Outline

- Conversion of sampling rate with a non-integer
- Interchanging orders of filtering and decimation and interpolation
- Polyphase implementation of systems
- Efficient implementation for decimation and interpolation

Conversion of sampling rate with a non-integer

- Conversion of sampling rate with a non-integer is possible by performing interpolation first and then decimation

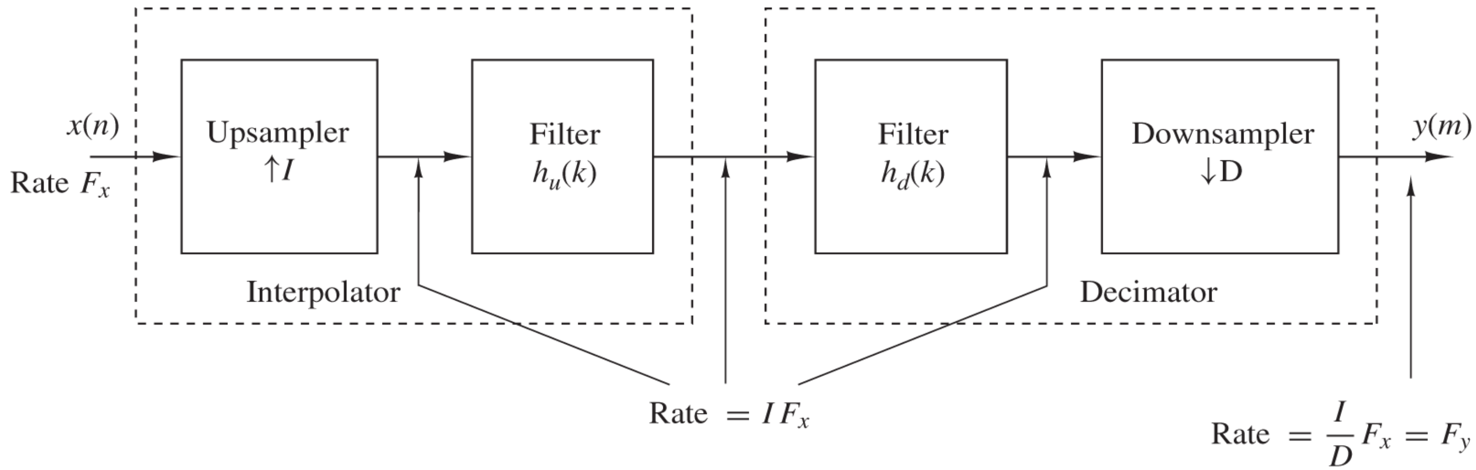


Figure 11.4.1 Method for sampling rate conversion by a factor I/D.

- The two filters in between can be combined to form a upsample-filter-downsample structure for resampling

Conversion of sampling rate with a non-integer

- Mathematically we have

$$w[l] = \sum_{k=-\infty}^{\infty} h[l-k]v[k]$$

which can be written as

$$w[l] = \sum_{k=-\infty}^{\infty} h[l-kI]x[k]$$

since $v[l]$ is simply $x[l]$ with zeros placed in between samples of $x[l]$

- The final output can be written as

$$x_r[l] = w[mD] = \sum_{k=-\infty}^{\infty} h[mD-kI]x[k]$$

Conversion of sampling rate with a non-integer

- Similar to decimation and interpolation, we want to see resampling as a linear system
- To be able to write the relation between $x[l]$ and $x_r[l]$ as a linear system we make a change of variable

$$k = \text{floor}[mD/I] - n$$

resulting in

$$x_r[n] = \sum_{n=-\infty}^{\infty} h[mD - \text{floor}[mD/I]I + nI]x[\text{floor}[mD/I] - n]$$

- Note that $mD - \text{floor}[mD/I]I = (mD)_I$ where $_I$ denotes modulo I
- With these we obtain

$$x_r[n] = \sum_{n=-\infty}^{\infty} h[(mD)_I + nI]x[\text{floor}[mD/I] - n]$$

Conversion of sampling rate with a non-integer

- This relationship in the frequency domain is

$$X_r(w) = \begin{cases} \frac{I}{D} X(w/D) & 0 \leq |w| \leq min(\pi, \pi D/I) \\ 0 & \text{otherwise} \end{cases}$$

- This is obtained simply by cascading the interpolation and decimation operations

Interchanging Orders in Resampling

- We said that sampling rate conversion can be performed by interpolation followed by decimation
- In general, this cannot be interchanged because these operations are tim-variant
- However, we can derive a rule that allows for interchanging
- We will derive rules for both decimation and interpolation

Interchanging the orders of filtering and decimation

- For downsampling (without filtering) we have

$$X_D(z) = \frac{1}{D}\sum_{i=0}^{D-1} X(z^{1/D}W_D^i)$$

- Then if this downsampled signal is passed through a system with $H(z)$

$$Y(z) = \frac{1}{D}H(z)\sum_{i=0}^{D-1} X(z^{1/D}W_D^i)$$

- If the signal is passed through a filter first (call that $H'(z)$ and then downsampled we would have

$$Y(z) = \frac{1}{D}\sum_{i=0}^{D-1} H'(z^{1/D}W_D^i)X(z^{1/D}W_D^i)$$

- For these two to be equivalent $H'(z) = H(z^D)$

Interchanging the orders of filtering and decimation

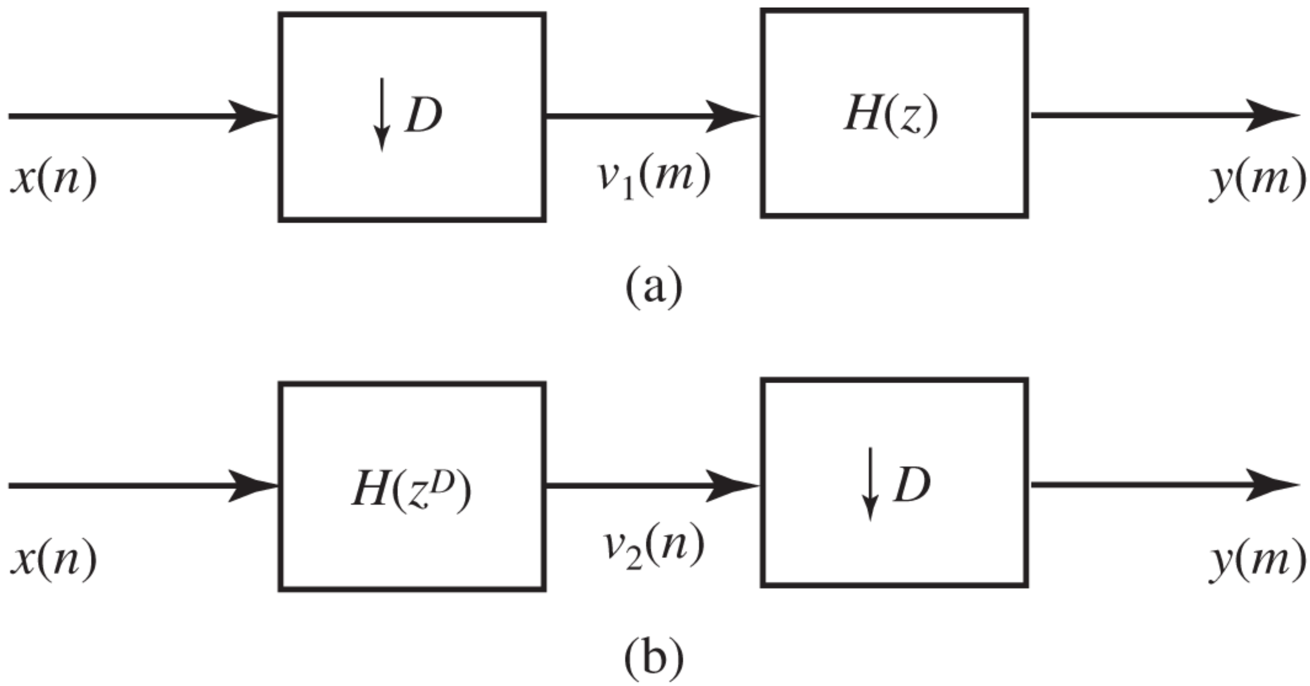


Figure 11.5.3 Two equivalent downsampling systems (first noble identity).

Interchanging the orders of filtering and interpolation

- Similar property can be derived for interpolation
- Assume a signal is passed through a filter first and then upsampled

$$Y(z) = H(z^I)X(z^I)$$

which is equivalent to upsampling first and then passing through a filter $H(z^I)$

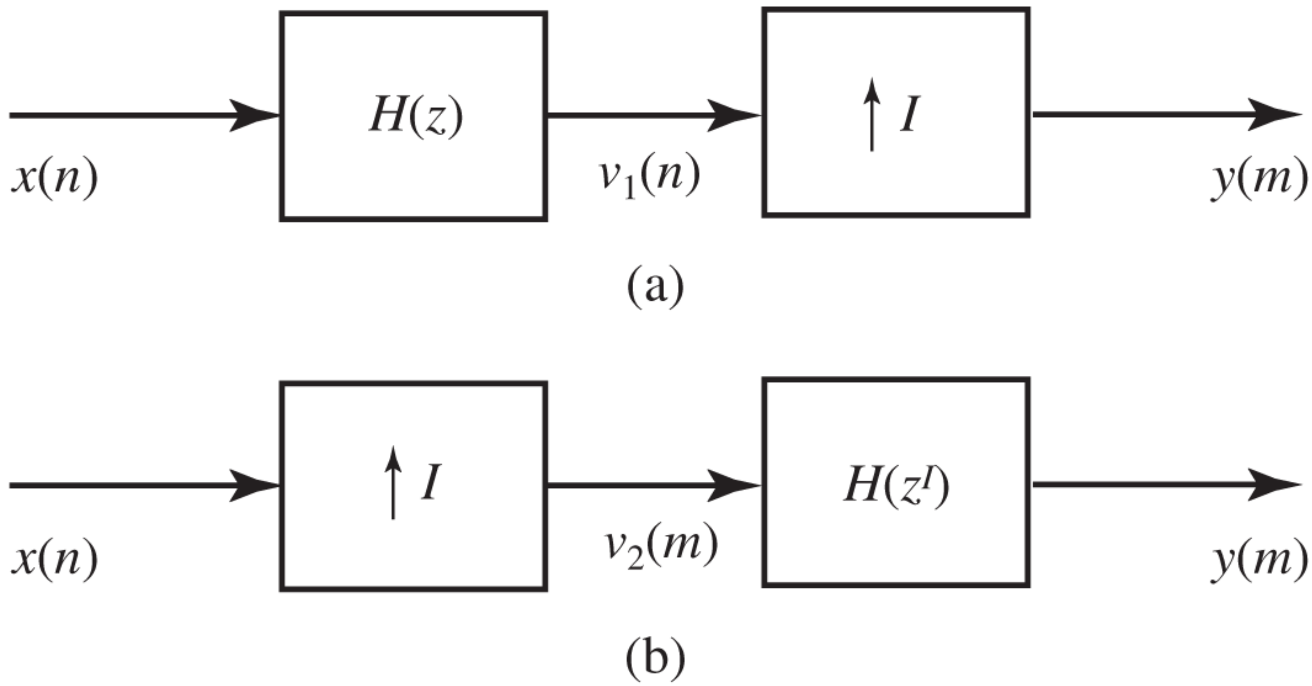


Figure 11.5.4 Two equivalent upsampling systems (second noble identity).

Polyphase implementation for linear systems

- Any system can be implemented by reshaping the filter coefficients as

$$H(z) = \sum_{i=0}^{M-1} z^{-i} \sum_{n=-\infty}^{\infty} h[nM+i] z^{-nM}$$

which can be written as

$$H(z) = \sum_{i=0}^{M-1} z^{-i} P_i(z^M)$$

with

$$P_i(z) = \sum_{n=-\infty}^{\infty} h[nM+i] z^{-n}$$

Polyphase implementation for linear systems

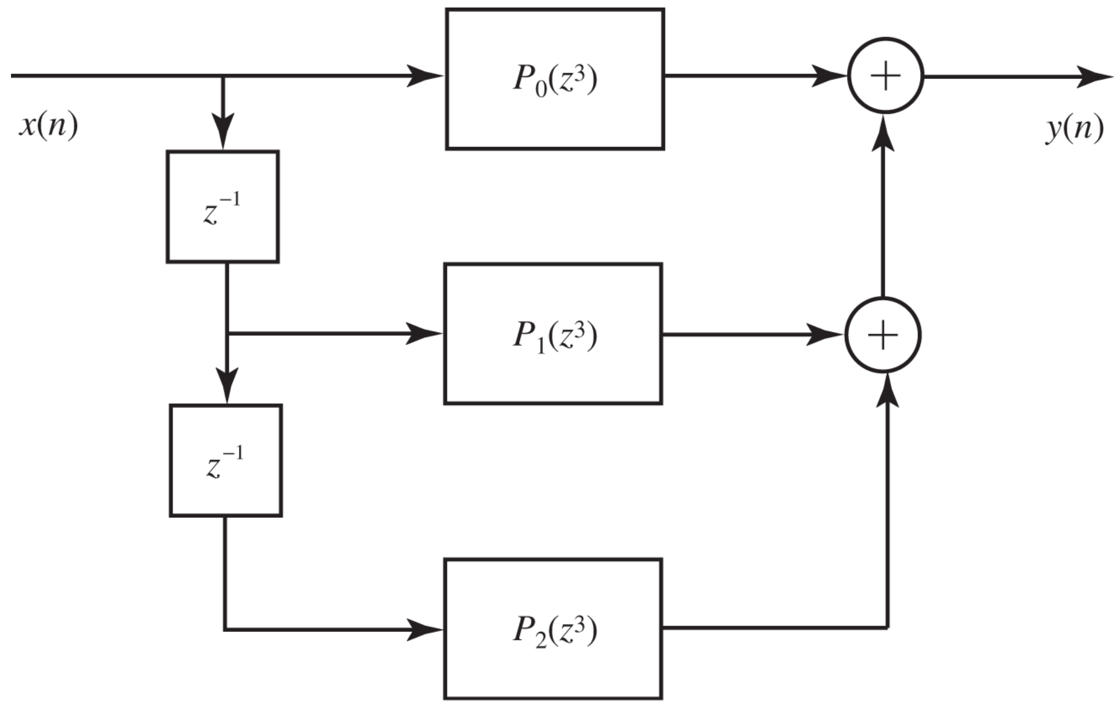


Figure 11.5.1 Block diagram of polyphase filter structure for $M = 3$.

Efficient Implementation for Decimation

- Consider a simple decimator

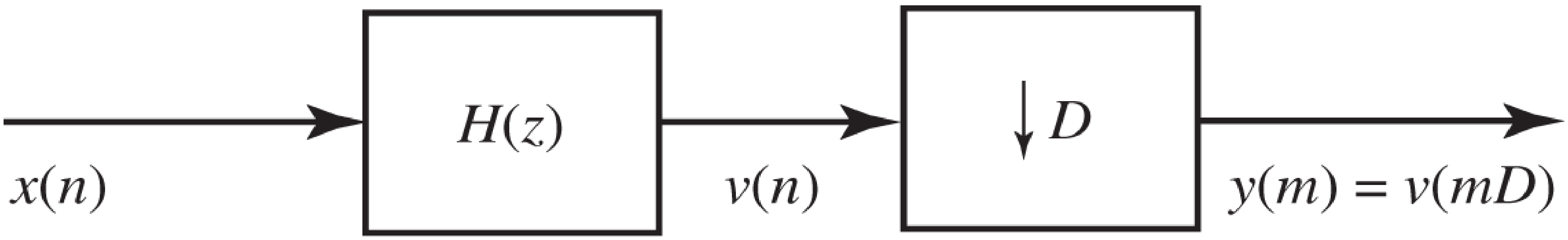
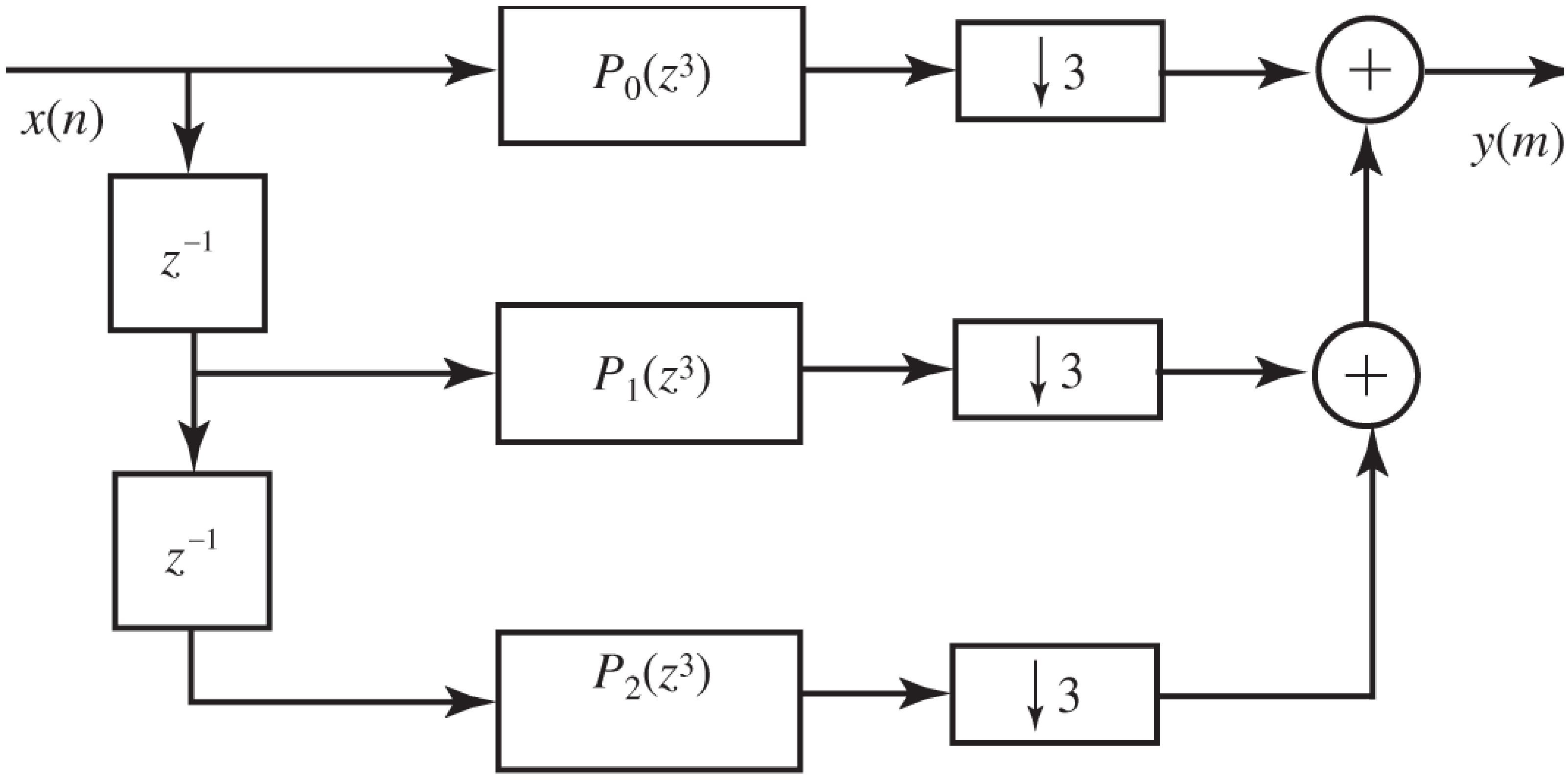


Figure 11.5.8 Decimation system.

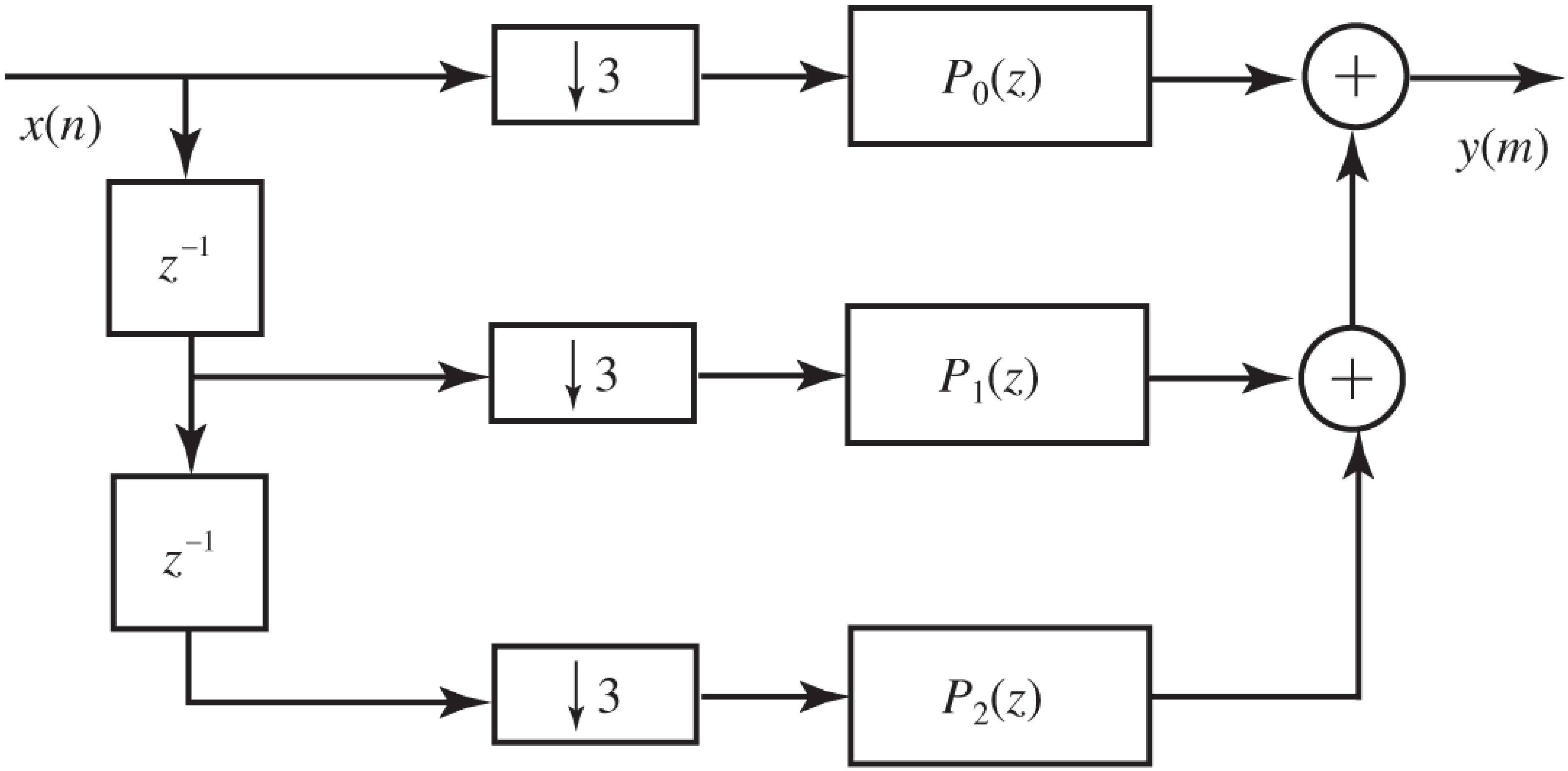
- We do not need every sample of the filter output, since it is decimated
- That is where a polyphase structure can help us
- Let us use a polyphase implementation for filtering

Efficient Implementation for Decimation



- This can be further simplified by the identities that we derived

Efficient Implementation for Decimation



- All of a sudden we are doing filtering in the downsampled world, a huge savings!

Efficient Implementation for Interpolation

- Similarly for an interpolator

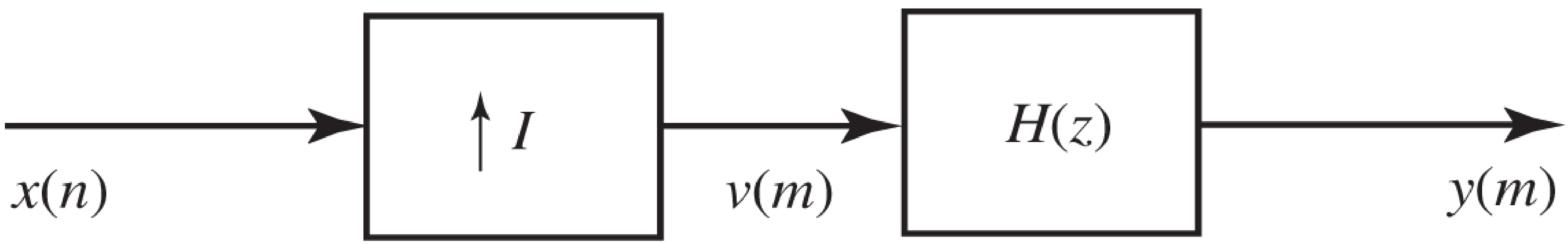
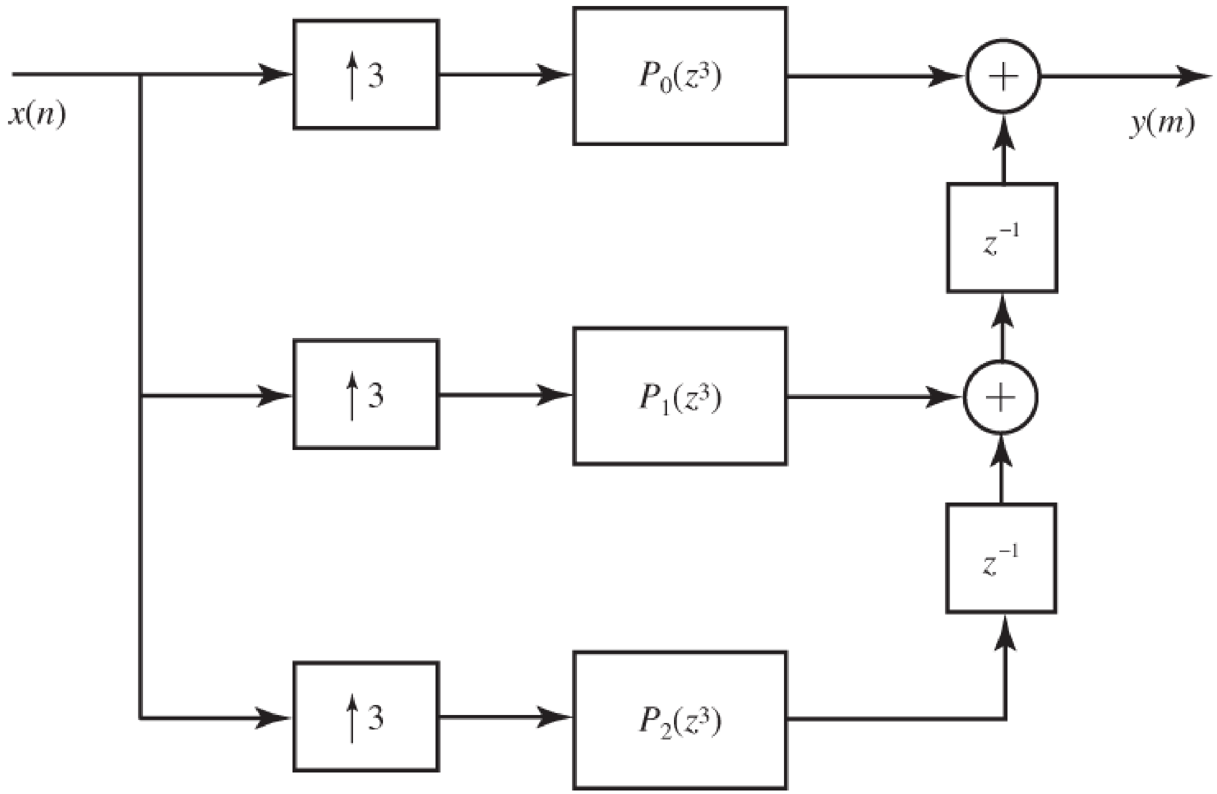


Figure 11.5.11 Interpolation system.

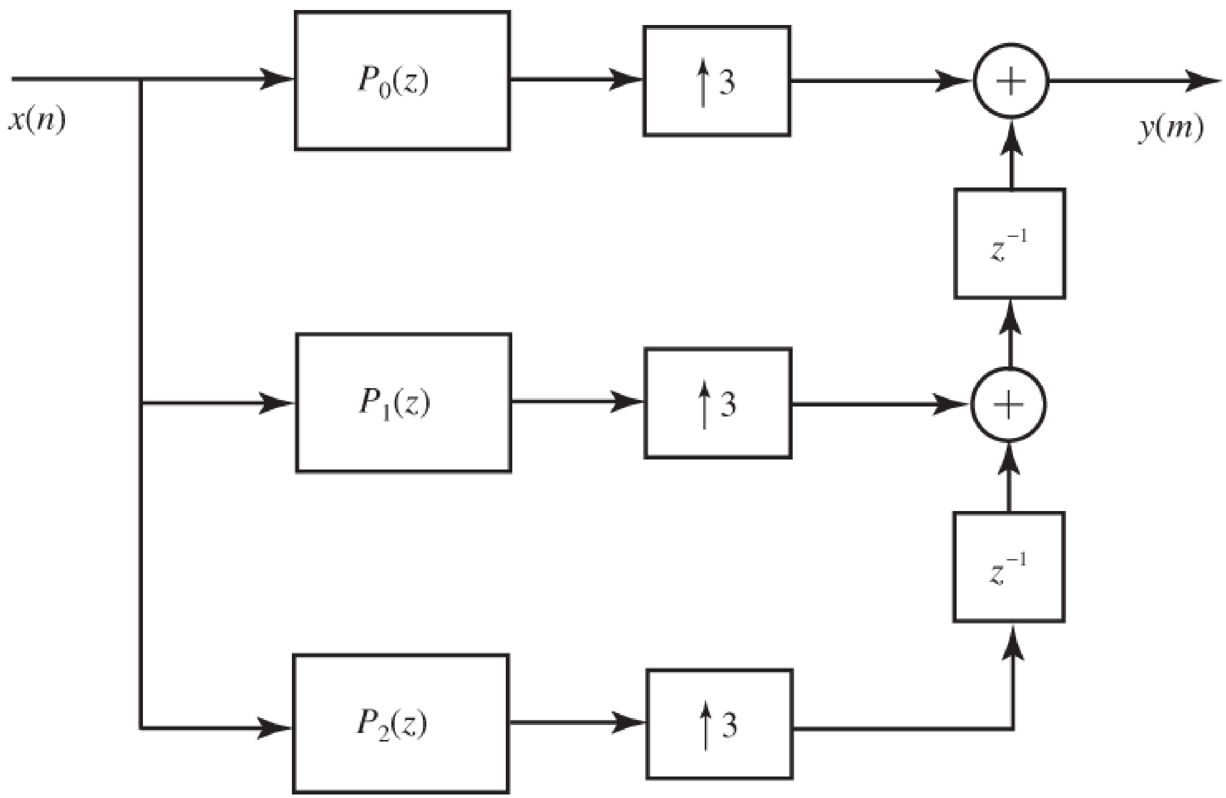
- We can use polyphase structure

Efficient Implementation for Interpolation



- This can be further simplified by the identities that we derived

Efficient Implementation for Interpolation



- All of a sudden we are doing filtering in the not upsampled world, a huge savings! Again..