Random Bit Generator in TEX

Hans van der Meer

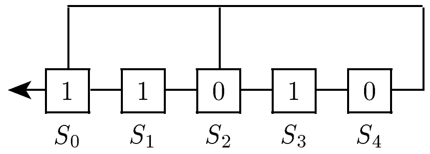
1 Introduction

When I started the use of TEX for my collection of exam questions, the need of a random bit generator arose. With such a generator it is easy to randomly permute items of multiple choice questions, choose between different variants, etc.

Since part of my interests are in the field of cryptography it was most natural to look for a convenient source of a random bitstream in that field. Such a source is provided by shiftregisters, the simplest form of which is the linear variety. Allthough not strong enough for direct use in cryptographic applications, their random properties are nevertheless excellent. Furthermore they are easily implemented, a real asset because of TEX's limited abilities in arithmetics. The prime reference for shiftregisters is the famous book by Golomb[1].

2 Linear Shiftregisters

Before describing how such a shiftregister can be implemented in TEX, it is necessary to have a modest look at their construction. The figure shows a small linear shiftregister. It consists of five socalled *stages* $S_0 \ldots S_4$ and is therefore called a five stage register. Each stage is a memory unit capable of holding one bit. The values of all the stages together make up the *state* of the register; in the figure the current state $S = (11010)$.



The register is operated in the following way. At each step the bits in the stages are shifted to the stage at their left. The bit in stage S_0 is thereby produced as the output bit. Of course the vacancy left in the rightmost stage must be filled up. Therefore all stages which in the figure have an exit at the top of the stage box, also spawn their bit through this exit just before the bit migrates to the left. These exits are called *taps*. The bits spawned are combined by the exclusive-or operator and the resultant bit fills the rightmost stage. E.g. with taps at $S_i, S_j, S_k, \ldots$ the mod 2 sum $S_i \oplus S_j \oplus S_k \oplus \ldots$ is formed. Thus the register produces an output bit and a new state at each operation step. In the example the output bit will be a 1 and the next state $S = (10101)$.

It is easily understood that eventually the bitstream must repeat itself. Because a n-stage register holds a n-bit quantity it can exist in 2^n different states only. Since new states are produced by a strictly deterministic process, a periodic pattern of successive states must result. Thus the output stream will be periodic. Of course it is desirable that the length of the cycle be as long as possible.

These registers can also be described with a polynomial in a bit variable $x \in \{0, 1\}$, called the *characteristic polynomial*. The example register has characteristic polynomial

$$f(x) = 1 + x^2 + x^5$$

It turns out that the length of the cycle produced by a register characterized by a given polynomial is connected to certain properties of this polynomial. Particular usefull are the socalled primitive polynomials.[1] One is able to show that primitive polynomials lead to the longest possible period for a linear shiftregister of a given size. In fact two cycles are produced: (1) a cycle of period 1 consisting of a stream of zeroes, (2) a fine random stream of zeroes and ones of length $2^n - 1$. The first cycle, the zero cycle, is not entirely useless as it offers a natural way for shutting off the random stream.[2]

After having explained how a shiftregister works, it is easy to see why I did choose the register based on

$$f(x) = 1 + x^{21} + x^{22}$$

for the implementation of a random bit generator in TEX. It is a primitive polynomial and therefore has a longest period of 4,194,303 bits. More than enough for all but the most exotic applications. And another important fact is that it has only two taps, located at the extremities of the register. This simplifies the implementation significantly.

3 Implementation

We are arriving at the implementation of all this stuff. At last! The simplicity of the implementation is in part due to the choice of an exponent below 32, making it possible to represent the complete state

[1] Roughly the equivalent of a prime number among polynomials plus an additional condition.

[2] I am using this stream when typesetting the full collection of exam questions. The absence of random shuffling makes it easier to connect the printed output with the TEX input.

Published in TUGboat vol.15 (1994) nr.1, p.57–58

of the register with a single count register. Since the character @ is used in internal macros, don't forget the catcode change with `\makeatletter` or `\catcode`\@=11` and changing it back afterwards.

```
\newcount\@SR
```

Furthermore we need a constant, necessary for handling the case where a 1-bit fills the vacancy in rightmost stage. Our choice $n = 22$ dictates the value $2^{21} = 2,097,152$.

```
\def\@SRconst{2097152}
```

Initialization of the stream is done by simply setting the count register (globally) to the intended start value. Keeping this value between 1 and 4,194,303 can be left the responsibility of the user.[3]

```
\def\SRset#1{\global\@SR#1\relax}
```

Each step in the register cycle needs the calculation of the exclusive-or of the stages having a tap. The form of the characteristic polynomial chosen confines this to the bits corresponding to x^{21} and x^0. Intricate calculations are therefore not needed. The value of the tap at the highest coefficient can be tested by comparing the register contents with the constant `\@SRconst` and jotting down the result in a scratch register. With `\ifodd` we take a look at the parity of the state which provides the value for x^0. A division by 2 then conveniently shifts the contents of all stages one place to the left. We place a 1 in the highest stage by adding `\@SRconst`, if there is an odd number of 1's in the two taps examined. Finally note that the new status is assigned `\global` and that the whole process is enclosed in a group which localizes the changes to the scratch register.

```
\def\@SRadvance{\begingroup
        % examine value of highest tap
    \ifnum\@SR<\@SRconst\relax \count@=0
    \else \count@=1 \fi
        % examine value of lowest tap
    \ifodd\@SR \advance\count@ by 1 \fi
        % all stages advance
    \global\divide\@SR by 2
        % place 1 in highest stage
    \ifodd\count@
        \global\advance\@SR\@SRconst\relax
    \fi
    \endgroup}
```

Production of an output bit and advancing the register one step is done by:

```
\def\SRbit{\@SRadvance
    \ifodd\@SR 1\else 0\fi
    }
```

[3] It is not difficult to write a macro that takes of its argument the value modulo 4,194,304 but one has to be careful not to end up with the null cycle.

The bit thus produced can be used in decision making. An example is the macro below which chooses between its first and second argument on the value of the next output bit of the register. With it we can write

```
\SRtest{choice 1}{choice 2}
```

and effect a random choice between the arguments. The implementation of `\SRtest` is

```
\def\SRtest#1#2{%
    \@SRadvance
    \ifodd\@SR #1\else #2\fi
    }
```

Another application is the permutation of items. Two items will be randomly interchanged by

```
\def\permtwo#1#2{\SRtest{#1#2}{#2#1}}
```

Those who are interested in the current value of the register state can obtain this by looking at the count register:

```
\def\SRvalue{\number\@SR}
```

References

[1] Golomb, S.W. 1967 *Shift Register Sequences*, San Francisco: Holden-Day.

◇ Hans van der Meer
University of Amsterdam
Faculty of Mathematics
and Computer Science
Plantage Muidergracht 24
1018 TV Amsterdam
Netherlands
hansm@fwi.uva.nl