

Chapter 2: Shift Register Generators

Chapter Contents (partial)

2.1. The Maximum Period of a Shift Register Generator

Definitions: state vector period, output period

Fact 2: In a shift register generator with k registers each of which can store Q symbols, neither the state vector period nor the output period can exceed $Q^k - 1$.

2.2 The State Vector Sequence of a Shift Register Generator

The Figure 2 Iteration Equation

A note on Symbols

Galois Field Review

Fact 3: Any monic $h(x)$ of degree k that is irreducible over $GF(p)$ is the minimum polynomial of the element $\{x\}$ of $GF(q=p^k)$.

The Figure 2 Galois Iterator

Fact 5: The state vector of a (Figure 2) shift register generator steps periodically through some subset N of the non-zero elements of $GF(p^k)$, provided it is initialized in some non-zero state. The state vector period N must integrally divide $p^k - 1$, so $N \leq p^k - 1$. The integer N is the order of $\{x\}$.

Fact 6: If $p^k - 1$ happens to be a prime number, the state vector period is $p^k - 1$.

Fact 8: If $h(x)$ is a *primitive polynomial* of $GF(q=p^k)$, then the state vector period is $p^k - 1$.

An example : $GF(q=2^9)$

The non-terminating Remainder in polynomial division.

Fact 10: The *state vector period* of a Figure 2 shift register generator is the same as the *period* of $h(x)$.

Fact 11: The period of any particular register of the Figure 2 circuit must integrally divide the state vector period N .

Corollary: The *output period* of a Figure 2 shift register generator must be a divisor of the state vector period N .

2.3 The Output Sequence of a Shift Register Generator

Fact 1: There are exactly p^k solutions to the difference equation (2.3.2).

Fact 5: All p^k solutions of the difference equation (2.3.2) are of the form $\{o_i\} = \{c, c, c, c \dots\}$.

An example : $GF(16)$

Fact 8: If $h(x)$ of degree k is a *primitive polynomial* of $GF(p^k)$, then all $p^k - 1$ non-zero solutions of the difference equation (2.3.2) have period $n = p^k - 1$.

Fact 9: If $h(x)$ of degree k is a *primitive polynomial* of $GF(p^k)$, then there is essentially only *one* characteristic non-zero output pattern of the shift register generator, its period is $p^k - 1$. **Fact 10:** For a Figure 1 shift register generator, the *state vector period* is the same as the *output period*.

(contents continues on next page)

2.4. Characteristic Sequences (general p)

Fact 1: Each possible state vector occurs exactly once somewhere in a characteristic sequence.

Fact 4: In a characteristic sequence, every non-zero symbol appears exactly (p^{k-1}) times, and the 0 symbol appears exactly $(p^{k-1} - 1)$ times.

Fact 5: If we line up our characteristic sequence with any delayed version of itself and compare over one period, we will find that the two sequences differ in exactly $(p-1)p^{k-1}$ places.

Binary Characteristic Sequences (p=2).

Facts 1-5: Restatements of previous facts for $p = 2$.

Fact 6: A string of $(k-1)$ 1's appears exactly once in a binary characteristic sequence.

The product of two shifted binary characteristic sequences.

Fact 7: Exactly 2^{k-2} ones appear in (one period's worth of) the product of two shifted binary characteristic sequences. If the sequences are unshifted, or are shifted by a multiple of the period, then exactly 2^{k-1} ones appear in the product.

Probability of strings of 1's and 0's.

Fact 8: In a characteristic sequence of reasonably large k , the distribution of string lengths is very close to the theoretical white sequence distribution, up to a length where strings are no longer allowed to exist. Up to this length, the relative probability of a string of exactly n identical symbols is given by $P(n) \approx 1/2^n$. For $k = 9$, the error in this estimate is $< 0.5\%$.

2.5 Autocorrelation, Probability, and Statistics for a sequence in GF(2)

A few Basics of Probability and Statistics Applied to GF(2)

Fact 1: The first and second order statistics are completely determined by the autocorrelation function.

Fact 2: For a statistically independent sequence, all higher order statistics are determined by the first order statistics

Definition: A *white* sequence over GF(2) is one which is statistically independent and has $p(a_n) = 1/2$.

Statistics for Modified GF(2): Coefficients are $(+1, -1)$.

Plots of autocorrelation functions for white sequences.

2.6 The Spectral Power Density of a Shift Register Generator

Case 1 Analysis: Singled Ended Output

Fact 1: Comparison of Figure 2.6.3 with Figure 2.5.1 shows that the characteristic sequence has an autocorrelation function that differs only slightly from that of a white sequence. For $k = 9$, $P = 511$ so $(1 + 1/P) \approx 1.002$. Thus, the first and second order statistics of a characteristic sequence are very close to those of a white sequence.

Case 2 Analysis: Differential Output

Fact 2: Apart from a possible DC line, the spectrum of the characteristic sequence output by a shift register generator is entirely continuous, and it is the envelope of the pulse shape used.

Appendix 2.1: Proof of Fact 4 of Section 2.3

Appendix 2.2: A List of Primitive Polynomials over GF(2)

Chapter 2: Shift Register Generators

2.1. The Maximum Period of a Shift Register Generator

Figure 1 of Chapter 1 shows the "scrambler-style" polynomial divider circuit, and Figure 2 shows the "standard" divider circuit. In either circuit, if we set the input to zero, preset the k registers to some values, and just let the thing run, we get some pattern out the output. Such a device is usually referred to as a *shift register generator*.

Definition: The *state vector* of any state machine circuit consisting of k registers each of which can store any of Q symbols is a k -tuple consisting of the symbol stored in each register. There are therefore Q^k possible values that this k -tuple (state vector) can have. In other words, there are Q^k states.

Definition: The *state vector period* of a state machine is the minimum number of clocks it takes to get from some starting state vector back to that same state vector. This period may or may not depend on the starting state.

Definition: The *output period* of a state machine is the minimum period of the output symbol sequence. This period is the same no matter where you start counting in the output symbol sequence. Note that the output sequence can in general be a combinatoric function of the state vector, but it is often just the output of one of the registers, as in Figure 1 or Figure 2.

Fact 1: For a state machine with k registers which can store any of Q symbols, neither the state vector period nor the output period can exceed Q^k .

Proof: The state machine takes some periodic path through its state space. Assume there are N states on this path. In this case, the state vector period is N , and the output period must be some number which integrally divides N , call it $M = N/k$. Thus, $M \leq N$. In any event, since $N \leq Q^k$, we know that both the state vector period N and the output period M cannot exceed Q^k .

Example: Consider a decade counter built from 4 flipflops. In this circuit, the state vector period is $N = 10$, while $Q^k = 2^4 = 16$. We can consider any register to be an "output". The output period of the least significant counter bit is $M = 2$. The output period of each of the other bits is $M = 10$.

Now we apply the above notions for general state machines to shift register generators. As is clear from Figure 1 and Figure 2, if the state vector becomes zero, the generator "dies". All future state vectors are 0, and the output sequence is 0. All periods are 1.

We therefore *exclude* this state vector from the following discussion. We shall for the moment assume that this state vector is never "hit", so we do not worry about the machine dying. Later we shall show why it is never hit. Thus, we have:

Fact 2: In a shift register generator with k registers each of which can store Q symbols, neither the state vector period nor the output period can exceed $Q^k - 1$.

Proof: This is just Fact 1, but we are saying that there are at most $Q^k - 1$ states, since the zero state is excluded.

2.2 The State Vector Sequence of a Shift Register Generator

In this section we discuss only the "standard" Figure 2 style shift register generator. We will apply our knowledge of Galois fields to learn what the state vector does after it is initialized in some non-zero state. In subsequent sections, we will discuss the output symbol sequence and also the state vector sequence of the Figure 1 type generator.

A spinoff of this section will be a knowledge of what happens to the sequence of remainders of a polynomial division when the input sequence of a polynomial divider is set to zero, after there has been some non-zero input.

The Figure 2 Iteration Equation

Figure 2 shows k registers q_1 through q_k . Construct the following polynomial:

$$q(x) = q_1 + q_2 x + q_3 x^2 + \dots + q_k x^{k-1} \quad (2.2.1)$$

One can regard $q(x)$ as a representation of the *state vector* of the shift register generator.

As shown in Section 1.4, the equation which determines the behavior of a register q_j in Figure 2 is

$$q_{j+1}(s+1) = q_j(s) - h_j o(s)$$

where the subscript denotes a register stage, and the argument s is a time index. We have changed this time index from n to s , because we have another use for symbol n below. We now simplify this as:

$$q'_{j+1} = q_j - h_j o \quad (2.2.2)$$

where prime means one clock after no-prime. Thus, $q(x)$ above records the register state before a clock, and $q'(x)$ records the state after one clock,

$$q'(x) = q'_1 + q'_2 x + q'_3 x^2 + \dots + q'_k x^{k-1} \quad (2.2.3)$$

If we insert (2.2.2) into (2.2.3) we arrive at this iteration equation for Figure 2:

$$q'(x) = x q(x) - o h(x) + i \quad (2.2.4)$$

Here, o = output which is q_k/h_k , and i = input = q_0 . See Figure 2. $h(x)$ is the polynomial we called $H(x)$ in Chapter 1, namely

$$h(x) = h_0 + h_1 x + \dots + h_k x^k \quad (2.2.5)$$

The fact that $q(x)$ and $h(x)$ have a relative offset in their indices is just a quirk of the way we chose to label the registers in Figure 1.

Equation (2.2.4) describes how the state vector of a shift register generator moves in time.

A note on Symbols

In Section 1.2 we discussed the generality of our various circuits and the meaning of the *symbols* that flow around in these circuits. In the most general case, each register could hold one of $Q = p^m$ different symbols, as discussed in Example 1 and Example 2 of Section 1.2. This is the Q used in Section 2.1, and this Q is also applicable to the present section up to this point.

For the rest of this Chapter, we shall consider the simpler Example 3 situation where each register holds only $Q = p$ different values, so the symbols are now elements of $GF(p)$. This means, for example, that the coefficients of $h(x)$ lie in $GF(p)$. Eventually, we will only care about $p=2$.

We can generalize everything below to the fully general case $Q = p^m$ by making a global substitution of $p \rightarrow p^m$. This would be necessary, for example, if we were analyzing a Reed-Solomon style shift register generator. In this case, our Galois construction generates a higher order extension field from a lower order extension field.

This seems like an unnecessary level of complexity to maintain in all the following work, so we settle for symbols in $GF(p)$.

Galois Field Review

Recall the Galois Field construction discussed at great length in our Galois notes. [We replace m with M in order to avoid confusion with the m used in the above note on symbols.]

$$GF(p^M) = \text{Polys}[x, GF(p)] / (f(x))$$

By way of review, here is what this says:

" If you are given *any* polynomial $f(x)$ of degree M which is irreducible over $GF(p)$ -- meaning that it cannot be factored -- , you can build a structure whose elements and operations $+$ and $\bullet$ are equivalent to the abstract Galois Field $GF(p^M)$. The elements of this structure are the remainders of polynomials of any degree defined over $GF(p)$ that one gets by dividing these polynomials by $f(x)$. The ring of such polynomials of any degree over the field $GF(p)$ is $\text{Polys}[x, GF(p)]$, and then the set of remainders is denoted as this ring $/ (f(x))$. The implied operations $+$ and $\bullet$ are what naturally arise when you add or multiply polynomials and then take a remainder."

The usual notation in dealing with the above structure is to put the raw polynomials of $\text{Polys}[x, GF(p)]$ inside curly brackets. Then the curly bracketed thing represents one of the remainders which is then regarded as an element of $GF(p^M)$. The following line should be clearly understood:

$$\{x f(x)\} = \{f(x)\} = \{0\} = 0$$

Clearly, all three polynomials $xf(x)$, $f(x)$, and 0 have remainder 0 when divided by $f(x)$. This element of $GF(p^M)$ must be the additive identity 0. If you add $f(x)$ or a multiple of $f(x)$ to some other polynomial, you do not change the remainder.

This is of course just the beginning of Galois Theory and the reader is referred to earlier notes, especially in regard to things like minimum polynomials, primitive elements and polynomials, etc.

We shall replace $f(x)$ with the $h(x)$ of our shift register generator, and M with k to get:

$$GF(q = p^k) = \text{Polys}[x, GF(p)] / (h(x))$$

This implies that the degree of $h(x)$ is k , so we must have some $h_k \neq 0$. If $p > 2$ in $GF(p)$, it is possible to have $h_k \neq 1$. In this case, one could divide through all the coefficients of $h(x)$ and write

$$h(x) = h_k h'(x)$$

where $h'_k = 1$. If $h(x)$ was irreducible, then so is $h'(x)$. Since $h'(x)$ has its highest degree coefficient equal to one, it is called a *monic* polynomial. From now on, we shall restrict our interest to monic $h(x)$. Our real interest in doing this is that we can then identify $h(x)$ with some minimum polynomial of an element of $GF(q)$, since all minimum polynomials are monic. We use q as a shorthand for p^k . The following fact is a combination of Fact 3 of Galois Chapter 5, and the first Fact of Chapter 6:

Fact 3: Any monic $h(x)$ of degree k that is irreducible over $GF(p)$ is the minimum polynomial of the element $\{x\}$ of $GF(q = p^k)$.

Proof: We know that $\{h(x)\} = 0$ (see above for $f(x)$), and therefore $h(\{x\}) = 0$. Thus, $\{x\}$ is a root of $h(x)$. Since $h(x)$ is monic and irreducible, it must be (from Fact 3 of Chapter 5) a minimum polynomial of some element α of $GF(q)$. But we have found a candidate element $\{x\}$. In general, a given minimum polynomial $m(x)$ is the minimum polynomial for several elements of $GF(q)$, namely, all elements of the conjugate set of α . This set can contain up to k elements.

Observation: The coefficient h_0 of $h(x)$ must be non-zero. Otherwise you could factor out x from $h(x)$ and then $h(x)$ would be reducible. In terms of Figure 2, the implication is that some feedback must always go *to* the leftmost register. And in Figure 1, some feedback must always come *from* the rightmost register. Thus, an implication of $h(x)$ being irreducible is that all k registers are in the feedback loop.

Reminder: In Chapter 4 it was found that the field $GF(q)$ was cyclic and that therefore there exist certain elements called primitive elements in terms of which all q elements of $GF(q)$ can be enumerated as follows:

$$GF(q) = \{0, 1, \alpha, \alpha^2, \alpha^3, \dots, \alpha^{q-2}\} \quad \alpha^{q-1} = 1$$

In general, any non-zero element of $GF(q)$ has an *order* -- a minimum integer n such that $\alpha^n = 1$. Primitive elements have order $n = q-1$. The order of any element must integrally divide $q-1$.

Fact 4: The order of $\{x\}$ must exceed 1.

Proof: If $n=1$, then $\alpha = 1$, so that $\alpha = \{x\} = \{1\}$. But this is a contradiction, because $\{x\} \bullet \{f(x)\} = \{xf(x)\} \neq \{f(x)\}$ for arbitrary $f(x)$. One loophole is if $k = 1$, so that $h(x) = ax+b$ and all remainders are constants, so $\{f(x)\} = K$. In this case only, it is possible to have $\{xK\} = \{K\}$. This means $\text{Rem}(xK/h(x)) = K$ which in turn means that $h(x) = ax-a$. Here is a sketch of this fascinating special case of Figure 2:



We are not surprised to learn that the period of this circuit is 1. If we are willing to ignore this less than interesting case, we may conclude that $n > 1$.

The Figure 2 Galois Iterator

In light of the Galois field $GF(p^k)$ discussion above, and the $\{..\}$ notation so defined, we can write the above state vector iteration equation (2.2.4) as

$$\{q'(x)\} = \{x\} \bullet \{q(x)\} - \{o\} \bullet \{h(x)\} + \{i\}$$

$\{o\} = o\{1\}$ and $\{i\} = i\{1\}$ are multiples of the Galois $\bullet$ identity element $\{1\}$. Since $\{h(x)\} = 0$, we get

$$\{q'(x)\} = \{x\} \bullet \{q(x)\} + \{i\} \quad (2.2.7)$$

To make a closer connection to the Galois fields, we now use Greek letters to denote field elements, as was done in our Galois notes. Define

$$\alpha = \{x\} \quad \beta = \{q(x)\} \quad \{i\} = i\{1\} = i \cdot 1 \quad (2.2.8)$$

Then we get this Galois statement of what happens with a single clocking of the shift register:

$$\beta' = \alpha \bullet \beta + i \cdot 1 = \alpha \beta + i \cdot 1$$

Here, $\beta = \{q(x)\}$ represents the starting state vector of the register set. And i is the input bit at this time. We can now go back add add back in our time s subscripts as follows:

$$\beta_{s+1} = \alpha \beta_s + i_s \cdot 1 \quad (2.2.9)$$

We are now able to study the progression of the state vector, starting at time $s=0$. After examining the first few terms, we quickly arrive at the general solution of our difference equation (2.2.9) for β_s :

$$\begin{aligned} \beta_1 &= \alpha \beta_0 + i_0 \\ \beta_2 &= \alpha \beta_1 + i_1 = \alpha^2 \beta_0 + [\alpha i_0 + i_1] \cdot 1 \end{aligned}$$

$$\beta_s = \alpha^s \beta_0 + [\alpha^{s-1} i_0 + \alpha^{s-2} i_1 + \dots + \alpha i_{s-2} + i_{s-1}] 1 \quad (2.2.10)$$

This last equation is an exceedingly powerful result and says a lot about what is going on in our polynomial divider circuit of Figure 2. We shall refer to it as the Galois Iterator for Figure 2.

If there is no input, as in a "shift register generator", the result simplifies:

$$\beta_s = \alpha^s \beta_0 \quad (2.2.11)$$

Fact 5: The state vector of a (Figure 2) shift register generator steps periodically through some subset N of the non-zero elements of $GF(p^k)$, provided it is initialized in some non-zero state. The state vector period N must integrally divide $p^k - 1$, so $N \leq p^k - 1$. The integer N is the order of $\{x\}$.

Proof: If $\beta_0 = 0$, then $\beta_s = 0$ and the shift register has "died", so assume $\beta_0 \neq 0$. In this case, we know that $\alpha \neq 0$. The non-zero elements of $GF(q)$ form a cyclic subgroup under $\bullet$, therefore no product of non-zero elements can ever give the zero element. Thus, we can never have $\beta_s = 0$ if $\beta_0 \neq 0$.

We know that the element $\alpha = \{x\}$ has some order N which integrally divides $p^k - 1$. Thus, N is the smallest integer such that $\alpha^N = 1$. Therefore we can write,

$$\beta_{s+N} = \alpha^{s+N} \beta_0 = \alpha^s \beta_0 = \beta_s$$

which says that N is the period of the state vector.

Fact 6: If $p^k - 1$ happens to be a prime number, the state vector period is $p^k - 1$.

Proof: From Fact 5, the period n must evenly divide $p^k - 1$. In Fact 4 we have rejected $N=1$, therefore we conclude that $N = p^k - 1$.

Example: For $GF(2^3)$ we have $p^k - 1 = 2^3 - 1 = 7$ which is a prime number. Thus, if you start a 3-register version of Figure 2 in any non-zero state, it will cycle through all 7 non-zero states. This is true regardless of what irreducible $h(x)$ is selected.

Example: For $GF(2^9)$ we have $2^9 - 1 = 511 = 7 \bullet 73 = \text{non-prime}$. In this case, if we start a 9-register version of Figure 2 in any non-zero state, it will cycle through some number N of states before returning to the initial state. There are only three possibilities: $N = 7$, $N=73$, $N=511$.

Comment: The preceding example shows the power of the Galois analysis. It would be extremely painful and time-consuming to obtain this kind of result by trial and error analysis.

Fact 7: If $\alpha = \{x\}$ is a *primitive element* of $GF(q=p^k)$, then the state vector period is $q-1$. In this case, the state vector "counts" through all $q-1$ non-zero elements of $GF(q)$.

Proof: By definition, the powers of a primitive α enumerate all $q-1$ elements of $\{GF(q) - 0, \bullet\}$, and $\alpha^{q-1} = 1$. Since α = primitive element is a generator of the cyclic group $\{GF(q) - 0, \bullet\}$, we can represent β_0 as some power of α , say $\beta_0 = \alpha^r$. Then from $\beta_s = \alpha^s \beta_0$ we have:

$$\{\beta_0, \beta_1, \beta_2, \beta_3, \beta_4, \dots, \beta_{q-2}\} = \{\alpha^r, \alpha^{r+1}, \alpha^{r+2}, \dots, \alpha^{r+q-2}, \alpha^{r+q-1} = 1, \alpha, \alpha^2, \dots, \alpha^{r-1}\}$$

There are $q-1$ elements in each set. All elements in the right set are different, so this is true of the left set as well. The next element on the left would be $\beta_{q-1} = \alpha^r = \beta_0$, so the period is $q-1$. In this case $\beta_s = \alpha^s \beta_0$ counts through all non-zero elements of $GF(q)$ so the period β_s is then equal to the maximum set in Fact 5, $n = q-1$.

Fact 8: If $h(x)$ is a *primitive polynomial* of $GF(q=p^k)$, then the state vector period is p^k-1 .

Proof: If $h(x)$ is a primitive polynomial of $GF(q)$, then all its roots are primitive elements. Since $\{x\}$ is a known root of $h(x)$, $\{x\}$ is then a primitive element, so the results follow from Fact 7.

An example : $GF(q=2^9)$

Consider again $GF(q=2^9) = GF(512)$. If we select $h(x) = 1 + x^4 + x^9$ or $1 + x^5 + x^9$, these being the primitive polynomials having the fewest non-zero coefficients, then if we start our shift register off in any non-zero state β_0 , it will not return to that state until 511 clocks have gone by. In other words, the state vector period is $N = q-1 = 2^9 - 1 = 511$.

We know that the number of primitive polynomials of a Galois Field is equal to the number of distinct conjugate sets which contain primitive elements, this was Fact 11 of Galois Chapter 5. Off hand, we do not know this number for $GF(512)$, but it may be somewhat large, since there are 512 elements to deal with.

For assistance, we now peruse Appendix C of Peterson and Weldon. In this list, we count a total of 24 primitive polynomials for $GF(2^9)$, and this does not include the reflected ones mentioned in Galois Chapter 5 Fact 12, so there are 48 primitive polynomials in all. Each set of conjugates containing a primitive element contains $m=9$ distinct elements (Fact 8 of Chapter 5), so we conclude that there are a total of $9 \cdot 48 = 432$ primitive elements out of the total number of 512 elements of $GF(2^9)$.

In Appendix C, the primitive polynomial with the *fewest* non-zero coefficients is listed first, let us call its corresponding primitive element a . The remaining primitive polynomials are then given, along with the smallest power of a which is one of their roots. The polynomials are given in an octal notation. Here are the first two table entries:

- | | | |
|---|------------------------------|---------------------------------------|
| 1 | $1021h(x) = (x - a) \dots$ | = mimumum polynomial containing a |
| | $1021 = 001,000,010,001$ | $h(x) = 1 + x^4 + x^9$ |
| | reflected version is: | $h(x) = 1 + x^5 + x^9$ |
| 3 | $1131h(x) = (x - a^3) \dots$ | = mimumum polynomial containing a^3 |
| | $1131 = 001,001,011,001$ | $h(x) = 1 + x^3 + x^4 + x^6 + x^9$ |
| | reflected version is: | $h(x) = 1 + x^3 + x^5 + x^6 + x^9$ |

We are obviously interested in the $h(x)$ with the fewest non-zero coefficients since this minimizes the number of adders needed in Figure 2 (or Figure 1). So we shall select $h(x) = 1 + x^4 + x^9$, which means we are choosing our $\alpha = \{x\} = a$.

The non-terminating Remainder in polynomial division.

In Chapter 1 we discussed the use of either Figure 1 or Figure 2 as a polynomial divider. Both circuits perform the same function. In Figure 1, the remainder is "hidden", but in Figure 2 it is exposed: the state vector always holds the "current remainder". In equation (1.3.9) we wrote this as:

$$I_r(z) / H(z) = O_{r-k}(z) + R^{(0)}_{k-1}(z)/H(z)$$

To review, $I_r(z)$ was an Input polynomial of degree r , $H(z)$ was the divisor polynomial of degree k , and $O_{r-k}(z)$ was the Output quotient polynomial of degree $r-k$. The Remainder $R^{(0)}_{k-1}(z)$ is always of degree $k-1$. If some of its leading coefficients happen to vanish, it is of degree less than $k-1$.

In Chapter 1 we used upper-case polynomial names to stress the notion of Z Transform, and for the same reason we used the variable z . We were also trying to show explicitly how the degrees of all the polynomials come out right.

Here we shall ignore all this baggage and rewrite the above in the simple form

$$i_0(x) / h(x) = o_0(x) + r_0(x)/h(x)$$

We have not changed anything, we are just simplifying the notation. The 0 subscript is a relative time index. We think now of time $t=0$ just after the last non-zero symbol of the input stream has been shifted in to the divider. If the input stream is always zero from this point on, we can think of the circuit as the *shift register generator* discussed above which has its register state initialized with the remainder appearing in the above equation, $r_0(x)$. That is, the coefficients of this polynomial are the starting contents of the k registers. After one clock, we have:

$$i_1(x) / h(x) = [x i_0(x)] / h(x) = o_1(x) + r_1(x)/h(x)$$

The input polynomial $i_1(x) = x i_0(x)$, because we have added a trailing 0 to the input symbol stream. After s clocks we would have

$$i_s(x) / h(x) = [x^s i_0(x)] / h(x) = o_s(x) + r_s(x)/h(x)$$

If at some point the remainder vanished, the subsequent quotient symbols would be zeros. Conversely, if the remainder were never to vanish, the quotient symbol stream would go on forever.

Fact 9: If $h(x)$ of degree k is irreducible over $GF(p)$, and if the first remainder

$$r_0(x) = \text{Rem}[i_0(x) / h(x)]$$

is non-zero, then all subsequent remainders

$$r_s(x) = \text{Rem}[x^s i_0(x) / h(x)]$$

are non-zero, and the quotient sequence never terminates. Moreover, the sequence of remainders has period N which is the state vector period discussed above. This period N divides evenly into $p^k - 1$. Furthermore, if $h(x)$ is a primitive polynomial of $\text{GF}(p^k)$, then the state vector period is exactly $p^k - 1$.

Proof: Once we identify the remainder with the shift register state, everything follows directly from the Facts given above.

Corollary: We have therefore answered Question 5 posed in Appendix 1.1

Question 5: How do we know that, in polynomial division over some field F , it is possible that the quotient may never terminate?

We have seen that, not only is it possible, but for irreducible $h(x)$, it is very likely.

Fact 10: The *state vector period* of a Figure 2 shift register generator is the same as the *period* of $h(x)$.

Proof: The *period* of a monic irreducible $h(x)$ was defined rather mysteriously in Chapter 5 as being the smallest value of n such that $h(x)$ divides evenly into $x^n - 1$. Let us assume that $h(x)$ has *period* n , so we can then write $(x^n - 1)/h(x) = g(x)$, where $g(x)$ is whatever quotient polynomial we get by this division. Then consider the remainder sequence noted above, for time s clocks after the input polynomial coefficients resume being 0,

$$r_s(x) = \text{Rem}[x^s i_0(x) / h(x)]$$

Then we can evaluate

$$r_{s+n}(x) - r_s(x) = \text{Rem}[(x^n - 1)x^s i_0(x) / h(x)] = \text{Rem}[x^s g(x)i_0(x)] = 0,$$

This shows that $r_{s+n}(x) = r_s(x)$, which says that the state vector period is n . Recall that the *period* n refers to the smallest power n such that $(x^n - 1)/h(x) = g(x)$. Similarly, the *state vector period* is the smallest integer n that makes the register state (= remainder) repeat. Thus, we see that these two entities are one in the same.

Summary: This section has focused on the state vector as being an element of a Galois Field $\text{GF}(p^k)$. For the Figure 2 circuit, this state vector can be identified with the remainder of polynomial division. We learned that the state vector period N of a Figure 2 style shift register generator is determined by the nature of $h(x)$.

We did not in this section have much to say about any particular register. For example, the output symbol sequence is basically the contents of the rightmost register in Figure 2. One wonders what the periodicity of the contents of a particular register might be, and thus, one wonders about the periodicity of the output data stream of a shift register generator.

For the moment, we settle for the following:

Fact 11: The period of any particular register of the Figure 2 circuit must integrally divide the state vector period N .

Proof: Certainly the period of one register cannot exceed the period of the register set as a whole. If the whole register state repeats with period N , then so must any one register. However, it is easy to imagine that a particular register might have a smaller period. To be compatible with the state vector period, the state vector period would have to consist of an integral multiple of these smaller periods.

Corollary: The *output period* of a Figure 2 shift register generator must be a divisor of the state vector period N .

In the next section, we shall find a much stronger statement about the output period.

2.3 The Output Sequence of a Shift Register Generator

In the previous section, we examined the behavior in time of a *state vector* representing the contents of the k registers of a Figure 2 shift register generator.

In this section, we concentrate on the *output sequence* of a shift register generator, rather than on the state vector of its register set.

In contrast with the state vector situation, the output sequences of both types of shift register generators -- Figure 1 style and Figure 2 style -- obey the *same* fundamental equation.

The Figure 1 state vector behavior is then obtained by considering a grouping of k consecutive output symbols.

The fundamental equation of interest here is the time domain description of the polynomial divider which was derived in Section 1.8 and appears there in a summary box,

$$i_n = \sum_{j=0}^k h_j o_{n+j} \quad n = \text{any integer} \quad (2.3.1)$$

Recall that this is the time-domain view of polynomial division $O(z) = H(z)/I(z)$.

For the scrambler style divider circuit of Figure 1, we can identify the output stream o with the contents of the rightmost register q_0 . If we set the input sequence to zero, $i_n = 0$, we have the equation which governs a shift register generator:

$$\sum_{j=0}^k h_j o_{n+j} = 0 \quad n = \text{any integer} \quad (2.3.2)$$

If we separate out the highest term and put it on the left, and set time index $n=0$, we get:

$$h_k o_k = -h_0 o_0 - h_1 o_1 - \dots - h_{k-1} o_{k-1} \quad (2.3.3)$$

The reader should stare at Figure 1 to see how it directly implements the above equation. The output at time k is determined entirely by the previous k outputs, which are stored in the registers of Figure 1.

The above equation is a difference equation. If we solve it iteratively, as we have done with other equations several times earlier, we end up explaining once again how polynomial division works, and we end up again with Figure 6, with symbols explained in Figure 3. The dividend polynomial in this case takes its leading coefficients from the starting contents of the shift register generator.

Our purpose here is not to solve the difference equation, but to make some fairly astounding observations about the properties that solutions of the difference equation have.

Fact 1: There are exactly p^k solutions to the difference equation (2.3.2). One of these solutions is all zeros, so there are $p^k - 1$ non-zero solutions.

Proof: By "solution" we mean an infinite sequence

$$\text{solution} = \{o_j\} = \{o_0, o_1, o_2, \dots, o_{512,453}, \dots\}$$

Consider Eq. (2.3.3) above. This shows that o_k is fully determined by the numbers $\{o_{k-1}, \dots, o_0\}$. If we iterate, we find that o_{k+1} is then determined by the set of numbers $\{o_k, \dots, o_1\}$. But we just noted that o_k is determined by $\{o_{k-1}, \dots, o_0\}$, therefore o_{k+1} is determined by $\{o_{k-1}, \dots, o_0\}$. Similarly, all subsequent o_{k+j} are determined by $\{o_{k-1}, \dots, o_0\}$. In other words, the entire solution is completely determined by the starting set $\{o_{k-1}, \dots, o_0\}$. Thus, the number of solutions is equal to the number of starting sets. Since each symbol o_i is a number in $GF(p)$, it can take p values. Thus, there are p^k starting sets, and hence there are p^k solutions. No two of these solutions can be the same because they had different starting sets, and the starting set is part of the solution sequence.

Fact 2: The p^k solutions of the difference equation can be regarded as elements of a k -dimensional vector space V^k . The k "basis vectors" in this space can be regarded as the k solutions which have the starting sets $\{10000..\}, \{01000..\}, \{00100..\},$ etc.

Proof: Since the difference equation is linear, any multiple of a solution or any sum of solutions is also a solution. Thus, the solutions form a vector space. Since we have found k linearly independent basis vectors which span all solutions, the dimension of the space is k . Note that indeed, any solution can be written as a sum of the basis vector solutions simply because any starting set can be written as a linear combination of the basis vector starting sets.

Assumption : Assume for the moment that we can find an integer n such that $h(x)$ of degree k divides evenly into $x^n - 1$. It is not necessary that n be the smallest such integer (the so-called *period* of $h(x)$). Also, we put off any discussion as to whether or not an n always exists for an arbitrary $h(x)$. For those $h(x)$ that will be of interest to us, we will *know* that n exists.

Definition: Assuming that n exists, we define $g(x)$ to be the quotient: $g(x) = (x^n - 1)/h(x)$.

Aside: If $h(x)$ is of degree k and is the minimum polynomial of an element of $GF(q=p^k)$, then we know that $q-1 = p^k - 1$ is a workable candidate for the n mentioned in the above Assumption. See Galois Chapter 5. Any monic $h(x)$ of degree k that is irreducible in $GF(p)$ is such a minimum polynomial, so for any such $h(x)$ our candidate n is $n = p^k - 1$.

Fact 3: Since we then have $g(x)h(x) = x^n - 1$, we can consider $g(x)$ to be the generator of a *cyclic code*. All coefficients are assumed to lie in field $GF(p)$.

The subject of cyclic codes was elaborated in Galois Chapter 7. We summarize some basic facts:

- (a) $h(x)$ is of degree k , and $g(x)$ is of degree $n-k$, and $g(x)h(x) = x^n - 1$.
- (b) Code polynomials are formed as $c(x) = d(x)g(x)$, where the $d(x)$ have degree $k-1$ and are called data polynomials. Thus, code polynomials have degree $n-1$, and so have n coefficients in $GF(p)$.
- (c) Since the $d(x)$ have k coefficients in $GF(p)$, there are p^k possible data polynomials. Therefore, there are p^k possible code words.
- (d) All cyclic permutations of a code word are code words.

Since the coefficients of a code polynomial lie in $GF(p)$, we can think of an n -dimensional vector or n -tuple consisting of the coefficients of $c(x)$. We write this as $c = \{c_0, c_1, \dots, c_n\}$, and we refer to it as a code word. Sometimes we will loosely refer to $c(x)$ as a code word, but we really mean this vector c .

We now make the following unusual claim:

Fact 4: If c is any code word of the cyclic code generated by $g(x)$, then the infinite sequence

$$\{o_i\} = \{c, c, c, c, \dots\}$$

is a solution of the difference equation (2.3.2).

Proof: See Appendix 2.1. This is the crucial fact, so naturally the proof is more than two lines. It might be good to look at this proof to make sure the above notation is understood.

Fact 5: All p^k solutions of the difference equation (2.3.2) are of the form $\{o_i\} = \{c, c, c, c, \dots\}$.

Proof: In Fact 4 we found a solution of (2.3.2) for each code word c . We know from cyclic code theory that there are p^k such code words, so we have found p^k solutions of (2.3.2). According to Fact 1, (2.3.2) has exactly p^k solutions, so we have found them all, and they are therefore all of the form claimed.

Fact 6: Any solution of (2.3.2) has a period which evenly divides n . We do not rule out the possibility here that different solutions might have different periods which divide n .

Proof: Since $\{o_i\} = \{c, c, c, c \dots\}$, we know that all solutions have period n . If, for some particular solution, it happens that the symbols of the code word c are periodic with some period which divides n , then of course that solution $\{o_i\}$ will have this smaller period.

An example : GF(16)

According to the Peterson and Weldon Appendix C, here are some candidate $h(x)$ polynomials, and α is a primitive element.

1 23	$h(x) = (x - \alpha)(x - \alpha^2)(x - \alpha^4)(x - \alpha^8)$ 23 = 010,011	$h(x) = 1 + x + x^4$	order 15
3 37	$h(x) = (x - \alpha^3)(x - \alpha^6)(x - \alpha^{12})(x - \alpha^9)$ 37 = 011,111	$h(x) = 1 + x + x^2 + x^3 + x^4$	order 5
5 07	$h(x) = (x - \alpha^5)(x - \alpha^{10})$ 07 = 000,111	$h(x) = 1 + x + x^2$	order 3

We have enumerated some of the minimum polynomials. For each one, we indicate the order of the field elements that go with that polynomial.

Consider $h(x) = 1 + x + x^2 + x^3 + x^4$ which is of degree 4, and which corresponds to a field element α^3 which is blatantly *not* a primitive element (since its order is 5 and not 15). Thus, this $h(x)$ is *not* a primitive polynomial.

We know that $h(x)$ divides into $x^{15} - 1$ to give $g(x)$. We would like to find this $g(x)$. Consider,

$$h(x) = 1 + x + x^2 + x^3 + x^4 = (x^5 - 1) / (x - 1)$$

$$g(x) = (x^{15} - 1) / h(x) = (x - 1) [(x^{15} - 1) / (x^5 - 1)] = (x - 1) (x^{10} + x^5 + 1)$$

$$g(x) = x^{11} + x^{10} + x^6 + x^5 + x + 1$$

Now we can write down some of the code words of $g(x)$. Our first one is $g(x)$ itself, and then we do all cyclic permutations. Notice for this $g(x)$ code word we start at the right and then fill with three zeros on the left to get it from $n-k$ to n coefficients. Please do not miss the point here. Code words are multiples of $g(x)$, so the unit multiple is one of the code words.

$$\{ 0,0,0,1,1, \quad 0,0,0,1,1, \quad 0,0,0,1,1 \}$$

We have artificially introduced gaps to reveal the repeating pattern that results. We can do cyclic permutations on the above to get 5 of the 16 codes words.

$$\begin{array}{lll} \{ 0,0,0,1,1, & 0,0,0,1,1, & 0,0,0,1,1 \} \\ \{ 0,0,1,1,0, & 0,0,1,1,0, & 0,0,1,1,0 \} \\ \{ 0,1,1,0,0, & 0,1,1,0,0, & 0,1,1,0,0 \} \\ \{ 1,1,0,0,0, & 1,1,0,0,0, & 1,1,0,0,0 \} \end{array}$$

$$\{ 1,0,0,0,1, \quad 1,0,0,0,1, \quad 1,0,0,0,1 \}$$

To get the other code words, add two of the above and then take its 5 cyclic forms, and then add another two and get its 5 cyclic forms. The 16th code word is all zeros.

Now, construct a shift register generator with 4 registers and use $h(x) = 1 + x + x^2 + x^3 + x^4$. We know that all output sequences are of the form $\{o_i\} = \{c, c, c, c, \dots\}$. Looking at the above code words, we quickly conclude that the period of all possible non-zero output streams is 5 !

This result is consistent with Fact 5 of Section 2.1. There, we showed that the state vector period of the shift register was equal to the order of the Galois element $\{x\}$ which is a root of $h(x)$. For our $h(x)$ here, $\{x\} = \alpha^3$ which has order 5. Since the state vector repeats every 5 clocks, it is not too surprising that the output sequence has this same period.

• end of example

Fact 7: If the integer n happens to be the *period* of $h(x)$, then at least n solutions of the difference equation (2.3.2) have period n .

Proof: We already know from Fact 6 that the period of any solution must divide n . The question here is whether there are some solutions which have the full period n .

Consider the code word $c(x)$ obtained by multiplying the generator $g(x)$ by unity. We claim that this c has the full period n . To show this, assume first that it does not, and that it has some smaller period $m = n/N$ (as happened in our example above). In this case we must be able to group the elements of c into N identical groups of m symbols. (In the example we had 3 identical groups of 5 symbols). In this case we can write:

$$c(x) = g(x) = f(x) [1 + x^m + x^{2m} + x^{3m} + \dots + x^{(N-1)m}]$$

where $f(x)$ is a polynomial of degree m representing the pattern that repeats N times. We recognize the above [...] as $(x^n - 1) / (x^m - 1)$ where $n = Nm$. And of course $(x^n - 1) = g(x)h(x)$ as usual. Thus we have

$$g(x) = f(x) (x^n - 1) / (x^m - 1) = f(x)g(x)h(x) / (x^m - 1)$$

Cancelling $g(x)$ we find that $f(x)h(x) = (x^m - 1)$ where $m < n$. But by definition, assuming that the *period* of $h(x)$ is n means that there is no such $m < n$ such that $h(x)$ divides $(x^m - 1)$. Thus we arrive at a contradiction, so it must be that this $c(x) = g(x)$ has the full period n , and therefore the corresponding solution to the difference equation (2.3.2) must have the full period n as well.

Furthermore, we know we can form $n-1$ distinct other code words by taking cyclic permutations of this $g(x)$ code word, and all of these code words will have period n as well.

Fact 8: If $h(x)$ of degree k is a *primitive polynomial* of $GF(p^k)$, then all $p^k - 1$ non-zero solutions of the difference equation (2.3.2) have period $n = p^k - 1$.

Proof: This is the big result we have been waiting for. The proof is so simple that it is deceptive.

From Galois Chapter 5 Fact 4, we know that if $h(x)$ is a primitive polynomial of $GF(p^k)$, then $h(x)$

has *period* $n = p^k - 1$. From Fact 7, we know that in this case we can produce n solutions of the difference equation (2.3.2) which have the full period n -- these are just the n cyclic permutations of the code word $g(x)$ (filled out to have n coefficients).

On the other hand, from the discussion following Fact 3, we know that the cyclic code generated by $g(x)$ has exactly p^k code words. One of these is the zero code word, so this leaves $p^k - 1$ non-zero code words. But this is our number n . We know from Fact 5 that every solution of (2.3.2) corresponds to a code word. Since there are n non-zero code words, there are only n non-zero solutions, and we have found them all, and they all have period n .

Fact 9: If $h(x)$ of degree k is a *primitive polynomial* of $GF(p^k)$, then there is essentially only *one* characteristic non-zero output pattern of the shift register generator, and its period is $p^k - 1$. This pattern is often called a *maximum length sequence*, but we will just call it the *characteristic sequence*.

Proof: Pick any of the $p^k - 1$ non-zero solutions mentioned in Fact 8. Write it down. Then all the other solutions are sub-sequences obtained by doing delayed starts on the original solution. For example,

```

Selected solution =      abcdefgabcdefgabcdefg.....
another solution =      bcdefgabcdefgabcdefga.....
another solution =      cdefgabcdefgabcdefgab.....
another solution =      defgabcdefgabcdefgabc.....
another solution =      efgabcdefgabcdefgabcd.....
another solution =      fgabcdefgabcdefgabcde.....
another solution =      gabcdefgabcdefgabcdef.....

```

In this sketch, we have $k = 3$, and the period of all (non-zero) solutions is $2^3 - 1 = 7$. The 7 solutions are all listed. The 8th solution is the boring one that is all zeros.

Example: We know that $h(x) = 1 + x^4 + x^9$ is a primitive polynomial of $GF(2^9) = GF(512)$. If we implement a shift register generator with nine registers using this $h(x)$, we know that there is one characteristic output sequence and it has period $n = 2^9 - 1 = 511$ bits.

Moreover, we know exactly what this sequence looks like: divide $h(x)$ into $x^{511} - 1$ to get $g(x)$ which has degree $n - k = 511 - 9 = 502$. The output pattern is then the 503 coefficients of $g(x)$ with a filler of 8 zeros on the end to make the full characteristic pattern of 511 bits.

This suggests the following fact;

In Section 2.1 we found that in the Figure 2 type shift register generator, if $h(x)$ is a primitive polynomial, then the register state vector cycles through all possible non-zero elements of $GF(q)$, so the *state vector period* was then $q - 1 = p^k - 1$. We concluded that the pattern of any particular register must have a period that divides evenly into $q - 1$. It seemed quite possible that some particular register might have some smaller period than $q - 1$.

What we have learned is that, for the Figure 2 circuit, the rightmost register must have the full period $n = q - 1$. This is because it generates the output sequence directly ($h_k = 1$ since $h(x)$ monic).

With regard to Figure 1, we know that *all* registers have period $n = q - 1$! This is because all registers contain the output sequence, we just have a time delay between them. We have thus proved:

Fact 10: For a Figure 1 shift register generator, the *state vector period* is the same as the *output period*.

Corollary: If $h(x)$ is a primitive polynomial of $GF(p^k)$, then the state vector period of a Figure 1 shift register generator is the maximal period $p^k - 1$.

Here is a drawing to illustrate this fact, which one can relate to the previous sketch above:

```

output sequence=      abcdefgabcdefgabcdefg.....
state vectors =       abc
(Figure 1 circuit)    bcd
                      cde
                      def
                      efg
                      fga
                      gab
                      abc

```

There is not much distinction between the state vector sequence and the output sequence. In the following discussion, we will refer to the state vectors as being obtained by "sifting through" the output sequence, one symbol at a time. What we mean by this inexact language is what is illustrated in the figure above. In the next section, we shall explore the nature of a characteristic sequence

2.4. Characteristic Sequences

In the previous section, we considered a shift register generator having k registers, having symbols in $GF(p)$, and having for its $h(x)$ a primitive polynomial of $GF(p^k)$. Such a shift register generator has essentially one output sequence known as a characteristic or maximum-length sequence. The circuit can be implemented as either Figure 1 or Figure 2.

Here we shall see what we can learn about such a characteristic sequence.

Observation: Different primitive polynomials $h(x)$ of degree k are likely to produce different characteristic sequences. We do not want to leave the impression that there is only one such sequence. In our Section 2.2 example of $GF(512)$, we noted that there were 48 distinct primitive polynomials. one usually selects one of the two possible primitive polynomials which have the minimum number of non-zero coefficients.

Fact 1: If one were to sift through one period of a characteristic sequence, moving one symbol position at a time, one would find each k -symbol *state vector* once and only once. In other words, each possible state vector occurs exactly once somewhere in the characteristic sequence.

Aside: What we do not know is the exact order in which these state vectors occur.

Proof: Fact 1 was demonstrated at the end of the previous section. If the same state vector were found twice within one period of the output sequence, then the period would equal the difference between the two findings of the same state vector, which is a contradiction. The last figure in the previous section exactly demonstrates this fact.

We now add another fact which we basically proved already:

Fact 2: In the characteristic sequence of a primitive $h(x)$ shift register generator, the maximum number of consecutive zeros is $k-1$, and this sequence occurs only once per period.

Proof: We can think of the characteristic sequence as belonging to code word $g(x)$ which has $n-k+1$ coefficients. The total period is n , so we have to fill with $n - (n-k+1) = k-1$ zeros. This pattern must occur once per period. It cannot occur twice, because if it did, and we including an adjacent 1, we would have a k -bit state vector pattern repeating twice per period, which we showed in Fact 11 is impossible.

By the way, it should be obvious that we could never have a string of k sequential zeros. This would kill the sequence altogether.

Fact 3: The maximum number of consecutive identical non-zero symbols in the characteristic output sequence is k , and this pattern occurs once per period.

Proof: Since every non-zero state vector must be included once in the characteristic sequence (Fact 11), the vector consisting of k identical symbols for any non-zero symbol in $GF(p)$ [such as 1] must occur once. If there were a string of $k+1$ identical symbols, then the vector of k identicals would appear twice, but Fact 11 says this cannot happen.

Fact 4: In a characteristic sequence , every non-zero symbol appears exactly $(p^{k-1} - 1)$ times, and the 0 symbol appears exactly $(p^{k-1} - 1)$ times. As a check on these numbers ,

$$(p^{k-1} - 1)(p - 1) + (p^{k-1} - 1) = p^k - 1 = \text{total symbols in characteristic sequence}$$

Proof: Take all p^k state vectors and lay them side by side in a huge set which then has kp^k symbols. Since each symbol can take p values, and since all symbols are treated equally in this full enumeration of the state vectors, in this huge set there must be $(kp^k) / p$ occurrences of each symbol. Thus, each symbol appears kp^{k-1} times.

Now form another set which includes all state vectors except the all-zero vector. All non-zero symbols occur in this set kp^{k-1} times, just as in the last set, because we did not eliminate any of them. However, we removed k occurrences of the symbol 0, so it occurs $kp^{k-1} - k$ times in this new set.

Consider once again the "sifting" process where you walk through the entire characteristic sequence looking at one vector's worth of symbols (k -symbols) at a time, but you shift only one symbol on each step of this walk. During this "walk" you encounter precisely all the symbols of our second set described above. Ie, we encounter all non-zero vectors. On the other hand, each symbol of the characteristic sequence is repeated k times by this process.

Therefore, each non-zero symbol appears in the characteristic sequence exactly $(kp^{k-1} - k)/k$ times, and the zero symbol appears $(kp^{k-1} - k)/k$ times.

Note on the minimum weight and distance of the cyclic code generated by $g(x)$.

We have seen that any period's worth of a characteristic sequence can be regarded as a code word of the code generated by $g(x)$, where $h(x)g(x) = x^n - 1$ with $n = p^k - 1$. The entire cyclic code consists of cyclic permutations of this code word, plus the zero code word. Thus, all non-zero code words have the same *weight*, which is the number of non-zero elements. From Fact 5 above, this weight is $(p-1)p^{k-1}$. Thus, this

is also the minimum weight of the code, which we know is the code *distance* as described in Galois Chapter 7.

Fact 5: If we line up our characteristic sequence with any delayed version of itself and compare over one period, we will find that the two sequences differ in exactly $(p-1)p^{k-1}$ places.

Proof: A characteristic sequence and a delayed version of itself over one period may be regarded as two code words of the cyclic code generated by $g(x)$, as discussed in the above paragraph. Assume that two code words differ in exactly N places. When you subtract these two code words, you then get a result which has N non-zero places. But this result is also a code word, and it has weight N . But we know this weight must be $(p-1)p^{k-1}$, as discussed above. Q.E.D.

Binary Characteristic Sequences ($p=2$).

First, we restate all the facts derived in Section 2.3, but specialized to the case $p=2$.

Fact 1': If one were to sift through one period of a binary characteristic sequence, moving one bit position at a time, one would find each k -bit *state vector* once and only once. In other words, each possible state vector occurs exactly once somewhere in the characteristic sequence.

Fact 2': In the binary characteristic sequence of a primitive $h(x)$ shift register generator, the maximum number of consecutive zeros is $k-1$, and this sequence occurs only once per period.

Fact 3': The maximum number of consecutive 1's in a binary characteristic output sequence is k , and this pattern occurs once per period.

Fact 4': In a binary characteristic sequence, 1 appears exactly 2^{k-1} times, and 0 appears exactly $2^{k-1} - 1$ times.

Fact 5': If we line up a binary characteristic sequence with any delayed version of itself and compare over one period, we will find that the two sequences differ in exactly 2^{k-1} places.

Now we add some new facts which apply only to binary characteristic sequences:

Fact 6: A string of $(k-1)$ 1's appears exactly once in a binary characteristic sequence.

Proof: The state vector 0111...11 must appear somewhere, having $(k-1)$ 1's. If the thing on the right of it begins with a 0, then we have found our string of $(k-1)$ 1's, and we have also located the state vector 1111...110. If the thing on the right begins with a 1, then we have found the single allowed occurrence of k 1's (see Fact 3'). In this case we look for the state vector 1111...110. Then this must have a 0 on the left to avoid a second occurrence of k 1's. Then this is our occurrence of $(k-1)$ 1's.

Example: Consider again GF(512). The characteristic sequence has length 511, as noted above. The number of 1's in this sequence is 256, the number of 0's is 255. The largest string of consecutive 0's is 8. The largest string of consecutive 1's is 9, and there is another string of 8 1's.

The product of two shifted binary characteristic sequences.

We have already discussed the *difference* of two such sequences, which is the same as the sum in the binary world, see Fact 5 of the previous section. Here we wish to examine the *product* of two shifted sequences. Our motivation is that such a product is involved in the autocorrelation function of a sequence, and this in turn is related to the spectral density of a sequence.

Consider our two shifted sequences as adjacent column vectors, each containing $n = 2^k - 1$ components. Of course each component is either a 1 or a 0. To the right imagine a third column vector which will be the sum of the two sequences, and a fourth column which is the product. We know that the first two columns are two code words of the cyclic code generated by $g(x)$, and we know these code words have weight 2^{k-1} , this is Fact 4'. Since the third column (the sum) is also a code word, it too has this same weight.

As we scan down the left two column vectors, there are four possible combinations we can get at each row: (0,0), (0,1), (1,0), and (1,1). Let us assign the following counts to these possibilities. In each case, we note the sum and product that will appear in the third and fourth column.

<u>The Counts</u>	<u>Column 3</u>	<u>Column 4</u>	<u>Shifted n's</u>	<u>Unshifted n's</u>
n_1 = number of (0,0)'s	sum = 0 product = 0	$(2^k - 1) - 3 \cdot 2^{k-2}$	$2^{k-1} - 1$	
n_2 = number of (0,1)'s	sum = 1 product = 0	2^{k-2}		0
n_3 = number of (1,0)'s	sum = 1 product = 0	2^{k-2}		0
n_4 = number of (1,1)'s	sum = 0 product = 1	2^{k-2}		2^{k-1}

It is easy to calculate these four counts. Let $w = 2^{k-1}$ = the weight of either code word. Then,

$$n_2 + n_4 = w = \text{number of 1's in second column}$$

$$n_3 + n_4 = w = \text{number of 1's in first column}$$

$$n_2 + n_3 = w = \text{number of 1's in third column}$$

These three equations tell us that $n_2 = n_3 = n_4 = w/2 = 2^{k-2}$. The total number of all combinations is equal to the number of rows, so

$$n_1 + n_2 + n_3 + n_4 = n = 2^k - 1$$

We solve this for n_1 , using the previous results for the other n's, to get $n_1 = (2^k - 1) - 3 \cdot 2^{k-2}$. We have added these solved-for counts as the Shifted n's in the above table.

Now look back at the above discussion. We assumed that the two starting sequences were *shifted* by some non-zero amount. More specifically, the sequences had to be shifted by an amount not equal to a multiple of the period $2^k - 1$ of the characteristic sequence. If this were not true, then the sum code word (column 3) would be all zeros, and its weight would then not be w , as we assumed above. In this case the above counts are wrong, but we know what the right ones are. They appear in the Unshifted n's column of the table, and they just follow from Fact 4'.

We have now proved the following:

Fact 7: Exactly 2^{k-2} ones appear in (one period's worth of) the product of two shifted binary characteristic sequences. If the sequences are unshifted, or are shifted by a multiple of the period, then exactly 2^{k-1} ones appear in the product.

Probability of strings of 1's and 0's.

In the next section we shall argue that, for reasonably large k , any characteristic sequence is very close to being a so-called white sequence. For such a totally random sequence, we can arrive at certain conclusions which we then apply to the characteristic sequence. One of these has to do with the probability of the occurrence of strings of repeated symbols.

Obviously the characteristic sequence is not perfectly white in that it never has more than k ones in a row, and never more than $k-1$ zeros in a row. These facts were proved above. So we take the following results as a reasonable estimates, not exact facts. We shall estimate the error as well.

Consider an operating shift register generator. We let it run for a long time and we store all of the output data. Then later on, we scan through the data and count the strings of 1's of various lengths, and we can then compute the relative probability of each length. In doing this scan of the data, we run our finger along the data until we find a 0-1 transition, then we count the number of 1's in the string.

With the "white" assumption above, we can estimate the relative probability of the occurrence of 1-strings of various lengths. Let p = probability of a 1, and q = probability of a 0. From Fact 4':

$$p = 2^{k-1} / (2^k - 1) \quad q = (2^{k-1} - 1) / (2^k - 1) \quad p + q = 1 \quad (2.4.1)$$

For $k = 9$, $p = 256/511$ and $q = 255/511$. In general, $p \approx q \approx 1/2$.

Here is a picture outlining the possibilities for 1-strings of lengths 1, 2 and 3. The vertical bar indicates the 0-1 transition in the data stream. On the right we indicate the relative probability of each situation. This probability is conditioned on the fact that we found a 0-1 transition:

...	x	x		0	1		0	x	x	...	q
...	x	x		0	1		1	0	x	...	pq
...	x	x		0	1		1	1	0	...	ppq

In general, the probability of our finding a string of 1's of length n is:

$$P(n) = q p^{n-1} = (q/p) p^n \quad (2.4.2)$$

Roughly setting $p = q = 1/2$, we get

$$P(n) \approx 1/(2^n) \quad (\approx \text{ for shift register generator, } = \text{ for white sequence})$$

Thus, the probability of finding strings of lengths 1 through 10 is

$P(1) = 1/2$	$P(6) = 1/64$
$P(2) = 1/4$	$P(7) = 1/128$
$P(3) = 1/8$	$P(8) = 1/256$
$P(4) = 1/16$	$P(9) = 1/512$
$P(5) = 1/32$	$P(10) = 1/1024$

We would now like to sum these probabilities to see how much in error our estimate is for a characteristic sequence. First consider the elementary series sum

$$\sum_{n=1}^m p^n = \frac{p - p^{m+1}}{1-p} = (p/q) (1 - p^m)$$

Now consider the sum of our 1-string probabilities, and make use of the above result:

$$\text{total probability of all 1-strings} = \sum_{n=1}^m P(n) = (q/p) \sum_{n=1}^m p^n = 1 - p^m$$

For a true random sequence of any $p < 1$, we would add up all terms to $m = \infty$ and the total probability is 1. In particular, for a white sequence having $p = 1/2$, this is true.

For a characteristic sequence, the longest possible string of 1's is k , see Fact 3'. Thus, we must truncate the sum at k , so we find:

$$\text{total probability of all 1-strings for char sequence} = 1 - p^k \quad (2.4.3)$$

Since $p \approx 1/2$, the error is $(1/2)^k$. For $k = 9$, this is $1/512 \approx 0.2\%$. Because there are no strings of ones longer than k , we have to raise the probability of all smaller strings by about 0.2%. This then gives an idea of the error involved in comparing a characteristic sequence to a white sequence.

As for strings of *zeros*, we can make an identical argument, except p and q are reversed. We get,

$$P(n) = p q^{n-1} = (p/q) q^n \approx 1/(2^n)$$

In this case, the longest allowed string is $k-1$ zeros, so our total probability sum stops one short of the 1's case. Adding up the total probability, we get:

$$\text{total probability of all 0-strings for char sequence} = 1 - q^{k-1} \quad (2.4.4)$$

Since $q \approx 1/2$, the error is $(1/2)^{k-1}$. For $k = 9$, this is $1/256 \approx 0.4\%$. So here we have to raise all our estimates of 0-string lengths by about 0.4%.

We can summarize the above in the following fact:

Fact 8: In a characteristic sequence of reasonably large k , the distribution of string lengths is very close to the theoretical white sequence distribution, up to a length where strings are no longer allowed to exist. Up to this length, the relative probability of a string of exactly n identical symbols is given by $P(n) \approx 1/2^n$. For $k = 9$, the error in this estimate is $< 0.5\%$.

2.5 Autocorrelation, Probability, and Statistics for a sequence in GF(2)

The autocorrelation function for a continuous signal $a(t)$ was defined in Spectral (6.1.1), where we omitted any kind of normalizing factor,

$$r(t) = \int_{-\infty}^{\infty} dt' a(t') a(t' + t)$$

Although we did not use this terminology in Spectral Chapter 6, it should be clear that we can define an exactly corresponding "digital" autocorrelation function having the following form:

$$r_k = \sum_{n=-\infty}^{\infty} a_n a_{n+k} \quad (2.5.1)$$

At this point, we still imagine that $a_n = a(t_n)$ is a sample of an analog signal. The sum converges if the sequence tapers off sufficiently in the past and future.

For a sequence of numbers in GF(2), the only way to make something "taper off" is to have it be zero before some time in the past and after some time in the future. For a periodic sequence the above definition cannot possibly converge, so we go with an alternative definition,

$$r_k = (1/P) \sum_{n=1}^P a_n a_{n+k} \quad (2.5.2)$$

where $P = 2^k - 1$ = the length of a characteristic sequence. Here it is implied that if a_{n+k} takes you off the end of the sequence, you wrap to the beginning of the sequence, just as if the sequence repeated forever in time. The only difference between this and the above definition is an infinite normalization factor.

The sum shown above is the average value of the product $a_n a_{n+k}$ for a characteristic sequence. This would also be true for any periodic sequence, or an aperiodic one as well as long as P is taken sufficiently large. If we were to take a statistical ensemble of shift register generators started in random starting states, we would identify the above sum as the statistical expectation value of $a_n a_{n+k}$, as it appeared in Spectral Chapter 6. Alternatively, we could examine the output of one shift register generator over random time intervals.

Thus, we now make a connection to Spectral Chapter 6 by associating the quantity $\langle a_n a_m \rangle$ with our digital autocorrelation function above:

$$\langle a_n a_{n+k} \rangle = r_k + (1/P) \sum_{n=1}^P a_n a_{n+k} \quad (2.5.3)$$

In particular, in Spectral Chapter 6 we derived a fairly general formula for the spectral density of a random pulse train in terms of $\langle a_n a_m \rangle$. The formula of interest is (6.4.7), and we now relate the coefficients α and β to our digital autocorrelation function:

$$\begin{aligned} \alpha &= \langle a_m a_n \rangle = r_{m-n} \quad \text{for } m \neq n \\ \beta &= \langle a_n^2 \rangle = r_0 \end{aligned} \quad (2.5.4)$$

Formula (6.4.7) assumes that the autocorrelation function r_k is constant for all non-zero k . If this is not the case, we have to back up to the earlier formula (6.3.9) which is completely general. In fact, this was what we had to do for our AMI line code example on Section 6.6 (d).

We shall soon compute the digital autocorrelation function r_k for a characteristic sequence, and then use the result in the formulas of Chapter 6 to obtain the frequency spectrum of the characteristic sequence. But first, we insert the following brief review of "probability and statistics" in relation to $\langle a_m a_n \rangle$ and r_k .

A few Basics of Probability and Statistics Applied to GF(2)

The probability of *events* A and B both being true is denoted by $P(A \leftrightarrow B)$, or $P(A, B)$. This is known as the joint second-order probability density function (pdf). A related concept is $P(A|B)$ meaning the conditional probability that A is true given that B is true. Here are the relations:

$$P(A, B) = P(A \leftrightarrow B) = P(A|B) P(B) = P(B|A) P(A) \quad (2.5.5)$$

If events A and B are *statistically independent*, one gets:

$$P(A, B) = P(A \leftrightarrow B) = P(A) P(B) \quad \square \quad P(A|B) = P(A), \quad P(B|A) = P(B) \quad (2.5.6)$$

The result $P(A|B) = P(A)$ means that the probability of A being true is not influenced by whether or not B is true.

Similarly one can define joint pdf's of third order and beyond. For example, statistical independence for a third-order pdf is indicated by,

$$P(A, B, C) = P(A \leftrightarrow B \leftrightarrow C) = P(A)P(B)P(C) \quad (2.5.7)$$

We now apply these general ideas to a sequence of numbers a_n in GF(2). The "event" of interest is that $a_n = 1$. The following short hand notation is useful:

$$p(a_n) + P(a_n=1)$$

$$p(a_n, a_m) + P(a_n=1, a_m=1) \quad (2.5.8)$$

We can now express various mean values in terms of these pdf's:

$$\langle a_n \rangle = 1 \cdot P(a_n=1) + 0 \cdot P(a_n \neq 1) = P(a_n=1) = p(a_n)$$

$$\langle a_n^2 \rangle = 1^2 \cdot P(a_n=1) + 0^2 \cdot P(a_n \neq 1) = P(a_n=1) = p(a_n)$$

$$\langle a_m a_n \rangle = 1^2 \cdot P(a_m=1, a_n=1) + 3 \text{ zero terms} = P(a_m=1, a_n=1) = p(a_m, a_n) \quad (2.5.9)$$

If $m \neq n$, it is possible that a_m and a_n could be statistically independent. In this case, one would have:

$$\langle a_m a_n \rangle = p(a_m, a_n) = p(a_m) p(a_n) = \langle a_m \rangle \langle a_n \rangle \quad (2.5.10)$$

For $m=n$, it is impossible that a_m and a_n be statistically independent, and one has:

$$\langle a_n a_n \rangle = p(a_n, a_n) = \langle a_n^2 \rangle = p(a_n) \neq p(a_n) p(a_n) \quad (2.5.11)$$

In other words, the "diagonal" second-order statistical quantity $p(a_n, a_n)$ is really the first-order quantity $p(a_n)$.

We can write down a few related statistical quantities, using $x = a_n$:

The moments for $N = 1, 2, 3, \dots$:

$$m_N = \langle x^N \rangle = \langle a_n^N \rangle = 1^N \bullet P(a_n=1) = p(a_n) = \langle x \rangle \quad (2.5.12)$$

The central moments for $N = 1, 2, 3, \dots$:

$$\mu_N = \langle (x - \langle x \rangle)^N \rangle = \sum_{j=0}^N \binom{N}{j} (-1)^j \langle x^{N-j} \rangle \langle x \rangle^j \quad (2.5.13)$$

The $N=2$ central moment (= variance = dispersion = σ^2 = square of standard deviation):

$$\mu_2 = \langle (x - \langle x \rangle)^2 \rangle = \langle x^2 \rangle - \langle x \rangle^2 = \langle x \rangle - \langle x \rangle^2 = p(a_n)[1 - p(a_n)] \quad (2.5.14)$$

The statistical quantities listed above are all expressible in terms of the first order pfd $p(a_n)$. In general, a full characterization of a statistical sequence requires specification of the probability density functions of all orders. However, much useful information is contained in the first two orders, and this information is sufficient for many applications.

Fact 1: The first and second order statistics are completely determined by the autocorrelation function.

Proof: Assume that the autocorrelation function r_k is known. Then,

first order:

$$p(a_n) = \langle a_n \rangle = \langle a_n^2 \rangle = p(a_n) = r_0$$

second order:

$$p(a_n, a_n) = \langle a_n^2 \rangle = p(a_n) = r_0$$

$$p(a_m, a_n) = \langle a_m a_n \rangle = r_{m-n} \quad m \neq n$$

Corollary 1: Third and higher order statistics are not determined by the autocorrelation function. One needs a fancier entity to get these higher statistics.

Example: Consider the third-order density:

$$p(a_n, a_{n+k}, a_{n+k'}) = \langle a_n a_{n+k} a_{n+k'} \rangle$$

One could relate this to some kind of fancier 2-variable autocorrelation function defined on a characteristic sequence of period P :

$$r_{k,k'} + (1/P) \sum_{n=1}^P a_n a_{n+k} a_{n+k'}$$

Fact 2: For a statistically independent sequence, all higher order statistics are determined by the first order statistics:

$$\langle a_m a_n \rangle = p(a_m, a_n) = p(a_m) p(a_n) = [p(a_m)]^2 \quad m \neq n$$

$$\langle a_m a_n a_k \rangle = p(a_m, a_n, a_k) = p(a_m) p(a_n) p(a_k) = [p(a_m)]^3 \quad m \neq n \neq k$$

etc.

Proof: This follows from the definition of "statistical independence" given above.

Definition: A *white* sequence over GF(2) is one which is statistically independent and has $p(a_n) = 1/2$.

For such a sequence, we have:

$$\langle a_n \rangle = p(a_n) = 1/2$$

$$\langle a_m a_n \rangle = p(a_m, a_n) = p(a_m) p(a_n) = 1/4 \quad m \neq n$$

$$\langle a_m a_n a_k \rangle = p(a_m, a_n, a_k) = p(a_m) p(a_n) p(a_k) = 1/8 \quad m \neq n \neq k$$

$$\text{etc.} \quad (2.5.15)$$

Here is a plot of the autocorrelation function for a white sequence a_n defined over GF(2):



Figure 2.5.1: Autocorrelation function of a white sequence over GF(2) = {0,1}.

Modified GF(2): Coefficients are (+1, -1).

It is sometimes convenient to think of the elements of the sequence a_n as belonging to the set $\{-1, +1\}$ instead of to the official GF(2) set $\{0, 1\}$. In this case, various statements made above must be modified. For example,

$$P(a_n = 1) = p(a_n)$$

$$P(a_n \neq 1) = P(a_n = -1) = 1 - p(a_n)$$

$$\langle a_n \rangle = 1 \cdot p(a_n) + (-1) \cdot [1 - p(a_n)] = P(a_n=1) = [2 p(a_n) - 1]$$

$$\langle a_n^2 \rangle = 1^2 \cdot p(a_n) + (-1)^2 \cdot [1 - p(a_n)] = 1$$

$$\begin{aligned} \langle a_m a_n \rangle &= 1^2 \cdot P(a_m=1, a_n=1) + 1 \cdot (-1) P(a_m=1, a_n=-1) + \\ &\quad (-1) \cdot 1 \cdot P(a_m=-1, a_n=1) + (-1)^2 P(a_m=-1, a_n=-1) \end{aligned} \quad (2.5.9)'$$

The moments for $N = 1, 2, 3, \dots$:

$$m_N = \langle x^N \rangle = \langle a_n^N \rangle = 1^N p(a_n) + (-1)^N [1 - p(a_n)] \quad (2.5.12)'$$

$$\begin{aligned} &= 1 && \text{for } N \text{ even} \\ &= [2 p(a_n) - 1] && \text{for } N \text{ odd} \end{aligned}$$

The $N=2$ central moment (= variance = dispersion = σ^2 = square of standard deviation):

$$\mu_2 = \langle (x - \langle x \rangle)^2 \rangle = \langle x^2 \rangle - \langle x \rangle^2 = 1 - [2 p(a_n) - 1]^2 \quad (2.5.14)'$$

For a *white sequence*, we have:

$$\langle a_n \rangle = p(a_n) = 0$$

$$\langle a_m a_n \rangle = p(a_m, a_n) = p(a_m) p(a_n) = 0 \quad m \neq n$$

$$\langle a_m a_n a_k \rangle = p(a_m, a_n, a_k) = p(a_m) p(a_n) p(a_k) = 0 \quad m \neq n \neq k$$

$$\text{etc.} \quad (2.5.15)'$$

And here is a plot of the autocorrelation function for a white sequence a_n defined over $\{-1, +1\}$:

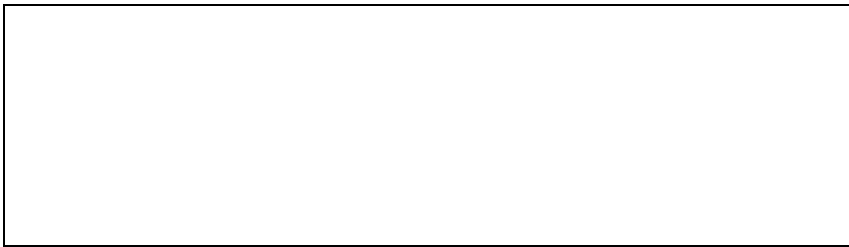


Figure 2.5.2: Autocorrelation function of a white sequence over $\{-1, +1\}$.

2.6 The Spectral Power Density of a Shift Register Generator

Consider a shift register generator of k stages having a primitive feedback polynomial $h(x)$. What is the frequency spectrum of the characteristic sequence output by this piece of hardware? There are really

two different answers to this question, depending on what the output driver looks like. We consider the following two cases:

Case 1: (single-ended) On a 1 we make a pulse $x_{\text{pulse}}(t)$, and on a 0 we make no pulse. For 1101,



Figure 2.6.1: Encoding of 1101 using arbitrary pulse shape, singled ended drive.

Case 2: (differential) On a 1 we make a pulse $x_{\text{pulse}}(t)$, and on a 0 we make a pulse $-x_{\text{pulse}}(t)$. For 1101,

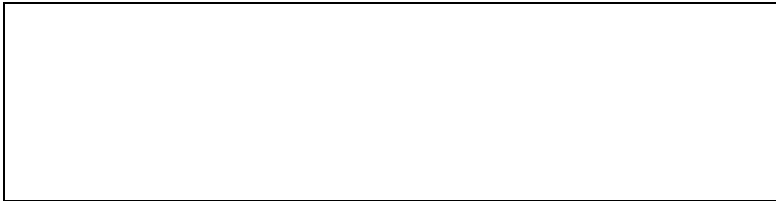


Figure 2.6.2: Encoding of 1101 using arbitrary pulse shape, differential drive.

When $x_{\text{pulse}}(t)$ is a box of width T_1 (as it would be for an ideal NRZ line code), Case 1 and Case 2 are related by a factor of 2 scale change and a DC offset. T_1 is the spacing between dotted lines in the above figures. We prefer to deal with an arbitrary pulse shape in the following analysis, so Case 1 and Case 2 are quite different.

In either case, our method of analysis will be the same. We compute the two quantities $\alpha = \langle a_m a_n \rangle$ and $\beta = \langle a_n^2 \rangle$, and we make use of our general formula given in Spectral Chapter 6, boxed equation (6.4.7). We anticipate that these averages will be independent of the subscripts, and so (6.4.7) applies. If this were not the case, we would have to back up to equation (6.3.9) as we did in the analysis of the AMI line code in Spectral Section 6.6(d). In quantity $\alpha = \langle a_m a_n \rangle$ it is implied that $m \neq n$.

Case 1 Analysis: Singled Ended Output

We replicate our table from Section 2.4 above. Recall that this table describes the product of two characteristic sequences, and just such a product sequence is what appears summed in the definition of the autocorrelation function, Equation (2.5.2).

<u>The Counts</u>	<u>Column 3</u>	<u>Column 4</u>	<u>Shifted n's</u>	<u>Unshifted n's</u>
n_1 = number of (0,0)'s	sum = 0 product = 0	$(2^k - 1) - 3 \cdot 2^{k-2}$	$2^{k-1} - 1$	
n_2 = number of (0,1)'s	sum = 1 product = 0	2^{k-2}		0
n_3 = number of (1,0)'s	sum = 1 product = 0	2^{k-2}		0
n_4 = number of (1,1)'s	sum = 0 product = 1	2^{k-2}		2^{k-1}

The product of two characteristic sequences consists of 0's and 1's as shown in Column 4. To get $\alpha = \langle a_m a_n \rangle$, we use the Shifted column for the values of the n_i counts, while for $\beta = \langle a_n^2 \rangle$ we use the Unshifted counts column. In these sums, the product=0 combinations make no contribution, so we add up only the product=1 combinations. We state the results for general k , and for $k = 9$ as an example:

$$\begin{aligned}
 \alpha = \langle a_m a_n \rangle &= n_4 / P = 2^{k-2} / (2^k - 1) &= (1/4)(1 + 1/P) &= 128/511 \\
 \beta = \langle a_n^2 \rangle &= n_4 / P = 2 \cdot 2^{k-2} / (2^k - 1) &= (1/2)(1 + 1/P) &= 256/511 \\
 (\beta - \alpha) &= 2^{k-2} / (2^k - 1) &= (1/4)(1 + 1/P) &= 128/511 \quad (2.6.1)
 \end{aligned}$$

Recall that $P = 2^k - 1$. Here is a plot:

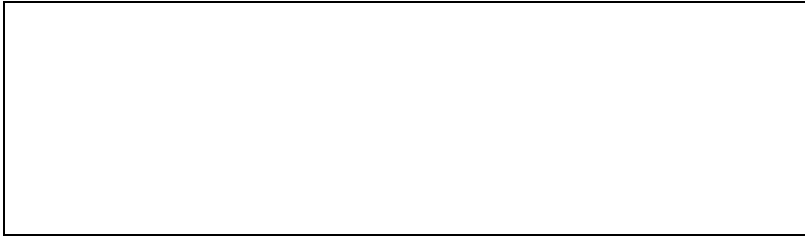


Figure 2.6.3: Autocorrelation function for Case 1 characteristic sequence

Fact 1: Comparison of Figure 2.6.3 with Figure 2.5.1 shows that the characteristic sequence has an autocorrelation function that differs only slightly from that of a white sequence. For $k = 9$, $P = 511$ so $(1 + 1/P) \approx 1.002$. Thus, the first and second order statistics of a characteristic sequence are very close to those of a white sequence.

Proof: See Fact 1 of Section 2.5 above.

The corresponding spectral power density from Spectral (6.4.7) is,

$$\frac{\langle |X(\omega)|^2 \rangle}{T} = \frac{|X_{\text{pulse}}(\omega)|^2}{T_1} \left\{ (1/4)(1 + 1/P) + (1/4)(1 + 1/P) \sum_{m=-\infty}^{\infty} 2\pi\delta(\omega T_1 - 2\pi m) \right\} \quad (2.6.2)$$

Notice that the continuous and discrete spectra have exactly equal strengths, and these strengths are just slightly greater than $1/4$. For $k=9$, we get coefficients $128/511 = .0250489$.

The above result is in terms of a general pulse shape. If we set $P = \infty$ in the above formula, we recover the spectral density of a white sequence. If we also specialize to the case of a square wave pulse, we replicate our earlier result for an NRZ line code having $p = 1/2$. See Spectral equation (6.5.1).

Case 2 Analysis: Differential Output

We modify the above table by replacing (0,1) with (+1,-1). We are basically scaling our coefficients by 2 and giving them an offset of -1 unit compared to Case 1. It was already noted that the resulting analog signals are considerably different between Case 2 and Case 1.

<u>The Counts</u>	<u>Column 3</u>	<u>Column 4</u>	<u>Shifted n's</u>	<u>Unshifted n's</u>
n_1 = number of (-1,-1)'s	sum = -2	product = 1	$(2^k - 1) - 3 \bullet 2^{k-2}$	$2^{k-1} - 1$
n_2 = number of (-1,1)'s	sum = 0	product = -1	2^{k-2}	0
n_3 = number of (1,-1)'s	sum = 0	product = -1	2^{k-2}	0
n_4 = number of (1,1)'s	sum = 2	product = 1	2^{k-2}	2^{k-1}

The product of two characteristic sequences consists of -1's and 1's as shown in Column 4. To get $\alpha = \langle a_m a_n \rangle$, we use the Shifted column for the values of the n_i counts, while for $\beta = \langle a_n^2 \rangle$ we use the Unshifted counts column. This time, we get contributions from all four possible combinations. We state the results for general k, and for k = 9 as an example: (recall P = length of sequence = $2^k - 1$)

$$\begin{aligned}
\alpha &= \langle a_m a_n \rangle = (n_1 - n_2 - n_3 + n_4) / P = -1/P &= -1/511 \\
\beta &= \langle a_n^2 \rangle = (n_1 - n_2 - n_3 + n_4) / P = 1 &= 1 \\
(\beta - \alpha) &= 1 + 1/P &= 1 + 1/511
\end{aligned} \tag{2.6.3}$$

Here is a plot:



Figure 2.6.4: Autocorrelation function for Case 2 characteristic sequence

Upon comparing Figure 2.6.4 with Figure 2.5.2, one sees again that the autocorrelation function for the characteristic sequence (now defined over $+1, -1$) is just slightly different from that of a white sequence.

The corresponding Case 2 spectral power density from (6.4.7) is,

$$\frac{\langle |X(\omega)|^2 \rangle}{T} = \frac{|X_{\text{pulse}}(\omega)|^2}{T_1} \left\{ 1 + 1/P + (-1/P) \sum_{m=-\infty}^{\infty} 2\pi \delta(\omega T_1 - 2\pi m) \right\} \tag{2.6.4}$$

where $P = 2^k - 1$. Now almost all the power is in the continuous spectrum. For a white sequence, we set $P = \infty$ and find that all power is in the continuous spectrum.

For an NRZ signal, Case 1 and Case 2 are essentially the same. They differ only by a factor of 4 due to the factor-of-2 scale change on the height of the pulse. The apparent discrete spectrum in (2.6.2) vanishes due to zeros in the quantity $|X_{\text{pulse}}(\omega)|^2$. Therefore:

Fact 2: Apart from a possible DC line, the spectrum of the characteristic sequence output by a shift register generator is entirely continuous, and it is the envelope of the pulse shape used.

Appendix 2.1: Proof of Fact 4 of Section 2.3

First, we restate the thing we want to prove:

Fact 4: If c is any code word of the cyclic code generated by $g(x)$, then the infinite sequence

$$\{o_i\} = \{c, c, c, c, \dots\}$$

is a solution of the difference equation (2.3.2).

Proof:

(1) In Section 7.1 we gave the following simple proof that the code words of any linear block code G are orthogonal to the code words of the dual code H ,

$$c' \cdot c^T = (d' \cdot H) \cdot (d \cdot G)^T = (d' \cdot H) \cdot G^T \cdot d^T = d' \cdot (H \cdot G^T) \cdot d^T = d' \cdot (0) \cdot d^T = 0$$

Recall that the T transpose symbol is present just to make one vector a column and the other a row, so we can dot them together. We could also write the above as

$$(c')^T \cdot c = 0$$

(2) In Section 7.2 we went on to discuss cyclic codes. Certainly $h(x)$ is a code word of the dual code of $g(x)$. Code words are just multiples of their generator, and $h(x)$ is a multiple of $h(x)$. Thus, we can think of the coefficients of $h(x)$ as the c' in the above equation. Of course $h(x)$ has k coefficients, and we have then to add $n-k$ zeros to make a true dual code vector $\mathbf{h}$. Consider this done. Now, if c is any code word of the cyclic code generated by $g(x)$, we have

$$\mathbf{h}^T \cdot \mathbf{c} = 0$$

If we write this out as a summation, we get

$$\sum_{j=0}^k h_j c_j = 0$$

where the numbers $\{c_0, c_1, c_2, \dots, c_k\}$ are the first k coefficients of any code word of $g(x)$.

We have just proved this fact:

Fact : The dot product of the coefficients of h with the first $k+1$ coefficients of any code word gives 0.

(3) What happens if we dot h with any $k+1$ consecutive coefficients of a code word c ? We know that there is some other code word c' for which these $k+1$ consecutive coefficients are the *first* $k+1$ coefficients. We know this because we know that all cyclic permutations of code words are also code words. Thus we have proved that

Fact: The dot product of the coefficients of h with any consecutive $k+1$ coefficients of any code word gives 0. Consecutive means in the cyclic sense, so that c_{k-1}, c_k, c_0, c_1 are consecutive.

(4) Now consider what it *means* to show that an infinite sequence $\{o\}$ is a solution of the difference equation (2.3.2)

$$\sum_{j=0}^k h_j o_{n+j} = 0 \quad (2.3.2) \text{ repeated}$$

Picture the sequence $\{o_i\}$ as an infinite row of numbers. Picture the sequence $\{h_j\}$ as a finite row of numbers. Place the row $\{h_j\}$ anywhere under the infinite row $\{o_i\}$, then take the dot product of the abutting numbers. This is then the left side of (2.3.2), and the result is supposed to be zero. Varying n means varying the position of the row $\{h_j\}$. Here is a picture drawn for $k=3$, and $n=5$:

$$\begin{array}{cccccccccccccc} o_0 & o_1 & o_2 & o_3 & o_4 & o_5 & o_6 & o_7 & o_8 & o_9 & o_{10} & o_{11} & \dots \\ & & & & & h_0 & h_1 & h_2 & h_3 & & & & \end{array}$$

Now consider our candidate solution of the difference equation:

$$\{o_i\} = \{c, c, c, c, \dots\}$$

Consider the above process. No matter where we position $\{h_j\}$, we will be dotting h with a consecutive set of $k+1$ coefficients of the code word c . We have just shown that this gives 0. Thus, $\{o_i\} = \{c, c, c, c, \dots\}$ is a solution of the difference equation (2.3.2). Q.E.D.

Appendix 2.2: A List of Primitive Polynomials over GF(2)

Data is taken directly from E.J. Watson, **Math. Comp.** 16, pp 368-369 (1962).

This list gives a primitive polynomial for each degree $k = 1 - 100, 107, 127$. As discussed earlier, there are in general many such polynomials for a given k . Recall that reversing the coefficients also gives a primitive polynomial. Notice that many values of k , including odd values, lack 3 term polynomials.

Examples: $9 \ 4 \ 0$ $h(x) = x^9 + x^4 + 152 \ 3 \ 0$ $h(x) = x^{52} + x^3 + 1$