1 Random number generators

In the fifties, computer programming had become of interest for simulating natural phenomena, thereby introducing the need for generating random numbers. Applications in physics, games, and cryptography are ubiquitous nowadays. True random number generators do exist - you can take a look at `https://www.random.org/` if you are interested - but the phenomena for generating them are nowhere in the vicinity of the hardware platforms we use typically indoors. Therefore, clever algorithms named Pseudo-Random Number Generators (PRNGs) were developed, requiring just one (true) random number as an input for generating at their outputs sequences of numbers that are random from a statistical perspective. Thanks to PRNGs your game, testbench or simulator will become much more interesting.

2 Linear feedback shift registers

The list of linear PRNGs includes linear congruential generators, XORshift algorithms, R250 (part of `RAND` in C), ARC4 (8-bit logic), the Mersenne twister, XABC, and XABC-2. You are free to deploy any of them for your microcontroller project, though we recommend a class of PRNGs called linear feedback shift registers. These electronic components materialize Galois arithmetics (they are thus based on mathematical theorems instead of years of fine-tuning), generate pseudo-random integers with an arbitrary bit length drawn from a uniform distribution, and require just a few lines of assembly code for emulating the electronic components with software.

A rigorous explanation of Galois arithmetics is outside the scope of this document, therefore we demonstrate the procedure of generating paseudo-random bits by an example. You find the document `lfsr_table.pdf` on Pointcarré. It lists the configuration of some maximum-cycle LFSRs with n stages and k taps. Let us take $n = 5$ and $k = 4$ as an example. We retrieve the taps {5,4,3,2} from the table in `lfsr_table.pdf`. Besides, we store the state of the 5-bit wide shift register as the integer $s_5s_4s_3s_2s_1$, where s_j is the j-th bit in the shift register. Take a look at Figure 1 for the configuration of the corresponding LFSR. We explain in the subsections ahead how to translate the number of stages n and the list of taps into an LFSR and how it is employed for the generation of pseudo-random bits.

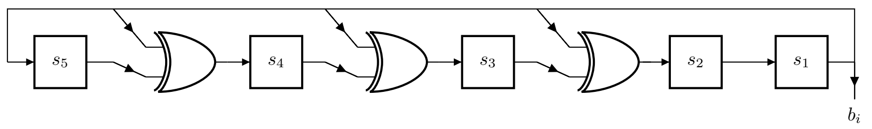

Figure 1: An LFSR with 5 stages and 4 taps.

2.1 Initialization

An n-stage LFSR requires n (pseudo-)random bits we call the seed. The seed must not be zero, but should be initialized such that (quasi) all values between 1 and $2^n - 1$ are drawn uniformly at random. Therefore, you have to use whatever bits in the registers of the microcontroller. You have to find a way such that the value of the seed is unpredictable. The seed will never be drawn from a uniform distribution, but you should aim at the largest amount of variety and unpredictability possible. As an example, we take the seed $s_5s_4s_3s_2s_1 = 11011$. The corresponding state of the LFSR is depicted in Figure 2.

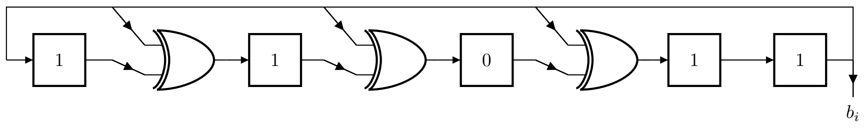

Figure 2: Example of an initialized LFSR with 5 stages and 4 taps.

2.2 Positioning the taps

A tap refers to a feedback loop.

- One of the taps has to be n ($n = 5$ in our example). It means that the value of s_1 is passed to s_n (s_5 in our example) in the next iteration.
- All other taps have to be smaller than n, greater than 0, and occur only once. A tap at position p means that

$$s_p \leftarrow s_{p+1} \oplus s_1,$$

  where $\oplus$ denotes the exclusive OR operation.

2.3 Iterating

Once the configuration of the LFSR is determined, we can iterate and as such change its state. We know that the state is equal to the seed at iteration 0. Thereafter, we find the states at iteration i as listed in Table 1.

iteration i	state	b_i	iteration i	state	b_i	iteration i	state	b_i
0	11011	1	12	01000	0	24	01100	0
1	10011	1	13	00100	0	25	00110	0
2	10111	1	14	00010	0	26	00011	1
3	10101	1	15	00001	1	27	11111	1
4	10100	0	16	11110	0	28	10001	1
5	01010	0	17	01111	1	29	10110	0
6	00101	1	18	11001	1	30	01011	1
7	11100	0	19	10010	0	31 = 0	11011	1
8	01110	0	20	01001	1	32 = 1	...	.
9	00111	1	21	11010	0			
10	11101	1	22	01101	1			
11	10000	0	23	11000	0			

Table 1: All states of the LFSR in Figure 1. The bits that are typed in a blue font are output as pseudo-random bits b_i.

After $2^n - 1$ iterations (31 in our example), the state of the LFSR is equal to its initial state (the seed) again and cycles through the same states over and over again. In order to avoid this cyclic behaviour in a sequence of pseudo-randomly generated bits, n should be as large as possible!

2.4 Random bits and bytes

During each iteration, s_1 is output as b_i, which is a pseudo-randomly generated bit. A combination of 8 outputs results in a pseudo-random byte. Depending on the range for which you want to generate pseudo-random integers, you will have to aggregate N pseudo-random bits, where 2^N is greater or equal to the number of distinct integers you want to generate.