduino not a feasible target using our methods
- We created a seed discovery program

What does this mean?

- Arduino not a feasible target using our methods
- We created a seed discovery program
  - Runs quickly

What does this mean?

- Arduino not a feasible target using our methods
- We created a seed discovery program
  - Runs quickly
  - Almost always finds the seed

What does this mean?

- Arduino not a feasible target using our methods
- We created a seed discovery program
  - Runs quickly
  - Almost always finds the seed
  - Tested X sequences, Y found seed

What does this mean?

- Arduino not a feasible target using our methods
- We created a seed discovery program
  - Runs quickly
  - Almost always finds the seed
  - Tested X sequences, Y found seed

Future work

- Find out what factors cause it to pass tests

Future work

- Find out what factors cause it to pass tests
- Implement more algorithms to look for entropy

Future work

- Find out what factors cause it to pass tests
- Implement more algorithms to look for entropy
- ?