GFSR Pseudorandom Number Generators (RNGs)
VLP RNGs

• Recap: the linear congruential method for pseudorandom number generation
  – Commonly used
  – Easily implemented in any high level language
  – The period length of its random number stream is limited by the underlying machine’s word size
  – A serious issue, since at present day computer speeds, a simulation run could exhaust such a random number stream in a few hours
    • Distribution sampling methods such as accept-reject, consume multiple random values

• Very long period (VLP) pseudorandom number generators remedy the relatively short period deficiency of LCM generators
  – Generalized feedback shift register (GFSR) pseudorandom number generators are one class of VLP pseudorandom number generators
    • Based on algebraic manipulation of primitive trinomials of high order.
Simulation Use of RNGs

- Frequently need to isolate (or duplicate) a sequence of events
  - Technique is to manage sequence of events by its own pseudorandom number stream
    - Eliminates possibility for the action of another sequence representing events of some other type altering the stream
- Simulation languages need to provide for multiple pseudorandom number streams
  - General purpose high level languages provide a single pseudorandom number stream
    - If an LCM generator is used, the stream is likely to be too short
  - Multiple entry points along a VLP stream effectively provides for multiple streams
LCM vs. GFSR RNGs

- The quality of a pseudorandom number stream is determined by applying various statistical tests for randomness
  - The linear congruential method (LCM) of Lehmer, has been shown (by Marsaglia) to have n-space uniformity problems
  - LCM generators are popular because their computational characteristic are simple
    - Period is too short for modern application
    - Randomness characteristics can be quite good except for short period
- The generalized feedback shift register (GFSR) pseudorandom number generator is generally traced to a seminal paper by Lewis and Payne ["Generalized Feedback Shift Register Pseudorandom Number Algorithm," Journal of the ACM (3), 1973, pp. 456-468]
  - It provides a pseudorandom number generator whose randomness characteristics under the various statistical tests for randomness match those of LCM generators, with better n-space uniformity
  - Basis for much of the development work in VLP pseudorandom number generation
  - More easily implemented VLPs have been derived along similar lines
Basis for GFSR RNGs

• The underlying basis for GFSR generators is couched in modern algebra
  – An added mathematical element to be understood (as we shall see).

• GFSR generators do not have an obvious approach for initializing the table employed in the feedback-shift process
  – The procedure utilized by Lewis and Payne was one of brute force
  – Collings and Hembree ["Initializing GFSR Pseudorandom Number Generators,” Journal of the ACM (33), 1986, pp. 706-711.] provided for a much superior initialization strategy involving manipulation of polynomials over the Galois field of order 2 ($\mathbb{Z}_2$).

• Multiple pseudorandom number streams are provided by simply generating as many tables as necessary
Algebraic Background

• The two element field $\mathbb{Z}_2$ is trivial in the sense that in and of itself there isn't much to it
  – Only has two elements
  – Provides the basic building block from which all digital computer hardware and software logic derives
  – Implicit algebraic connection to the integers modulo $m$ (the basis for finite mathematics and computer arithmetic)
  • Reason we often work with $\mathbb{Z}_2$ rather than an alternate viewpoint, such as Boolean logic.
Two element field $\mathbb{Z}_2$

- The two operations $+$ and $\cdot$ are used
  - Correspond to addition and multiplication of ordinary integers
  - Separate the integers into the two equivalence classes of the even integers and the odd integers
  - Represent the even integers by 0 and the odd integers by 1, then the two operations are "inherited" from their behavior on the equivalence classes (i.e., even $+$ even is even, even $+$ odd is odd, even $\times$ odd is even, etc)
  - Result is the operation tables

<table>
<thead>
<tr>
<th>$+$</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\cdot$</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

- The $+$ operation corresponds to the Boolean $\oplus$ operation (exclusive OR) and the $\cdot$ operation is Boolean AND. Note that Boolean $\oplus$ is also the logical operation $\neq$. 
Polynomials over $\mathbb{Z}_2$

- Polynomials have been extensively studied in mathematics for centuries
  - If we restrict attention to polynomials with coefficients in $\mathbb{Z}_2$ and extend the $+$ and $\cdot$ operations to apply to the polynomials
    - Since $1 + 1 = 0$ in $\mathbb{Z}_2$, we have the nice reduction $x + x = 0$ when looking at polynomials over $\mathbb{Z}_2$

- The collection of all polynomials with coefficients restricted to 0 and 1 from $\mathbb{Z}_2$ and using the $+$ and $\cdot$ operations is denoted by $\mathbb{Z}_2[x]$

- The term **ring** is used to describe algebraic systems of two operations that have the same kinds of properties as addition and multiplication of numbers
  - $\mathbb{Z}_2[x]$ is called the **ring of polynomials over $\mathbb{Z}_2$**
Ring Types

• Algebraists classify rings into different types, generally based on properties they share with common algebraic systems
  – The ring of matrices is a non-commutative ring

• The ring of polynomials $\mathbb{Z}_2[x]$, has properties analogous to ones normally associated with the integers
  – There are polynomials in $\mathbb{Z}_2[x]$, called **irreducible polynomials**, that are analogous to the prime numbers of the ring of integers $\mathbb{Z}$ ($\mathbb{Z}$ is from the German word for number, *Zahl*)
Computer Arithmetic

• The manner in which computer memory is addressed dictates that a power of 2 is the natural word size \( n \) to employ (the word size is the number of bits used to represent an integer).

• The most natural operation at the circuit level for a computer word size of \( n = 2^k \) is arithmetic modulo \( n \)
  – Represents the quotient ring \( \mathbb{Z} / 2^k \)
  – The elements of \( \mathbb{Z} / 2^k \) are just the equivalence classes of the integers modulo \( 2^k \) with operations of addition and multiplication as inherited from the (infinite) algebraic system we know as the integers
  – The number of elements (the order of \( \mathbb{Z} / 2^k \)) is automatically \( 2^k \)
    • Represents all possible sequences of 0's and 1's that can be stored in a computer word.
Zero Divisors

- A practical issue of computer arithmetic is the presence of "0-divisors", numbers which when multiplied together give 0 (the circuitry reports this as an overflow condition).
  - Overcoming this limitation requires adding additional features, such as floating point arithmetic (based on "scientific notation"), an inherently imprecise mechanism if used for integer arithmetic.

- The "arithmetic precision" can be increased by using algorithms for processing bit-strings whose length exceeds the "natural integers" of the machine's $2^k$ word size.

- The natural representation of integers in computer memory as integers modulo $2^k$ eliminates any natural means of eliminating 0-divisors, since integer arithmetic modulo $p$ has 0-divisors unless $p$ is a prime or $p = 1$ (a trivial case).
Finite Fields

• 0-divisors do not occur if the modulo base is a prime $p$, in which case the integers modulo $p$ (designated $\mathbb{Z}_p$) not only has no zero divisors, but also has the property that non-zero elements have multiplicative inverses
  – This type of ring is called a field, the most familiar examples of which are infinite, the real numbers $\mathbb{R}$, and the complex numbers $\mathbb{C}$

• Algebraists have been able to show that every finite field has order $p^k$ for some prime $p$ and that finite fields are unique; i.e., any two finite fields of the same order are equivalent

• Additionally, the multiplicative group of a finite field is cyclic, meaning it has an element $c$ such that the sequence of powers $1, c, c^2, c^3, \ldots$ generate the elements of the field

• For $p=2$, there is a natural finite field structure for 32-bit memory in addition to integer arithmetic mod 32
Finding Finite Fields

• We get a finite field of order $p$ for $p$ a prime from the integers modulo $p$

• Similarly we get a finite field of order $p^k$ from $\mathbb{Z}_p[x]$ by factoring modulo an irreducible polynomial order $k$
  – The question is what element is a cyclic generator for the field’s multiplicative group?

• For the sequence $1, x, x^2, x^3, \ldots$ some power $x^s$ must cycle back to 1 modulo the irreducible polynomial $f(x)$
  
  You can see this since for some $p$ you have to re
Fields of order $2^k$

- For irreducible polynomial $f(x) \in \mathbb{Z}_2[x]$ of order $k$, $\mathbb{Z}_2[x]/(f(x))$ is the finite (Galois) field of order $2^k$
  - In general, the notation $\mathbb{R}[x]/(f(x))$ represents the ring of polynomials $\mathbb{R}[x]$ modulo the polynomial $f(x)$

- This tells us how to get our hands on all finite fields whose order is a power of 2, subject to being able to find the necessary irreducible polynomials

- This makes the issue of finding irreducible polynomials that are primitive pretty important
  - Keep in mind that people continue to try to unravel the mystery of prime numbers and you will gain some additional appreciation for the potential degree of difficulty this may entail
Special Properties of Finite Fields

- Recall that in general, the multiplicative group of any finite field is **cyclic**, which means that there is some non-zero element whose powers cycle through all of the non-zero elements of the field.

- We now know that a finite field has order $n = p^k$ for some prime $p$, so for $p = 2$ we just need an irreducible polynomial $f(x)$ in $\mathbb{Z}_2[x]$ of degree $k$ to generate the field of order $2^k$.

- Within $\mathbb{Z}_2[x]/(f(x))$ the obvious cyclic generator works if $f(x)$ is primitive; namely, the powers of $x$ modulo $f(x)$. In other words, we get the $n - 1$ non-zero elements of $\mathbb{Z}_2[x]/(f(x))$ by computing

  $$1 = x^0, x^1, x^2, x^3, ..., x^{n-2}, x^{n-1} = x^0 = 1,$$

  where $n = 2^k$,

  reducing modulo $f(x)$ as we get powers larger than $k-1$.

- This is algebraically guaranteed to cycle through all the non-zero polynomials of degree less than $k$ (the degree of $f(x)$).
Representing the Polynomials of Degree $< k$

- The polynomials of degree less than $k$ can be represented by the $2^k$ sequences of 0's and 1's representing their coefficients; i.e.,

\[
x^0 \quad x^1 \quad x^2 \ldots \quad x^{k-1}
\]

<table>
<thead>
<tr>
<th>coefficients</th>
<th>corresponding polynomial</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 . . . 0</td>
<td>$\equiv 0$</td>
</tr>
<tr>
<td>1 0 0 . . . 0</td>
<td>$\equiv 1$</td>
</tr>
<tr>
<td>0 1 0 . . . 0</td>
<td>$\equiv x$</td>
</tr>
<tr>
<td>1 1 0 . . . 0</td>
<td>$\equiv 1 + x$</td>
</tr>
<tr>
<td>0 0 1 . . . 0</td>
<td>$\equiv x^2$</td>
</tr>
<tr>
<td>1 0 1 . . . 0</td>
<td>$\equiv 1 + x^2$</td>
</tr>
<tr>
<td>0 1 1 . . . 0</td>
<td>$\equiv x + x^2$</td>
</tr>
<tr>
<td>1 1 1 . . . 0</td>
<td>$\equiv 1 + x + x^2$</td>
</tr>
</tbody>
</table>

- These are the base representatives of the elements of $\mathbb{Z}_2[x]/(f(x))$ just as 0, 1, 2, ..., $2^k-1$ are the base representatives for the integers modulo $2^k$. 
Example: Irreducible not a Primitive

- For $\mathbb{Z}_2[x]$, $1+x+x^2+x^3+x^4$ is an irreducible polynomial of degree 4
  - For $f(x) = 1+x+x^2+x^3+x^4$, $x^4 \equiv (1+x+x^2+x^3) \mod f(x)$
  - The field is of order $16 = 2^4$
  - The sequence of powers of $x$ is
    
    \[
    \begin{array}{c|c|c|c|c}
      n & 1 & x & x^2 & x^3 \\
    \hline
      x^4 & \equiv & 1+x+x^2+x^3 & \equiv & 1+x+x^2+x^3+x^4 \equiv 1 \\
    \end{array}
    \]

- $f(x)$ is not primitive since $x$ is not a cyclic generator for the multiplicative group of $\mathbb{Z}_2[x] \mod f(x)$
- $g(x) = (1+x)$ is a cyclic generator for the multiplicative group of this field
Example: Irreducible a Primitive

- For $\mathbb{Z}_2[x]$, $1+x+x^4$ is irreducible of degree 4
  - For $f(x) = 1+x+x^4$, $x^4 \equiv (1+x) \mod f(x)$
  - The sequence of powers of $x$ is
    
    $\begin{align*}
    1 & \quad \rightarrow \quad 1000 \\
    x & \quad \rightarrow \quad 0100 \\
    x^2 & \quad \rightarrow \quad 0010 \\
    x^3 & \quad \rightarrow \quad 0001 \\
    x^4 & \equiv 1+x \\
    x+x^2 & \quad \rightarrow \quad 1100 \\
    x^2+x^3 & \quad \rightarrow \quad 0110 \\
    1+x+x^3 & \quad \rightarrow \quad 0011 \\
    1+x^2 & \quad \rightarrow \quad 1101 \\
    1+x^3 & \quad \rightarrow \quad 1010 \\
    x+x^3 & \quad \rightarrow \quad 0101 \\
    1+x+x^2 & \quad \rightarrow \quad 1110 \\
    x+x^2+x^3 & \quad \rightarrow \quad 0111 \\
    1+x+x^2+x^3 & \quad \rightarrow \quad 1111 \\
    1+x^2+x^3 & \quad \rightarrow \quad 1110 \\
    1+x^3 & \quad \rightarrow \quad 1001 
    \end{align*}$
  - It is primitive
Tausworthe Sequence

- If primitive polynomial \( f(x) \) over \( \mathbb{Z}_2[x] \) has degree \( k \), there are \( 2^k - 1 \) non-zero elements since the field has \( 2^k \) elements (including 0)
- When the sequence of polynomials (all of degree < \( k \)) is evaluated at \( x=1 \) (addition is mod 2; i.e., exclusive OR), the Tausworthe sequence results [ "Random Numbers Generated by Linear Recurrence Modulo Two," Math. Comput. (19), 1965, pp. 201-209]
- This is the sequence of 0's and 1's of period \( 2^k - 1 \) described by Lewis and Payne for defining GFSR RNGs
A Small Example

- $f(x) = 1 + x^2 + x^5$ is a primitive trinomial in $\mathbb{Z}_2[x]$ (take it on faith)
  - Small enough to illustrate the computational processes involved because the multiplicative group of $\mathbb{Z}_2[x]/(f(x))$ in this case only has $2^5 - 1 = 31$ terms (note that all irreducibles are primitive)

- Since $f(x) \equiv 0 \mod (f(x))$
  - $f(x) = 1 + x^2 + x^5 = 0$

and over $\mathbb{Z}_2$ we always have $t + t = 0$

it follows that

$\sqrt{x} + x^2 + x^5 + \sqrt{x} + x^2 = 0 + 1 + x^2$

so we get the reduction

$x^5 = 1 + x^2$
Tausworthe Sequence Example

• Then using this reduction on the powers of x we get:

\[x^0 = 1, \ x, \ x^2, \ x^3, \ x^4,\]
\[x^5 = 1 + x^2,\]
\[x^6 = xx^5 = x(1 + x^2) = x + x^3,\]
\[x^7 = xx^6 = x(x + x^3) = x^2 + x^4,\]
\[x^8 = xx^7 = x(x^2 + x^4) = x^3 + x^5 = 1 + x^2 + x^3,\]
\[x^9 = xx^8 = x(1 + x^2 + x^3) = x + x^3 + x^4,\]
\[x^{10} = xx^9 = x(x + x^3 + x^4) = x^2 + x^4 + x^5 = x^2 + x^4 + 1 + x^2 = 1 + x^4,\]
\[\ldots\]
\[x^{20} = x^2 + x^3,\]
\[\ldots\]

• If we evaluate each of these at \(x = 1\) we get the Tausworthe sequence

\[1\ 1\ 1\ 1\ 1\ 0\ 0\ 0\ 1\ 1\ 0\ 1\ 1\ 1\ 0\ 1\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 1\ 0\ 1\ 1\ 0\ 0\]

which has an appearance of randomly distributed 0's and 1's once past the initial run of 1's
General Procedure for Getting the Powers of $x$

- It is always the case for polynomial $f(x)$ of degree $p$ in $\mathbb{Z}_2[x]$ that $f(x) = g(x) + x^p$ for some $g(x)$ of degree $< p$, so $x^p = g(x) \mod (f(x))$.
- Hence, $x^p$ may be obtained from those of the $p$ terms $x^0, x^1, x^2, \ldots, x^{p-1}$ for which $g(x)$ has non-zero coefficients.
- This is the basis for a linear recurrence relation, where the $i^{\text{th}}$ term is obtained from those of the preceding $p$ terms that correspond to the non-zero positions in $g(x)$.
- Computation is simplified if $f(x)$ is chosen to be a primitive trinomial of the form $f(x) = 1 + x^q + x^p$, in which case the linear recurrence is given by $x^i = x^{(i-p)} + x^{(i-p)+q}$.

$$x^i = x^p x^{(i-p)}$$
$$= (1 + x^q)x^{(i-p)} = x^{i-p} + x^{(i-p)+q}$$
Primitive Trinomials

- Primitive trinomials are important enough that lists of them have been published:
  - Florent Chabaud’s exhaustive lists are at

- For example,
  - $1 + x^{15} + x^{49}$ is primitive and
  - $1 + x^{32} + x^{521}$ is primitive

- The first one produces a Tausworthe sequence of period $2^{49} - 1$ and the other of period $2^{521} - 1$, a very long period indeed
  - Continued study has revealed enormously larger polynomials in this class
Recent Work on Trinomials

• Richard Brent and Paul Zimmermann

The largest known primitive trinomials as of June-July, 2007 are

\[ 1 + x^{8785528} + x^{24036583} \quad \text{and} \]
\[ 1 + x^{8412642} + x^{24036583} \]

both of degree 24,036,583
Using the Tausworthe Sequence to Produce Random Values

• The Tausworthe sequence always will start out predictably, since the first p terms are obtained from \( x^0, x^1, x^2, ..., x^{p-1} \) which evaluates to a sequence of p 1's
  – Generally desirable randomness properties only begin to occur after progressing well out into the sequence

• Primitive trinomials of the form \( f(x) = 1 + x^q + x^p \) are used for simplicity

• The idea is to maintain a window of length p into the Tausworthe sequence in a shift register of length p
Shift Register Use

• The \( + \) operation of \( \mathbb{Z}_2 \) is the same as Boolean XOR.
• The bit to be shifted in is calculated via feedback from positions 0 and \( q \) using the linear recurrence
  \[
  x_i = x_{i-p} + x_{(i-p)+q}
  \]
  position 0 of register  position \( q \) of register
• With the bit shifted in calculated in this manner by the Boolean XOR operation applied to the \( 0^{th} \) and \( q^{th} \) bits, the effect is to shift a window of length \( p \) along the Tausworthe sequence
• If the hardware register support is provided, at the microcode level this can be accomplished in a single CPU cycle!
Shift Register Example

- \( x^{i-5} + x^{i-3} = x^i \)
- In the earlier example, from the irreducible polynomial \( f(x) = 1 + x^2 + x^5 \) we obtained the Tausworthe sequence:

\[
1 \ 1 \ 1 \ 1 \ 1 \ 0 \ 0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 1 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 1 \ 0 \ 1 \ 1 \ 0 \ 0
\]

\( x^{i-5} + x^{i-3} = x^i \)

\[ \text{left shift, } x^i \text{ shifted in} \]
\[ \text{(window moves right)} \]

- From \( x^5 = 1 + x^2 \) we compute \( x^{i-5} x^5 = x^{i-5} + x^{i-5} x^2 \) giving

\[
x^i = x^{i-5} + x^{i-5+2}
\]

so the recurrence relation is given by

\[
x^i = x^{i-5} + x^{i-3}
\]
Getting an RNG from the Tausworthe Sequence

• Lewis and Payne ["Generalized Feedback Shift Register Pseudorandom Number Algorithm," *Journal of the ACM* (3), 1973, pp. 456-468] turned the basic idea into an RNG

• Procedure is to arrange \( p \) segments of the Tausworthe sequence each of length \( p \) in columns, each segment "delayed" by some constant amount \( d \) from the preceding one
  – \( p \) shift registers are arranged in this fashion to produce a \( p \times p \) array of \( p \)-bit words.

• It can be shown that maximum period length on these words \( (2^p-1) \) is achieved if the \( p \times p \) matrix has rank \( p \) (linearly independent columns)

• This condition is satisfied if \( 0 < d < 2^p - 1 \) and \( d \) is relatively prime to \( 2^p - 1 \)

• For a pseudorandom number generator producing \( L \)-bit words \( (L \leq p) \), only the first \( L \) columns are employed
Obtaining Table Columns

- For $f(x) = 1 + x^2 + x^5$, recurrence relation $x^i = x^{i-5} + x^{i-3}$, and its sequence starting from position 5 with delay 20

<table>
<thead>
<tr>
<th>Column 1</th>
<th>Column 2</th>
<th>Column 3</th>
<th>Column 4</th>
<th>Column 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**delay** 20
Matrix of 5-bit Registers

- The matrix with columns starting from position 5 in the Tausworthe sequence with a column to column delay of 20 is given by

<table>
<thead>
<tr>
<th>21</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>18</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>25</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Getting a VLP RNG

• For the primitive polynomial
  \[1 + x^{32} + x^{521}\]
  with \(p = 521\), we clearly get a random stream using the generalized feedback shift process of enormous period \((2^{521} - 1)\), a clear example of a VLP RNG

• With the typical machine word size of \(L = 64\), it is clear that we don't need but the first 64 columns of the \(p \times p\) array to produce 64-bit random values.
  – Normally, the \(p \times L\) array consisting of the first \(L\) columns is used
  – 457 of the possible 521 columns in the generating matrix are not used
  – In the complete cycle of length \(2^{521} - 1\) for this generator, any combination of 0's and 1's in the first 64 columns occurs \(2^{457}\) times when considering all possible combinations across the full width of 521
    • Each of the \(2^{64}\) integers that can be represented on the machine occurs \(2^{457}\) times in the sequence!
    • Since the basic LCM generator of period length \(2^{31} - 1\) is inherently limited to the values in its 32-bit computation space, it does not have the capacity for providing repetition in its random number stream
GFSR Weaknesses

• One weakness for the GFSR generator for the above polynomial is that the control matrix of 64 521 bit registers (about 4K bytes) needs to be provided for in the hardware

• Another weakness is the lack of means for easily initializing the generator
  – Lewis and Payne provided only a brute force mechanism
  – A much superior approach is described in
GFSR Table Initialization

- Lewis and Payne chose to work directly from the basic Tausworthe sequence in initializing the $p \times L$-bit matrix
  - Their initialization applied the linear recurrence repeatedly
    - starting from the initial sequence of $p$ 1's as the first column, and employing a delay of $d = 100p$
    - To offset the effect of the initial column of all 1's, they cycled the matrix an additional $r$ times ($r = 5000p$)
    - With these values, Collins and Hembree ("Initializing GFSR Pseudorandom Number Generators," *Journal of the ACM* (33), 1986, pp. 706-711) note that 637,098 terms must be computed when $p=98$ and $L=15$
- In a similar vein, Arvillias and Maritsas ("Partitioning the Period of a Class of m-Sequences and Application to Pseudorandom Number Generation," *Journal of the ACM* (25), 1978, pp. 675-686) argue that the limitations of these somewhat arbitrary delay values may produce correlation problems in some instances (lack of independence between evenly spaced numbers in the sequence)
  - They suggest a delay $d$ of $(2p-1)/L$
    - Impractical when $p = 521$ and $L = 64$ and a brute force initialization is employed
Modern Algebra to the Rescue (1)

Collings and Hembree provide a computationally reasonable means of providing initialization at any distance out in the Tausworthe sequence

1. It is easy to calculate the polynomials $x^p$, $x^{2p}$, $x^{4p}$, ... where each is the square of the preceding one

   – Since $1 + 1 = 0$ for $\mathbb{Z}_2$, squaring is accomplished by simply doubling the exponents (then reducing modulo $f(x)$ to get a polynomial of degree less than $p$)

   – This process is algorithmically easier to deal with than it might first appear

   – For a polynomial $T_i(x) = x^{1p}$ for $t = 2^i$, represent it by a bit string $T$ of length $p$

   – Prior to reduction modulo $f(x)$, the active bit positions in $T_{i+1}(x) = (T_i(x))^2$ are given by $2$ times the active positions in $T_i(x)$; e.g.,

     if $T_i(x)$ is given by $1\ 0\ 0\ 1\ 0\ 1$ corresponding to $1 + x^3 + x^5$

     it has active positions at $0, 3, 5$

     $T_{i+1}(x) = (T_i(x))^2$ is given by $1\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 1$

     corresponding to $1 + x^6 + x^{10}$ with active positions $0, 6, 10$
Modern Algebra to the Rescue (2)

2. If we cycle out the initial amount $r$, then delay another $d$ in the Tausworthe sequence, we get the $<1^{st\text{-}row},1^{st\text{-}column}>$ element of the GFSR table

- The power of $x$ that produces this element is $x^d x^r$
- If we are clever and take as the values for the initial delay of $r$, and the repeating delay of $d$ numbers of the form $2^n p$, say $d=2^n p$ and $r=2^m p$, then we will be able to cast this using the $T_i$'s
  - What we really want is the polynomial of degree $< p$ that we get by reducing this power of $x$ modulo our primitive trinomial
  - This is obtained by reducing

$$S_{d+r}(x) = x^d x^r = T_n(x) T_m(x) \text{ modulo } f(x).$$

- If $p=521$ and $n=100$, then 101 iterations, $x^p$, $x^{2p}$, $x^{4p}$, ..., will be required to obtain $T_n(x)$
  - $T_m(x)$ may be picked up along the way
  - Compared to the brute force method, relatively few iterations gets us a long way out in the sequence!
Modern Algebra to the Rescue (3)

3. Moving along the 1st row, we delay another d in getting to the 1st element of the next column
   - This means that the value comes from $x^{2d+r}$ so we need to reduce
     $$S_{2d+r}(x) = x^{2d+r} = S_{d+r}(x) \quad T_n(x) = (T_n(x))^2 \quad T_m(x) \quad \text{modulo } f(x)$$
   - Successive multiplication by $T^n(x)$ (with reduction modulo $f(x)$) produces the sequence of polynomials for generating the 1st row of the GFSR table
     - Call this sequence of polynomials $W_0$
Modern Algebra to the Rescue (4,5)

4. The sequence of polynomials for the next row of the GFSR table is produced by multiplying each element of $W_0$ by $x$
   - The polynomials generating the Tausworthe sequence still correspond to powers of $x$ even when reduced modulo $f(x)$
   - Iteratively repeating this process will produce the generating polynomials for each row $W_i$ of the GFSR table.

5. For each row, the sequence of $L$ polynomials $W_i$ can be represented as an $L \times p$ matrix, each row representing the corresponding polynomial of degree $< p$ in $W_i$
   - The $L$ terms generated by applying XOR across each row of the $L \times p$ matrix produces the $i^{th}$ row of the GFSR table
   - ie., each of the polynomials in $W_i$ is evaluated at $x = 1$
Implementation Considerations

- The actual reduction modulo $f(x)$ and polynomial multiplication over $\mathbb{Z}_2$ are just bit manipulation exercises easily accommodated in a high level language such as C.
- After calculation of $W_0$, for the primitive trinomial $x^{521} + x^{32} + 1$ and a 64 bit table, there is a single polynomial reduction required for each of the remaining 33,343 positions ($33,343 = 521 \times 64 - 1$).
The Small Example Revisited

- Our small example used
  \[ f(x) = 1 + x^2 + x^5 \]

- This polynomial is an primitive trinomial in \( \mathbb{Z}_2[x] \)
  - Although there are only 31 elements in \( \mathbb{Z}_2[x]/(f(x)) \), there are enough terms to fully illustrate the construction process without overwhelming amounts of computation

- The powers of \( x \) are
  
  \[
  \begin{align*}
  x^5 &= T_0(x) = 1 + x^2, \\
  x^6 &= x + x^3, \\
  x^7 &= x^2 + x^4, \\
  x^8 &= 1 + x^2 + x^3, \\
  x^9 &= x + x^3 + x^4, \\
  x^{10} &= T_1(x) = 1 + x^4, \\
  x^{11} &= 1 + x + x^2, \\
  x^{12} &= x + x^2 + x^3, \\
  x^{13} &= x^2 + x^3 + x^4, \\
  x^{14} &= 1 + x^2 + x^3 + x^4, \\
  x^{15} &= 1 + x + x^2 + x^3 + x^4, \\
  x^{16} &= 1 + x + x^3 + x^4, \\
  x^{17} &= 1 + x + x^4, \\
  x^{18} &= 1 + x, \\
  x^{19} &= x + x^2, \\
  x^{20} &= T_2(x) = x^2 + x^3, \\
  x^{21} &= x^3 + x^4, \\
  x^{22} &= 1 + x^2 + x^4, \\
  x^{23} &= 1 + x + x^2 + x^3, \\
  x^{24} &= x + x^2 + x^3 + x^4, \\
  x^{25} &= 1 + x^3 + x^4, \\
  x^{26} &= 1 + x + x^2 + x^4, \\
  x^{27} &= 1 + x + x^3, \\
  x^{28} &= x + x^2 + x^4, \\
  x^{29} &= 1 + x^3, \\
  x^{30} &= x + x^4
  \end{align*}
  \]

  which produces the Tausworthe sequence
  \[
  1 1 1 1 1 0 0 0 1 1 1 0 1 0 1 0 0 0 0 1 0 0 1 0 1 1 0 0
  \]
Determining $W_i$'s

- For delay $d = 22 \times 5 = 20$ and $r = 20 \times 5 = 5$, the GFSR table generating polynomials $W_0, W_1, W_2, W_3, W_4$ are given by

$$W_0 = (x^{25} = T_2(x) T_0(x) = 1 + x^3 + x^4), (T_2(x)^2 T_0(x) = 1 + x^2 + x^3 + x^4),$$
$$\quad (T_2(x)^3 T_0(x) = x^3), (T_2(x)^4 T_0(x) = 1 + x + x^2 + x^3),$$
$$\quad (T_2(x)^5 T_0(x) = x + x^2 + x^3)$$

$$W_1 = (1 + x + x^2 + x^4), (1 + x + x^2 + x^3 + x^4), (x^4),$$
$$\quad (x + x^2 + x^3 + x^4), (x^2 + x^3 + x^4)$$

$$W_2 = (1 + x + x^3), (1 + x + x^3 + x^4), (1 + x^2),$$
$$\quad (1 + x^3 + x^4), (1 + x^2 + x^3 + x^4)$$

$$W_3 = (x + x^2 + x^4), (1 + x + x^4), (x + x^3),$$
$$\quad (1 + x + x^2 + x^4), (1 + x + x^2 + x^3 + x^4)$$

$$W_4 = (1 + x^3), (1 + x), (x^2 + x^4),$$
$$\quad (1 + x + x^3), (1 + x + x^3 + x^4)$$
GFSR Table

- Evaluating $W_0, W_1, W_2, W_3, W_4$ at $x=1$ yields the GFSR table

\[
\begin{align*}
1 & 0 & 1 & 0 & 1 & \equiv_{10} & 21 & \leftarrow \text{feedback position} \\
0 & 1 & 1 & 0 & 1 & \equiv_{10} & 13 \\
1 & 0 & 0 & 1 & 0 & \equiv_{10} & 18 & \leftarrow \text{feedback position} \\
1 & 1 & 0 & 0 & 1 & \equiv_{10} & 25 \\
0 & 0 & 0 & 1 & 0 & \equiv_{10} & 2
\end{align*}
\]
1st Iteration

- The 1st iteration of the GFSR pseudorandom number generator will combine the 2 feedback positions to yield 7 and the GFSR table becomes:
  
  \[
  \begin{align*}
  01101 & \equiv_{10} 13 \\
  10010 & \equiv_{10} 18 \\
  11001 & \equiv_{10} 25 \\
  00010 & \equiv_{10} 2 \\
  00111 & \equiv_{10} 7
  \end{align*}
  \]

- Once the table is in place, actual random number generation is computationally efficient, since it involves only shift and exclusive OR operations.
  
  - Can be accomplished in 1 machine cycle at the hardware level!

- Significant computational time is necessary to generate GFSR tables, but only needs to be done once for each random number stream employed.
What Next?

- The Mersenne Twister (MT) RNG is a derivative of the twisted GFSR RNG of Matsumoto and Kurita [“Twisted GFSR generators II,” ACM Transactions on Modeling and Computer Simulation (4), 1994, pp. 254-266]
  - The technique overcomes the initialization and seeding problems associated with the original GFSR approach.

- Openware code for a Mersenne Twister of period $2^{19937}-1$. can be obtained from http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html
  - At this site, Matsumoto provides a C implementation requiring only 624 words of memory
    - The generator benchmarks at 4 times faster than rand().
  - The sequence produced is not cryptographically secure
    - for cryptographic use an appropriate secure hashing algorithm (non-invertible function compressing several words into one word) has to be additionally employed

- There are other VLP RNGs that have now been developed

- Research is ongoing
  - A battery of statistical tests (Marsaglia’s “Diehard” collection) is available from http://stat.fsu.edu/pub/diehard/