

## Notes on Linear Feedback Shift Registers

## Linear Congruential Number Generators

The problem of obtaining long streams of pseudo-random numbers has application in many fields of electronics (see the 'uses of Linear Feedback Shift Registers' below). Of particular use are predictable, repeatable bit patterns which have the useful properties of truly random numbers, such as the probability of a particular bit being 0.5, the probability of each bit being uncorrelated with its predecessor, and the autocorrelation function of the pattern being essentially flat.

To obtain a set of random numbers  $X_n$  restricted to a space  $[0, m-1]$  the following formula can be used.

$$X_{n+1} = (aX_n + c) \bmod m \quad (1)$$

This method of obtaining random numbers is referred to as the linear congruential method. It, or other methods extending it, are useful because the maths involved in choosing  $a, c, m$  and  $X_0$  is solvable, so predictions of the length of the number pattern generated and its statistical properties can be found. More complex methods may look impressive, but usually turn out to be less effective than this method with carefully chosen  $a, c, m$  and  $X_0$ . We restrict ourselves to finding how to get the maximal length of pattern before the numbers generated by (1) repeat.

General case,  $c \neq 0$ , maximum length of pattern must be  $m$ . This will be the case if;

$c$  is relatively prime to  $m$  (i.e. they have no prime factors in common)

$a-1$  is a multiple of  $p$  for every prime  $p$  dividing  $m$

$a-1$  is a multiple of 4 if  $m$  is a multiple of 4

Note that this suggests values of  $m$  that are prime (or prime<sup>2</sup>, or prime<sup>3</sup>) tend to be good, as more values of  $a$  might be available from the second condition above. This is borne out in practice, as a randomly chosen  $a$  and non prime  $m$  will usually not give a long pattern.

Restricted case,  $c=0$ . Obviously  $X_0=0$  always gives pattern length 1, so the maximum pattern length is  $m-1$ . Actually the maximum pattern length is  $\lambda(m)$ , where

$$\lambda(2) = 1, \lambda(4) = 2, \lambda(2^e) = 2^{e-2} \text{ etc.}$$

$$\lambda(p^e) = p^{e-1}(p-1) \text{ for } p \text{ prime and } > 2$$

$\lambda(m)$  can be obtained in general by breaking down  $m$  into its constituent primes. For instance  $20 = 5 \times 2^2$ .  $\lambda(20)$  is then the lowest common multiple of  $\lambda(5)$  and  $\lambda(2^2)$ .

The maximum pattern length is then obtained by setting  $X_0$  to some value relatively prime to  $m$ , and by choosing  $a$  to be a 'primitive element modulo  $m$ '. Such a primitive element is defined by the formula  $a^\lambda = 1 \bmod m$ . For  $m$  a power of 2 greater than or equal to 8,  $\lambda = (3 \text{ or } 5) \bmod 8$ .

For the proofs of all these formulae, see Don Knuth, *The Art of Computer Programming, Seminumerical Algorithms* vol2 1973 Addison Wesley.

## Linear Feedback Shift registers

From above, the maximal pattern length of a linear congruential number generator where  $m$  is  $2^e$ , is always  $2^{e-2}$ . Drat, not long enough. However, we can extend the theory to deal with more complex linear forms than (1). We could try

$$X_{n+1} = (aX_n + c) \bmod m$$

## Uses of Linear Feedback Shift Registers

*Encryption and decryption:* You can use the pseudorandom sequence of values generated by an LFSR in the encryption (scrambling) and decryption (unscrambling) of data in communication systems. In the case of digital data, you can simply XOR the data stream with the output of an LFSR to generate an encrypted signal. (A similar process at the receiver decrypts the incoming signal.)

*Data communications:* For certain communication applications in environments with significant background noise, modulating the data using a maximal-length LFSR provides the modulated signal an auto-correlation function that lets you recover the data despite a S/N ratio of many decibels.

*Data integrity:* A traditional application for LFSRs is in cyclic-redundancy-check (CRC) calculations. CRC calculators let you detect errors in data communications by using the stream of transmitted data bits to modify the values fed back to the LFSR. The final CRC value stored in the LFSR, which is known as the checksum, depends on every bit in the data stream. The receiver compares an internally generated checksum with the checksum sent by the transmitter to determine whether any corruption occurred during transmission. This form of error detection is efficient because of the small number of bits that have to be transmitted in addition to the data. An offshoot of this application is used in the detection of *computer viruses*, although more sophisticated viruses have evolved that can counteract these measures.

*Fault test:* Another application for CRC calculators is in the circuit-board test strategy known as "functional test." You plug the board into a functional tester, and the tester applies a pattern of signals to the board's inputs. After allowing sufficient time for propagation effects to settle, the tester compares the actual values observed on the outputs with a set of expected values that are stored in the system. This process is repeated for a large number of input sequences.

If the board fails the preliminary tests, you can employ a more sophisticated form of analysis, called a "guided probe," to identify the cause of the failure. The tester instructs the operator to place the probe at a location on the board and then to rerun the entire sequence of test patterns. The tester compares the actual sequence of values the probe sees with a sequence of expected values stored in the system. You repeat this process until the tester locates the faulty component or track.

A major consideration when using guided-probe testing is the amount of data you expect to store. One way to minimize the expected data is to use LFSR-based CRC calculators. You can pass the sequence of expected values through a CRC calculator implemented in software. You can also pass the sequence of actual values seen by the guided probe through an identical CRC calculator implemented in hardware. The calculated checksum values are also known as "signatures," and the guided probe test based on this technique is known as "signature analysis." Irrespective of the number of test patterns used, the system has to store only a single signature for each track. In addition, for each application, the tester has to compare only the signature gathered by the probe with the expected signature stored in the system. Therefore, the compressed data result in storage requirements that are much smaller and comparison times that are much shorter than the uncompressed approach.

*Built-in self-test:* Another test strategy you may want to employ is built-in self-test (BIST). Devices using BIST contain special test generators and result-gathering circuits, both of which you can implement using LFSRs.

*Pseudorandom numbers:* Many computer programs, including games, digital and analog simulators, and graphic packages, rely on random sequences. You can generate pseudorandom sequences using LFSRs. In fact, pseudorandom numbers have an advantage over truly random numbers, because many computer applications typically require repeatability. However, designers also need the ability to modify the seed value of the pseudorandom-number generator, to create alternative pseudorandom sequences.

*Optimised 'counters'.* Use in FIFOs instead of binary counters. The position flags in a FIFO do not need to actually count in binary, they just need to span the whole memory hierarchy. So build them from LFSR and then you have less logic in the critical logic paths and a faster final design.

### Properties of Linear Feedback Shift Registers

One full cycle has the number of '1's one greater than the number of '0's

In one full clock cycle the probability of having a '1' follow another '1' or '0' follow another '0' is approximately half.

In one full cycle a cyclical shift has the number of disagreements one greater than the number of agreements - the autocorrelation function has no side lobes and therefore no area more likely to give false readings in noisy environments.

### 'Famous' Linear Feedback Shift Registers

*Modem Checksum Protocols.* CRC for the Comité Consultatif Internationale Télégraphique et Téléphonique (CCITT) has  $x^{16}+x^{12}+x^5+1$  for many standard modem protocols.  $x^{16}+x^{15}+x^2+1$  used by IBM.

*Navstar Global Positioning System.* The codes transmitted by the Navstar satellites are deterministic binary sequences with noise-like properties. Two codes are transmitted: the C/A, or coarse/acquisition, code and the P, or precision, code. Both codes are generated using linear feedback shift registers (LFSRs).

The C/A-code is generated by two 10-bit LFSRs. One of these is a  $1 + x^3 + x^{10}$  register referred to as G1. The

other, G2, has polynomial representation  $1 + x^2 + x^3 + x^6 + x^8 + x^9$

$+ x^{10}$ . In this case, the output comes not from cell 1 but from a second set of taps. Various pairs of these second taps are binary added. The different pairs yield the same sequence but with different delays or shifts (as given by the 'shift and add' or 'cycle and add' property of LFSR sequences: a chip-by-chip sum of a LFSR sequence and any shift of itself is the same sequence except for some shift). The delayed version of the G2 sequence is binary added to the output of G1. That becomes the C/A-code. The G1 and G2 shift registers are set to the all ones state in synchronism with the epoch of the X1 code used in the generation of the P-code (see below). The various alternative pairs of G2 taps (delays) are used to generate the complete set of 36 unique PRN C/A-codes. This family of codes is a subset of codes known as Gold codes which have the property that any two have a very low cross correlation (are nearly orthogonal).

There are actually 37 PRN C/A-codes, but two of them (34 and 37) are identical. The first 32 codes are assigned to satellites. Codes 33 through 37 are reserved for other uses such as for ground transmitters.

The P-code generation follows the same principles as that for the C/A-code, except four, 12-cell shift registers are used. Two registers are combined to produce the X1 code, which is 15 345 000 chips long and repeats every 1.5 seconds; and two registers are combined to produce the X2 code, which is 15 345 037 chips long. The X1 and X2 codes can be combined with 37 different delays on the X2 code to produce 37 different one-week segments of the P-code. Each of the first 32 segments is associated with a different satellite. The remaining 5 are reserved for other uses in conjunction with the last 5 C/A-codes.

### Other checksums

International Book Standard Number ISBN

$$10d_1 + 9d_2 + 8d_3 \dots = 0 \pmod{11}$$

Mastercard

$$2^1 d_1 + d_2 + 2^2 d_3 \dots = 0 \pmod{10} \text{ where } ^\wedge \text{ is multiplication and digit add.}$$

These methods could not in all honesty be said to be particularly elegant.

## VHDL for Shift Register (after Wakerly)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

entity shift_reg is
  port (
    CLK, RESET, RIN, LIN: in STD_LOGIC;
    S: in STD_LOGIC_VECTOR (2 downto 0);
    D: in STD_LOGIC_VECTOR (7 downto 0);
    Q: in STD_LOGIC_VECTOR (7 downto 0)
  );
end shift_reg;

architecture behavior of shift_reg is
  signal IQ: STD_LOGIC_VECTOR (7 downto 0);
begin
  process (CLK,RESET,IQ)
  begin
    if (RESET='1') then IQ <= "00000000";
    elsif (CLK'event and CLK='1') then
      case unsigned(S) is
        when 0 => null; -- Hold
        when 1 => IQ <= D; -- Load
        when 2 => IQ <= RIN & IQ(7 downto 1); -- Shift Right
        when 3 => IQ <= IQ(6 downto 0) & LIN; -- Shift Left
        when 4 => IQ <= IQ(0) & IQ(7 downto 1); -- Rotate Right
        when 5 => IQ <= IQ (6 downto 0) & IQ(7); -- Rotate Left
        when 6 => IQ <= IQ(7) & IQ (7 downto 1); -- Arithmetic Right
        when 7 => IQ <= IQ (6 downto 0) & '0'; -- Arithmetic Left
        when others => null;
      end case;
    end if;
    Q <= IQ;
  end process;
end behavior;
```

## VHDL for simple ring counter (realistic, but not self correcting)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

entity ring_counter2 is
  port ( CLK, RESET: in STD_LOGIC;
         Q: in STD_LOGIC_VECTOR (3 downto 0) );
end ring_counter2;

architecture behavior of ring_counter2 is
begin
  process(CLK,RESET)
  begin
    if RESET='1' then
      Q <= '0001';
    else
      Q <= Q(2 downto 0) & Q(3);
    end if;
  end process;
end behavior;
```

## VHDL for simple ring counter (completely over the top)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

entity ring_counter is
  port (
    CLK, RESET: in STD_LOGIC;
    Q: in STD_LOGIC_VECTOR (3 downto 0)
  );
end ring_counter;

architecture behavior of ring_counter is

  type state is (one,two,three,four);
  signal PRESENT_STATE, NEXT_STATE : state;

begin

  -- process makes sure that the state machine ticks every clock cycle
  -- and that those ticks happen on the +ve edge. Generic stuff

  process(CLK,RESET)
  begin
    if RESET='1' then
      PRESENT_STATE <= one;
    elsif CLK='1' and CLK'event then
      PRESENT_STATE <= NEXT_STATE;
    end if;
  end process;

  -- process that makes this state machine unique.

  process(PRESENT_STATE)
  begin
    case PRESENT_STATE is
      when one =>
        Q <= "0001";
        NEXT_STATE <= two;
      when two =>
        Q <= "0010";
        NEXT_STATE <= three;
      when three =>
        Q <= "0100";
        NEXT_STATE <= four;
      when four =>
        Q <= "1000";
        NEXT_STATE <= one;
      when others =>
        Q <= "0001";
        NEXT_STATE <= two;
    end case;
  end process;

end behavior;
```