Hardware & Embedded Security

Prof Daniele Rossi


Via G. Caruso 16, room B-1-03


daniele.rossi1@unipi.it


050 221 7611

1

Random Number Generators

Lecture 8 - DR

2

Brief Outline:

- Principles and architecture of Random Number Generators
- Source of entropy
- Pseudo-random (Deterministic) and True Random Number Generators
- Entropy in digital systems: clock jitter and metastability
- Examples

3

Random Numbers in Cryptography

- The keystream in the one-time pad
- The secret key in the DES encryption
- The prime numbers p, q in the RSA encryption
- Session keys
- The private key in digital signature algorithm (DSA)
- The initialization vectors (IVs) used in ciphers

4

Possible applications: OTP, secret keys for sessions, prime number generation etc.

To generate true RN, we need sources of entropy. How do we define entropy?

- Entropy can be related to disorder and uncertainty. Something that cannot be predicted, that provides a certain amount of surprise in the result of a system.

This is the property we need to design a TRNG.

What properties we would like to have from RNG:

- Good statistics in terms of uniform prob. Distrib. of output should be as uniform as possible. This means # of 1s and 0s is more or less balanced and response is not biased and prone to frequency attacks. (Uniform probability distribution)
- Unpredictability! you must not be able to predict the next values according to the generated ones.

Generic prop. for RNG.

PRNG: they are not true random, they are deterministic. The sequence generated is generated by an algorithm. Why are we using PRNG? As a possible solution why is this okay? A PRNG generates a sequence that from statistical viewpoint looks like a random sequence. 1st property ok.

List of properties helps us understand why a PRNG is actually used:

A source of entropy

- Entropy is the measurement of uncertainty or disorder in a system
- Good entropy comes from the surrounding environment which is unpredictable and chaotic
- You can think of entropy as the amount of surprise found in the result of a randomized process: the **higher the entropy**, the **less the certainty** found in the result
- **Random number generators** or RNGs are hardware devices or software programs which take **non-deterministic inputs** in the form of physical measurements of temperature or phase noise or clock signals etc and generate **unpredictable numbers as its output**

5

Random number generators

- Despite there being a lot of different applications of random numbers in a cryptographic system they all share two basic requirements:
  - **Good statistical properties** (**uniform probability** distribution)
    - Every value of **any random number** used in a cryptographic system **must be equally likely** to appear. This requirement is of utmost importance since a biased probability distribution would open the door to an attacker e.g. by making frequency attacks possible
  - **Unpredictability of random numbers**
    - Random numbers, especially those used for secret parameters such as keys, must be **unpredictable** to prevent an attacker from being able to compute future or preceding values from the already generated and captured data

6

Pseudo-random Number Generator

- Pseudo-random Number Generator (Deterministic Random Number Generator)
  - A polynomial-time computable function f(x) that expands a short random string x into a long string f(x) that **appears random**
- Not truly random in that:
  - **Deterministic algorithm**
  - Dependent on initial values (seed)
- Objectives
  - Fast
  - Uniform

Principal differences between TRNGs and PRNGs

Characteristic	*TRNG*	*PRNG*
Way of generation	External source of entropy	Mathematic algorithm
Efficiency	X	√
Deterministic	X	√
Periodicity	X	√
Reproducibility	X	√

All the other properties (except not True randomness) are good for our usages.

7 It is efficient, deterministic (not good, but easy to implement); it is periodic, a weak point if sequence is not long enough and if we don't reset algorithm by changing up the seed. What you usually do is reseed the PRNG to change internal state and start with a new sequence. This is good for unpredictability. Of course it is reproduceable because it is deterministic. GOOD THINGS they are fast and they provide uniform number generation.

Pseudo-random Number Generator

- A hardware pseudo random number generator (PRNG) is a device capable of generating a **sequence of binary numbers** that **imitates the properties of random numbers**.
- A PRNG usually require to be fed by an initial input value called a '**seed**'
- The output sequence generated depends on the seed and appears to be random while it is not truly unpredictable
- A PRNG's output sequence of binary numbers is a **deterministic function** of the **seed value**, meaning that sequence can be reproduced later if the seed is known (the term "pseudorandom" refers to the deterministic nature of the generator)

8

It is periodic and we can determine max length of the period.
When you determine algorithm, you should try to get period
to be as long as possible. Period is defined as maximum length
of the pattern we can generate.

Pseudo-random Number Generator

- PRNGs are also **periodic**; as randomness is limited to seed generation, the output sequence of binary numbers will start repeating at regular intervals
- The period of a PRNG is defined as **maximum length** of the non-repetitive pattern in the sequence
- PRNG's containing *n* bits of internal state cannot have its period longer than 2^n (actually, it generates max **$2^n - 1$ different patterns**) or sometimes have a period that is much shorter depending on the given seed

We cannot get an all 0 pattern, otherwise in an system we get stuck on that pattern

9

Pseudo-random Number Generator

Properties of a good PRNG

P1	The sequence of values it provides should resemble a sequence of independent realizations of a $U(0,1)$ r.v.
P2	From the point of view of statistical simulation: - the results must be reproducible.[1] From the point of view of cryptographic applications: - the sequences must be unpredictable.[2]
P3	The sequence of generated values should have a non-repetitive cycle as long as possible.
P4	The generator - must be fast and - must occupy a small amount of internal memory.

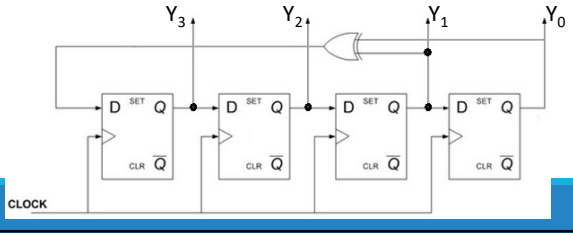
- 1. That is, starting with the same initial conditions, the same sequence must be obtained. This would allow to debug possible failures of the model or to simulate different alternatives of the model in the same conditions;
- 2. We are emphasizing the necessary security of the systems that make use of the generated sequences.

10

Results must be reproducible and sequences must be unpredictable. Period must be as long as possible. They need to be fast and as small as possible. We implement them in silicon; size and speed is important.

Pseudo-random Number Generator

- A good example of hardware-based pseudo random number generator is a **Linear Feedback Shift Register** (LFSR)
- An LFSR is basically a shift registers with FFs connected in series with the outputs of some of FFs combined in exclusive-OR configuration to provide a feedback mechanism
- When the inputs of the FFs are fed with a seed value and the LFSR is clocked, it generates a pseudorandom pattern of 1s and 0s



Seed: initial state of the register

FF are connected in a Shift Reg fashion. Last output is feedbacked in a linear combination of the previous outs with a XOR.

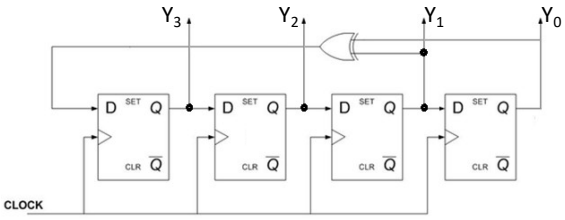
Seed = initial state of FF (sets of 0s and 1s) and we go on and generate a sequence.

At a given clock cycle, the output of the FFs will get the previous FF output we have. The sequence changes according to the feedback loop.

12

Pseudo-random Number Generator

- An LFSR is called a **maximal length LFSR**, if it can generate a stream of random numbers of **maximum length** of $2^n - 1$ before it starts to repeat, where ***n*** is number of register elements in it
- A 32-bit maximal length LFSR can generate about 4 billion (pseudo-)random numbers before it begins to repeat the sequence of numbers again



Exercise: write the pseudo-random sequence generated by the LFSR in the figure for a seed equal to s3 s2 s1 s0 = 1011

13

You have 2 ways to generate using a LFSR: a Galois or Fibonacci LFSR. What is the difference between a Galois and Fibonacci LFSR? They are built starting from the same polinomial. And we have some feedback. In Fib, you have that the XOR for linear comb are combinary output of the rightmost Flop with some intermediate results of internal flop. Then you feed result to the leftmost F. Gal: output is combined with output of some intermediate flop and combined as input to the next flop.

Which of the two is better performance wise? In Galois you don't have cascaded XORs. Or multiple input XORs. And delay introduced is longer than delay you have for Galois. True if you design an ASIC. But might not be so true for FPGAs, with LUT. And delays introduced by them overshadows the XORs. Plus you can implement XORs in LUT. And in FPGA delay is given mainly by local interconnections.

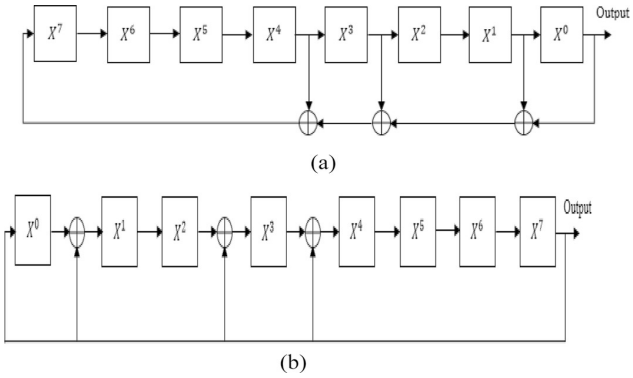
- Degree of polinomial is 8 in this case, we need 8 FFs. The highest order term and the constant term are always present (x^8, 1). Look at Fibonacci: for your feedback network you work w/ polinomial. $f(x) = x^8 + x^4 + x^3 + x + 1$. The feedback is provided with combinatorial taking outputs of 4th, 3rd, 1st FF (x^4, x^3, x^1). For Galois the feedback XOR is provided as input to 1st, 3rd, 4th polinomial.

Okay but how do we get maximum length?

Example of LFSR implementation

- The length and feedback structure of an LFSR depends on the primitive polynomial that is selected to generate the pseudo-random sequence
- Example: primitive polynomial

$$f(x) = x^8 + x^4 + x^3 + x + 1$$



LFSR representations: **(a)** Fibonacci; **(b)** Galois

14

Maximal length polynomials

- The bit positions that affect the next state are called the *taps*.
- Exercise: determine the taps for the LFSR structure in the previous slide

from Wikipedia

Bits (n)	Feedback polynomial	Taps	Taps (hex)	Period ($2^n - 1$)
2	$x^2 + x + 1$	11	0x3	3
3	$x^3 + x^2 + 1$	110	0x6	7
4	$x^4 + x^3 + 1$	1100	0xC	15
5	$x^5 + x^3 + 1$	10100	0x14	31
6	$x^6 + x^5 + 1$	110000	0x30	63
7	$x^7 + x^6 + 1$	1100000	0x60	127
8	$x^8 + x^6 + x^5 + x^4 + 1$	10111000	0xB8	255
9	$x^9 + x^5 + 1$	100010000	0x110	511
10	$x^{10} + x^7 + 1$	1001000000	0x240	1,023
11	$x^{11} + x^9 + 1$	10100000000	0x500	2,047
12	$x^{12} + x^{11} + x^{10} + x^4 + 1$	111000001000	0xE08	4,095
13	$x^{13} + x^{12} + x^{11} + x^8 + 1$	1110010000000	0x1C80	8,191
14	$x^{14} + x^{13} + x^{12} + x^2 + 1$	11100000000010	0x3802	16,383
15	$x^{15} + x^{14} + 1$	110000000000000	0x6000	32,767
16	$x^{16} + x^{15} + x^{13} + x^4 + 1$	1101000000001000	0xD008	65,535
17	$x^{17} + x^{14} + 1$	10010000000000000	0x12000	131,071
18	$x^{18} + x^{11} + 1$	10000001000000000	0x20400	262,143
19	$x^{19} + x^{18} + x^{17} + x^{14} + 1$	11100010000000000000	0x72000	524,287
20	$x^{20} + x^{17} + 1$	10010000000000000000	0x90000	1,048,575
21	$x^{21} + x^{19} + 1$	10100000000000000000	0x140000	2,097,151
22	$x^{22} + x^{21} + 1$	11000000000000000000	0x300000	4,194,303
23	$x^{23} + x^{18} + 1$	10000100000000000000	0x420000	8,388,607
24	$x^{24} + x^{23} + x^{22} + x^{17} + 1$	11100001000000000000	0xE10000	16,777,215

15

Taps are used to describe feedback net.
They tell you how to combine outputs in feedback.

TRNG: in order to be TR, the process adopted shouldn't be algorithmic, but it needs to exploit some kind of randomness source from random phenomena.
We can exploit random phenomena that are non predictable.

True Random Number Generator

- TRNGs are **not algorithmic**, but instead they are systems, which extract randomness from non-algorithmic random phenomena (temperature fluctuations, radioactive decay, ambient radio noise, etc.)
- Based on the source used, they can be further divided to:
  - **Physical TRNG** (PTRNG) uses physical noise on electron level present in all semiconductors. A PTRNG is a physical device and uses physical noise
  - **Non-physical TRNG** (NPTRNG) may not be a physical device, but instead a piece of software. It uses non-physical randomness source such as user interactions with an operating system

16

Hybrid Random Number Generator

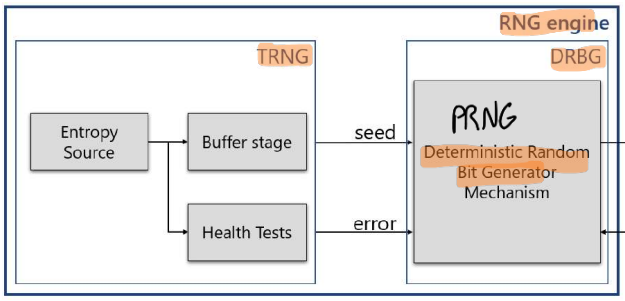
- Both TRNGs and DRNGs (PRNGs) have their advantages and disadvantages and hence many cryptographic systems use **Hybrid RNGs**, which combine the strengths of TRNGs and DRNGs
- Example of Hybrid DRNG: They use a TRNG to periodically generate seeds for a DRNG
  - Since the output of a DRNG is predictable if we know its seed, often **reseeding using a TRNG** can reduce predictability of a hybrid DRNG
  - Additionally, the **output sequence** of such a generator is **perfectly uniform**, which might not be a case for a pure TRNG
  - Their output bit rate is determined by the **bit rate** of the underlying **DRNG** because (pseudo)-random numbers may be produced until the repetition period of a DRNG is not reached

17

Then we have hybrid RNG, that use both a TRNG and a PRNG. How to combine them? Why do you need a PRNG? Why don't we just use a TRNG? Efficiency is the main reason, that goes along with performance. Plus, we cannot guarantee uniformity with a TRNG, while we can show that for PRNG. Let's look at the architecture:

How are these blocks combined? TRNG provides a true RN. The generator provides a seed for the PRNG. Then you can run some health tests: tests to assess quality of entropy source for generation. A buffer stage is just to store generated seed.

Hybrid RNG Architecture



- Health tests: required for the assessment of the level of entropy of the Entropy Source. Examples:
  1. The repetition count test: triggers an alarm if the same output value is maintained too many times in a row.
  2. The adaptive proportion test: it is built to check if the frequency of occurrence of every possible output value fall within a certain confidence interval → If an output occurs too much frequently, it means that the statistical quality of the randomness source is lowered, and an alarm shall be triggered.

18

Good TRNG Design

Thermal noise, also called **Johnson–Nyquist noise**, is the random electrical noise generated by the thermal agitation of electrons in a conductor (like a resistor). This noise is *completely natural* and unavoidable as long as the temperature is above absolute zero. The cool thing? It's fundamentally unpredictable at the quantum level.

- Entropy (Randomness) Source:
  - Randomness present in physical processes such as thermal and shot noise[1] in circuits, Brownian motion, or nuclear decay
- Harvesting Mechanism:
  - The mechanism that does not disturb the physical process but collects as much entropy as possible
- Post-Processing (optional):
  - Applied to mask imperfections in entropy sources or harvesting mechanism or to provide tolerance in the presence of environmental changes and tampering

[1]Shot noise, the time-dependent fluctuations in electrical current caused by the discreteness of the electron charge, is well known to occur in solid-state devices

20 What kind of process does a TRNG need to go through? Identify the entropy source. Noise is a good source: thermal noise, shot noise based on fact that current provided by transistor is given by superposition of discrete particles and you can have changes that are random.

To improve randomness of TRNG you can post process to increase randomness.

Let's ID the main req. We ask our sys: → to be simple

1. We would like to have harvest mech and entropy source to work purely digital and have something simple. We don't want correction circuits, of course they could damage randomness.
- Of course efficiency and simplicity that allows analysis.

NOTE: What is main diff. between behavior of a PUF and the one of a TRNG? In both cases output is unpredictable (for the PUF only for the first time. If you apply same chl you expect same response. You want consistency. This is not the case for a TRNG). You could use PUF as a one time TRNG, but PUFs don't get analyzed for randomness as much as TRNG.

NOW: main phenomenon at base of PUF is process variation. For TRNG you want something that no matter when used, acts as a source of randomness.

We will use Clock Jitter and Metastability.

- Clock Jitter: describes a variation of the clock edge from its ideal position. Clock has a certain period, no matter the duty cycle, you have edges in which you commute. The time distance between those edges can vary in a random way more or less.
- Metastability: with a storage element in which constraints are not satisfied, you don't know which value you will be able to provide. Metastability is the ability of a circuit to persist in an undefined state for indefinite period of time.

These are inherent to any digital sys. Great!

CS: The instant in which edges occur can slightly vary. We can explore this phenomenon for our system. Ideal clock signal is given by $t(n) = n \cdot T$. But the edges do not happen exactly at integer multiples of the clock period.

Set of Requirements

- The design should be **purely digital**
- The harvesting mechanism should be **simple**
  - The unpredictability of the TRNG should not be based on the complexity of the harvesting mechanism, but only on the unpredictability of the entropy source
- No correction circuits are allowed
- Compact and efficient design (high throughput per area and energy spent)
- The design should be sufficiently simple to allow rigorous analysis

21

Sources of randomness in digital systems

- In logic devices (digital systems), physical noise sources are quite limited, since logic devices are supposed to always be in a well-defined state
- In order to generate random numbers, we need an uncontrollable random phenomenon
- Physical phenomena commonly used to generate random numbers in logic devices are:
  - **Clock jitter**, which is a variation of the clock edge from its ideal position
  - **Metastability** is an ability of a circuit to persist in an undefined state for indefinite period of time
  - **Analog signals** such as shot noise of a diode, thermal noise, etc.

22

Clock Jitter

- An ideal clock signal in digital logic devices is supposed to be a rectangular signal with a 50% duty cycle and stable period.
- Due to various noises affecting electronic devices, the **clock signal is never absolutely stable,** and its edges move from their stable position → the **phase of the clock signal fluctuates**.
- This fluctuation can be seen as a **clock jitter** in time domain and as a **phase noise** in the frequency domain.
- In logic devices, the clock jitter is usually unwanted, but inevitable. Since the jitter is negatively affecting high-frequency communications and high-speed systems, it has been well-studied and characterized.

23

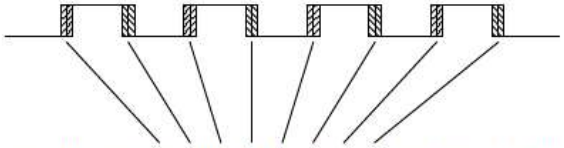
Clock Jitter

- Clock jitter in a digital system is a deviation of the actual clock edge from an ideal clock edge.
- Ideal clock signal is defined by the equation: $t(n) = n \cdot T$

t(n): time of n-th period of a clock signal

T: period of the clock signal

- A real clock signal does not arrive always at integer multiples of its period, as the ideal one does, but its edges are fluctuating around this value because of the jitter.

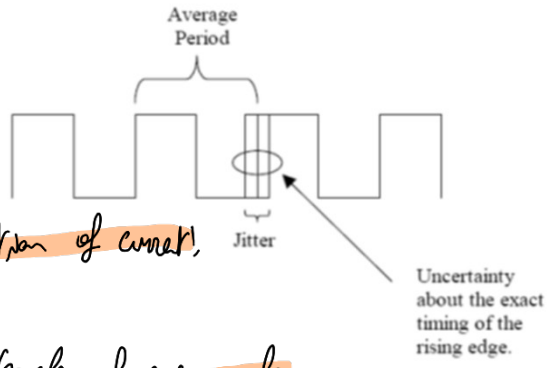


Depending on the jitter size, the clock edge may arrive anywhere within these regions

24

Clock Jitter

- Jitter is variations in the significant instants of a clock
- Jitter is nondeterministic (random)
- **Sources of Jitter:**
  - Semiconductor noise
  - Cross-talk
  - Power supply variations
  - Electromagnetic fields



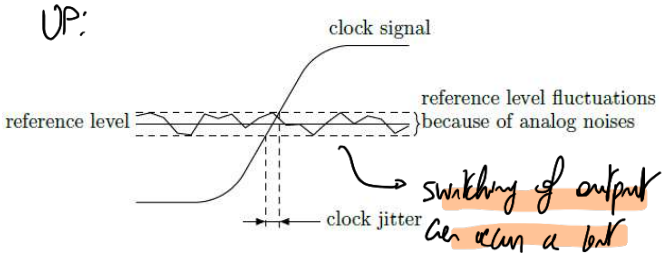
Inductive effects : because of alternation of current,

Phenomenon when you have a transfer of energy from one part of system to another because of capacitive/inductive coupling effects.

Noises causing clock jitter in digital circuits

- Digital circuits use a reference level, usually in the middle of operating voltage range, in order to detect clock edges (to switch from L-H or H-L)
- This reference level should be as stable as possible, but in reality it fluctuates because of various noises.
- When the reference level shifts, it causes the clock edge to be detected sooner or later as it normally would be → this temporal shift in clock detection moment is observed as the clock jitter.

UP:



Switching of output can occur a bit earlier or later give the noise we have. Noise is somehow random, so effect stays random.

RET

Let's focus on noise power supply variation:
let's say you have a buffer composed of 2 cascaded inverters.
This buffer is supplied by means of V_{DD} and GND:

Imagine you have a switch in input. The output usually starts to change when input has reached a certain point, usually $V_{DD}/2$.
So if power supply changes, your time for switching will change V_{DD}.
JMP UP.
Switching instant turns out to be random.

change in out is much faster than for input

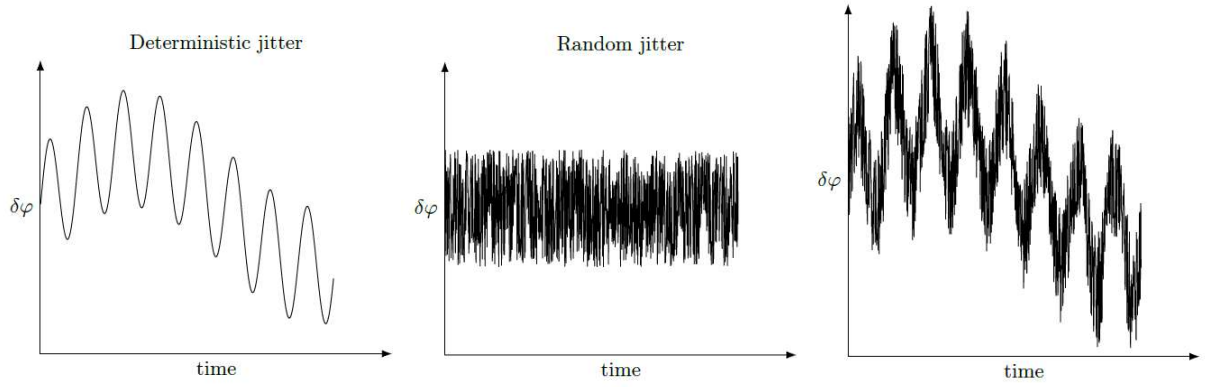
Jitter components

- Jitter has various components, which are caused by different phenomena:
  - **random**, or
  - **deterministic**.
- Random components, such as those originating from the thermal or 1/f noise, are unpredictable and follow some kind of statistics
- **Deterministic components are implementation dependent**, which means they depend on specific effects such as processed data and power supplies used

30

Jitter components

Note that we can have random jitter or det. If noise depends on deterministic event, the jitter will not be truly random. Most of the time we have random here,



31

But why do we have power supply variation in the 1st place? This varies according to the ops we are performing in our system, but why does it vary? We mentioned how PS is distributed to our sys. We use metal for distribution, that introduces resistive and inductive losses, but also capacitance losses and parasites.

A current flowing in RC circuit provides output variations: inductive parasitic effects.

If the current provided by PS changes, data dependent, this causes the unpredictable variations we have.

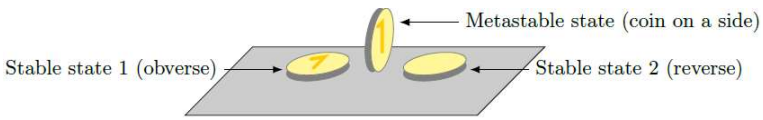
Useful jitter for TRNGs (1)

- In the context of TRNGs, the deterministic jitter components are unwanted, since they do not provide any real randomness.
- The randomness can be extracted only from the jitter caused by random noises.
- Before generating random numbers from the jitter, we need to know its statistical properties.

32

Metastability

- Metastability is an ability of a system to persist in an "illegal" (unstable) state for an indefinite period of time.



- When we toss a coin, we want it to land on either of the two faces → its faces are the two legal (stable) states.
- But when a coin lands on its side, the result of a coin flip is indecisive, and thus illegal. This state is called a metastable state.
- In order for a coin to stay in a metastable state, it must be in perfect equilibrium. Even a slightest force applied to it will cause it to fall on either one of the faces.

35

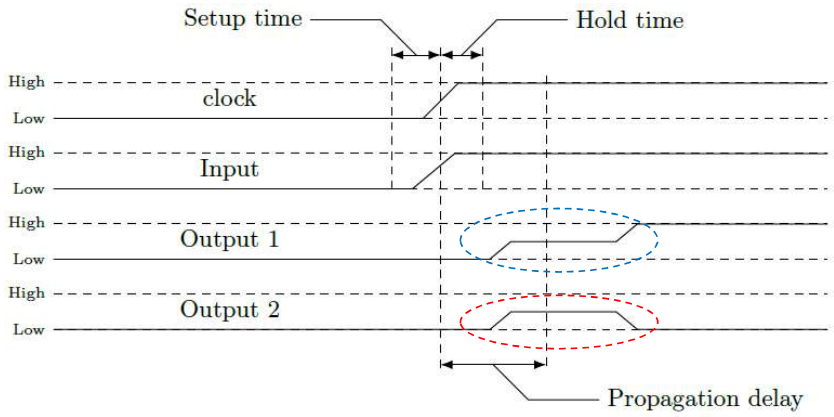
Metastability in logic devices

- In logic devices, metastability can occur in memory cells or registers (flip-flops) during normal operation of the device when constraints are not satisfied
- Flip-flops require an input signal to be stable for certain time before the clock edge (**setup**) and also for some time after the clock edge (**hold**)

  → the register is guaranteed to have a well-defined value within a specified delay at the output

36

Metastability in logic devices



input changes in this period

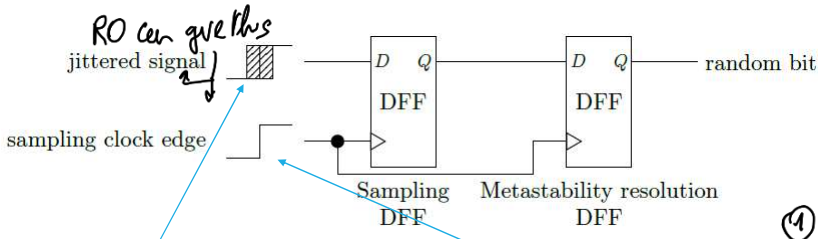
- If the setup and hold times are not respected, the register can fall into a **metastable state**, where it hovers for an unspecified period of time before it resolves to either its previous value or the new one

37

How can I take advantage of Metastasis and Invasion?

① Design techniques will guarantee that whenever we sample input it will be affected by Jitter. We don't know whether we sample 1 or 0, and we will write ②

Extraction of randomness from the clock jitter



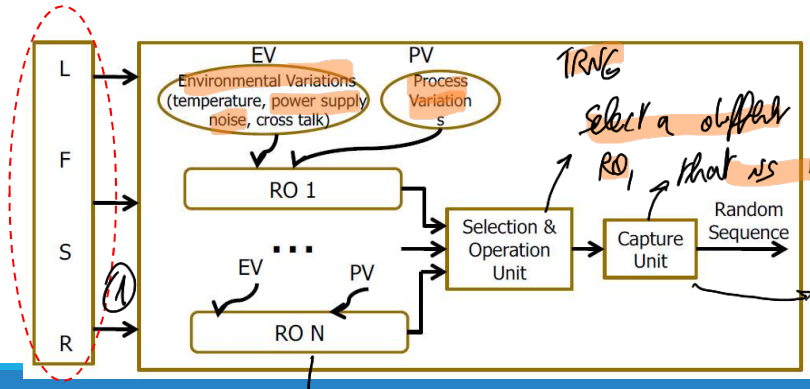
- When generating random numbers from the jitter, we need to digitize it in some way in order to produce random bits.
- To produce random bits, the **clock signal must arrive during the jitter-affected edge** of the jittered signal → this requires high precision of clock timing, because the jitter is very small (usually in order of picoseconds).
- As we already mentioned, the jitter is inevitable, so even the sampling clock signal is not jitter-free → This also adds to the difficulty of precise timing

41 Why do we have a second DFF? Due to metastab., the 1st FF could provide a value in between 0 and Vdd. We don't want to provide this signal as output. So we sample final value of 1st FF and will generate either a 0 or 1 as output.

For 2nd FF Vco is enough to control Metastab. Note that Vp is usually very small

Example of TRNG Structure

- **LFSR**: Generate random patterns, causing random switching noise



Select a different RO, that is then captured by means of FF

Can be the previous one we saw. And we can generate Random sequ

42

Affected by process variation and noise too. Eventually they oscillate w/ given frequency. They generate slightly different signals ③

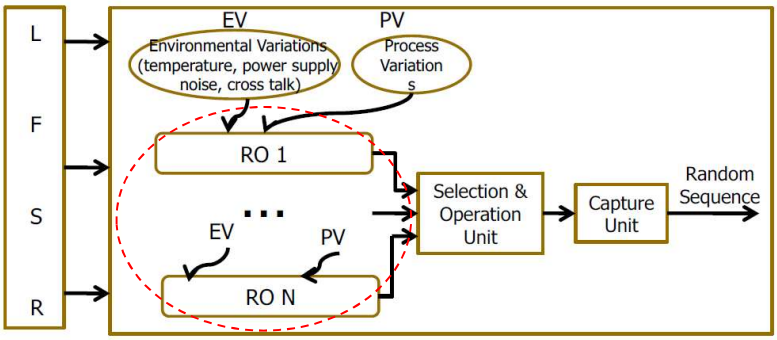
This is a circuit that generalizes previous one.

① Not directly related to functionality of TRNG. Why do we have it?
LFSR connected to the same Power supply of circuit. It generates a pseudorandom sequence. This can generate power supply noise that is apparently random.
LFSR injects random noise on the sys, via random cross talk noise.
The 0, 1s interact with rest of the system. If supplied w same PS network, it will also inject random PS variation noise.

③ and they might provide Jitter because of this.

Example of TRNG Structure

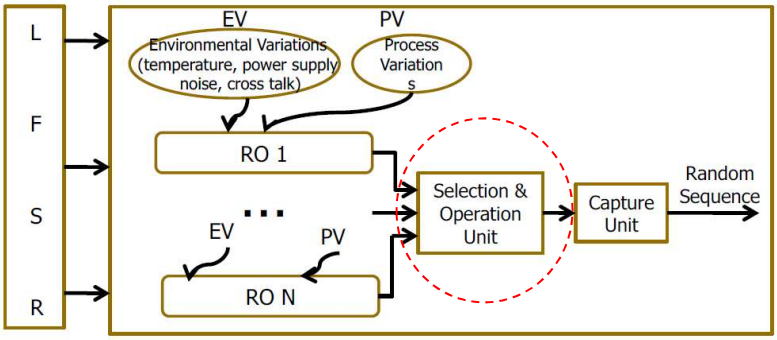
- **Ring Oscillators**
  - Process variations & environmental variations
  - Random phase jitter



43

Example of TRNG Structure

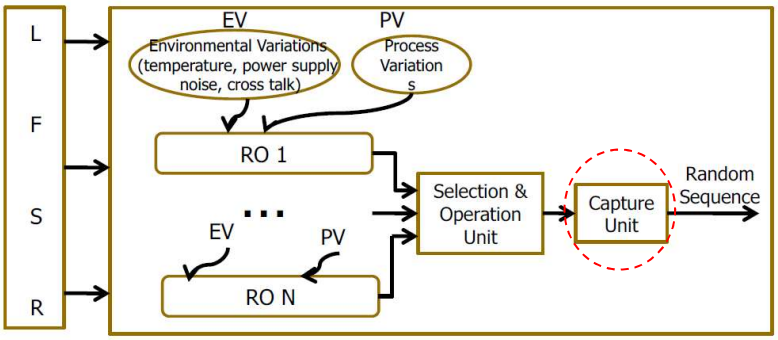
- **Selection & Operation Unit**: The random phase of ring oscillators could be translated into digital values by this unit, such as XOR operation



44

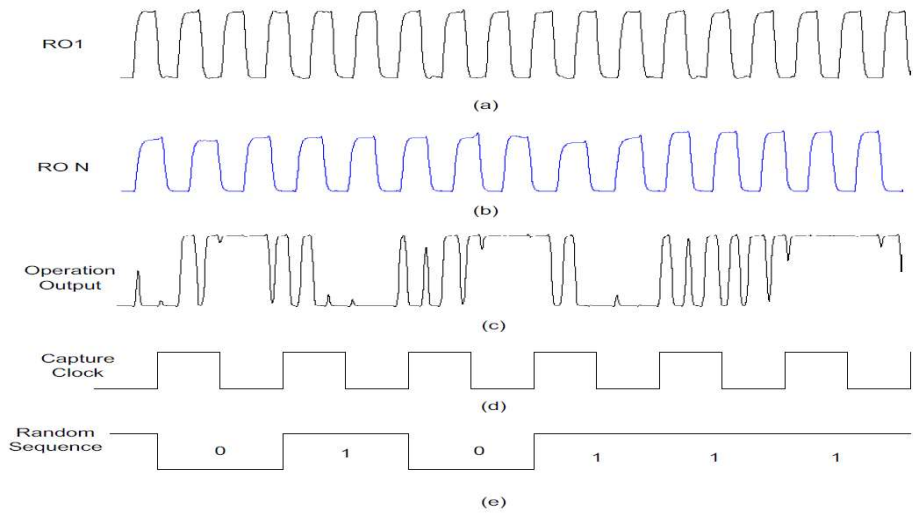
Example of TRNG Structure

- **Capture Unit**: Make sure the digital value is sampled with the frequency of the required true random number



45

TRNG Output



46

Thank you!

Any questions?

54