Old Section 1.5 and similar for mult PhL 6.17.13

I am just storing older work here, not really needed. My newer stuff is MUCH more detailed.

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. We already know from the above discussion that everything works properly, but here we try to understand exactly how each circuit does the division process in the time domain.

Figure 3 shows Long Division

I will later scan in Fig 3 and other Fig's as needed.

In Figure 3 we show the long division of a divisor polynomial H(z) into a dividend polynomial of the form $I_r(z)$ as shown in (1.3.8),

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

where we are dealing with the three proper polynomials (assuming $r \geq k$)

$$I_r(z) = i_0 z^r + i_1 z^{r-1} + i_2 z^{r-2} + \ldots\ldots + i_r \qquad (1.3.4)$$

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

$$O_{r-k}(z) = o_k z^{r-k} + o_{k+1} z^{r-k-1} + \ldots + o_{k+r} \qquad (1.3.5)$$

Sometimes one forgets how this algorithm works. At each level of the division process there is a "current dividend". The first such dividend is $I_r(z)$ which appears under the bar in Fig 3. We find the first term 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 k of H(z). We then stop and declare that last current dividend to be the "remainder". Note that we have made use of $\alpha_j \equiv (h_j / h_k)$ as a shorthand in Fig 3.

Although each current dividend may be very long, 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 2 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, which we repeat here:

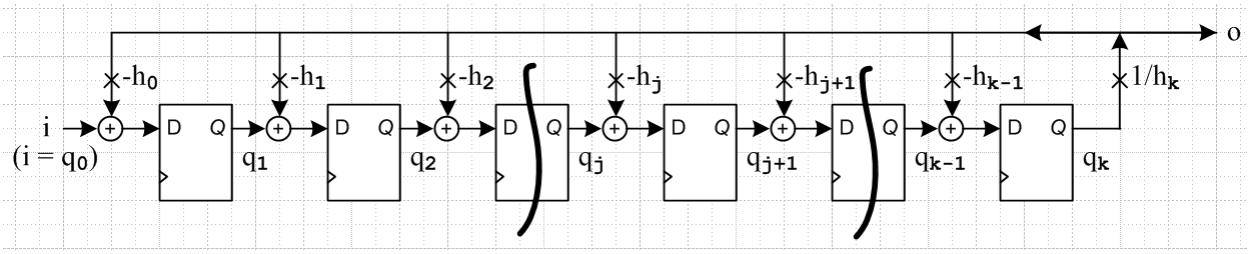


Figure 2: The standard polynomial divider circuit.

The left column of Fig 4 shows the count of clocks, while the second column shows the output o. The next k columns show the contents of the k registers, but in reverse order relative to Fig 2. The final column shows the input stream i_n.

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 (starting with clock k, we show the powers of z). 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. Details of the quantities like i'_j are then shown below the table in Fig 4.

For example, look at the item $i'_1 z^{r-1}$ in the q_k column at clock k+1. 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 in Fig 2, 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.

When the current dividend has degree ≤ that of the divisor H(z), we stop and we interpret that last current dividend as the remainder. So the registers end up holding the remainder after the quotient is shifted out.

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, ... h_{k-1}} times o 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) times the current output o from this vector to get the new vector current dividend. From this vantage point, we see that Figure 2 is a very standardized form of the divider, because we can clearly see how each successive subtraction occurs.

Application of Figure 2 to CRC .

In this little section the degree of polynomial H(z) is K ≡ n-k instead of k. This K is then the number of registers in a Figure 2 polynomial divider set up to divide a polynomial by H(z). The symbols n and k have the following new meaning: a set of k data symbols is used to construct a proper polynomial d(z). A set of n code symbols (known as a block) is transmitted where n > k. The block contains the k data symbols and n-k "parity check symbols" which are used to allow detection of errors. At the transmitting end, a Figure 2 polynomial divider is used to divide the polynomial $[z^{n-k}d(z)]$ by the polynomial H(z) of degree K = n-k. In this context, H(z) is referred to as a "generator" polynomial. This division can be thought of this way

$$z^{n-k}d(z) = D(z)H(z) - \gamma(z)$$

where D(z) is the division quotient and -γ(z) is the division remainder (which has degree ≤ n-k). The quotient D(z) has degree k, the same as that of d(z). A "code word polynomial" (degree n) is given by

$$C(z) \equiv D(z)H(z) = \gamma(z) + z^{n-k}d(z)$$

where we just use the previous line to get the right side. If we write out the coefficients of C(z) (from low to high power), we find that the first n-k of them are the coefficients of γ(z), and the last k of them are the data symbols of d(z). For example, if n=5 and k=3,

$$C(z) = (\gamma_0+\gamma_1 z) + z^2(d_0+d_1 z+d_2 z^2) = \gamma_0+\gamma_1 z+d_0 z^2+d_1 z^3+d_2 z^4 \rightarrow (\gamma_0, \gamma_1, d_0, d_1, d_2)$$
$$= (c_4, c_3, c_2, c_1, c_0).$$

Thus in this scheme, the K = n-k parity check symbols γ_i are the negative of the remainder coefficients obtained by doing the division $[z^{n-k}d(z)]/H(z)$. This remainder is obtained using a Figure 2 divider with K = n-k registers. When the proper polynomial division has completed, the registers contain the remainder -γ(z). By grounding the feedback line in Figure 2 and directly accessing the output of the rightmost register, the next n-k clocks shift out -γ(z) and the coefficients are then negated and appended to the data stream as shown in the example above to create code polynomial C(z). The data stream of coefficients c_i, which we think of as the polynomial C(z), is then transmitted (symbol d_2 first in our example). At the receiving end, the received C(z) is divided by H(z) by another Figure 2 divider. Since C(z) ≡ D(z)H(z), there should be no remainder! If there was an error in the transmission so that some erroneous code word C'(z) was received, one gets C'(z) = D(z) H(z) + s(z) where s(z) is an anomalous remainder (the syndrome) which is found sitting in the Figure 2 registers after the division. This error

detection scheme is known as a Cyclic Redundancy Check (CRC). Notice that both the transmitting end (encoder) and receiving end (decoder) have Figure 2 dividers which divide by H(z).

The subject is discussed more in Chapter 8 of Ref [Galois]. There, variable z is called x, and H(z) is called g(x).

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

As a reminder, here is the scrambler style divider,

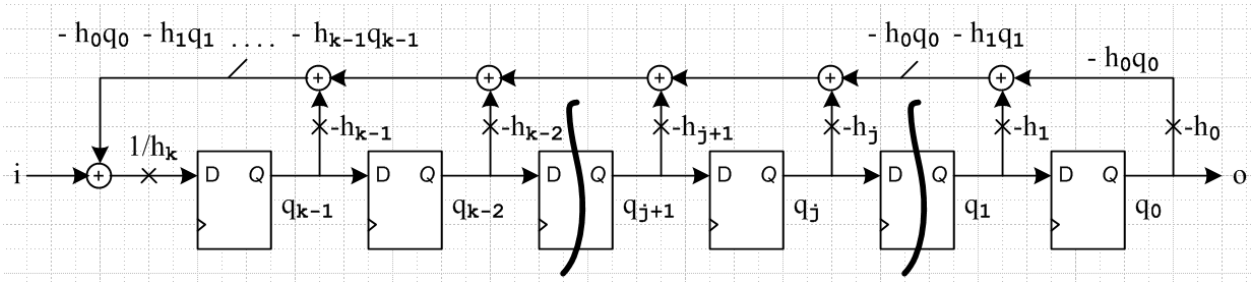


Figure 1: A polynomial divider circuit.

In Figure 6 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 6 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 $1/h_k$ 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.

So imagine that all registers start with 0 and the first input symbol i_0 is clocked into q_{k-1}. Only the leftmost adder then does anything interesting (since all other adder inputs are 0) and it adds 0 to i_0 and this is represented by the leftmost column in Fig 6. On the next clock, i_0 moves into q_{k-2} and now the left two adders become active, and they form the sum shown in the second column of Fig 6. On the next clock, three adders become active, and so on. So the column sums shown in Fig 6 indicate what the adders in Fig 1 are adding on each clock.

The items in Figure 6 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 height never exceeds k lines. 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. Again, these column sums are exactly what all those adders in Figure 1 are computing!

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

****

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

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

In Figure 9 we show the multiplication of H(z) by $I_r(z)$ written out long hand using the elementary school add and shift method. Perversely, we have put the highest power on the right for each of the polynomials to be multiplied together. This has no effect on the algorithm.

In the corresponding division circuit of Figure 1 we did subtractions. Here, for multiplications, as everyone knows, we are doing a set of additions, so there are no longer any minus signs. The circuit of Figure 7 is simply doing the column sums shown in Figure 9. For example, when the first non-zero input bit i_0 is sitting at Fig 7 input, the set of adders obtains $o_0 = i_0 h_k$ (all other registers are 0), and this is the polynomial coefficient seen in the rightmost column in Fig 9. One clock later, i_0 moves to the q_{k-1} position in Fig 7 and i_1 is at the input. In this case the output will be $i_1 h_k + i_0 h_{k-1}$ and this matches the second column from the right in Fig 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 according to the traditional add and shift algorithm. 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 8 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.