

Chapter 1: Polynomial Processors

Chapter Contents

1.1. Review of the Z Transform

1.2. A Polynomial Divider

Figure 1: a polynomial divider circuit, Symbols, Analysis of Figure 1

Registers: hold the next k symbols of $O(z)$

Example: The Scrambler

1.3 Interpretation of Polynomial Division

Finite Division, The Hidden Remainder

Impulse Response

1.4 The standard form of the polynomial divider.

Figure 2: The standard polynomial divider circuit, Analysis of Figure 2

Registers: hold the current dividend

1.5 Comparison to Long Division of Polynomials

Figure 3: long division

Figure 4 : register contents of Figure 2 after each clock.

Figure 5 : vectorized action of the divider of Figure 2.

Application of Figure 2 to CRC :

Figure 6 : explanation of Figure 1

1.6. A Polynomial Multiplier

Figure 7: A polynomial multiplier circuit, Analysis of Figure 7

Interpretation of Polynomial Multiplication

Impulse Response

Registers: hold the most recent k input symbols of $I(z)$.

Example: The Descrambler

1.7 The standard form of the polynomial multiplier.

Figure 8: The standard polynomial multiplier circuit, Analysis of Figure 8

Registers: hold the current partial product

Figure 9: Comparison of Figure 7 with Long Multiplication

Figure 10: Comparison of Figure 8 with Long Multiplication

Figure 11: Vectorized view of Figure 8 Long Multiplication

1.8 Polynomial Processors in the Time Domain

The Multiplier of Figure 7

The Divider of Figure 1

Convolution Theorem Approach, Summary box

1.9 Simultaneous Polynomial Multiply and Divide

Figure 12: A simultaneous polynomial multiplier and divider.
 Analysis of Figure 12
 Application: A CRC checksum calculator.

Appendix 1.1. Processing polynomials vs. processing integers.

Chapter 1: Polynomial Processors

1.1. Review of the Z Transform

This subject was discussed in much detail in Chapter 3 of our notes on Spectral Theory. There we defined the Z transform of a sequence as

$$F(z) = \sum_{n=-\infty}^{\infty} f_n z^{-n}$$

Here $f_n = f(t_n) = f(n\Delta t)$, a function of time $f(t)$ sampled at discrete times. If we restrict our interest to sequences f_n which vanish for negative time $n < 0$, we can write this as

$$F(z) = \sum_{n=-\infty}^{\infty} \theta(n) f_n z^{-n} = \sum_{n=0}^{\infty} f_n z^{-n} \quad (1.1.1)$$

Consider the sequence f_{n-1} . This is simply f_n *delayed* 1 time unit (ie, shifted 1 time unit Δt into the future). For example, if $f(t)$ has a strong peak at zero argument, then f_n peaks at $t=0$, whereas f_{n-1} peaks at $t = 1 \Delta t$, so the peak moved to a later time.

Let us denote the Z transform of f_{n-1} by $F_{-1}(z)$. The -1 suggests a delay of 1 unit, and happens to match the notation in the next section. Then we might refer to $F(z)$ as $F_0(z) = F(z)$.

How is this $F_{-1}(z)$ related to $F(z)$?

$$F_{-1}(z) = \sum_{n=0}^{\infty} f_{n-1} z^{-n} = z^{-1} \sum_{n=0}^{\infty} f_n z^{-n} + f_{-1} = z^{-1} F(z)$$

Since we assumed f_n was zero for $n < 0$, we can set $f_{-1} = 0$. Similarly, we can go on to show:

$$F_{-m}(z) = \sum_{n=0}^{\infty} f_{n-m} z^{-n} = z^{-m} F(z) = z^{-m} F_0(z) \quad (1.1.2)$$

Here f_{n-m} is a sequence that is delayed m clocks after f_n . Its transform is z^{-m} times the transform of f_n . Thus, in the z -domain ($z = \exp(i\omega\Delta t)$), we can associate a factor of z^{-1} for each flipflop (register) delay

of a signal. This is such an important point, and there are going to be so many powers of z floating around, we will repeat:

Fact: One can interpret z^{-m} as a delay of a digital signal by m clocks.

Example: If you run f_n into a chain of m registers, then f_{n-m} is what comes out of the last register.

We can of course invert the above relationship for a function which is *advanced* m clocks relative to $f(n)$:

$$F_m(z) = \sum_{n=0}^{\infty} f_{n+m} z^{-n} = z^m F(z) = z^m F_0(z) \quad (1.1.3)$$

1.2. A Polynomial Divider

Consider the following circuit:

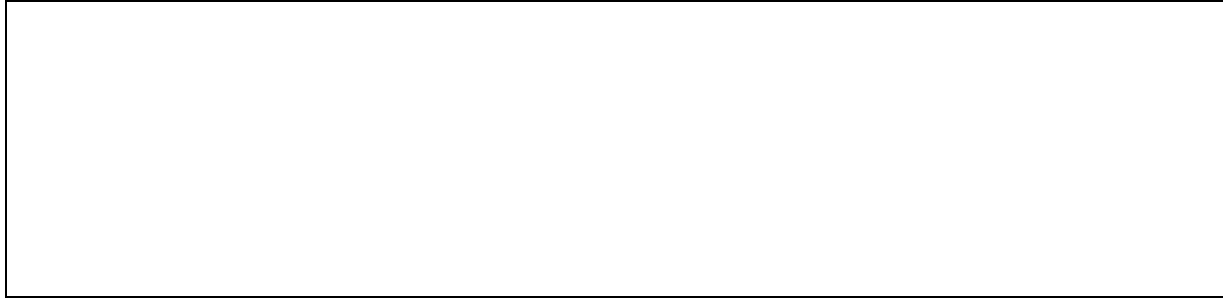


Figure 1: A polynomial divider circuit.

Before analyzing the above circuit, and others similar to it, we wish to make a clear statement of the degree of generality these circuits have. The reader should not jump to the conclusion that these circuits process only bits, but rather that they process *symbols*. Basically, we are dealing with a topology here that is not dependent on the specific nature of the symbols.

Symbols

We shall call the basic computational elements in the above circuit *symbols*. All lines in the drawing carry such symbols. The adders add two such symbols using an operation $+$; the X 's indicate places where a symbol being carried on a line is multiplied by some constant symbol $-h_j$ using an operation $\bullet$; each register holds a symbol; the input and output data streams consist of symbols.

A very general way to interpret a symbol is as an m -tuple of numbers which are elements of the ring $\text{Mod}(p)$. In this case, we can think of each register as a latch consisting of m "p-ary flipflops" which are special in that they can store not just 2 but p different numbers: $0, 1, 2, \dots, p-1$. In this case, one can regard a symbol as an element of a ring we will just call $\text{Ring}(p^m, \bullet, +)$. The definition of the circuit would then be complete once we specified how the operations $\bullet$ and $+$ act on the p^m ring elements.

Definition: A p -ary flipflop stores a *pit*. If $p=2$, a pit is a bit. That is, a binary pit is a bit. Current computer circuits tend to deal with bits and not $p>2$ pits; the future may be different.

Example 1: One specification for the operations $\bullet$ and $+$ would be to say $\text{Ring}(p^m, \bullet, +) = \text{Mod}(p^m)$. More explicitly, we might consider $p=2, m=16$ and write $\text{Ring}(2^{16}, \bullet, +) = \text{Mod}(2^{16}) = \text{Mod}(16384)$. In this case, the symbols in the above Figure are 16-bit binary numbers, and we would then refer to the circuit as a "digital filter". As we shall soon see, the transfer function for this filter is $1/H(z)$, where $H(z)$ is a polynomial whose coefficients are the h_j shown in the Figure.

Notice that in such a circuit, there is a "mixing" of the individual bit lines at each place where $+$ or $\bullet$ is performed. For example, an adder is not just 16 independent 1-bit adders. The adders have "carry" between the bit positions. Similarly for the $\bullet$ operations.

There is a subtle restriction on the circuit of this example. Notice that the circuit involves multiplication by $1/h_k = h_k^{-1}$. In general, not all elements of $\text{Mod}(p^m)$ have inverses. { For example, in $\text{Mod}(4)$, there is no element 2^{-1} since $2 \bullet 0 = 0, 2 \bullet 1 = 2, 2 \bullet 2 = 0, 2 \bullet 3 = 2$. } Thus, we would have to make sure that we choose some h_k which has an inverse. A good candidate is $h_k = 1$.

If we are interested in having h_k be an *arbitrary* one of our symbols, we need to restrict our interest to cases where $\text{Ring}(p^m, \bullet, +)$ is a *field*. In a field, every non-zero element always has an inverse. As shown in our Galois notes, $\text{Ring}(p^m, \bullet, +)$ will be a field if and only if p is a prime number. In this case, we have a special notation: $\text{Ring}(p^m, \bullet, +) = \text{GF}(p^m)$, where GF stands for Galois Field. This leads to:

Example 2: Let $\text{Ring}(p^m, \bullet, +) = \text{GF}(p^m)$ where $p = \text{prime}$. In this case, our symbols are m -tuples of numbers which are elements of $\text{Mod}(p)$. For $p=\text{prime}$, $\text{Mod}(p)$ is itself a field which we call $\text{GF}(p)$, and sometimes Z_p . It is just the modulo- p integer field we are all familiar with.

So in this example, our symbols are elements of $\text{GF}(p^m)$, and each number in the symbol m -tuple is an element of $\text{GF}(p)$. The operations $\bullet$ and $+$ for $\text{GF}(p^m)$ are unique and were studied in our Galois notes. A good example of a computer circuit of this type using $p=2$ is a Reed-Solomon encoder.

Observation: $\text{GF}(p^m) \neq \text{Mod}(p^m)$. Both these rings have p^m elements, but their $+$ and $\bullet$ operations are completely different. Moreover, $\text{GF}(p^m)$ is a field, whereas $\text{Mod}(p^m)$ is not a field except for $m=1$.

Example 3: This example is just Example 2 with $m=1$. In this case, a symbol is a 1-tuple -- just a single number. The symbol is an element of $\text{GF}(p)$. If $p=2$, the symbols of $\text{GF}(2)$ are just bits. Examples of this kind of circuit are binary polynomial dividers and scramblers.

Analysis of Figure 1

The equations that go with the above figure are quite straightforward. Notice that all the feedback accumulates and ends up at the input of the leftmost register. We write:

$$q_{k-1}(n+1) = d_{k-1}(n) = (1/h_k) \left[i(n) - \sum_{j=0}^{k-1} h_j q_j(n) \right] \quad (1.2.1)$$

We now have a small notational conflict. In the Z transform discussion above, the sequence index n was treated as a subscript, for example, f_n . Here, we wish to reserve the subscript location to label a *stage* of the shift register. We have to put the n somewhere, so we put it as an argument (n). Hopefully, this should cause no confusion.

The q output of the leftmost register at time $n+1$ will be its d input at time n . If we project the above equation into the z -plane, we get at once (see Z transform discussion above):

$$z Q_{k-1}(z) = D_{k-1}(z) = (1/h_k) [I(z) - \sum_{j=0}^{k-1} h_j Q_j(z)] \quad (1.2.2)$$

We can now use Equation (3) above to slide all these z -domain functions to $Q_0(z)$. We are taking the output as our reference point. Notice that the various Q 's above are all advanced relative to Q_0 .

$$Q_j(z) = z^j Q_0(z) \quad Q_{k-1}(z) = z^{k-1} Q_0(z) \quad (1.2.3)$$

We then get:

$$z^k Q_0(z) = (1/h_k) [I(z) - \sum_{j=0}^{k-1} h_j z^j Q_0(z)] \quad (1.2.4)$$

It is an easy matter to solve this equation for $Q_0(z) = O(z)$, our output function, in terms of $I(z)$, the input function. Here is the result:

$$O(z) = I(z) / H(z) \quad (1.2.5)$$

where

$$H(z) = h_k z^k + h_{k-1} z^{k-1} + \dots + h_1 z + h_0 \quad (1.2.6)$$

We have therefore arrived at the undeniable conclusion that the circuit shown in Figure 1 does in fact divide the incoming polynomial $I(z)$ by the polynomial $H(z)$ to generate an output polynomial $O(z)$. This circuit is a *polynomial divider*.

Registers : If we take a snapshot of the division process of Figure 1 at any clock, we find that the k registers always hold the next k symbols of the "quotient to be ". Just stare at Figure 1. Nothing can alter the contents of these registers as their contents shift to the right.

Example : The Scrambler . If we work in the field of a bit, $GF(2)$, we know that $- = +$ and $2(\text{anything}) = 0$. If we take the polynomial

$$H(z) = z^9 + z^4 + 1$$

then the circuit of Figure 1 becomes exactly the front end of the scrambler which appears in Appendix A of the SMPTE proposed Serial Digital Interface. Thus, one can interpret the action of the scrambler on the incoming signal as division of the incoming signal's polynomial by $z^9 + z^4 + 1$. The output of the scrambler section is the quotient of this division.

1.3 Interpretation of Polynomial Division

In general, the input data stream $I(z)$ entering the circuit of Figure 1 goes on forever:

$$I(z) = i_0 + i_1 z^{-1} + i_2 z^{-2} + \dots \quad (1.3.1)$$

It starts at $t=0$, and then goes on from there. Even if $I(z)$ consists of a finite number of non-zero symbols followed by all zeros, we can still regard it as going on forever. As the divider circuit clocks each input symbol in, it clocks one quotient symbol out. Even if the input stream terminates and becomes all zeros, we have no guarantee that the output stream might not go on forever with non-zero symbols. We can make the following distant analogy with numbers:

$$514.000000... / 37 = 13.891891891891... \quad (1.3.2)$$

Although the dividend 514 ends with all zeros, the quotient in this case goes on forever.

Finite Division

It is helpful now to think of *stopping* the circuit after a finite number of input symbols have been shifted in. In this case, we can regard the input polynomial as finite, namely,

$$I(z) = i_0 + i_1 z^{-1} + i_2 z^{-2} + \dots + i_r z^{-r} \quad (1.3.3)$$

Here, we have shifted in $r+1$ symbols $\{ i_0, \dots, i_r \}$, and then we stop the shift clock. We can rewrite this input sequence as:

$$I(z) = z^{-r} [i_0 z^r + i_1 z^{r-1} + i_2 z^{r-2} + \dots + i_r] + z^{-r} I_r(z) \quad (1.3.4)$$

where we have defined the bracketed polynomial as $I_r(z)$. One should understand that there is no absolute significance to the size of the powers, what matters is the relativity between powers. We have written $I(z) = z^{-r} I_r(z)$ so we can deal with something we are comfortable with, namely $I_r(z)$ which looks like a normal polynomial of degree r . Regardless of how we think of $I(z)$, we know that i_0 is the first symbol (coefficient) to enter the circuit.

Since $H(z)$ has degree k , and since $I(z)$ has degree 0, we know from (2.5) that $O(z)$ must have degree $0-k = -k$. This means that its first k symbols all vanish. This is easily interpreted in terms of Figure 1 since you have to wait k clocks before any non-zero symbols emerge from the circuit output, after i^0 hits the input. (This will also be true of our second circuit to be given below.)

Since we have shifted in $r+1$ symbols, we must have shifted out $r+1$ symbols, of which the first k vanished. Thus, the output sequence contains a total of $r-k+1$ non-zero symbols. So here is $O(z)$:

$$O(z) = o_k z^{-k} + o_{k+1} z^{-k-1} + \dots + o_r z^{-r} \quad (1.3.5)$$

It is now convenient to relabel these coefficients as follows,

$$O(z) = o_0 z^{-k} + o_1 z^{-k-1} + \dots + o_{r-k} z^{-k-r} \quad (1.3.6)$$

Now the first coefficient of our quotient will be called o_0 instead of o_k . Next, we would like to recast the quotient so that the leading power is $r-k$. We do this because we want to think of dividing $I_r(z)$ by $H(z)$. Thus we write:

$$O(z) = z^{-r} [o_0 z^{r-k} + o_1 z^{r-k-1} + \dots + o_{r-k}] + z^{-r} O_{r-k}(z) \quad (1.3.7)$$

All we did was factor out z^{-r} , nothing has been changed. We can now identify $O_{r-k}(z)$ as a polynomial of degree $r-k$ which looks like it ought to be the quotient. So far then we have:

$$O(z) = I(z) / H(z) \quad I(z) = z^{-r} I_r(z) \quad O(z) = z^{-r} O_{r-k}(z)$$

Insert the last two into the first, cancel the z^{-r} and we get:

$$O_{r-k}(z) = I_r(z) / H(z) \quad (1.3.8)$$

This now looks more like the kind of polynomial division we are familiar with. We can take the above equation, sit down, and do the long division by hand if we want, and try to make a comparison with what the circuit of Figure 1 is doing. We shall entertain this task a few sections hence.

The Hidden Remainder

We have seen that if we shift in a polynomial of degree r , and we divide by a divisor $H(z)$ of degree k , we end up with a quotient of degree $r-k$. There is of course a *remainder* that is hidden somehow within the circuit. In other words, the *information* of the remainder is somehow stored in the k registers. We know that the remainder is a polynomial of degree at most $k-1$. Mathematically, we can write:

$$I_r(z) / H(z) = O_{r-k}(z) + R_{k-1}(z)/H(z) \quad (1.3.9)$$

where O is the quotient polynomial, and R is the remainder polynomial. The circuit, however, does not output the second term, it only outputs the first term, the quotient.

Question: What happens if we now shift in one more input symbol?

Answer: In this case, we redo the above analysis replacing r with $r+1$ everywhere. We end up with the circuit then computing:

$$O_{r+1-k}(z) = I_{r+1}(z) / H(z)$$

and mathematically, we know there is again some hidden remainder, so we write:

$$I_{r+1}(z) / H(z) = O_{r+1-k}(z) + R'_{k-1}(z)/H(z)$$

The quotient has just gained one more symbol, but now the remainder is likely to be completely different, so we put a prime on R'.

The maximum possible degree of the remainder never changes, it is always $k-1$. So the remainder has k coefficients, some of which may be zero. And the circuit has k registers. The remainder is some linear combination of the symbols in the registers. If at some point, all registers become 0, the remainder is zero, and then the sequence of quotient symbols terminates.

So we can now summarize what happens when the input sequence contains only a finite number $r+1$ of non-zero symbols. For any number of clocks n after the input sequence goes to zero, we get

$$I_{r+n}(z) / H(z) = O_{r+n-k}(z) + R_{k-1}(z)/H(z)$$

On each clock we get a new quotient symbol, and some new hidden remainder $R^{(n)}_{k-1}(z)$. If at some point the remainder goes to zero, the quotient sequence terminates and the system returns to the idle state which it had at negative time. The other possibility is that symbols keep circulating forever, and the quotient sequence then is infinite.

Impulse Response

This is why our circuit is sometimes called an infinite impulse response filter (IIR). In fact, if the input sequence consists of a single non-zero 1 symbol, then the "filter" output is by definition the impulse response. In this case we would write:

$$I(z) = 1 \quad O(z) = I(z)/H(z) = 1/H(z) \quad (1.3.10)$$

so the circuit is now dividing $H(z)$ into an ever larger polynomial whose leading power has coefficient 1, and whose later powers have zero coefficients. If we wait for r clocks after the unit symbol has shifted in, we have:

$$z^r / H(z) = O_{r-k}(z) + R_{k-1}(z)/H(z)$$

Unless $H(z)$ is a single power of z , the impulse response will go on forever. We shall have more to say about this subject later on.

1.4 The standard form of the polynomial divider.

In the previous section we have given a "scrambler" type implementation of a polynomial divider. We proved that it really does divide polynomials.

Here, we present the more standard form of such a polynomial divider:



Figure 2: The standard polynomial divider circuit.

It is going to turn out that $O(z) = I(z)/H(z)$ exactly as before, but this is certainly not obvious at this point. Comparing this figure to Figure 1, we note several things. There are still k registers, but now we have feedback entering each stage. Also, we have reversed the labeling left to right, so that q_0 and h_0 are now on the left. The output is the output of the last register scaled down by $1/h_k$. This output also drives the feedback bus. It is convenient to identify the input signal i with q_0 of some imagined register off the left edge of the picture.

Analysis of Figure 2

For the j^{th} stage we can write, using notation similar to the previous section,

$$q_{j+1}(n+1) = d_{j+1}(n) = q_j(n) - h_j o(n) \quad j = 0, 1, 2, \dots, k-1 \quad (1.4.1)$$

Projecting this into the z -plane yields,

$$z Q_{j+1}(z) = Q_j(z) - h_j O(z). \quad (1.4.2)$$

It is helpful to examine the two extremes. At the left end of Figure 2, we set $j = 0$ and have $Q_0(z) = I(z)$. At the right end, if we set $j=k$, then we must set $Q_{k+1}(z) = 0 = Q_k(z) - h_k O(z)$.

We can now solve this difference equation by brute force:

$$zQ_1(z) = I(z) - h_0 O(z)$$

$$z^2 Q_2(z) = z Q_1(z) - z h_1 O(z) = I(z) - [h_1 z + h_0] O(z)$$

.....

$$z^{j+1} Q_{j+1}(z) = I(z) - [h_j z^j + \dots + h_1 z + h_0] O(z) \quad (1.4.3)$$

Setting $j=k$ and using the above noted fact that $Q_{k+1}(z) = 0$, we get

$$0 = I(z) - H(z) O(z)$$

which means that

$$O(z) = I(z) / H(z) \quad (1.4.4)$$

Thus, the circuit in Figure 2 is functionally identical to that in Figure 1.

Registers: If we take a snapshot of the division process of Figure 2 at any clock, we claim that the k registers of Figure 2 always contain the most significant k symbols of the "current dividend". We can always interpret the current dividend (contents of the k registers of Figure 2) as being the remainder of the polynomial which so far has been shifted in. We will elaborate on this in the next section. This is very different from what we said about Figure 1. There, the registers contained the next k symbols of the quotient to be. Thus, although the I/O of these circuits is the same, their internal registers have different meanings.

1.5 Comparison to Long Division of Polynomials

It is not difficult to develop an understanding of how the circuits of Figure 1 and Figure 2 actually implement long division. We shall deal first with the standard form of Figure 2, and then handle the scrambler form of Figure 1.

Figure 3 shows Long Division

In Figure 3 we show the long division of a divisor polynomial $H(z)$ into a dividend polynomial of the form $I_T(z)$ as discussed in Section 1.3. Sometimes one forgets how this algorithm works. At each level of the division process there is a "current dividend". The first such dividend is $I_T(z)$. We find the first coefficient of the quotient, multiply it by $H(z)$, and then subtract this product to get the second current dividend. We repeat the process until we arrive at a current dividend whose degree is less than the degree of $H(z)$. We then stop and declare that last current dividend to be the "remainder". Note that we have made use of $\alpha_i = (h_i / h_k)$ as a shorthand.

Although each current dividend may be very large, we only need the most significant k terms in order to proceed at each stage. This is because $H(z)$ has only k terms, and the only interesting part of forming a new current dividend occurs when you subtract k terms from the previous current dividend.

In Figure 3, we have done the first few stages, and have defined some convenient symbols. Notice that the first current dividend involves coefficients of the form i_n (the actual input symbols to our circuits), while the second current dividend has derived coefficients of the form i'_n , and so on. Figure 3 is completely self-contained, the reader is encouraged to verify that it makes sense.

Figure 4 shows the Figure 1 circuit register contents after each clock.

In Figure 4, we relate the long division of Figure 3 to the hardware circuit shown in Figure 2. For the first k clocks, nothing much happens, and the incoming symbols of $I(z)$ simply load up the k registers. When they are fully loaded, we can at once identify the contents of these registers as being the first current dividend in Figure 3. If we do one clock, we see that the registers then hold the second current dividend! At each register, the feedback does just the right thing to generate a coefficient of the next current dividend.

For example, look at the item $i'_1 z^{r-1}$ in the q_k column. By definition, $i'_1 = i_1 - i_0 h_{k-1}/h_k$. This is a sum of two terms. The i_1 came from the q_{k-1} register, while the second term is the feedback term from the q_k register. It must be understood that the powers of z in Figure 3 and 4 are implicit in the circuit, they are place-holders. One never stores something like z^5 in a register.

If you now look at the progression of items stored in register q_k , you see i_0, i'_1, i''_2 , etc. Looking at the equations for each of these, you see that only h_{k-1} is needed to compute the second term each time. This is a good thing, since h_{k-1} is only available to this particular register's input. Note also that the $1/h_k$ is taken care of at the feedback drive point.

To summarize, the registers of the circuit in Figure 2 always store the current dividend. Each register has a local feedback system designed to compute the correct term for the next current dividend. A division is accomplished by a series of successive subtractions. This is why there are minus signs on the various constants h_i . After each subtraction, the result is a new current dividend.

Figure 5 shows vectorized action of the divider.

One can see from Figure 2 that on each clock, a whole *vector* or k -tuple of symbols $\{h_0, h_1, \dots, h_{k-1}\}$ is being subtracted from the register contents on each clock. Figure 5 is geared to this simple way of thinking about Figure 2. Think now of each current dividend as a vector. On each clock, we are subtracting a new multiple of $H(z)$ from this vector to get the new vector current dividend. From this vantage point, we see that Figure 2 is very standardized form of the divider, because we can clearly see how each successive subtraction occurs.

Application of Figure 2 to CRC .

A CRC checksum is precisely the remainder you get by dividing a data polynomial by some generator polynomial $H(z)$. When the incoming polynomial is done being shifted in (data symbols), the feedback path of Figure 2 is switched to ground, and then the next k clocks shift out the remainder. These are then appended to the data symbols as the CRC checksum. The checksum symbols are the *parity check symbols* of a cyclic code in systematic form, as discussed elsewhere in these notes. Later, when it is time to do a CRC check, the entire code word consisting of data symbols followed by parity check symbols is treated

as $I(z)$ and is divided by $H(z)$. The remainder in this case is called the syndrome. If there were no errors in the data of parity check symbols, the syndrome remainder will be zero.

Figure 6 and an explanation of the scrambler style circuit of Figure 1

In Figure 5 we have merely duplicated the information in Figure 3. However, we have not shown the various current dividends. Instead, we imagine adding up the columns for *all* subtractions at once, not after each subtraction. Thus, the bottom line in Figure 5 represents the sum of each column. Notice that if you multiply each entry in this sum row by $1/h_k$, you get the quotient symbols. This multiplication is accomplished by the leftmost X in Figure 1. The result goes into the leftmost register, and it emerges from the right register as a quotient symbol k clocks later.

The items in Figure 5 under the top bar appear to form a triangular array. However, if more terms were filled out, one would find this to be a diagonal strip whose high never exceeds k items. This is because each row is a multiple of $H(z)$, and $H(z)$ only has k terms. It therefore follows that no column sum ever adds up more than k items. These column sums are exactly what all those adders in Figure 1 are doing!

The hard work is now done. We move on to multipliers which are similar but easier to understand.

1.6. A Polynomial Multiplier

Consider the following circuit:

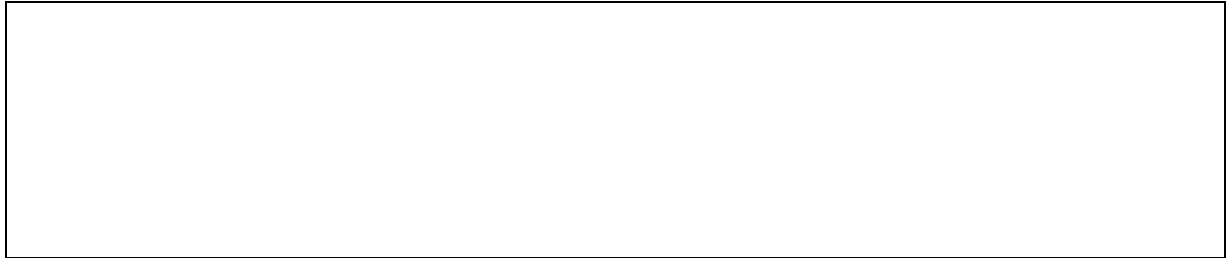


Figure 7: A polynomial multiplier circuit.

Analysis of Figure 7

The analysis of this and the next circuit are similar to the polynomial circuits discussed earlier, so we shall proceed with a minimum of comment. The equation for the output node is:]

$$o(n) = \sum_{j=0}^k h_j q_j(n) \quad (1.6.1)$$

Project into the z plane to get:

$$O(z) = \sum_{j=0}^k h_j Q_j(z) \quad (1.6.2)$$

Translate all $Q_j(z)$ back to the input node using

$$Q_j(z) = z^{-(k-j)} \quad Q_k(z) = z^{-(k-j)} I(z) \quad (1.6.3)$$

The result is

$$O(z) = z^{-k} \left(\sum_{j=0}^k h_j z^j \right) I(z) = z^{-k} H(z) I(z)$$

or

$$z^k O(z) = H(z) I(z) \quad (1.6.4)$$

The circuit shown in Figure 7 must indeed be a *polynomial multiplier*. The factor z^k means that when you think you are done multiplying H and I , the result you get $z^k O(z)$ is the desired product *advanced* k clocks. Thus, you have to do another k clocks to extract all of $O(z)$ from the circuit, as discussed below

Since $I(z)$ is degree 0, and $H(z)$ is degree k , we expect $O(z)$ to be degree 0.

Interpretation of Polynomial Multiplication

Proceeding as in Section 1.3, we consider a "finite multiplication". We shall assume that the input data polynomial $I(z)$ has $r+1$ non zero coefficients, and this is followed by all zeros. It takes $r+1$ clocks to clock in $I(z)$, and most of the product is then formed. It will take another k clocks to clock out the final k bits of the product, which have been left (information-wise) in the registers. So we assume that we are going to do a total of $r+k+1$ clocks to get our answer.

As before, we write the input polynomial in this form,

$$I(z) = z^{-r} [i_0 z^r + i_1 z^{r-1} + i_2 z^{r-2} + \dots + i_r] + z^{-r} I_r(z) \quad (1.6.5)$$

The output polynomial after $r+k+1$ clocks is of degree $r+k$ and thus has the form

$$O(z) = o_0 + o_1 z^{-1} + \dots + o_{r+k} z^{-(r+k)} \quad (1.6.6)$$

We factor out $z^{-(r+k)}$ to get a proper polynomial,

$$O(z) = z^{-(r+k)} [o_0 z^{r+k} + o_1 z^{r+k-1} + \dots + o_{r+k}] + z^{-(r+k)} O_{r+k}(z) \quad (1.6.7)$$

So far then we have:

$$z^k O(z) = H(z) I(z) \quad I(z) = z^{-r} I_r(z) \quad O(z) = z^{-(r+k)} O_{r+k}(z)$$

Installing the last two in the first then yields,

$$O_{r+k}(z) = H(z) I_r(z) \quad (1.6.8)$$

This now looks like the kind of polynomial multiplication we are familiar with. We can take the above equation, sit down, and do the long multiplication by hand if we want, and try to make a comparison with what the circuit of Figure 7 is doing. We shall carry out this task below.

Impulse Response

If we inject an $I(z)$ which has $i_0 = 1$ as its only non-zero coefficient, we have $I(z) = 1$ and the result is

$$z^k O(z) = H(z) \quad (1.6.9)$$

The impulse response is just $H(z)$. As usual, we have to supply k clocks to extract this thing from the circuit. The circuits of Figure 7 and Figure 8 to come have no feedback, so they are FIR "filters". They have a finite impulse response, as we have just seen. It lasts k clocks and is then gone, and the system is back to its steady state of idleness.

Registers: The registers in the circuit of Figure 7 hold the most recent k input symbols of $I(z)$.

Example: The Descrambler . If we work in the field of a bit, $GF(2)$, we know that $- = +$ and $2(\text{anything}) = 0$. If we take the polynomial

$$H(z) = z^9 + z^4 + 1$$

then the circuit of Figure 7 becomes exactly the back end of the descrambler which appears in Appendix A of the SMPTE proposed Serial Digital Interface. Thus, one can interpret the action of the descrambler on its incoming signal as multiplication of the incoming signal's polynomial by $z^9 + z^4 + 1$. The output of the descrambler section is the product of this multiplication.

1.7 The standard form of the polynomial multiplier.

In the previous section we have given a "descrambler" type implementation of a polynomial multiplier. We proved that it really does multiply polynomials.

Here, we present the more standard form of such a polynomial multiplier:



Figure 8: The standard polynomial multiplier circuit.

It is going to turn out that $z^k O(z) = I(z) H(z)$ exactly as before, but this is probably not obvious at this point. Just as Figure 7 was very similar to Figure 1, Figure 8 above is very similar to Figure 2.

Analysis of Figure 8

For the j^{th} stage we can write,

$$q_{j+1}(n+1) = d_{j+1}(n) = q_j(n) + h_j i(n) \quad j = 0, 1, 2, \dots, k-1 \quad (1.7.1)$$

Projecting this into the z -plane yields

$$z Q_{j+1}(z) = Q_j(z) + h_j I(z) \quad (1.7.2)$$

As before, we inspect the two extremes. At $j=0$ we note that $Q_0(z) = 0$. At $j=k$, we find

$$z Q_{k+1}(z) = Q_k(z) + h_k I(z) = O(z) \quad (1.7.3)$$

As before, we do a brute forces solution to the difference equation:

$$z Q_1(z) = h_0 I(z)$$

$$z^2 Q_2(z) = z Q_1(z) + h_1 z I(z) = (h_1 z + h_0) I(z)$$

...

$$z^{j+1} Q_{j+1}(z) = (h_j z^j + \dots h_1 z + h_0) I(z) \quad (1.7.4)$$

Setting $j=k$ we get:

$$z^{k+1} Q_{k+1}(z) = z^k O(z) = H(z) I(z)$$

Our result is then

$$z^k O(z) = H(z) I(z) \quad (1.7.5)$$

which is the same as that obtained from Figure 7. Thus, the circuit in Figure 8 is functionally identical to that in Figure 7.

Comparison of the descrambler style multiplier of Figure 7 with Long Multiplication

In Figure 9 we show the multiplication of $H(z)$ by $I_T(z)$ written out long hand. In the corresponding division circuit of Figure 1 we did subtractions. Here, for multiplications, as everyone knows, we are doing a pile of additions, so there are no longer any minus signs anywhere. The circuit of Figure 7 is simply doing the column sums shown in Figure 9. As noted earlier, the array of numbers which appears triangular in Figure 9 is really a diagonal strip, and there are never more than k numbers to add up in one column. These additions are carried out by the adders in the circuit of Figure 7.

Recall that things start in an idle state where all registers are 0. When i_0 is at the input, the first product symbol O_0 is already sitting at the output. After the next clock, i_1 is at the input, i_0 has moved one to the right, and we now have the desired O_1 at the output. As more input symbols shift in, more adders are activated. At the end, there is the little drainage phase alluded to above.

In Figure 7 it is clear what role the registers play. They just hold the most recent k input symbols. When a new input symbol is shifted in, the one in register q_0 shifts out into the "symbol bucket". The input symbols are held only as long as they are needed.

Comparison of the standard circuit of Figure 8 with Long Multiplication

In Figure 10 we show the long multiplication written out with "partial products". That is, instead of doing all the sums at once, as in Figure 9, we do them one at a time. What is left after each sum is a current partial product, analogous to the current dividend of the divider. The registers in the circuit of Figure 7 always contain the current partial product. The feed-forward adders at the input of each register do the necessary computation so that register holds the correct coefficient of the next current partial product. We have worked out a few details in Figure 10.

Figure 11 shows the vectorized picture which is in many ways easier to understand. At each clock, the current partial product is updated by adding to it a multiple of the vector $H(z)$. This vector is multiplied by the current input symbol, as Figure 8 shows.

1.8 Polynomial Processors in the Time Domain

In the previous sections of this Chapter we concentrated on the various circuits from a polynomial point of view. The circuits divided or multiplied two polynomials to produce an output polynomial.

In this section, we focus directly on the symbols of the input and output data streams in both cases, and we give a very concise restatement of the operation of our various circuits. We first do some mechanical derivations to convince the reader that the results are valid, then at the end, we show why these are the right results.

The Multiplier of Figure 7

Already from Eq. (1.6.1) we have most of our desired result,

$$o(n) = \sum_{j=0}^k h_j q_j(n)$$

Recall that q_j refers to register number j , and n is a time index. We are in the time domain here. Looking at Figure 7, it is fairly clear that we can map each $q_j(n)$ to the *left* in the Figure by going *backwards* in time.

$$q_j(n) = q_{j+1}(n-1) = q_{j+2}(n-2) \dots = q_k(n-k+j) = i(n-k+j) \quad (1.8.1)$$

Here, we have mapped a given q_j all the way to the left side of Figure 7 where we identify q_k with the input data stream. If we insert this expression for $q_j(n)$ into the above for $o(n)$ we get,

$$o(n) = \sum_{j=0}^k h_j i(n-k+j)$$

And since n is an arbitrary time index on both sides, we can take $n \rightarrow n+k$. At the same time, we put our time index as a subscript instead of as an argument. This gives our final result:

$$o_{k+n} = \sum_{j=0}^k h_j i_{n+j} \quad (1.8.2)$$

For each integer n , we get a very simple equation relating the output sequence to the input sequence. If we were solving for i in terms of o , this would be called a set of difference equations. However, for the multiplier of Figure 7, we are really interested in o as a function of i . Nevertheless, we shall loosely refer to this as a difference equation.

Claim: Since the Figure 8 multiplier performs the same function as the Figure 7 multiplier, the above equation applies to it as well.

Proof: We leave it to the reader to directly derive the above result for the Figure 8 multiplier. We know the result will be the same, and soon this will become very obvious.

The Divider of Figure 1

We start with (1.2.1) which reads

$$q_{k-1}(n+1) = (1/h_k) \left[i(n) - \sum_{j=0}^{k-1} h_j q_j(n) \right]$$

Move h_k to the left side, then realize that the left side is just the k^{th} term of the sum. Thus,

$$\sum_{j=0}^k h_j q_j(n) = i(n) \quad (1.8.3)$$

Now map the $q_j(n)$ to the *right* by going *forward* in time, see Figure 1:

$$q_j(n) = q_{j-1}(n+1) = q_{j-2}(n+2) = \dots = q_0(n+j) = o(n+j) \quad (1.8.4)$$

Inserting this into the above we get

$$\sum_{j=0}^k h_j o(n+j) = i(n)$$

Again going to subscripts for the time indices, we get our final result,

$$i_n = \sum_{j=0}^k h_j o_{n+j} \quad (1.8.5)$$

This is amazingly similar to the multiplier result given above. Here, however, since we are solving for o in terms of i , we really do have a set of difference equations.

Claim: Since the Figure 2 divider performs the same function as the Figure 1 divider, the above equation applies to it as well. The reader is welcome to derive this fact by brute force.

Convolution Theorem Approach

Consider equation (1.6.4) which says that a polynomial multiplier does in fact multiply polynomials:

$$z^k O(z) = H(z) I(z)$$

We showed that the circuits of Figures 7 and 8 each implement this operation.

The reader is now referred to the summary box at the end of Section 24 of Spectral Theory, Chapter 3. This box summarizes the properties of the Z Transform. Like all derivatives of the underlying Fourier Integral Transform, there is a convolution theorem which operates with respect to the Z transform. If we set our time step $\Delta t = 1$, meaning 1 clock period, the convolution theorem states:

$$A(z) = B(z)C(z) \quad \square \quad a_n = \sum_{j=-\infty}^{\infty} b_{n-j} c_j \quad (1.8.6)$$

We now make the following selections for A,B and C:

$$\begin{array}{lll} A(z) = z^k O(z) & a_n = o_{n+k} & / \text{ see Eq. (1.1.3)} \\ B(z) = I(z) & b_n = i_n & \\ C(z) = H(z^{-1}) & c_n = h_{-n} & \end{array}$$

Note: This last item deserves some comment. Recall that the Z transform is set up for polynomials in "Z transform format", as in Eq. (1.1.1) for example, where coefficients are aligned with negative powers of z. Our I(z) and O(z) have always been in this format, but H(z) is in "standard polynomial format" where we align with positive powers of z. Thus, we write H(z⁻¹) to put H back into the Z transform format. A glance at Eq. (1.1.1) shows that z⁰ ↔ z⁻¹ means that f_n ↔ f_{-n}.

Installing these time domain sequences into the convolution sum of (1.8.6), then taking j ↔ -j on the dummy summation index gives:

$$o_{n+k} = \sum_{j=-\infty}^{\infty} i_{n-j} h_{-j} = \sum_{j=-\infty}^{\infty} i_{n+j} h_j$$

Now we know that h_j vanishes for j<0 and for j>k, so our final result is then

$$o_{k+n} = \sum_{j=0}^k h_j i_{n+j}$$

which agrees with our derivation (1.8.2) above.

We quickly repeat the process for the division equation (1.2.5)

$$I(z) = O(z)H(z)$$

Make the following selections for A,B and C:

$$\begin{array}{ll} A(z) = I(z) & a_n = i_n \\ B(z) = O(z) & b_n = o_n \\ C(z) = H(z^{-1}) & c_n = h_{-n} \end{array}$$

Installing these into the convolution sum gives:

$$i_n = \sum_{j=-\infty}^{\infty} o_{n-j} h_{-j} = \sum_{j=-\infty}^{\infty} o_{n+j} h_j = \sum_{j=0}^k h_j o_{n+j}$$

which duplicates (1.8.5) above.

We now summarize the results of this section:

Polynomial Processors		
	<u>z-domain</u>	<u>time-domain</u>
Polynomial Divider (Figure 1 or Figure 2)	$O(z) = I(z)/H(z)$	$i_n = \sum_{j=0}^k h_j o_{n+j}$
Polynomial Multiplier (Figure 7 or Figure 8)	$z^k O(z) = I(z) H(z)$	$o_{k+n} = \sum_{j=0}^k h_j i_{n+j}$

1.9 Simultaneous Polynomial Multiply and Divide

The following circuit is both useful in cyclic code hardware, and serves as a check on all previous work of this chapter. Consider,

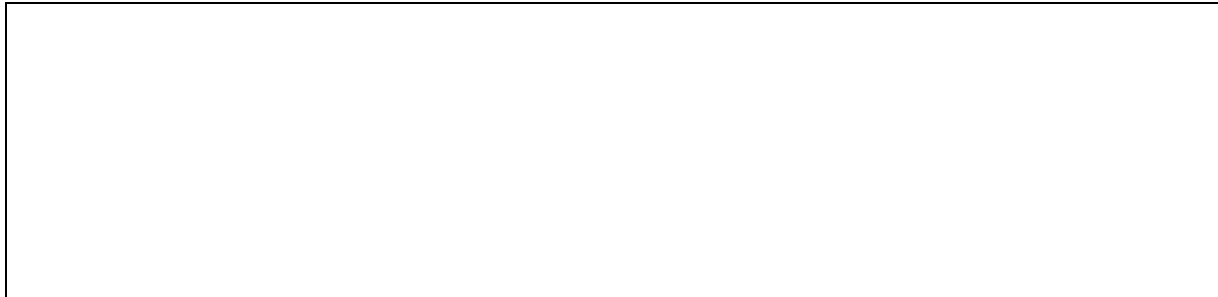


Figure 12: A simultaneous polynomial multiplier and divider.

This circuit looks like a superposition of a multiplier on the top by polynomial $H(z)$, and a divider on the bottom by polynomial $G(z)$. Both polynomials are of degree k .

Analysis of Figure 12

We are by now familiar with the technique. We start with:

$$q_{j+1}(n+1) = q_j(n) + h_j i(n) - g_j o(n) \quad (1.9.1)$$

Project into the z plane to get:

$$zQ_{j+1}(z) = Q_j(z) + h_j I(z) - g_j O(z) \quad (1.9.2)$$

Examine the two extremes. For $j=0$ we note that $Q_0(z) = 0$. For $j=k$, we get $Q_{k+1}(z) = 0$.

The brute force solution of the difference equation proceeds as usual:

$$zQ_1(z) = h_0 I(z) - g_0 O(z)$$

$$z^2 Q_2(z) = z Q_1(z) + h_1 z I(z) = (h_1 z + h_0) I(z) - (g_1 z + g_0) O(z)$$

...

$$z^{j+1} Q_{j+1}(z) = (h_j z^j + \dots h_0) I(z) - (g_j z^j + \dots g_0) O(z) \quad (1.9.3)$$

Setting $j=k$ and using the above limit that $Q_{k+1}(z) = 0$ we get:

$$0 = H(z) I(z) - G(z) O(z)$$

which we can then solve to get

$$O(z) = I(z) H(z) / G(z) \quad (1.9.4)$$

Sure enough, Figure 12 multiplies by $H(z)$ at the same time it divides by $G(z)$. We have seen in the derivation that it keeps track of the successive additions due to the multiplication, as well as the successive subtractions of the division.

We should now be able to take appropriate limits of this result to recover our earlier results. If we select $H(z) = 1$, so that $h_0 = 1$, then we recover the divider of Figure 2:

$$O(z) = I(z) / G(z)$$

On the other hand, to get a multiplier, we need to select $G(z)$ such that $g_k = 1$. This means that we have $G(z) = z^k$. Then we duplicate the results of Figure 8,

$$O(z) = I(z) H(z) / z^k \quad \text{or} \quad z^k O(z) = I(z) H(z)$$

Application: In an encoder for a cyclic code, one needs to compute the remainder of a power z^s times a an incoming polynomial. The above circuit can be used for this purpose. When the input stream is done shifting in, the registers will contain the desired remainder, as discussed earlier. We have:

$$O(z) = [I(z) z^s] / G(z)$$

Thus, $H(z) = z^s$, which means that the input stream is injected only stage s in Figure 12. Ie, $h_s = 1$, and all other coefficients vanish. If this point happens to align with a vanishing coefficient of $g(z)$, then no 3-input adders are needed in the circuit!

We leave as an exercise for the reader to build a simultaneous multiplier/divider out of Figure 1 and Figure 7. It seems that it ought to be possible.

Appendix 1.1. Processing polynomials vs. processing integers.

In this section, we will consider a series of questions which will bring out some distinctions between dealing with polynomials and dealing with integers.

Question 1: Can we make a correspondence between multiplying polynomials and multiplying base-10 integers?

For example, we might try to associate $5z^2 + 1z + 4$ with the base-10 integer 514. If we think of $z = 10$, this seems to be a reasonable association. Now let's multiply two polynomials:

$$(3z+8)(5z+9) = (15z^2 + 67z + 72)$$

On the other hand, we could write down this base-10 integer product:

$$38 \bullet 59 = 2242$$

If we evaluate $(15z^2 + 67z + 72)$ at $z=10$, we do in fact get 2242. *However*, note the following:

$$38 \bullet 59 = 2242 \quad / \quad (3z+8)(5z+9) = (2z^3 + 2z^2 + 4z + 2)$$

We will now give a formal statement of why $A \neq B$.

(1) We can represent a base-10 integer as an m-tuple of elements of the field $Z_{10} = \text{Mod}(10)$. Such m-tuples themselves form a ring which has a 1-to-1 correspondence with the ring of integers Z . Let's call this ring $Z\text{-base-10}$. We have a certain familiar rule for the multiplication of such m-tuples. The rule is that you convert each m-tuple to an integer, get the product, and then convert the result back to an m-tuple. This rule is very effectively implemented by a device known as a calculator.

(2) On the other hand, we can represent a polynomial $f(z)$ as an m-tuple of elements of the integer ring Z . Such m-tuples themselves form a ring known as $\text{Polys}(z, Z)$. We have a rule for multiplying these m-tuples, it is what we do when we do a long multiplication of polynomials. The result can also be represented as an m-tuple.

(3) What we have pointed out above in our $A \neq B$ counterexample is that the two rings Z and $\text{Polys}(z, Z)$ are not the same. Once again,

$$Z\text{-base-10}: \quad \{3, 8\} \bullet \{5, 9\} = \{2, 2, 4, 2\}$$

$$\text{Polys}(z, Z): \quad \{3, 8\} \bullet \{5, 9\} = \{15, 67, 72\}$$

Well, the reader might say, it is pretty obvious that these two rings are different. For example, we cannot represent 15 or 67 as a single base-10 digit. Also, we would be in trouble if we tried to deal with a polynomial with a negative coefficient:

$$(3z^2 - 7z + 2) = ? = 3[-7]2$$

The thing on the right does not look much like a base-10 integer. This leads to a refinement of our previous question:

Question 2: Can we make a correspondence between multiplying polynomials defined over the field $Z_{10} = \text{Mod}(10)$ and multiplying base-10 integers?

Now consider the same product of polynomials above, but coefficients are in Z_{10} :

$$(3z+8)(5z+9) = (15z^2 + 67z + 72) = (5z^2 + 7z + 2)$$

Now we have eliminated the abovementioned objections, namely, we no longer have negative coefficients, nor do we have coefficients that are larger than one digit. However, we still have no correspondence between the above polynomial product and this integer product $38 \bullet 59 = 2242$. So we now have:

$$\text{Z-base-10:} \quad \{3,8\} \bullet \{5,9\} = \{2,2,4,2\}$$

$$\text{Polys}(z, Z_{10}): \quad \{3,8\} \bullet \{5,9\} = \{5,7,2\}$$

Why are these rings not the same? At the point $(15z^2 + 67z + 72)$ we at least stood a chance, but then when we map each coefficient into Z_{10} to get $(5z^2 + 7z + 2)$, we throw out information. There is no way to connect this polynomial with the integer 2242. This leads to the following observation:

Fact: Doing polynomial multiplication in the ring $\text{Polys}(z, Z_{10})$ is the same as multiplying the corresponding base-10 integers *if you ignore all carries*.

For example,

$$\begin{array}{r} 59 \\ 38 \\ \hline 02 \\ 57 \\ \hline 572 \end{array}$$

This is exactly what you are doing when you multiply $(3z+8)(5z+9)$ with coefficients in Z_{10} . So we can now generalize the above to say:

Fact: Multiplying polynomials with coefficients in the ring Z_n corresponds to multiplying integers base- n provided that all carries are thrown out.

Corollary: Multiplying polynomials with coefficients in the field $\text{GF}(2) = Z_2$ corresponds to multiplying binary numbers provided all carries are thrown out.

We are now ready for,

Question 3: Is there some corresponding statement we can make about *dividing* polynomials?

Our first observation is one that we probably should have made earlier. Looking back at our long hand division of polynomials in Figure 5, or at the circuits of Figure 1 or 2, we realize that the coefficient h_k of $H(z)$ must have an inverse, or we are dead in the water. For example, the very first quotient symbol is $(i_0/h_k) = i_0 \cdot (h_k)^{-1}$. This leaves us with two options if we want to be thinking about integers:

(1) Make sure polynomial coefficients are in a *field*, not just a ring. In a field, all elements have inverses.

(2) Restrict to $H(z)$ which have $h_k = 1$. Such polynomials are called *monic*.

Now, assuming we do (1) or (2), we move to the analog for division of Question 2 above, namely:

Question 4: Can we make a correspondence between *dividing* polynomials defined over the field $Z_{10} = \text{Mod}(10)$ and dividing base-10 integers?

Here is an example:

$$(5z^2 + 3z + 1) / (z+2) = (5z + 3) + 5/(z+2)$$

where we give both the quotient and remainder polynomials. Note that coefficients are not in Z , they are restricted to Z_{10} . Now consider the division of two corresponding base-10 integers:

$$531/12 = 44.25 = 44 + 25/100$$

Is it hopefully clear that there is not much connection between $(5z+3)$ and "44", and (5) and ".25". To make this very clear we write:

$$531/12 = 44.25 \quad / \quad (5z^2 + 3z + 1) / (z+2) = (4z + 4) + (2z+5)/(z+2) \\ = (4z + 6) + 1/(z+2)$$

This is no doubt quite obvious to the reader. However, the next fact may be less obvious:

Fact: Doing polynomial division in the ring $\text{Polys}(z, Z_{10})$ is the same as dividing the corresponding base-10 integers *if you ignore all carries and borrows*.

First, we review the above example:

$$(5z^2 + 3z + 1) / (z+2) = (5z + 3) + 5/(z+2)$$

Now we do it with no-carry, no-borrow long division:

$$\begin{array}{r} 12 \overline{) 531} \\ \underline{50} \\ 31 \\ \underline{36} \end{array} \quad / \quad 5 \cdot 12 = 50 \text{ if no carry}$$

$$5 \quad / 31-36 = 5 \text{ if no borrow } [1 - 6 = 1 + (-6) = 1 + 4 = 5]$$

With multiplication of polynomials in $\text{Polys}(z, Z_{10})$ we never encountered borrows because there were never any subtractions. Multiplication consists of only additions. In contrast, the process of long division involves both multiplications (involving possible carries) and subtractions (involving possible carries).

We now generalize,

Fact: Doing polynomial division in the ring $\text{Polys}(z, Z_n)$ is the same as dividing the corresponding base- n integers *if you ignore all carries and borrows*.

Corollary: Doing polynomial division in the ring $\text{Polys}[z, \text{GF}(2)]$ is the same as dividing the corresponding binary numbers *if you ignore all carries and borrows*.

Summary: When dealing with multiplication and division of integers, there is an interaction between the digit positions. This interaction is known as borrow and carry. For example, when the coefficient in a base-10 digit becomes "too large", part of the information is transferred to the next digit on the left via a "carry". One can think of an integer base- z as a polynomial in powers of z , but there is always this interaction implied by the arithmetic rules $+$, $-$, $\bullet$, $/$.

In contrast, in the multiplication and division of polynomials in z , everything is carefully aligned with powers of the variable z . Information is *never* transferred between two unequal powers.

We close this section with one final question:

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

Back in Section 1.3 we made this claim, and appealed to an analogy with integer division by observing that

$$514.000000... / 37 = 13.891891891891...$$

gives a remainder which never terminates. The analogy is that dividing polynomials is "like" dividing integers, and we might compare a non-terminating remainder polynomial with this situation involving integer division:

$514/37 = 13$	Remainder = 33
$5140/37 = 138$	Remainder = 34
$51400/37 = 1389$	Remainder = 7
$514000/37 = 13891$	Remainder = 33
$5140000/37 = 138918$	Remainder = 34
$51400000/37 = 1389189$	Remainder = 7
$514000000/37 = 13891891$	Remainder = 33

In retrospect, we must now admit that dividing integers is really quite different from dividing polynomials, so our analogy is not very convincing.

A true answer to Question 5 will have to await the next chapter.