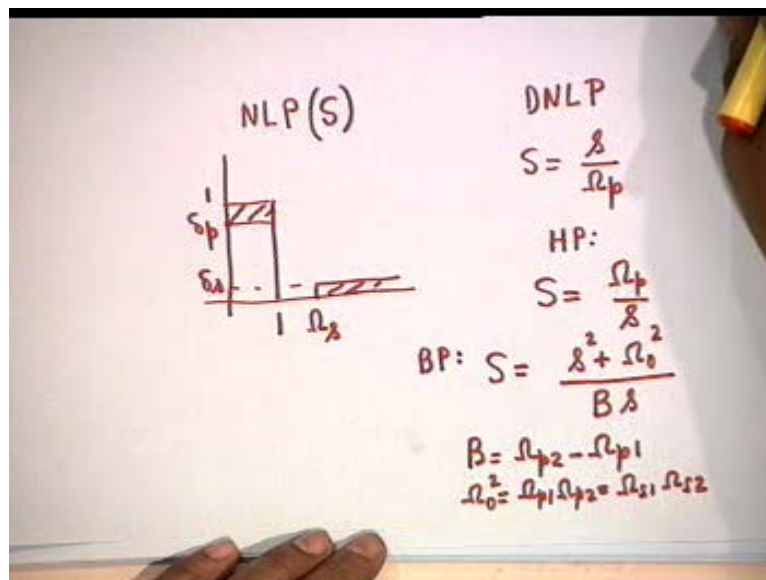


Digital Signal Processing
Prof: S. C. Dutta Roy
Department of Electrical Engineering
Indian Institute of Technology, Delhi
Lecture - 28
Digital Filter Structures

This is the 28th lecture on DSP and the topic for today's lecture is digital filter structures. The last lecture, No. 27, was a problem solving session and before that we had discussed analog frequency transformation where the problem was to transform a normalized low pass filter with normalization like what is shown in the Figure.

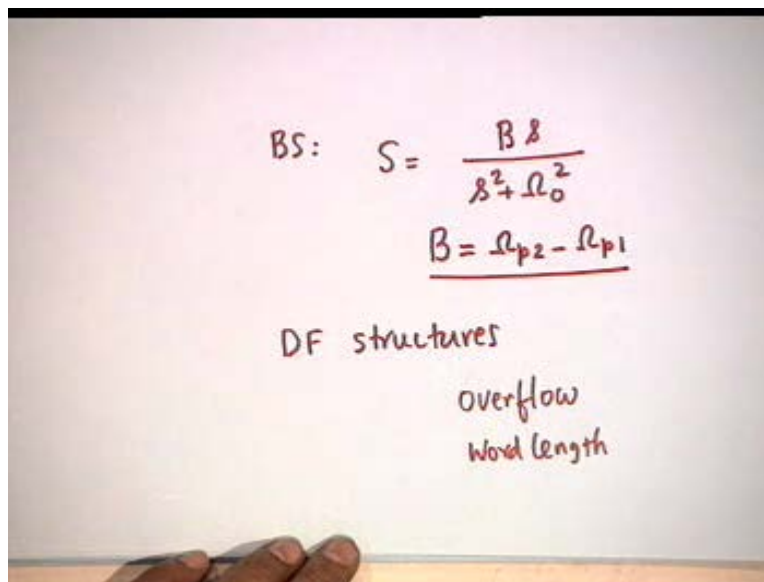
(Refer Slide Time: 01.32 – 03.33)



The tolerance in the passband was taken as $1 - \delta_p$ and the tolerance in the stopband was δ_s with stopband edge at Ω_s . This is to be transformed to de-normalized low pass. If the complex frequency variable in normalized low pass is S then S is to be replaced by s/Ω_p where s is the complex frequency variable in the de-normalized low pass with passband from 0 to Ω_p . In de-normalized high pass, the transformation is simply reciprocal of this, i.e. $S = \Omega_p/s$. In band-pass,

the transformation is $S = (s^2 + \Omega_0^2)/(Bs)$ where B is $\Omega_{p2} - \Omega_{p1}$ and $\Omega_0^2 = \Omega_{p1} \Omega_{p2} = \Omega_{s1} \Omega_{s2}$. If this equality does not happen with the given specs, then you shall have to alter Ω_{s1} or Ω_{s2} to satisfy this relation. You can alter both but that is risky because if you under satisfy one of these stop bands then the design is not acceptable. Your pass band is sacred; you have to satisfy this exactly. Stop band can be taken liberty of only in one direction; you can over satisfy, not under satisfy.

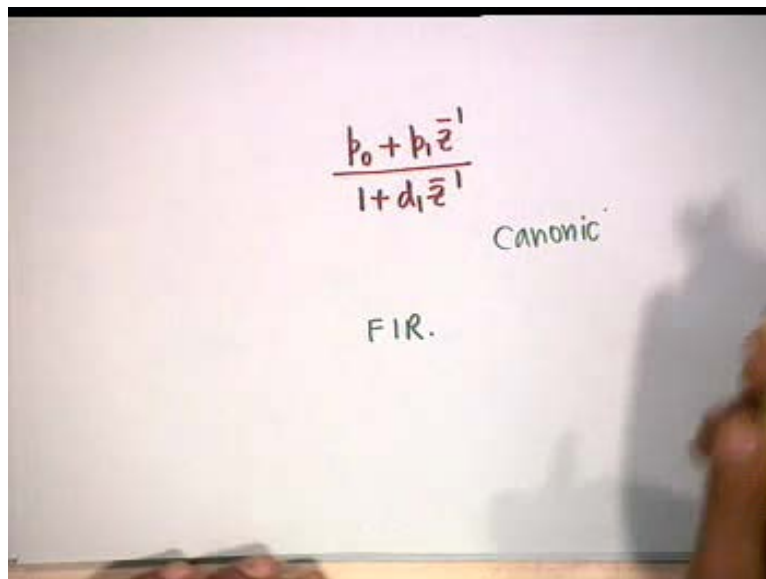
(Refer Slide Time: 03.52 – 06.28)



In a similar manner, if it is a band stop filter then $S = Bs/(s^2 + \Omega_0^2)$ but here B has no physical significance. It was bandwidth in the band-pass case, but not here. $\Omega_{p2} - \Omega_{p1}$ is not the bandwidth. The pass band here extends from 0 to Ω_{p1} on the lower side and Ω_{p2} to infinity on the higher side. The stop band width is $\Omega_{s2} - \Omega_{s1}$ where these frequencies are determined by the tolerance in the stop band. This is one of the important points to remember. Then we started discussion on digital filter structures and we considered the problem of analysis, which is very simple. One can write the equations and then eliminate the intermediate variables to get the overall transfer function $Y(z)/X(z)$. On the other hand, the synthesis problem is tougher than the analysis problem. And one of the characteristics of the synthesis problem is that, if there exists one solution, then there shall exist an indefinite number of solutions. One can argue: if I get one

solution, why do not I stick to it? Why do I go for alternative solutions? It turns out that different structures have different properties and one of the things that we have to worry about is overflow. That is, if at any stage in the filter, there is saturation then there shall be overflow and the filter shall cease to perform in the manner that you like it to perform. And the other problem is word length effects, which arises because of finite number of bits that you have to use: 8 or 16 or 32, and therefore arises the necessity that the coefficients as well as the signals have to be truncated. For example, an 8-bit number multiplied by another 8-bit number gives you a 16 bit number. And if your hardware is 8-bit, then you shall have to truncate to 8 bits. Different structures have different properties with regard to word length effects. One should choose the one which has the lowest quantization error and this is the reason why a multiplicity of structures has to be worked out and then you choose an optimum or near optimum structure.

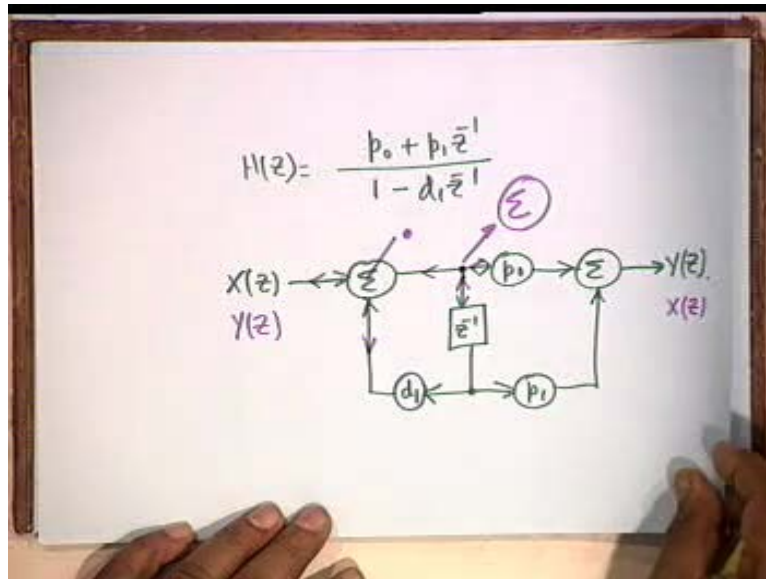
(Refer Slide Time: 07.14 – 08.26)



We also showed how to derive two structures for the transfer function $(p_0 + p_1 z^{-1}) / (1 + d_1 z^{-1})$ and we showed that one can make this structure canonic with respect to multipliers and delays. We shall expand on this today to see how a general IIR transfer function can be realized by a canonic structure, canonic in both delays and multipliers. But before that, let us introduce the topic of

transposition. This is an important topic which allows one structure to be converted to another. Transposition can be explained with reference to this particular transfer function.

(Refer Slide Time: 08.33 – 11.48)

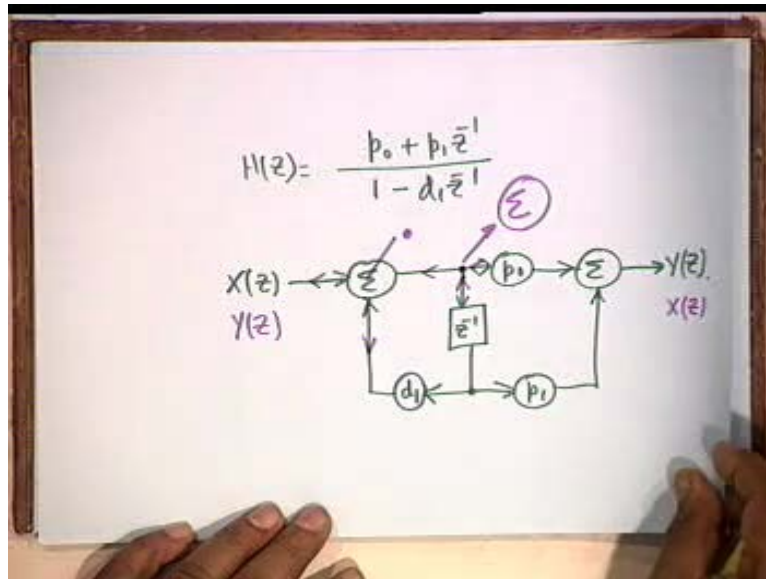


Let us change the transfer function a little bit. Let $H(z) = (p_0 + p_1 z^{-1}) (1 - d_1 z^{-1})$. Then you know that the structure that we get is as shown in the Figure where the multiplier in the feedback path is d_1 now, not $-d_1$. And then we have two multipliers p_0 and p_1 , the outputs of which add together to give you $Y(z)$. This is a canonic structure, canonic in multipliers, because only three multipliers, p_0 , p_1 and d_1 , are needed and also canonic in delay of because it is a first order filter and there is only one delay.

Now, in this structure, let us make the following changes. First, we interchange the input and output. Second, reverse all the arrows. I will draw the total structure later. But as soon as you reverse, the Σ node now becomes a pick up node. A signal comes and goes in two directions. All Σ nodes are replaced by pick up nodes and conversely, all pick up nodes are replaced by Σ nodes. This is the third step. If you do these, then you have transposed the structure. So the transposition involves three steps. First, interchange the input and output. Reverse all the arrows in the second step. In the third step, replace each Σ by a pick up point and each pick up point by

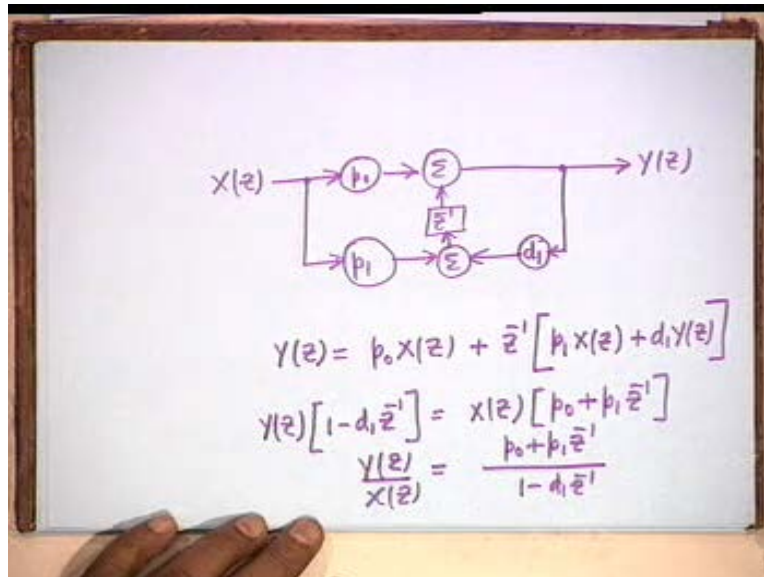
Σ . The structure that I get is something like this where the original structure and the transposed structure are superimposed. If the original structure is removed and we redraw the structure in the conventional manner, i.e. input at the left and output on the right, then

(Refer Slide Time: 11.56 – 13.28)



the structure shall look like the one shown in the next figure.

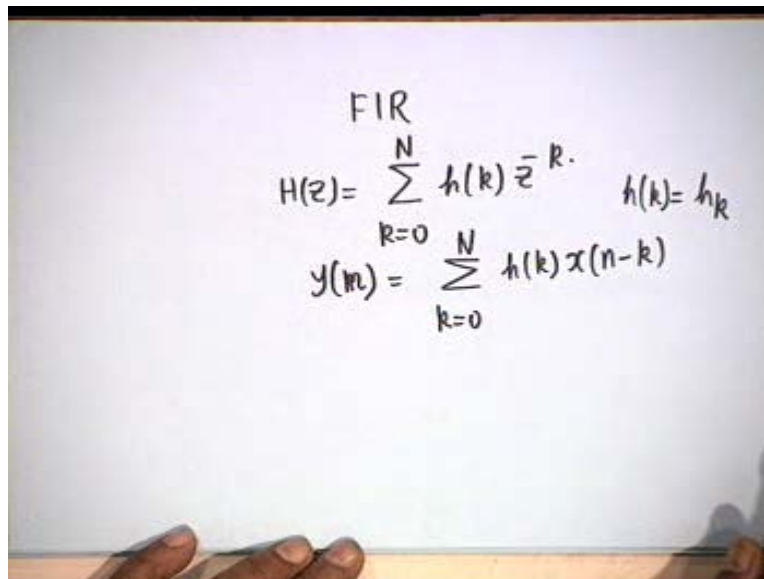
(Refer Slide Time: 13.29 - 15.33)



Now, if I find the transfer function, it is easy to show that this structure gives the same transfer function. This is in general true. Given any digital filter structure, you can always make a transposition to get an alternative structure.

The properties of overflow and word length effects are different for these two structures. Ideally, if we had infinite bit arithmetic and infinite storage capacity, then the structures would have been the same in theory. But in practical implementation, they have different properties and therefore it is worthwhile to determine the transposed structures. We shall give one or two more illustrations of transposed structures but before that let us study systematically how first FIR filters are realized and then we shall go to IIR filters.

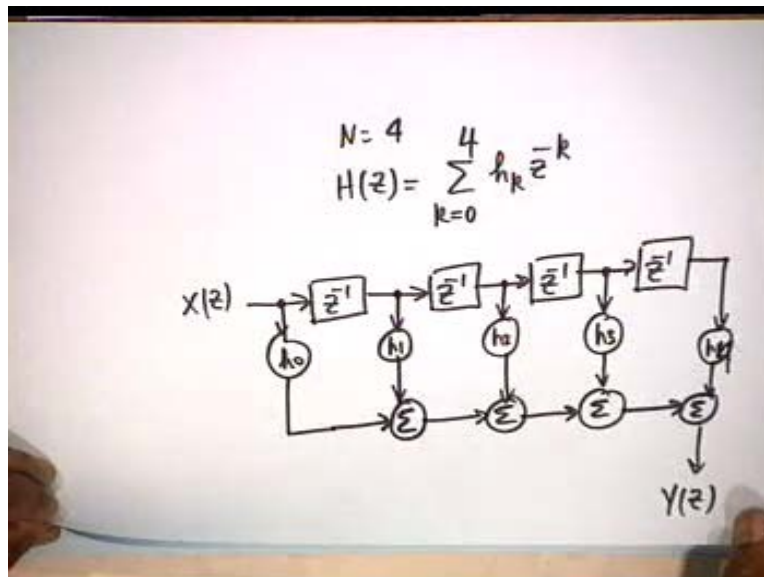
(Refer Slide Time: 16.19 – 17.56)



The image shows a whiteboard with handwritten mathematical formulas for an FIR filter. At the top, the word "FIR" is written. Below it, the transfer function is given as $H(z) = \sum_{k=0}^N h(k) z^{-k}$. To the right of this equation, it is noted that $h(k) = h_k$. Below the transfer function, the difference equation is written as $y(n) = \sum_{k=0}^N h(k) x(n-k)$.

For FIR, as you know, the transfer function is of the form summation $h(k)z^{-k}$, $k = 0$ to N . I shall use N for filter order sometimes, and length some other time; so you have to keep track of what it stands for. Here, obviously, the length is $N + 1$. I shall now derive a structure which uses these coefficients $h(k)$; incidentally, in order not to write these brackets again and again, I shall use a subscripted h_k to show that this h is a function of k . For simplicity, we shall do it for a specific example $N = 4$, i.e. a 4th order filter. You can easily generalize.

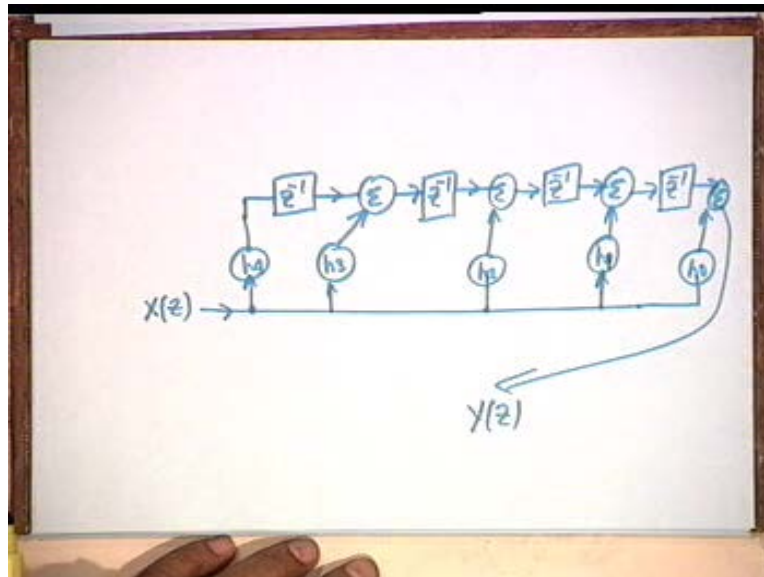
(Refer Slide Time: 18.02 - 19.49)



My transfer function is $H(z) = \sum_{k=0}^4 h_k z^{-k}$. A very simple minded realization can be obtained as follows. First, you draw all the four delays z^{-1} , z^{-1} , z^{-1} and z^{-1} in a chain, and put $X(z)$ at the left, and multiply the leftmost signal i.e. the input signal by h_0 ; then after one delay multiply the signal by h_1 , add the two signals. All summers shall have only 2 inputs and we obey that discipline.

The signal at the end of two delays is multiplied by h_2 and added to the previous result. Similarly, derive $h_3 z^{-3}$ and $h_4 z^{-4}$ and go on adding to get $Y(z)$. This is called a direct structure because the multipliers are obtained directly from the transfer function; no manipulation is needed. Now, if you make a transposition of this and then a flipping over, then obviously, you should get what is shown in the next figure.

(Refer Slide Time: 19.54 - 22.22)



Both of them are valid structures in their own right, but in practical implementation, the properties are different. They are also sometimes known in literature as Transversal structures. Current authors also call one of them as direct form one and the other as direct form two. These nomenclatures are not important for us as long as you can obtain one structure and its equivalent transposed structure. There are indirect structures also for FIR filters.

(Refer Slide Time: 22.31 - 25.45)

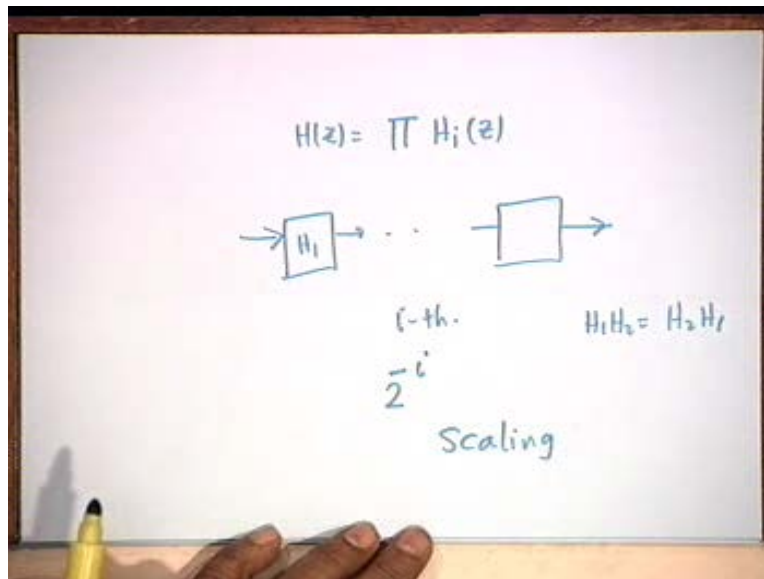
Cascade

$$H(z) = h_0 (1 + \beta_0 z^{-1}) \prod_{k=1}^{N/2} (1 + \beta_{1k} z^{-1} + \beta_{2k} z^{-2}) \quad N \text{ odd}$$

$$H(z) = \prod_{k=1}^{N/2} () \quad N \text{ even}$$

For example, if $H(z)$ is an odd order transfer function, then I can factorize this into one first order and a number of second order factors. Take h_0 common; then you can write the first order factor as $(1 + \beta_0 z^{-1})$. If N is 5, then I shall have two second order factors each of the form $1 + \beta_{1k} z^{-1} + \beta_{2k} z^{-2}$, $k = 1, 2$. Why can't I write $H(z)$ as product of only first order factors? What would be the disadvantage? I may get complex coefficients because it is not guaranteed that all the 0's are real; if all the 0's are real, then you can do with all first order factors. But complex 0's must be written together so that the coefficients are real because we want to implement the filter in real time with real coefficients. Each of these factors can now be realized in the direct form. And then you have to cascade them together. So this is called a cascade realization. On the other hand, if N is even, then all factors, in general, will be second order. Each of them now has to be realized in the transversal form, and the overall structure will be a cascade of them.

(Refer Slide Time: 25.50 – 29.32)



Obviously if one synthesis solution exists then there exists indefinite number of solutions. In implementation of cascade, you write $H(z) =$ continued product of $H_i(z)$ where $H_i(z)$ is no more complicated than a quadratic. It is either a linear factor or a quadratic one. The order of cascading is important. You can start with H_1 and go up to the last one and then you can make permutations. Ideally, in cascading the order of cascading is not important as far as the total transfer function is concerned because $H_1 H_2$ is the same as $H_2 H_1$.

Multiplication is commutative in theory but in practice, the order of cascading is important from the point of view of overflow and word length effects. So your job after finding a cascade realization is not complete. If you want to implement, you must study the quantization and the overflow behaviors of the structure. Normally the section with the lowest gain is put at the beginning so that its signal output does not saturate the next sections. Now, it may not be possible to avoid overflow particularly in fixed point arithmetic. You know the dynamic range of fixed point arithmetic is severely limited as compared to floating point. If you are keen on fixed point, then one may have to scale the transfer function, to avoid overflow.

In other words, suppose the i th stage the signal that you get has exceeded the dynamic range of storage in the accumulator. Then at $(i - 1)$ th stage you use a scaling constant and it is usually of the form 2^{-k} . You can divide by half; this is not a multiplier, it is only a shift. Then the overall transfer function is realized within a multiplying constant. So scaling is also an important consideration, you may have to do it. One might ask why not do the scaling at the input or the output? Well, if it is permissible you may do it. But at the input doing the scaling is prone to another disastrous effect.

Can you tell me what this effect is? If the signal becomes very weak, then noise may take over, and therefore your scaling has also to be done in a distributed manner, judiciously, so that you maintain a balance between overflow, signal strength and also word length effect, that is quantization errors. In digital implementation, we are not worried about the actual value of the number because the main function of a digital signal processor is filtering. If you get rid of the undesired frequencies, then digital amplification is no problem; it is simply multiplying by a power of 2. It is a shift in the proper direction.

(Refer Slide Time: 30:36 -32:46)

$$\begin{aligned} &\text{Parallel} \\ H(z) &= \sum_{k=0}^7 h_k z^{-k} \\ &= \sum H_i(z) \end{aligned}$$

Interpolation / Decimation

What about parallel implementation? Is it possible to do a parallel implementation of an FIR filter? Parallel is usually resorted to when you wish to speed up the process. Parallel processing is faster than serial processing. In the FIR case, however that does not happen because if you have $H(z)$ equal to summation $h_k z^{-k}$, $k = 0$ to 7 , then 7 delays are needed in whatever way you decompose. Parallel means you write $H(z)$ as $\sum H_i(z)$; so there shall be at least one H_i which shall contain seven delays and therefore the processing time cannot be reduced by parallel implementation in FIR case. This is not true in IIR, which we shall see later. However, parallel implementation is useful in the context of multirate signal processing by interpolation and decimation. Parallel decomposition helps there and the decomposition is done in a particular manner, that is called polyphase decomposition. We shall briefly touch upon this topic because it has now become an integral part of any programmable DSP. Interpolation and decimation are resorted to for various reasons but one of the main reasons is that it reduces the hardware complexity of the structure. We shall illustrate this with reference to a particular transfer function of order eight.

(Refer Slide Time: 32.58- 35.40)

$$\begin{aligned}
 H(z) &= h_0 + h_1 z^{-1} + \dots + h_8 z^{-8} \\
 &= (h_0 + h_2 z^{-2} + \dots + h_8 z^{-8}) \\
 &\quad + (h_1 z^{-1} + h_3 z^{-3} + \dots + h_7 z^{-7}) \\
 &= E_0(z^2) + z^{-1} E_1(z^2)
 \end{aligned}$$

Let us say we have $H(z) = h_0 + h_1 z^{-1} + \text{etc} + h_8 z^{-8}$. We can decompose this transfer function in various ways. We can write this as $(h_0 + h_2 z^{-2} + h_4 z^{-4} + h_6 z^{-6} + h_8 z^{-8})$ plus $(h_1 z^{-1} + h_3 z^{-3} + h_5 z^{-5} +$

$h_7z^{-7} = E_0(z^2) + z^{-1} E_1(z^2)$. Obviously, the implementation would be that of $E_0(z^2)$ and $E_1(z^2)$ delayed by one sample, and then added to give you the output. This, by itself, does not increase the speed. However, if I have a factor of two interpolation, then in the z transform, z is squared. And therefore if I decimate each component by a factor of two, I can realize $E_0(z)$ and $E_1(z)$. Therefore the speed increases. Read the chapter of Multirate Signal Processing in Mitra to find out how decimation and interpolation help.

There is nothing sacred about breaking it up into two transfer functions. Technically, what we did is called two phase decomposition. I can also make a three phase decomposition.

(Refer Slide Time: 35.43 – 38.00)

$$H(z) = E_0(z^3) + z^{-1}E_1(z^3) + z^{-2}E_2(z^3)$$

$$E_0(z^3) = h_0 + h_3z^{-3} + h_6z^{-6}$$

$$E_1(z^3) = h_1 + h_4z^{-3} + h_7z^{-6}$$

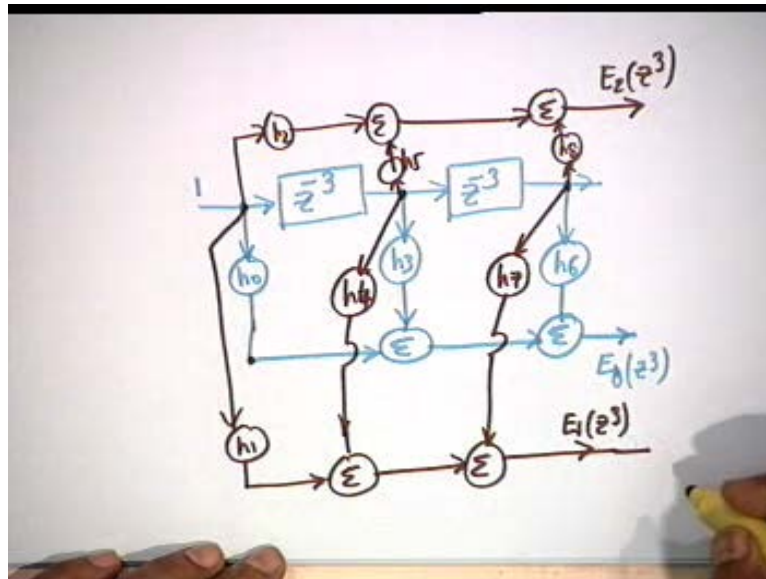
$$E_2(z^3) = h_2 + h_5z^{-3} + h_8z^{-6}$$

3-φ

I can write $H(z) = E_0(z^3) + z^{-1}E_1(z^3) + z^{-2}E_2(z^3)$ where E_0 shall be now $h_0 + h_3z^{-3} + h_6z^{-6}$, E_1 is $h_1 + h_4z^{-3} + h_7z^{-6}$ and E_2 would be $h_2 + h_5z^{-3} + h_8z^{-6}$ and the realization is obvious. We have three parts and we require two extra delays. But the total number of delays you are using is still eight. You can also appreciate that the total number of delays required in the implementation of polyphase decomposition is much greater than in direct realization or cascade realization. However, the number of delays can still be kept at eight for this structure by sharing delays. And I shall

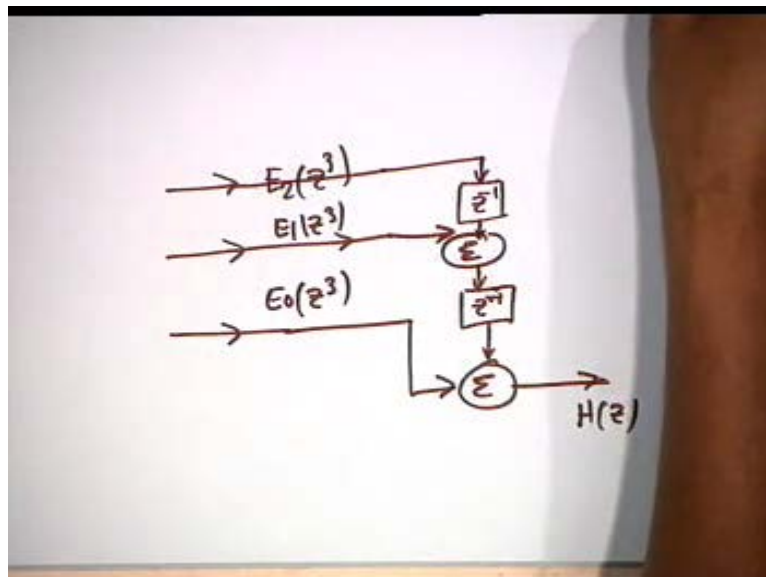
illustrate this with reference to this particular decomposition. This is a three phase decomposition.

(Refer Slide Time: 38.18 – 41.14)



We shall have two blocks, each having three delays. From here I first combine signals to obtain E_0 , E_1 and E_2 . For example, for E_0 I have taken h_0 multiplier from the input; then I have generated h_3z^{-3} and h_6z^{-6} . Then I combine these by two summers, to get the transfer function $E_0(z^3)$. Similarly, we construct $E_1(z^3)$ and $E_2(z^3)$ and combine the three E 's, as shown in the figure. This is a trick that one can employ to make the structure canonic.

(Refer Slide Time: 41.16- 42.35)



(Refer Slide Time: 42.48 – 44.58)

$$H(z) = h_0 + h_1 z^{-1} + \dots + h_{M-1} z^{M-1}$$

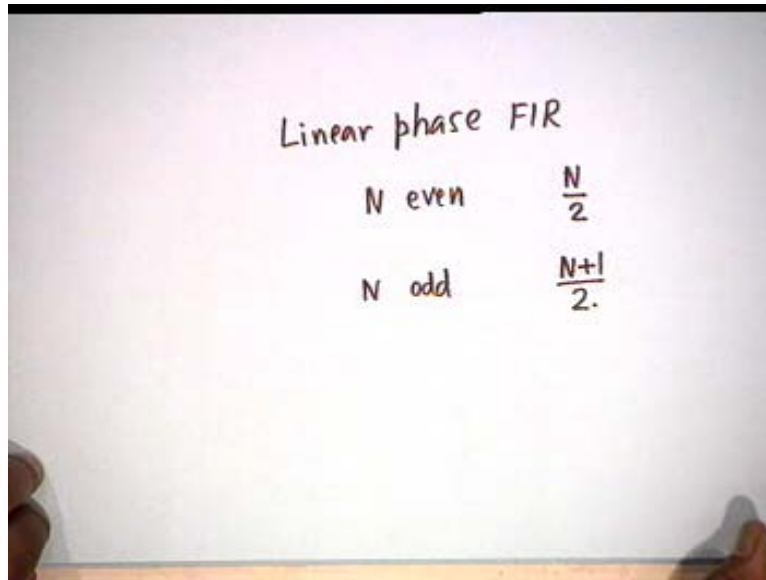
L- ϕ decomp.

$$H(z) = \sum_{m=0}^{[M/L]} z^{-m} E_m(z^L).$$
$$h(n) = 0, \quad n \geq M.$$

In general if your transfer function is $H(z) = h_0 + h_1 z^{-1} + \text{etc} + h_{M-1} z^{M-1}$ (the length here is M). And you make L phase decomposition, then your transfer function $H(z)$ shall be of the form

summation $\sum z^{-m} E_m(z^L)$, where m will go from 0 to integer part of M/L , usually. There is denoted as $\lfloor M/L \rfloor$. This is the general L phase decomposition of a length M FIR filter.

(Refer Slide Time: 45.04 - 46.20)



Now obviously if the FIR was linear phase then the number of multipliers can be reduced approximately by half. If the length N is even then because of symmetry or antisymmetry, the number of multipliers can be reduced to $N/2$. If N is odd you require $N + 1$ divided by two numbers of multipliers.

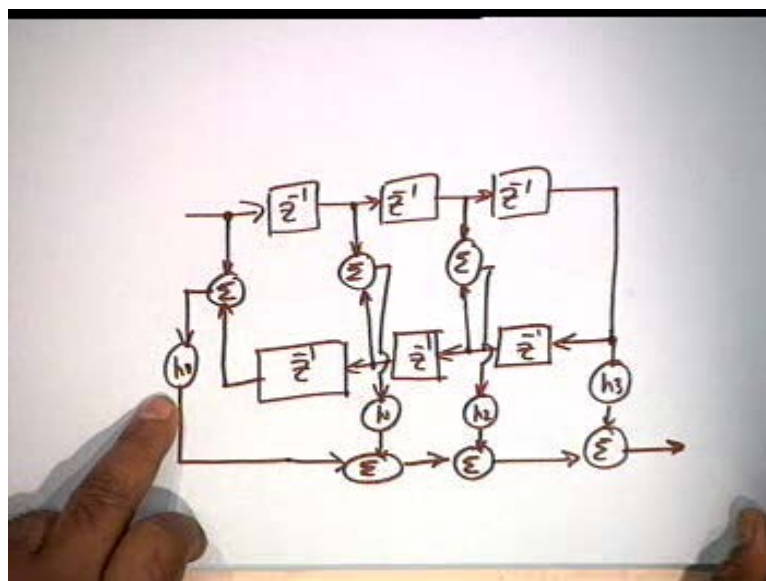
(Refer Slide Time: 46.25 – 47.53)

$$N=7 \text{ i.e. } H(z) = h_0 + h_1 z^{-1} + \dots + h_6 z^{-6}$$

$$H(z) = h_0 + h_1 z^{-1} + h_2 z^{-2} + h_3 z^{-3} + h_4 z^{-4} + h_5 z^{-5} + h_6 z^{-6}$$

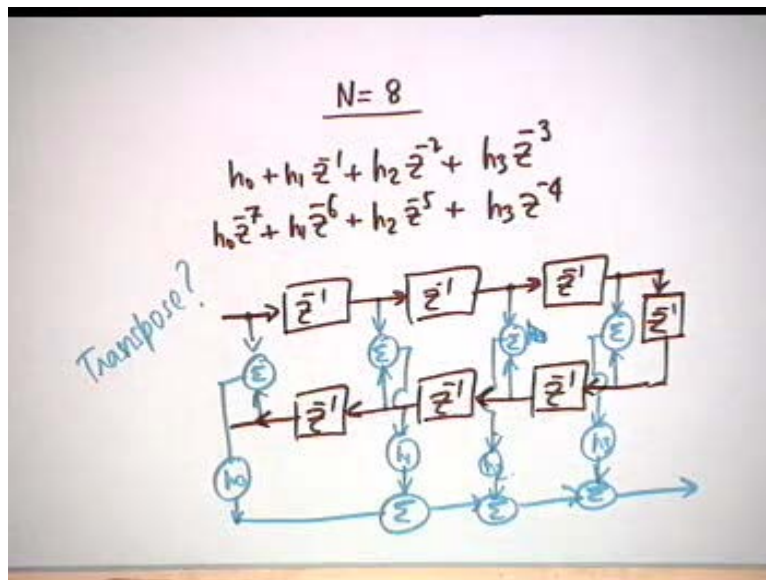
We shall illustrate the structure with reference to two examples; first consider $N = 7$ i.e. $H(z) = h_0 + h_1 z^{-1} + \dots + h_6 z^{-6}$, where $h_0 = h_6$, $h_1 = h_5$, $h_2 = h_4$.

(Refer Slide Time: 47.57 – 50.22)



We first draw all the delays in a chain, three in the upper line, and the other three in the lower line. The undelayed signal is now combined with six delayed signal, and the output is multiplied by h_0 ; you have to add it to other terms. Similarly, $X(z)z^{-1}$ is to be combined with $X(z)z^{-5}$, and then, multiplied by h_1 and combined with the previous signal. Similarly, $X(z)z^{-2}$ shall be combined with $X(z)z^{-4}$ and multiplied by h_2 , and then added to the previous signal. h_3z^{-3} is a loner, and you multiply $X(z)z^{-3}$ by h_3 and add to the output of the previous summer; the total number of multipliers is $(7 + 1)/2 = 4$; it is canonic in delays. This is also an advantage of linear phase filters; not only they give constant group delay, but the hardware requirement is also much less.

(Refer Slide Time: 50.27 - 53.13)



On the other hand, if $N = 8$, $H(z) = (h_0 + h_1z^{-1} + h_2z^{-2} + h_3z^{-3}) + (h_0z^{-7} + h_1z^{-6} + h_2z^{-5} + h_3z^{-4})$. We draw the delays in a different fashion. There are seven delays, so we draw three of them in the upper line, three of them in the lower line, and the extra one we draw in the vertical direction. Now we combine signals in the same manner, as shown in the figure. It shows not only the hardware structure, it also shows the software structure, i.e. how to write the program. The diagram will indicate to you the successive steps in the algorithm. What do you think its transpose shall be? Will it be identical to the previous one? No. Try it yourself.