Efficient structures for

Decimators and Interpolators

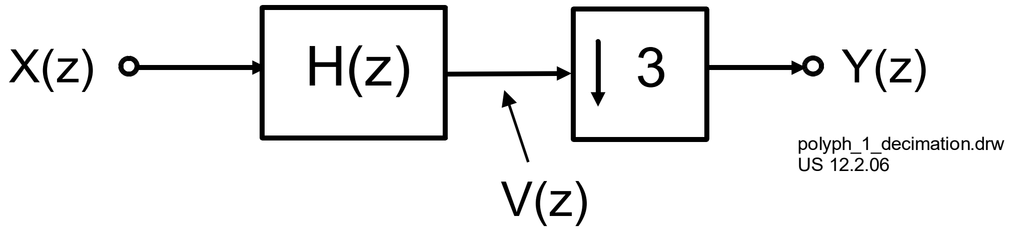
- An implementation example for an efficient structure for a decimator by a factor 3
- An implementation example for an efficient structure for an interpolator by a factor 3
- Multirate filters with optimum value for M for minimum computational effort

Änderungen/Korrekturen:

06-Nov-11	Folien 32-35 ergänzt (Kzr)
11-Nov-12	Folien 43-47 komplett überarbeitet, Zeichnungen geändert,
06-Nov-13	Folien 18 kein Delay erforderlich
14-Nov-13	Folien 35 Bild gelöscht, farbige Markierung
24-Oct-14	Folien 35 Bedeutung der Filter geändert
24-Oct-14	Folien 43 Low-pass und High-pass im Bild vertauscht
11-Oct-16	Folie 32 $X_i(n) \rightarrow X_i(z)$, i=1, 2, 3, 4 ersetzt im obigen Bild
14-Jun-19	Dateinamen. Struktur neu aufgebaut

Decimation by 3: efficient structure [1]

- **A decimation filter with decimation factor 3 (note: H(z) operates on higher Fs)**



- **We assume that H(z) is realized by using an FIR filter, transfer function H(z)**
- **For demonstration purposes, we assume a filter degree of the filter, N=8.**

- **Thus**

$$H(z) = V(z) / X(z)$$

$$H(z) = b_0 + b_1 \cdot z^{-1} + b_2 \cdot z^{-2} + b_3 \cdot z^{-3} + b_4 \cdot z^{-4} \ldots$$
$$+ b_5 \cdot z^{-5} + b_6 \cdot z^{-6} + b_7 \cdot z^{-7} + b_8 \cdot z^{-8}$$

Decimation by 3: efficient structure [2]

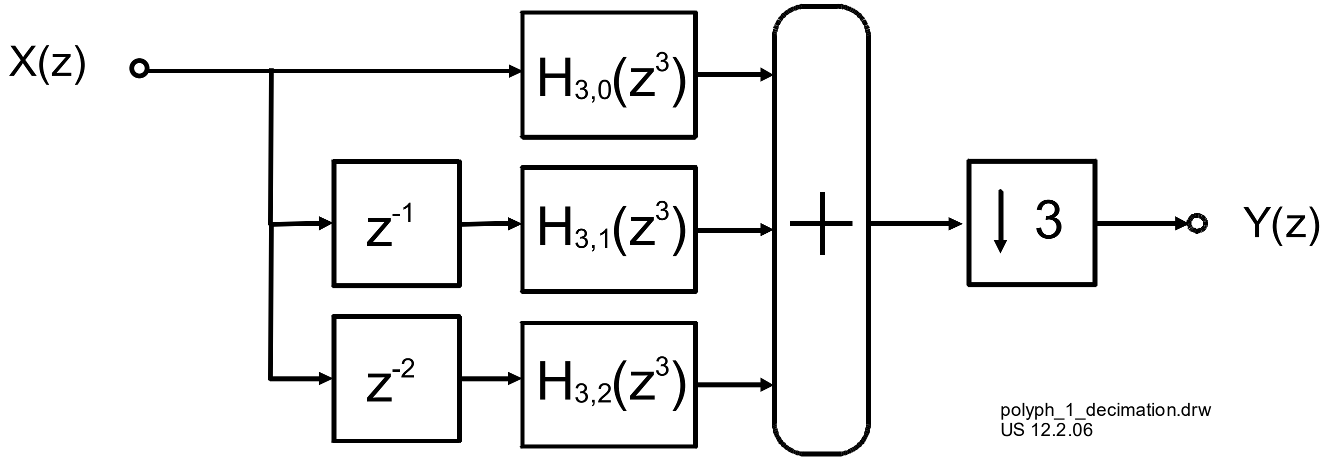
The polyphase decomposition for M=3 : (note : components are in z^3)

$$H(z) = H_{3,0}(z^3) + z^{-1} \cdot H_{3,1}(z^3) + z^{-2} \cdot H_{3,2}(z^3)$$

$$H_{3,0}(z^3) = b_0 + b_3 \cdot z^{-3} + b_6 \cdot z^{-6},$$

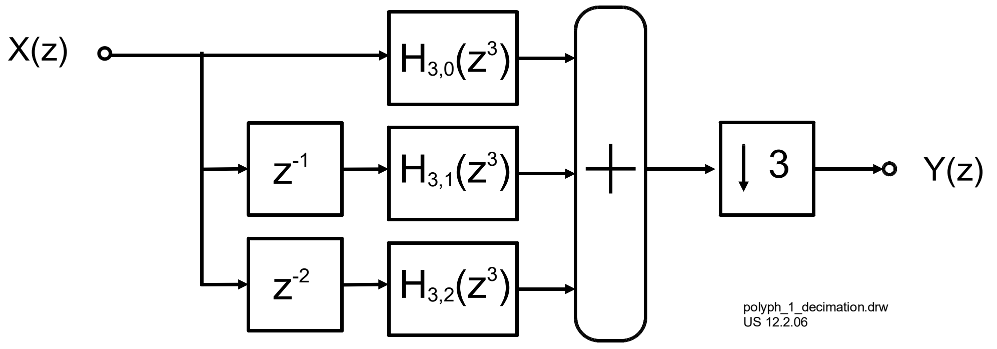
$$H_{3,1}(z^3) = b_1 + b_4 \cdot z^{-3} + b_7 \cdot z^{-6},$$

$$H_{3,2}(z^3) = b_2 + b_5 \cdot z^{-3} + b_8 \cdot z^{-6}$$

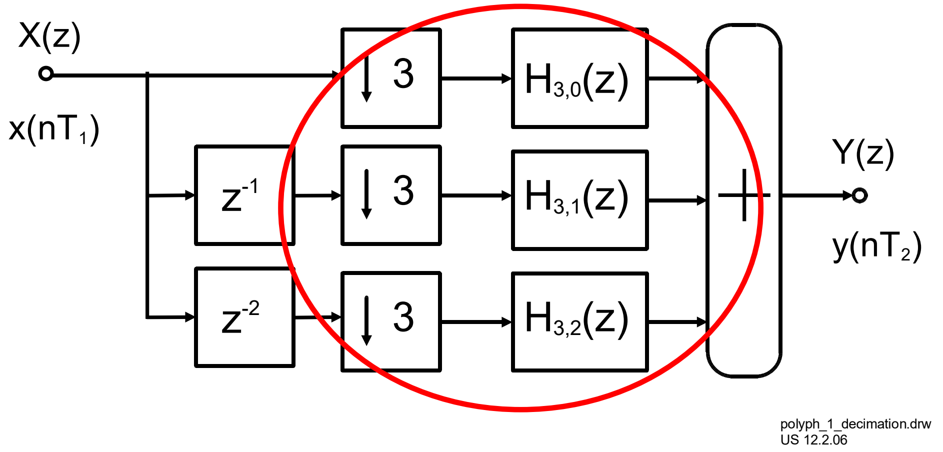


Decimation by 3: efficient structure [3]

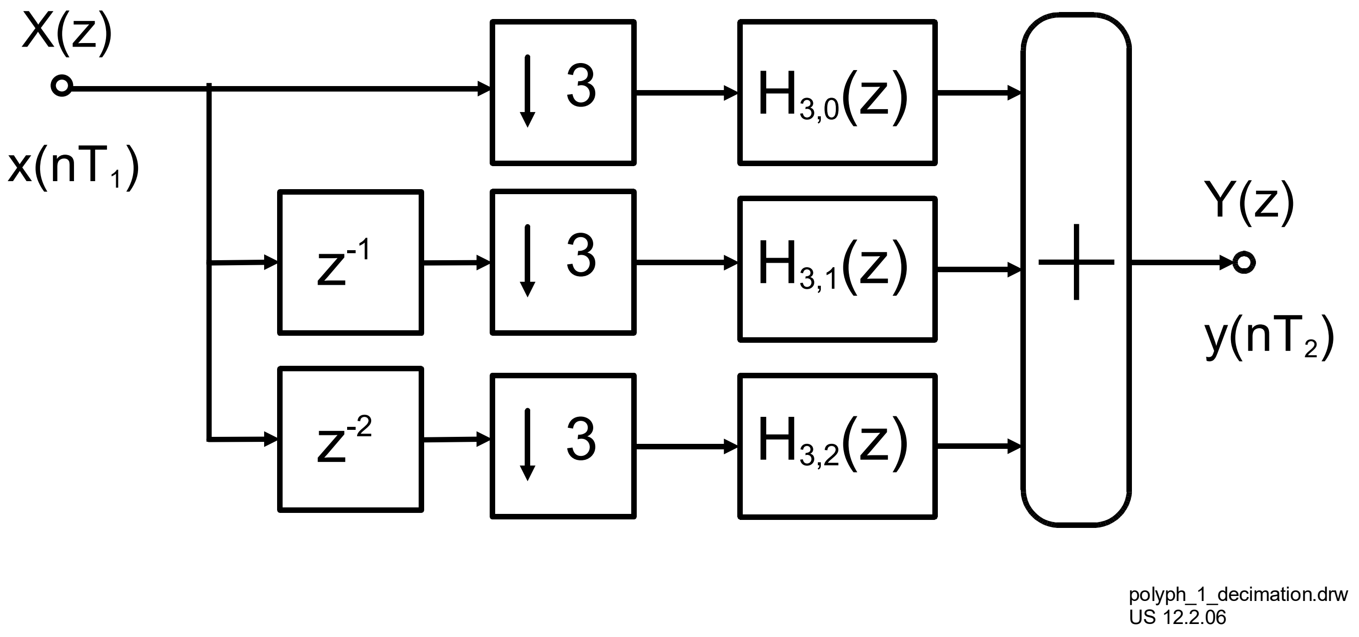
Again: Polyphase structure for decimation (delays are on input side)



Using first noble identity, we obtain (note H(.) is now in z, not z^3 any more !!)



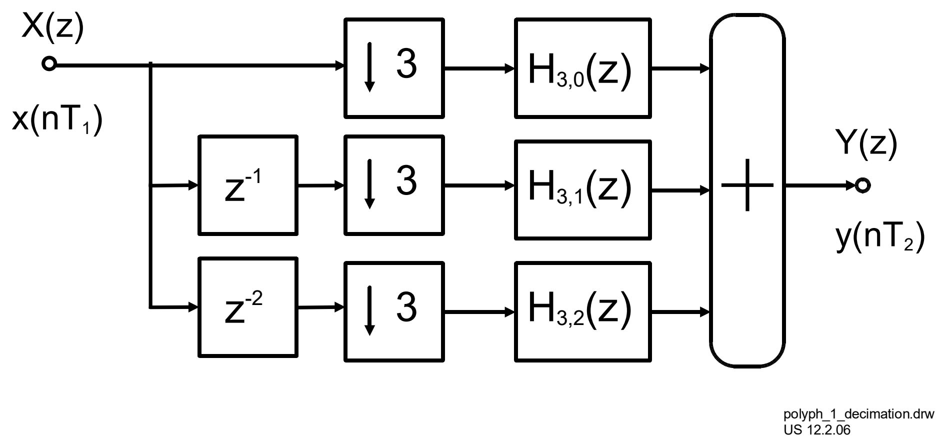
Decimation by 3: efficient structure [4]



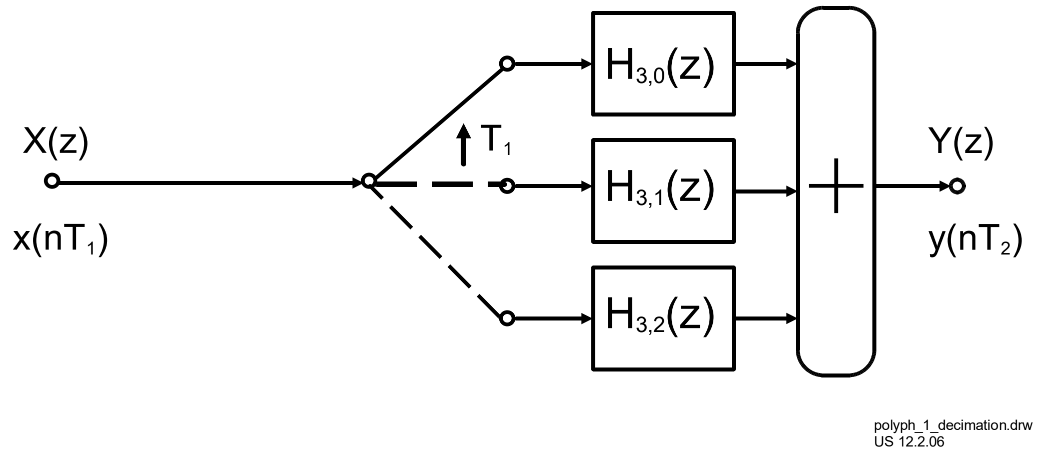
- **Some important remarks for decimation**
  - **Samples $x[3nT_1]$ are sent to $\rightarrow$ polyphase filter $H_{3,0}(z)$**
  - **Samples $x[3nT_1-1]$ are sent to $\rightarrow$ polyphase filter $H_{3,1}(z)$**
  - **Samples $x[3nT_1-2]$ are sent to $\rightarrow$ polyphase filter $H_{3,2}(z)$**

Decimation by 3: efficient structure [5]

Again the previous structure with $H_{3,0}$, $H_{3,1}$, $H_{3,2}$ in z



Efficient: Using a de-interleaver instead, which rotates anti-clockwise with $1/T_1$

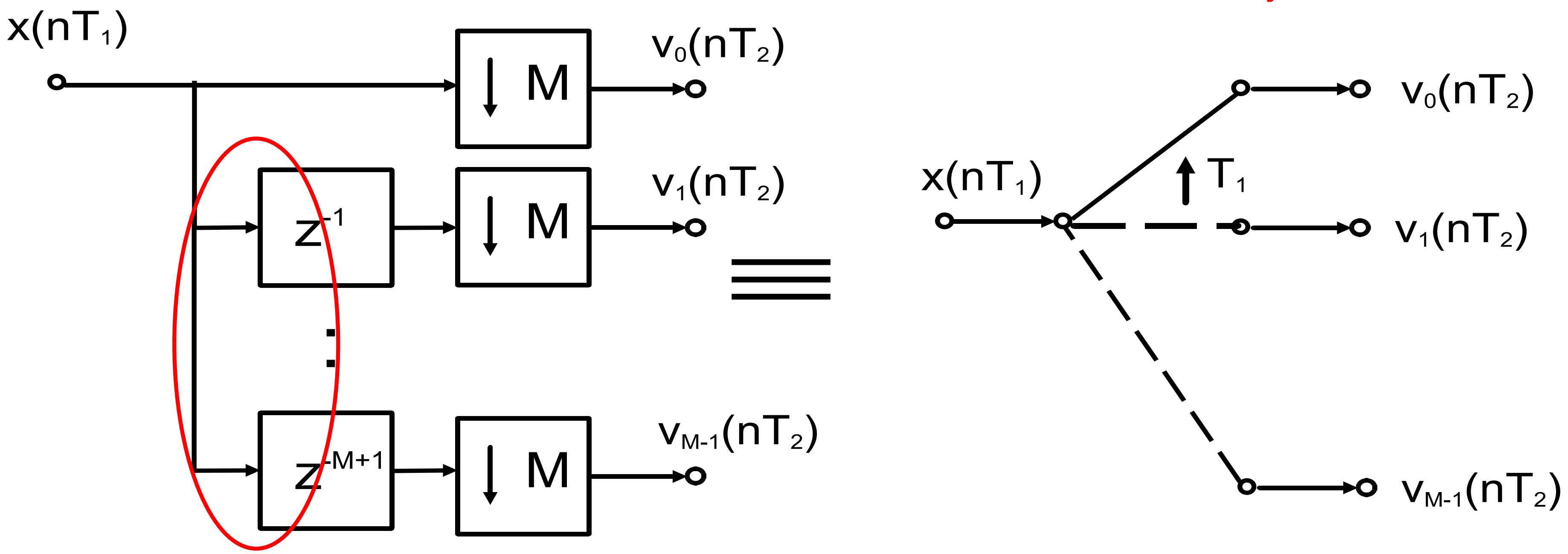


Important remark: the diagram to the left just shows the principle. It is not yet 100% correct.

General de-interleaver (for general M)

- **The two signal-flow diagrams for the de-interleaver on this and on the next slide are equivalent**
- **The switch rotates anti-clockwise with $1/T_1$**

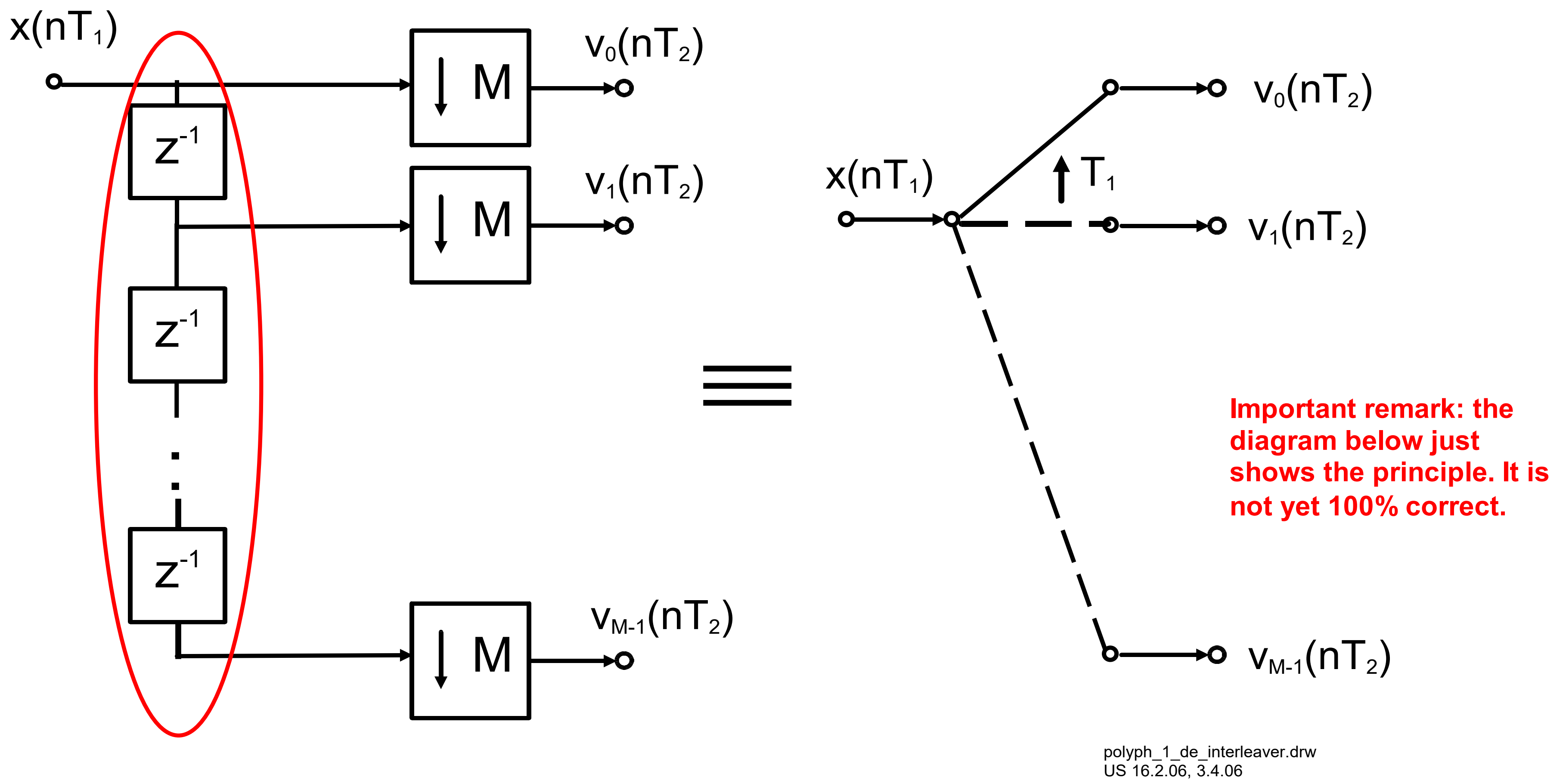
Important remark: the diagram below just shows the principle. It is not yet 100% correct.



polyph_1_de_interleaver.drw
US 16.2.06, 3.4.06

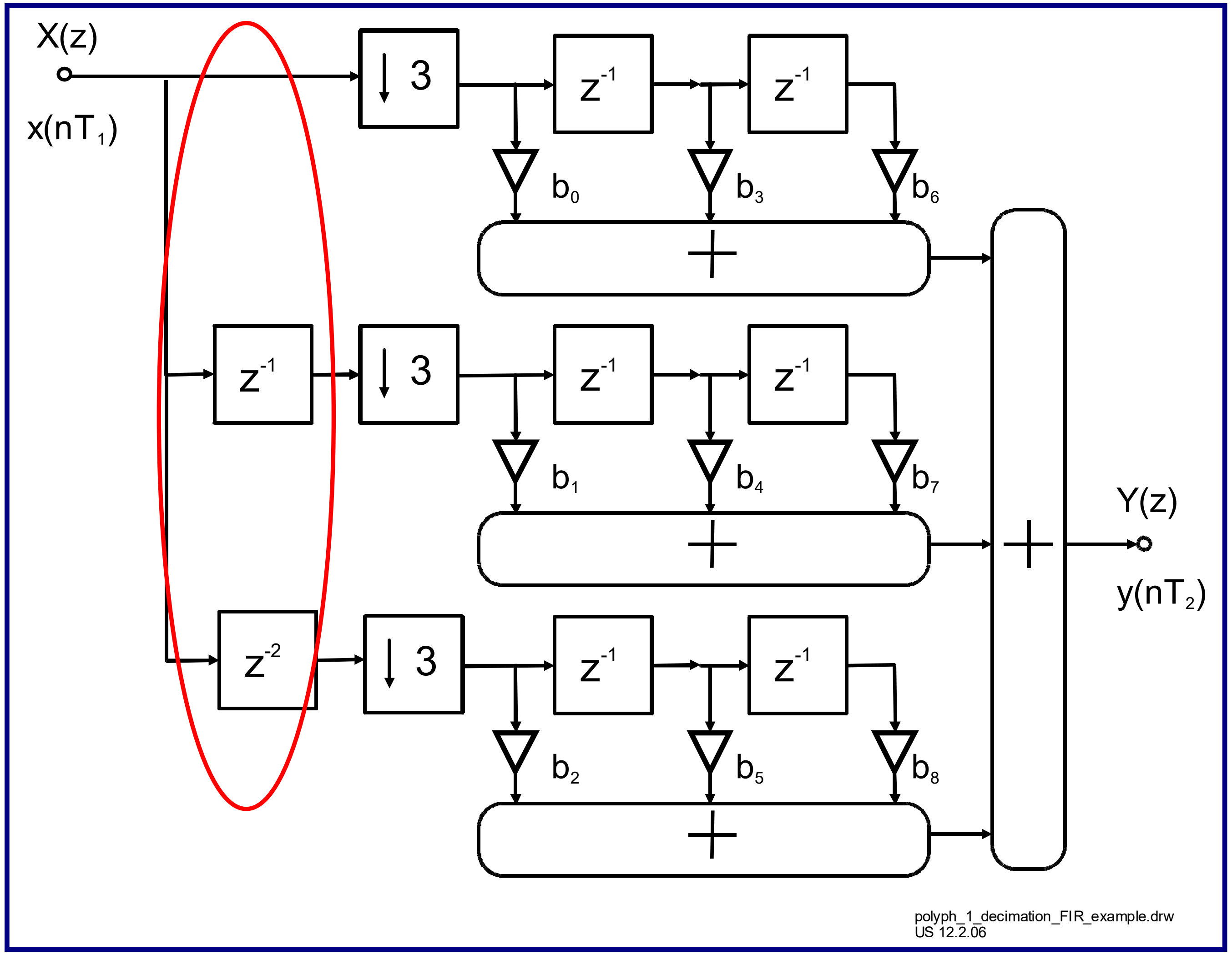
General de-interleaver (for general M)

- **The two signal-flow diagrams for the de-interleaver on this and on the previous slide are equivalent. The switch rotates anti-clockwise with $1/T_1$**



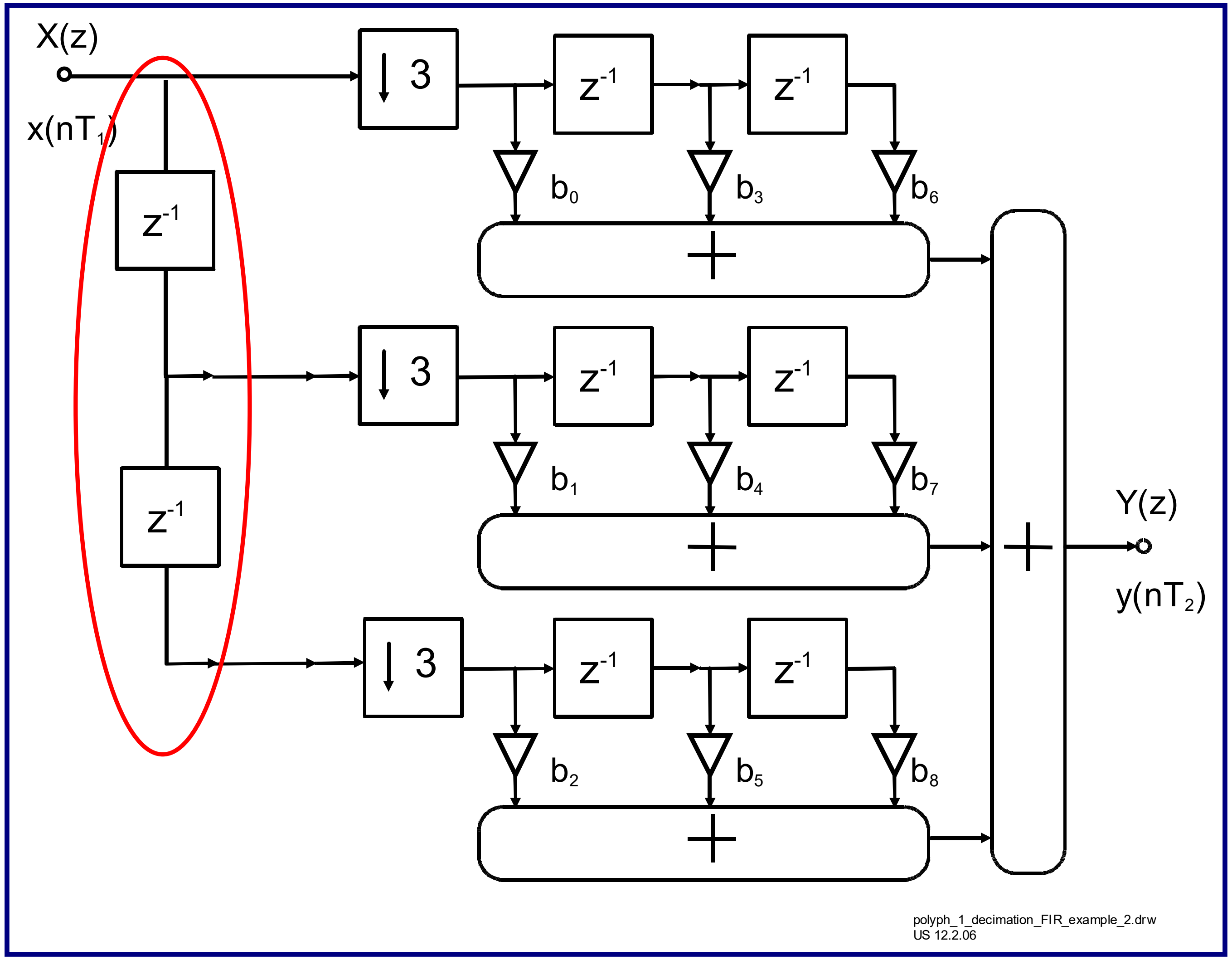
Important remark: the diagram below just shows the principle. It is not yet 100% correct.

Decimation by 3: a) final efficient structure for N=8 [6]



$$H_{3,0}(z) = b_0 + b_3 \cdot z^{-1} + b_6 \cdot z^{-2},$$
$$H_{3,1}(z) = b_1 + b_4 \cdot z^{-1} + b_7 \cdot z^{-2},$$
$$H_{3,2}(z) = b_2 + b_5 \cdot z^{-1} + b_8 \cdot z^{-2}$$

Decimation by 3: b) final efficient structure for N=8 [7]



Note:
different de-interleaver,
same behavior

$$H_{3,0}(z) = b_0 + b_3 \cdot z^{-1} + b_6 \cdot z^{-2},$$
$$H_{3,1}(z) = b_1 + b_4 \cdot z^{-1} + b_7 \cdot z^{-2},$$
$$H_{3,2}(z) = b_2 + b_5 \cdot z^{-1} + b_8 \cdot z^{-2}$$

Decimation by 3: b) MATLAB [8]

MATLAB implementation:

- **The delays are updated on Fs**
- **NO de-interleaver yet**
- **Quite Efficient decimation by 3, which can still be improved (see later)**

Important remark: the MATLAB code contains the corrections (delays…) already.

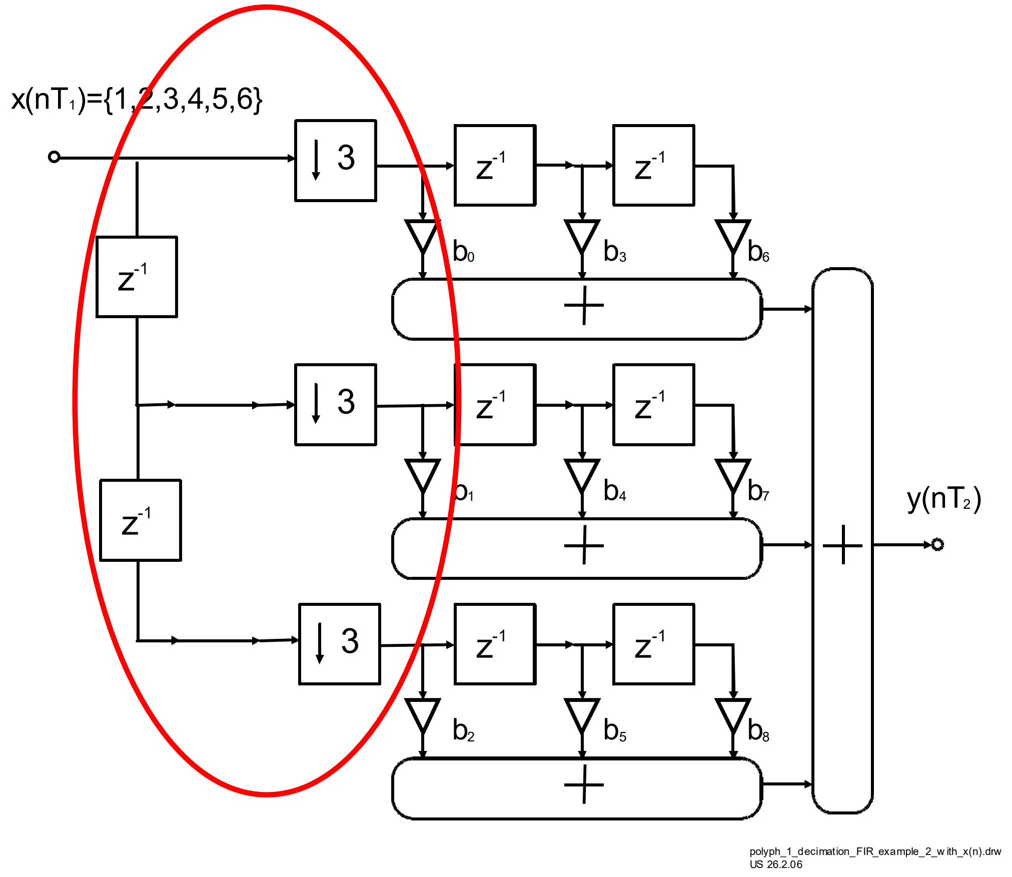
```
% kernel from sim_L1_M3.m

MM=3;
% the coefficients
b_filt_b0_b3_usw = b_fir(1:MM:length(b_fir));
b_filt_b1_b4_usw = b_fir(2:MM:length(b_fir));
b_filt_b2_b5_usw = b_fir(3:MM:length(b_fir));

% the delays
H_filt_30_delays = zeros(1,length(b_filt_b0_b3_usw));
H_filt_31_delays = zeros(1,length(b_filt_b1_b4_usw));
H_filt_32_delays = zeros(1,length(b_filt_b2_b5_usw));

% filter x(n)
y=zeros(1,N_samp/MM);
mx=0;

% delays
D1 = 0;
D2 = 0;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for kx=1:N_samp

    if rem(kx-1,MM)==0  % shift filter delays, calculate output only one out of 3 times
        mx = mx + 1;
        H_filt_30_delays = [x(kx), H_filt_30_delays(1:length(H_filt_30_delays)-1)];
        H_filt_31_delays = [D1, H_filt_31_delays(1:length(H_filt_31_delays)-1)];
        H_filt_32_delays = [D2, H_filt_32_delays(1:length(H_filt_32_delays)-1)];

        y_part1(mx)= b_filt_b0_b3_usw * H_filt_30_delays';
        y_part2(mx)= b_filt_b1_b4_usw * H_filt_31_delays';
        y_part3(mx)= b_filt_b2_b5_usw * H_filt_32_delays';

        y(mx) = y_part1(mx) + y_part2(mx) + y_part3(mx);
    end

    D2 = D1;            % shift D1 and D2 always
    D1 = x(kx);
end
```

Decimation by 3: b) how to program even more efficiently [1]

- **Looking again at the signal-flow diagram it is clear that the computation of the decimation filter output signal can be implemented in various modes**
  - **Block mode:**
    the filter output signal is computed in "one step". This means that a large number of operations has to be carried out during this "one step", while few operations (update of delays only …) have to be carried out during other time instances. It should be mentioned that "block mode" requires additional delay elements for buffering…
  - **SISO-mode (Serial-In-Serial-Out) mode:**
    The DSP receives input-INTerrupts and output-INTerrupts depending upon the input- and output sampling frequencies and processes the samples on a serial basis. Example: for $F_{s,in}$=48 kHz and $F_{s,out}$=16 kHz, there is one sample at the output for every block of three samples at the input.
- **We will consider the SISO-mode (Serial-In-Serial-Out) mode in this lecture only. For block-mode, see ref. [1].**
  - **SISO-mode: very efficient implementation, steering of the circuit more complex**
  - **Block mode: less efficient (more RAM required), "simple" steering**

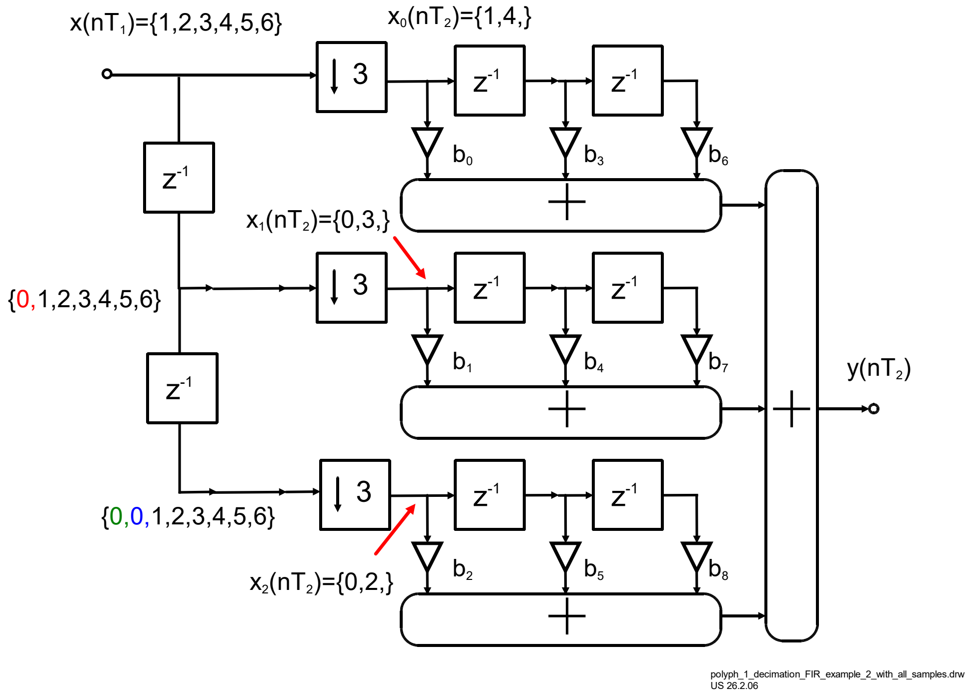
Decimation by 3: b) how to program even more efficiently [2]

- **We assume M=3 and x(n)={1,2,3,4,5,6}. In order to program efficiently, we distribute the computation of the filter output signal across three input samples.**
  - **If n=0, compute the upper FIR filter, save result**
  - **If n=1, compute the middle FIR filter, add result to result obtained for n=0, save result**
  - **If n=2, compute the lower FIR filter, add result to result obtained for n=1, output ready**
  - **It is however not right-away clear from the signal-flow diagram, how to distribute the computation for the delays and the three down-samplers on the input side**



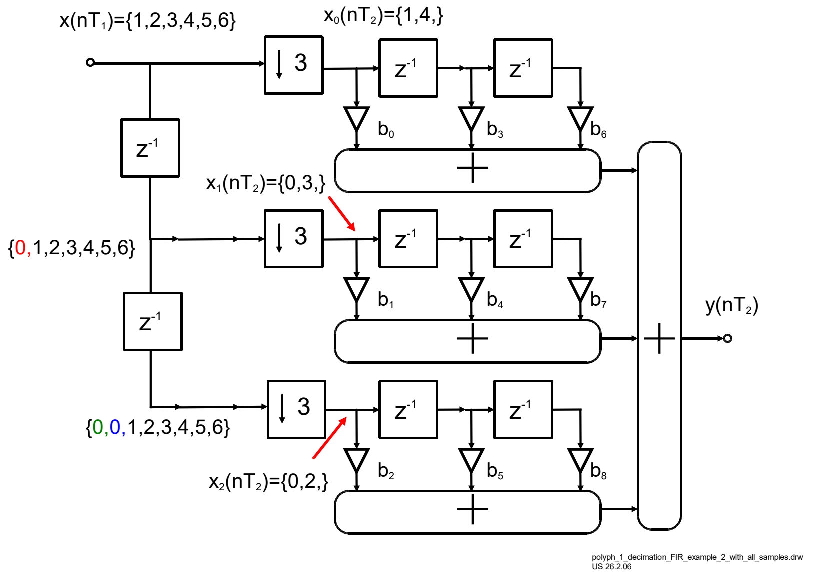
Decimation by 3: b) how to program even more efficiently [3]

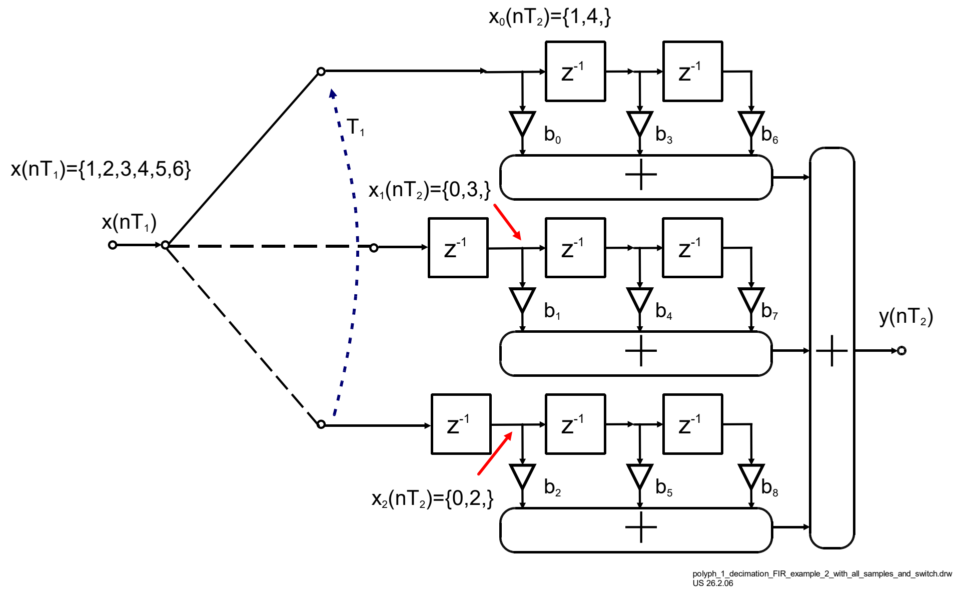
- **How to distribute the computation for the delays and the three down-samplers on the input side:**
  - **One possibility is again block processing: updating all delays of the 3 FIR filters only every M samples, but: this is not efficient**
- **Better: use a de-interleaver, but do this carefully !!**
  - **Assume x(n)={1,2,3,4,5,6}**
  - **Note how x_0, x_1, x_2 are obtained from the two delay elements and the down-samplers**
  - **This corresponds to a de-interleaver which rotates anti-clock-wise, plus a delay element except for x_0**



Decimation by 3: b) how to program even more efficiently [4]

- **These two signal-flow diagrams are equivalent.**
- **The computation of the FIR filter output signal is equally distributed across M input samples $\Rightarrow$ maximum efficiency**
  - **The upper FIR filter is computed first for n=0 on the input INT side (3x larger !)**
  - **followed by the lower (!) FIR filter for n=1, add upper output and save,**
  - **followed by the middle (!) FIR filter for n=2, add lower output, $y(nT_2)$ ready**
  - **Note : lower FIR filter computation is done before middle FIR filter computation**

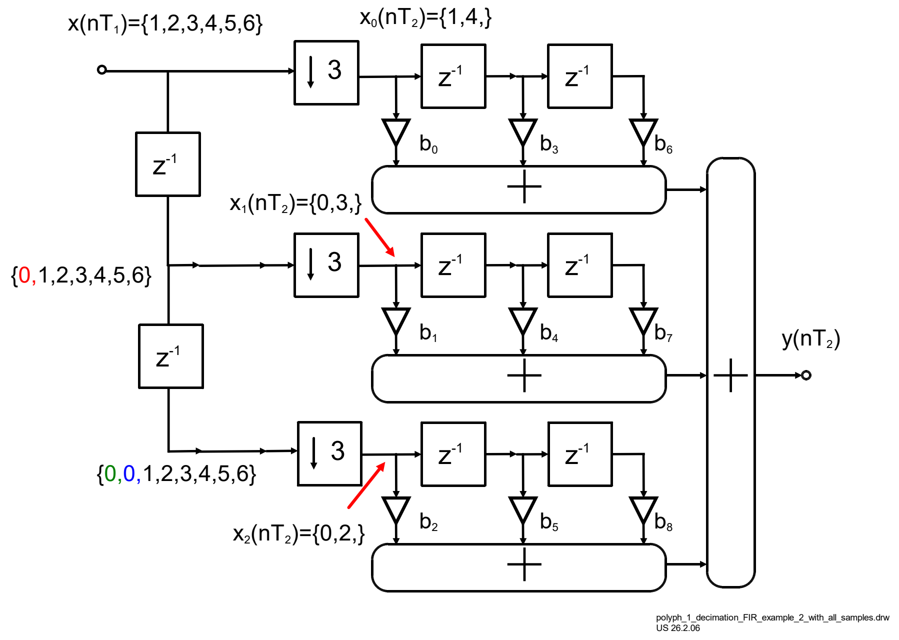


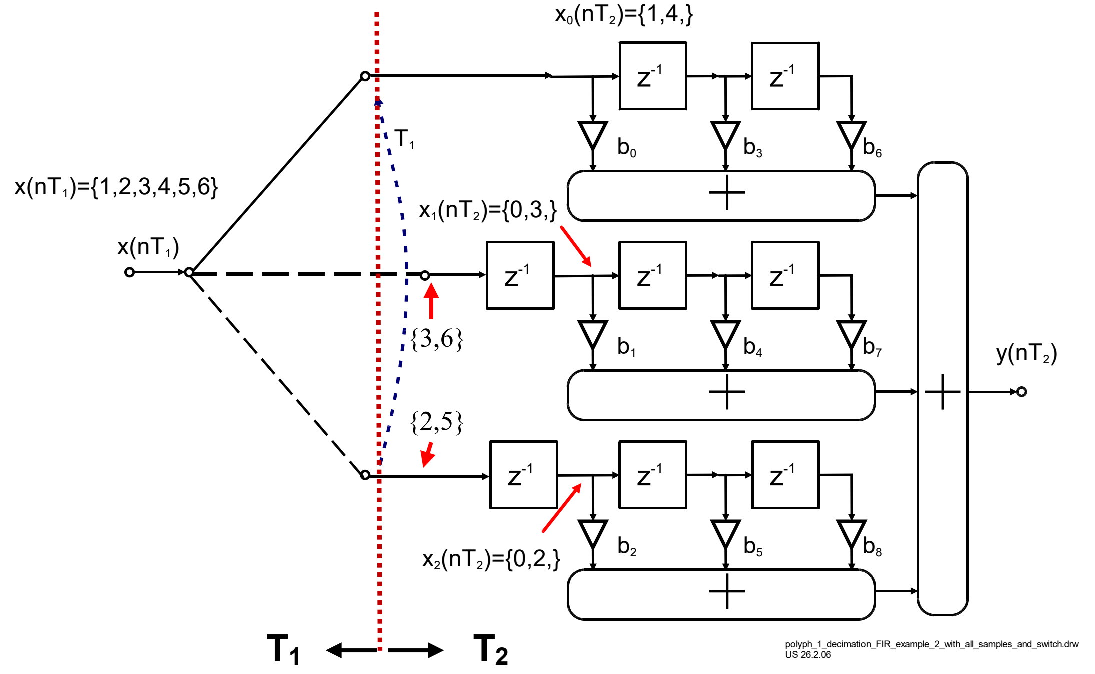


Decimation by 3: how to <u>program</u> even more efficiently [5]

For M=3, three polyphase signals can be defined

- **$x_0(m)=x(n)$ = {x(0), x(3), x(6), …} = {1, 4, 7,…}, n=0, 3, 6, … m=0, 1, 2, ...**
- **$x_1(m)=x(n-1)$ = {x(-1), x(2), x(5), …} = {?, 3, 6,…} daher gilt n=3m**
- **$x_2(m)=x(n-2)$ = {x(-2), x(1), x(4), …} = {?, 2, 5,…}**





(Remark : see P.P.Vaidyanathan, “Multirate Systems and Filter Banks”, 1993, pp. 131, Fig. 4.3-9 a, b, Göckler/Groth. S. 195)

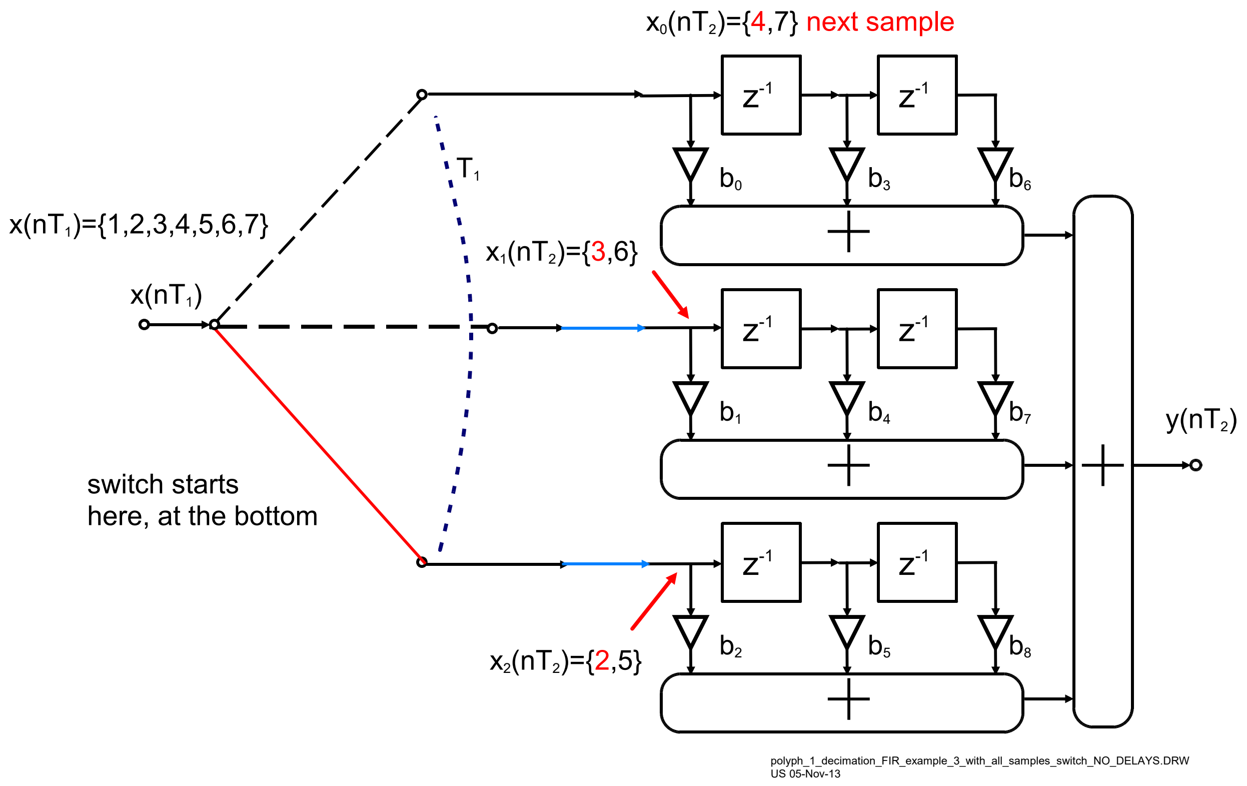
Decimation by 3: b) final, efficient MATLAB [5]

A MATLAB example

```
% kernel from decimate_by_3_FIR_1_stage.m

% the coefficients
b_filt_b0_b3 = b_fir_remez(1:MM:length(b_fir_remez));
b_filt_b1_b4 = b_fir_remez(2:MM:length(b_fir_remez));
b_filt_b2_b5 = b_fir_remez(3:MM:length(b_fir_remez));

% the delays
H_filt_30_delays = zeros(1,length(b_filt_b0_b3));
H_filt_31_delays = zeros(1,length(b_filt_b1_b4));
H_filt_32_delays = zeros(1,length(b_filt_b2_b5));

% filter x(n)
y=zeros(1,N_samp/MM);
mx=0;

% delays
D1 = 0;
D2 = 0;

% switch rotates anti-clockwise (!!!)
for kx=1:N_samp

    if rem(kx-1,MM)==0
        mx=mx+1;
        H_filt_30_delays = [x(kx),
           H_filt_30_delays(1:length(H_filt_30_delays)-1)];
        y_part1(mx)= b_filt_b0_b3 * H_filt_30_delays';

    elseif rem(kx-1,MM)==1
        H_filt_32_delays = [D2,
           H_filt_32_delays(1:length(H_filt_32_delays)-1)];
        y_part3(mx)= b_filt_b2_b5 * H_filt_32_delays';
        D2 = x(kx);

    elseif rem(kx-1,MM)==2
        H_filt_31_delays = [D1,
           H_filt_31_delays(1:length(H_filt_31_delays)-1)];
        y_part2(mx)= b_filt_b1_b4 * H_filt_31_delays';
        D1 = x(kx);
% add them up
        y(mx) = y_part1(mx) + y_part2(mx) + y_part3(mx);

    end; % if
end; % for
```

Decimation by 3: how to program even more efficiently [6]

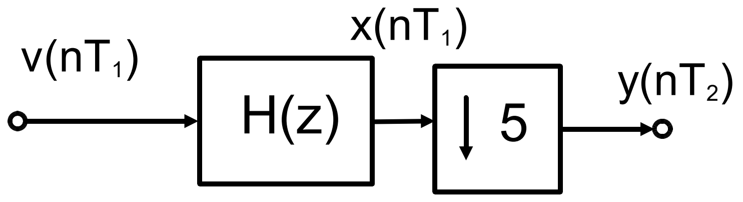
No additional delays in branch #1 … are required if the switch starts at the bottom of the arrangement and for branch #0, the next input sample is used.
Then, the switch position inherently contains the delay from previous slides.



$x_0(nT_2)=\{4,7\}$ next sample
z^{-1}
T_1
b_0
b_3
b_6
$x(nT_1)=\{1,2,3,4,5,6,7\}$
$x_1(nT_2)=\{3,6\}$
$x(nT_1)$
b_1
b_4
b_7
$y(nT_2)$
switch starts here, at the bottom
b_2
b_5
b_8
$x_2(nT_2)=\{2,5\}$
polyph_1_decimation_FIR_example_3_with_all_samples_switch_NO_DELAYS.DRW
US 05-Nov-13

Exercise: M=5 decimation, programming efficiently

A decimation filter for $F_{s,in}$= 36621.093750 Hz to $F_{s,out}$= $F_{s,in}$ /5, thus M=5 has to be implemented.



decimate_by_5.drw
US 26.2.06

- **Assume that the input x(n)={1,2,3,4,5,6,7,0,0,…} and develop/draw the detailed signal-flow diagram for M=5 with de-interleaver.**
- **Determine the polyphase components x_0, … ,x_4**
  Write the MATLAB code for the filter specification:

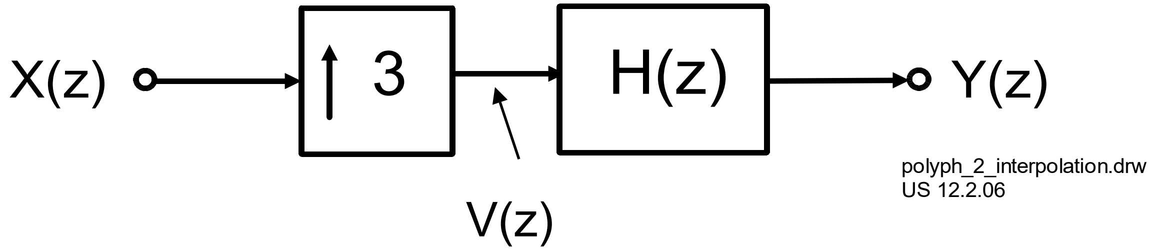
δ_{pass}	:	0.1,
δ_{stop}	:	0.1, thus 20 dB stop-band attenuation,
f_{pass}	:	1000 Hz

and test it with a 500 Hz sinus signal `decimate_by_5_FIR_1_stage_design_example.h`

(You may assume that the linear-phase FIR filter has 10 coefficients b_0,..,b_9)

Interpolation by 3: efficient structure [1]

An interpolation filter with interpolation factor 3 (note: H(z) operates again on higher Fs)



- **We assume (again) that H(z) is realized by using an FIR filter**
- **For demonstration purposes, we assume a filter degree of the filter, N=8.**

- **Thus**

$$H(z) = Y(z)/V(z)$$

$$H(z) = b_0 + b_1 \cdot z^{-1} + b_2 \cdot z^{-2} + b_3 \cdot z^{-3} + b_4 \cdot z^{-4} \ldots$$
$$+ b_5 \cdot z^{-5} + b_6 \cdot z^{-6} + b_7 \cdot z^{-7} + b_8 \cdot z^{-8}$$

Interpolation by 3: efficient structure [2]

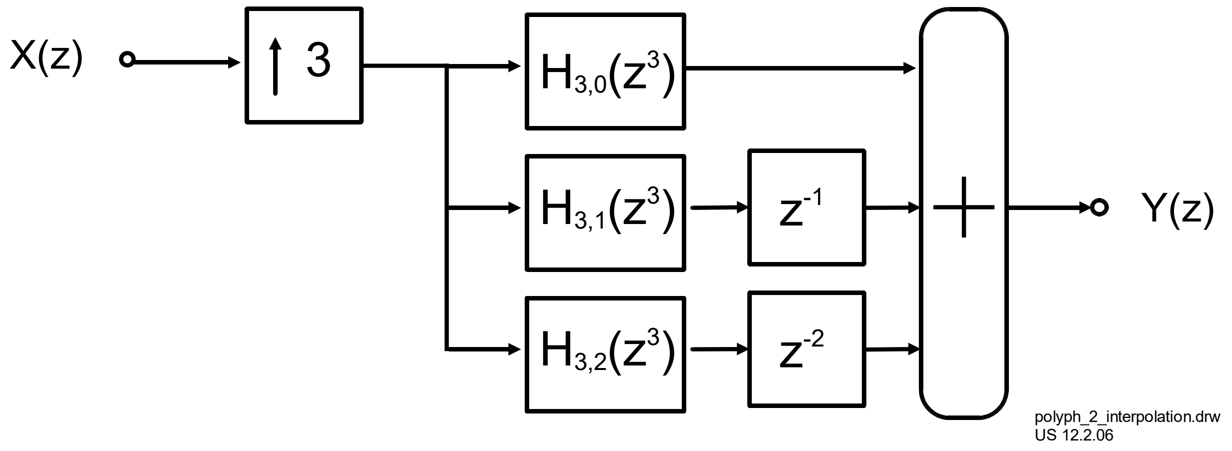
The polyphase decomposition for L=3 : (note : components are in z^3)

$$H(z) = H_{3,0}(z^3) + z^{-1} \cdot H_{3,1}(z^3) + z^{-2} \cdot H_{3,2}(z^3)$$

$$H_{3,0}(z^3) = b_0 + b_3 \cdot z^{-3} + b_6 \cdot z^{-6},$$

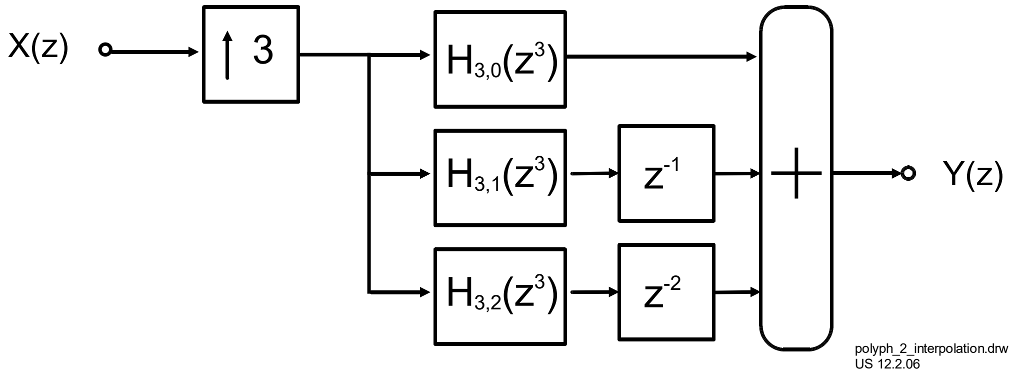
$$H_{3,1}(z^3) = b_1 + b_4 \cdot z^{-3} + b_7 \cdot z^{-6},$$

$$H_{3,2}(z^3) = b_2 + b_5 \cdot z^{-3} + b_8 \cdot z^{-6}$$

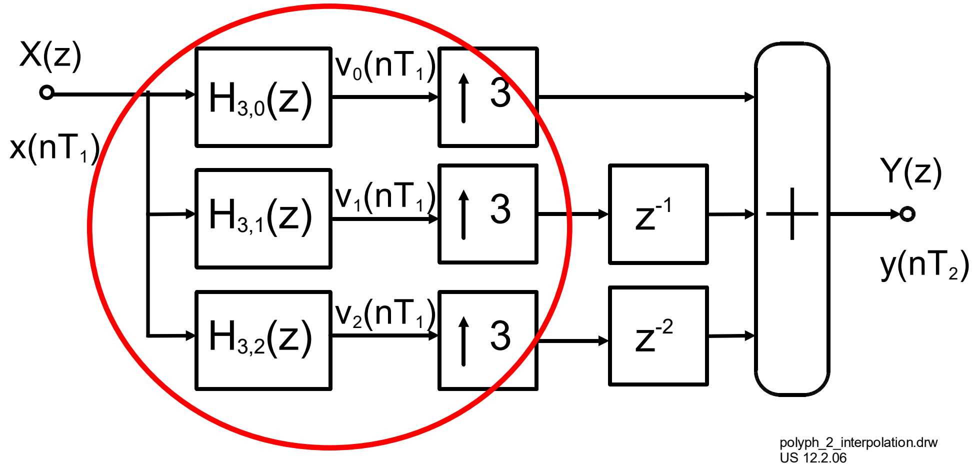


Interpolation by 3: efficient structure [3]

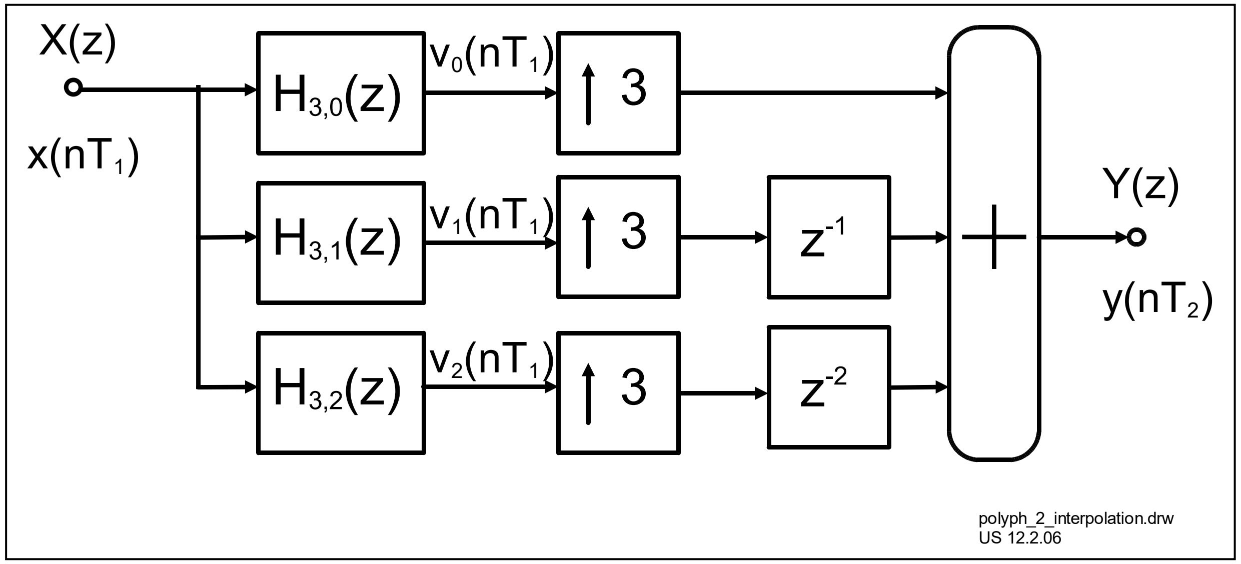
Again: Polyphase structure for interpolation (delays are on output side)



Using second noble identity, we obtain (note H(.) is now in z, not z^3 any more !!)



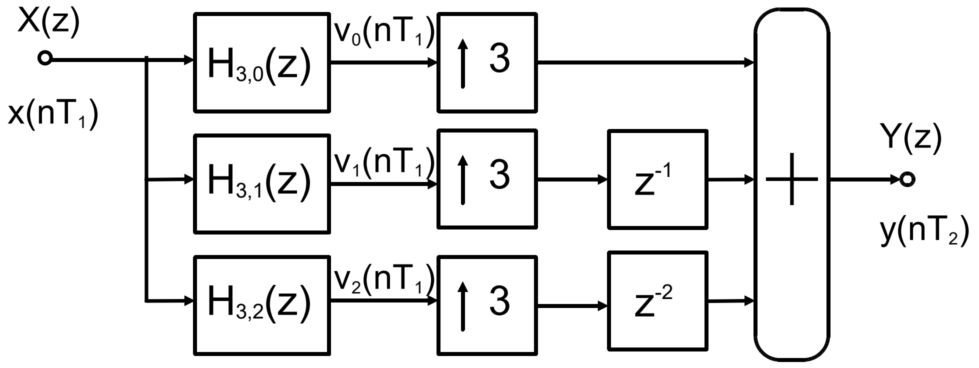
Interpolation by 3: efficient structure [4]



- **Some important remarks for interpolation**
  - **Interpolated samples of $v_0[nT_1]$ are not delayed** $\rightarrow$ **[y_1,0,0]**
  - **Interpolated samples of $v_1[nT_1]$ are delayed by T_2** $\rightarrow$ **[0, y_2,0]**
  - **Interpolated samples of $v_2[nT_1]$ are delayed by 2* T_2** $\rightarrow$ **[0,0, y_3]**
  - **These sequences are added,**
    - $\rightarrow$ **[y_1, 0, 0] + [0, y_2, 0] + [0, 0, y_3]=[y_1, y_2, y_3]**

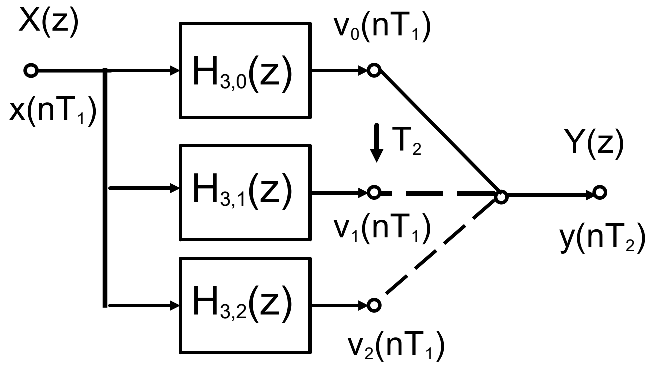
Interpolation by 3: efficient structure [5]

Again the previous structure with $H_{3,0}$, $H_{3,1}$, $H_{3,2}$ in z



polyph_2_interpolation.drw
US 12.2.06

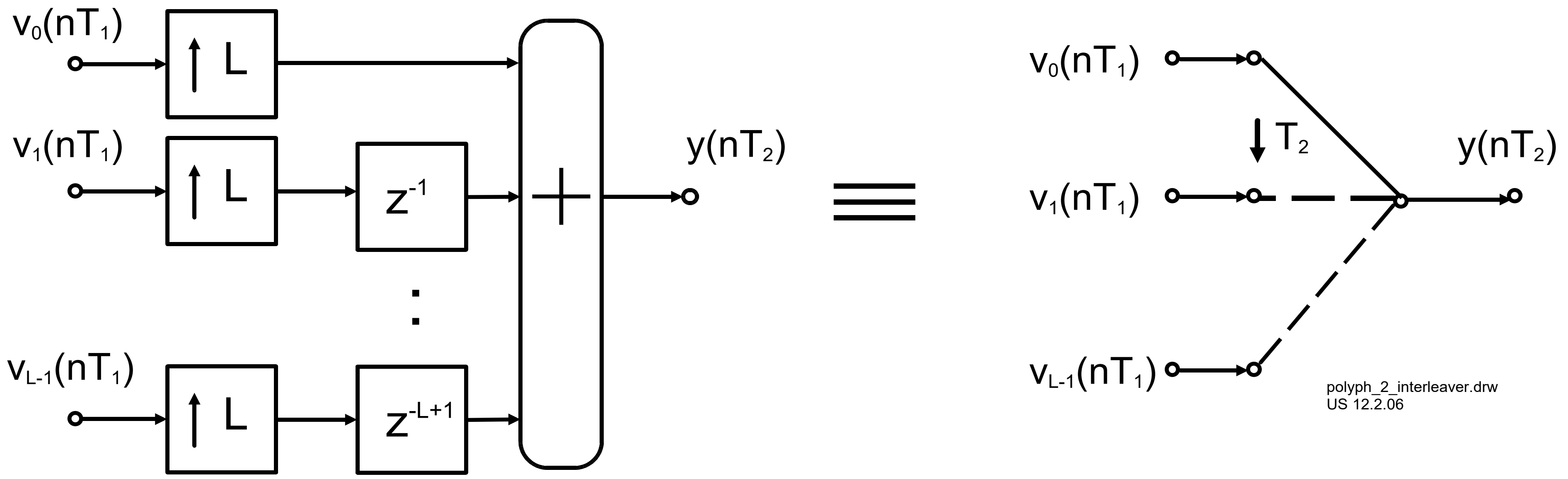
Using an <u>interleaver</u> instead, which rotates anti-clockwise with $1/T_2$



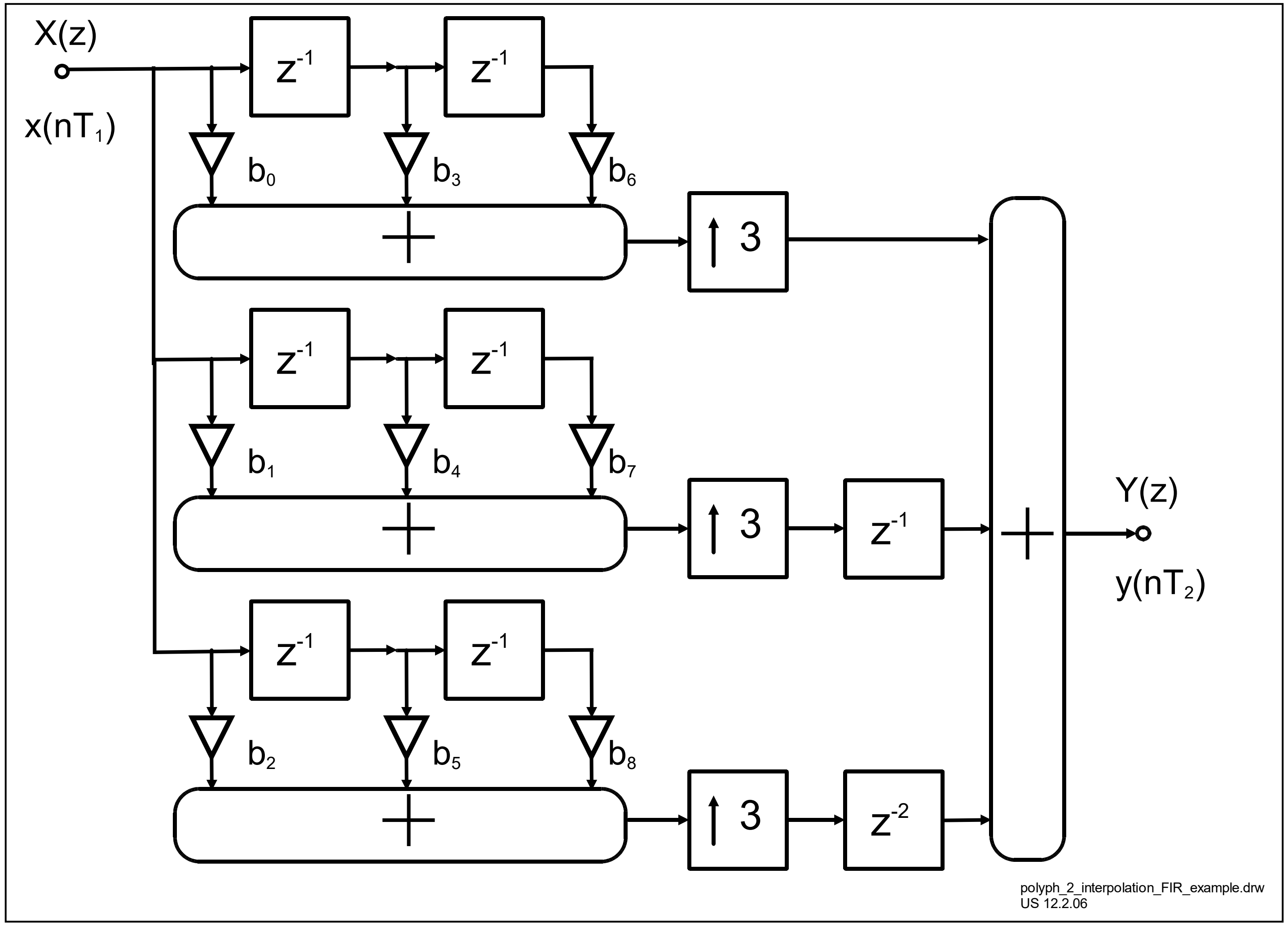
polyph_2_interpolation.drw
US 12.2.06

General interleaver (for general L)

The switch rotates anti-clockwise with $1/T_2$

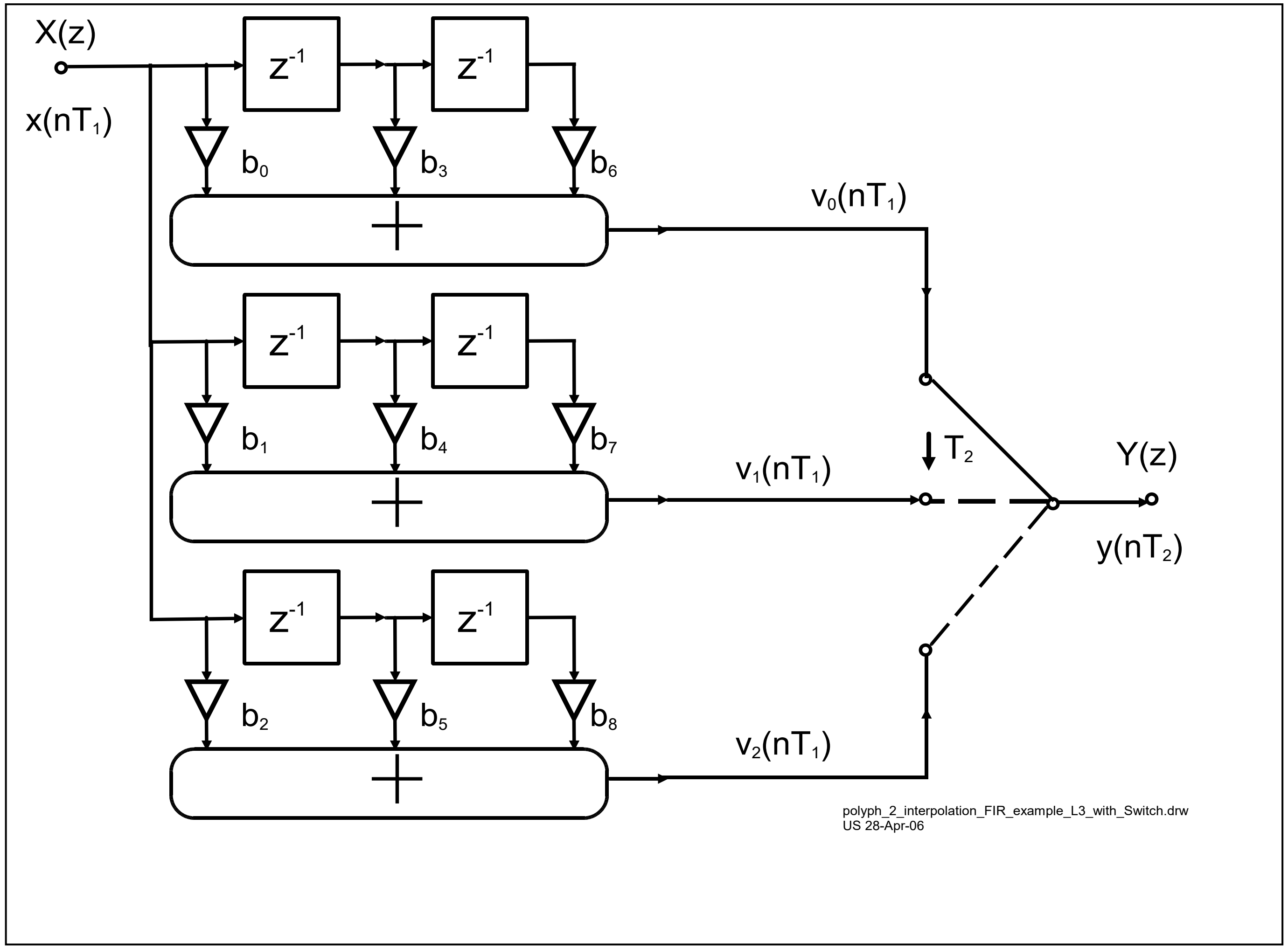


Interpolation by 3: final efficient structure for N=8 [6]



$$H_{3,0}(z) = b_0 + b_3 \cdot z^{-1} + b_6 \cdot z^{-2},$$
$$H_{3,1}(z) = b_1 + b_4 \cdot z^{-1} + b_7 \cdot z^{-2},$$
$$H_{3,2}(z) = b_2 + b_5 \cdot z^{-1} + b_8 \cdot z^{-2}$$

Interpolation by 3: final efficient structure for N=8 [7]



1) The three up-samplers by 3 and the delays have been replaced by a switch.
2) The switch rotates anti-clockwise with 1/T_2

$$H_{3,0}(z) = b_0 + b_3 \cdot z^{-1} + b_6 \cdot z^{-2},$$
$$H_{3,1}(z) = b_1 + b_4 \cdot z^{-1} + b_7 \cdot z^{-2},$$
$$H_{3,2}(z) = b_2 + b_5 \cdot z^{-1} + b_8 \cdot z^{-2}$$

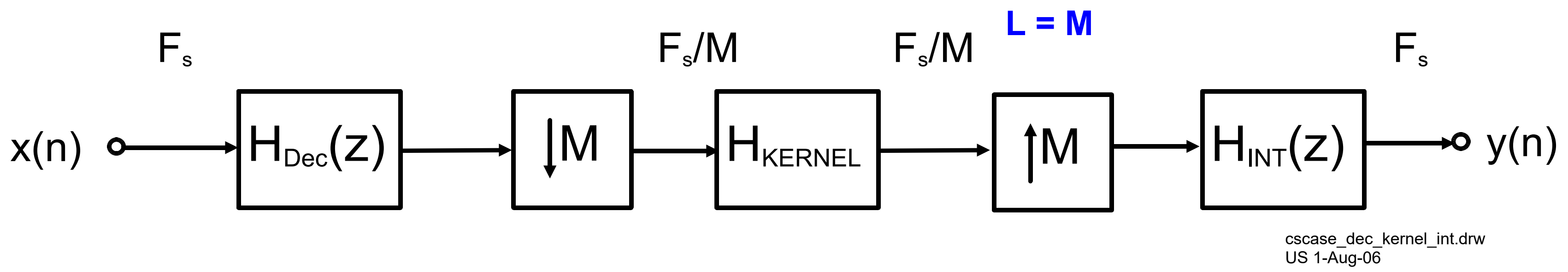
Interpolation by 3: b) MATLAB [8]

MATLAB implementation: Efficient interpolation by 3

```
% kernel from interpolate_by_3_FIR_1_stage.m


% the coefficients
b_filt_b0_b3 = b_fir_remez(1:LL:length(b_fir_remez));
b_filt_b1_b4 = b_fir_remez(2:LL:length(b_fir_remez));
b_filt_b2_b5 = b_fir_remez(3:LL:length(b_fir_remez));

% the delays
H_filt_30_delays = zeros(1,length(b_filt_b0_b3));
H_filt_31_delays = zeros(1,length(b_filt_b1_b4));
H_filt_32_delays = zeros(1,length(b_filt_b2_b5));

% filter x(n)
y=zeros(1,N_samp*LL);
mx=0;

for kx=1:N_samp
    mx=mx+1; % interpolate at kx
    H_filt_30_delays =
      [x(kx),H_filt_30_delays(1:length(H_filt_30_delays)-1)];
    y(mx) = b_filt_b0_b3 * H_filt_30_delays';

    mx=mx+1; % FIRST interpolated sample
    H_filt_31_delays =
      [x(kx),H_filt_31_delays(1:length(H_filt_31_delays)-1)];
    y(mx) = b_filt_b1_b4 * H_filt_31_delays';

    mx=mx+1; % SECOND interpolated sample
    H_filt_32_delays =
      [x(kx),H_filt_32_delays(1:length(H_filt_32_delays)-1)];
    y(mx) = b_filt_b2_b5 * H_filt_32_delays';

end; % for
```

Multi-rate filters

- **If the pass-band of a low-pass filter is rather small (<< Fs/4 or < Fs/4) and the transition band is small, the concept or multi-rate filtering may be applied.**



- **Using FIR polyphase representation, all filters can be operated at the lower sampling frequency Fs/M, as done before.**
- **Multirate filters can be implemented more efficiently than “normal” filters with this specification**
- **Note that the signal at the output of the kernel filter is usually not sent to a DAC. Thus, multi-rate only appears “inside” the filter (McBsp0 is sufficient, McBsp1 DAC for control purposes only).**

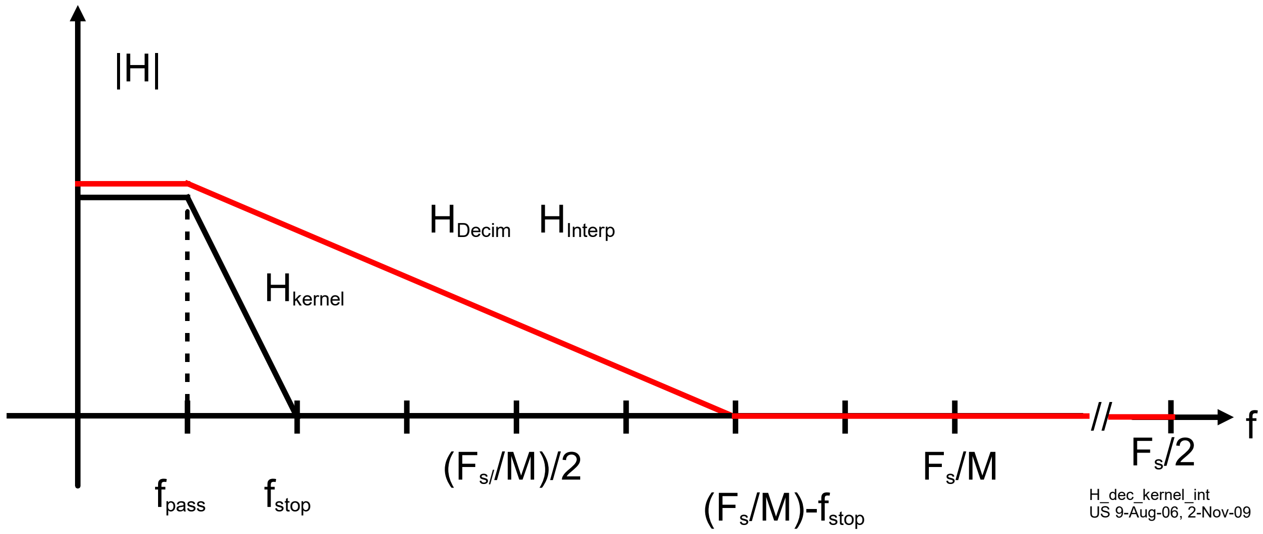
Multirate filters

Desired:

FIR filter, small passband 0..f_{pass}, small transition band f_{pass} .. f_{stop} , *wide* stop-band f_{stop}

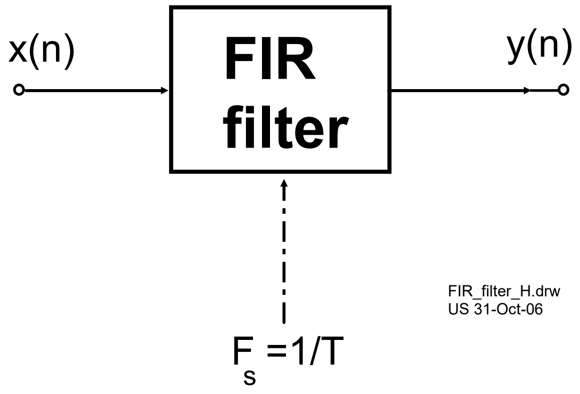
Instead:

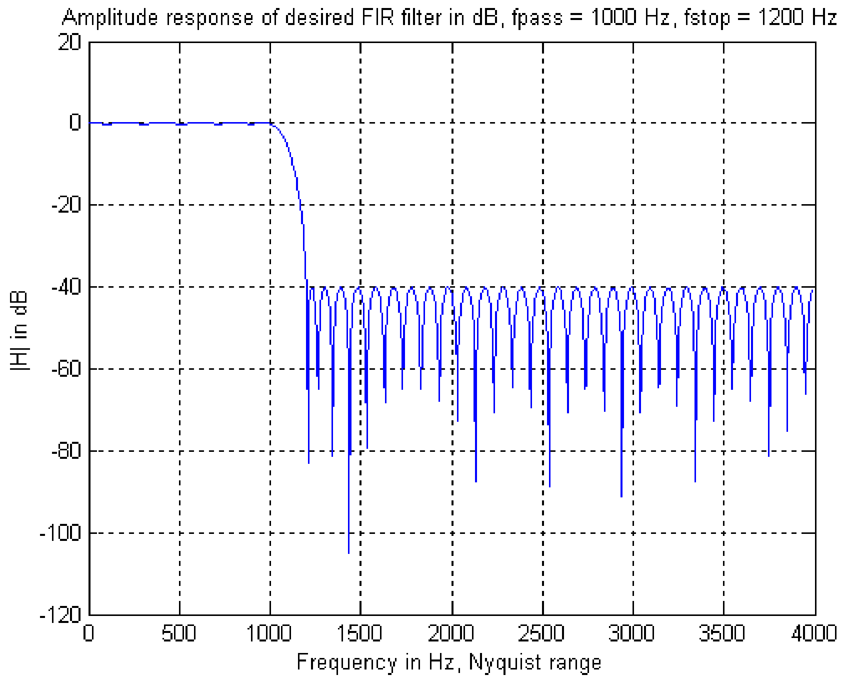
Use decimation, kernel and interpolation filter. No aliasing allowed in pass-band and transition band → specification to fulfill :



Multirate filters, example for M=2 [1]

For F_s = 8 kHz, it is desired to implement an FIR filter with transition band 1000 .. 1200 Hz.



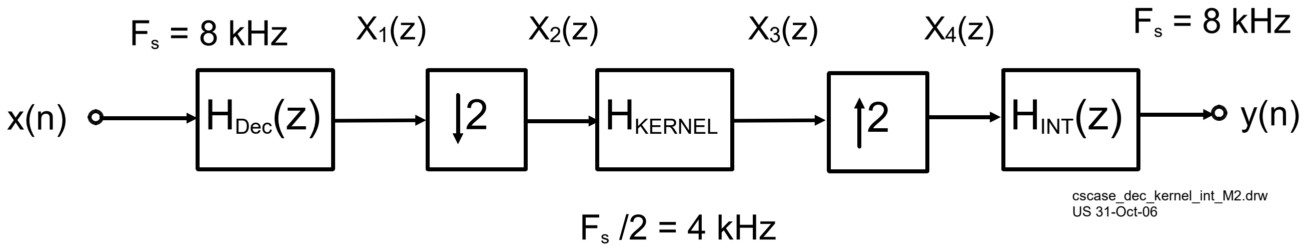


Since f_{pass} < Fs/4, a multi-rate filter is always advantageous compared to a "normal" FIR filter (requires less operations, without proof…)

`dec_kernel_int_M2_Bsp_Vorlesung.m`

Multirate filters, example for M=2 [2]

We therefore replace the <u>single</u> filter arrangement by 3 cascaded filters, a decimation filter, a kernel filter and an interpolation filter



We have to analyze the required amplitude responses of the three filters in order to design these filters

$$X_1(z) = X(z) \cdot H_{Dec}(z) \qquad X_2(z) = \frac{1}{2}\left[X_1(z^{\frac{1}{2}}) + X_1(-z^{\frac{1}{2}})\right] = \frac{1}{2}\left[X(z^{\frac{1}{2}}) \cdot H_{Dec}(z^{\frac{1}{2}}) + X(-z^{\frac{1}{2}}) \cdot H_{Dec}(-z^{\frac{1}{2}})\right]$$

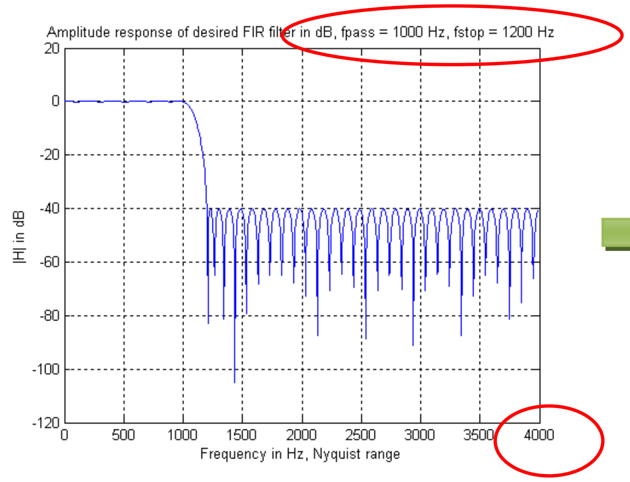
$$X_3(z) = \frac{1}{2}\left[X(z^{\frac{1}{2}}) \cdot H_{Dec}(z^{\frac{1}{2}}) \cdot H_{Kernel}(z) + X(-z^{\frac{1}{2}}) \cdot H_{Dec}(-z^{\frac{1}{2}}) \cdot H_{Kernel}(z)\right]$$

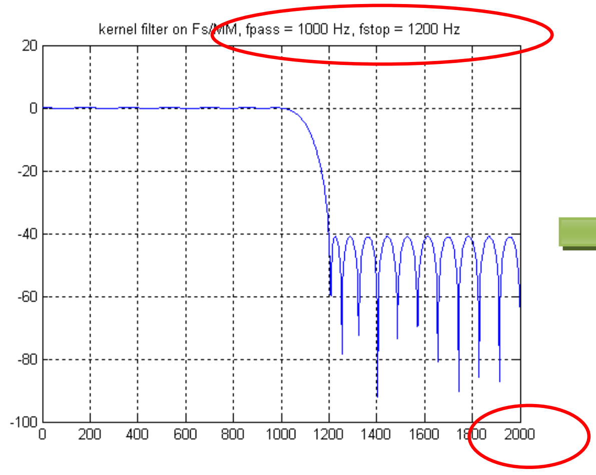
$$X_4(z) = X_3(z^2) = \frac{1}{2}\left[X(z) \cdot H_{Dec}(z) \cdot H_{Kernel}(z^2) + X(-z) \cdot H_{Dec}(-z) \cdot H_{Kernel}(z^2)\right]$$

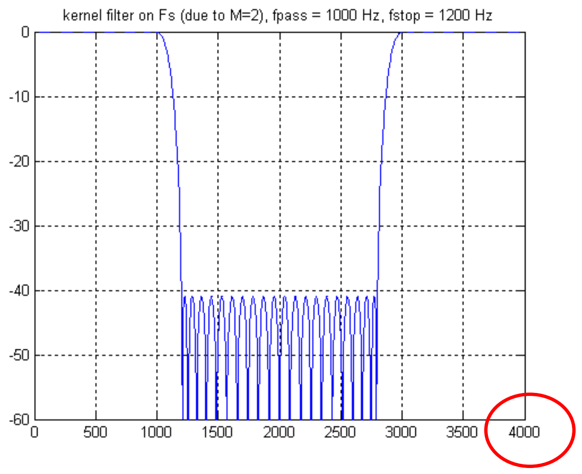
$$Y(z) = \frac{1}{2}\left[X(z) \cdot \boxed{H_{Dec}(z) \cdot H_{Kernel}(z^2) \cdot H_{Int}(z)} + X(-z) \cdot H_{Dec}(-z) \cdot H_{Kernel}(z^2) \cdot H_{Int}(z)\right]$$

Required !

Multirate filters, example for M=2 [3]

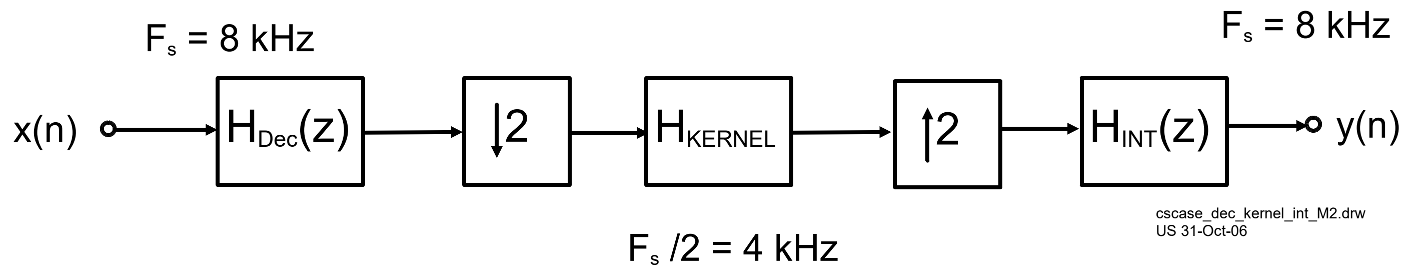




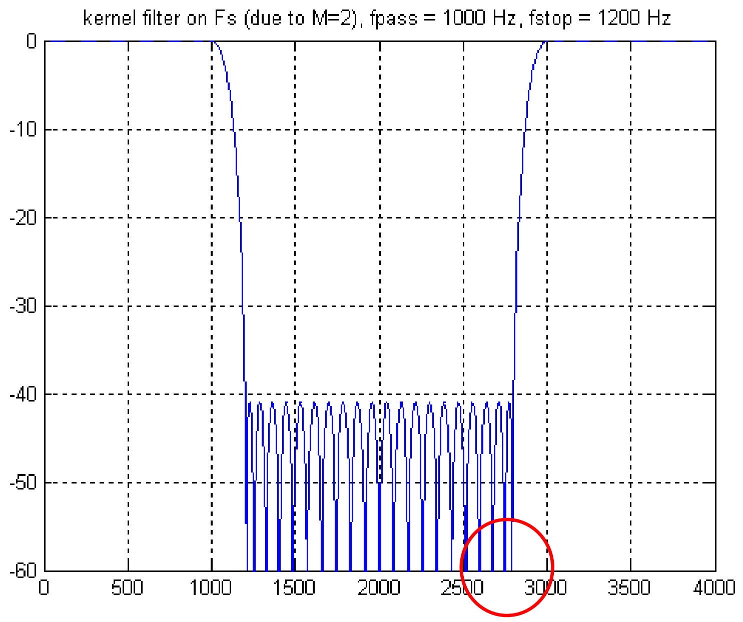


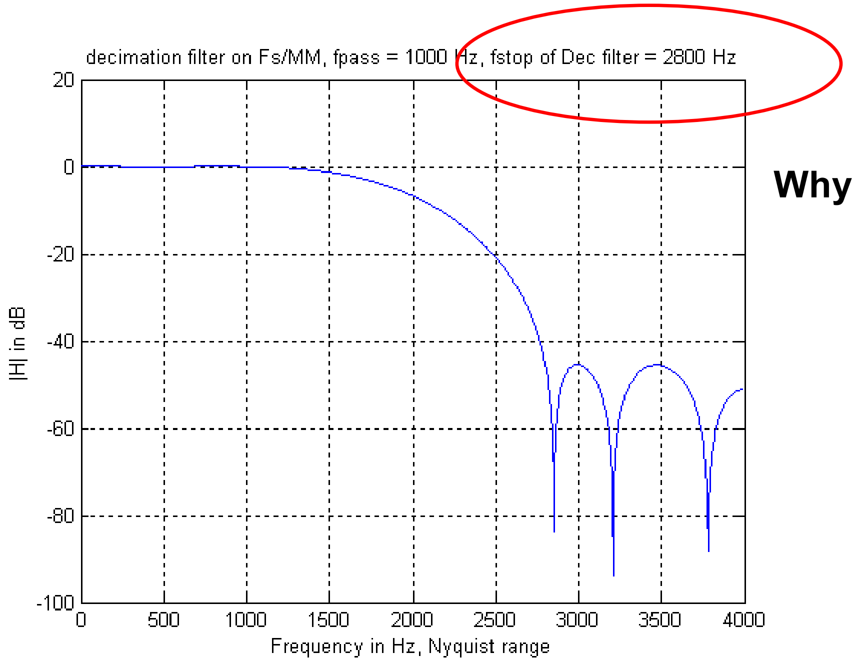
Desired FIR-Filter (F_S = **8** kHz) FIR-Filter $H_K(z)$ (F_S /2 = **4** kHz) FIR-Filter $H_K(z^2)$ ($\rightarrow$ F_S = **8** kHz)

(N = 80) **(N = 45)**

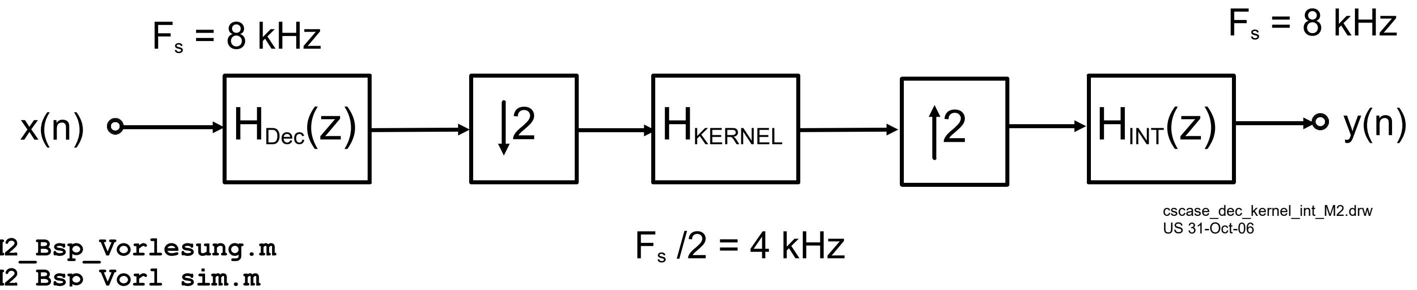


Multirate filters, example for M=2 [4]





Why 2800 Hz ??



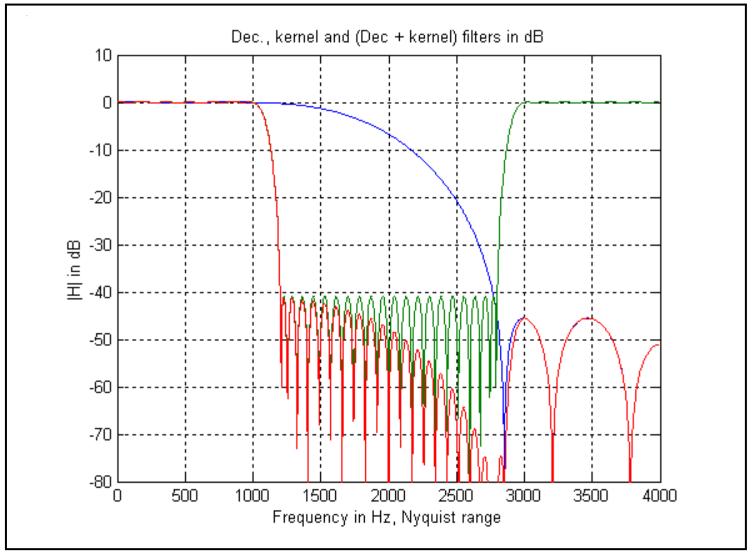
dec_kernel_int_M2_Bsp_Vorlesung.m
dec_kernel_int_M2_Bsp_Vorl_sim.m

Multirate filters, example for M=2 [5]

$H_{Dec}(z)$ ----

$H_{Kernel}(z^2)$ ----

$H_{Dec}(z) \cdot H_{Kernel}(z^2)$ ----

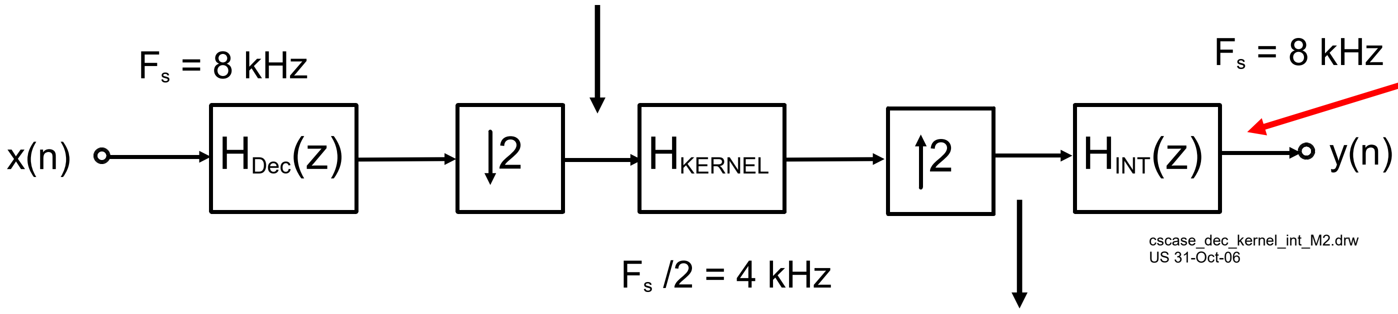


The <u>decimation</u> filter removes aliasing !!

The <u>kernel</u> filter does actual filtering on Fs/M !!

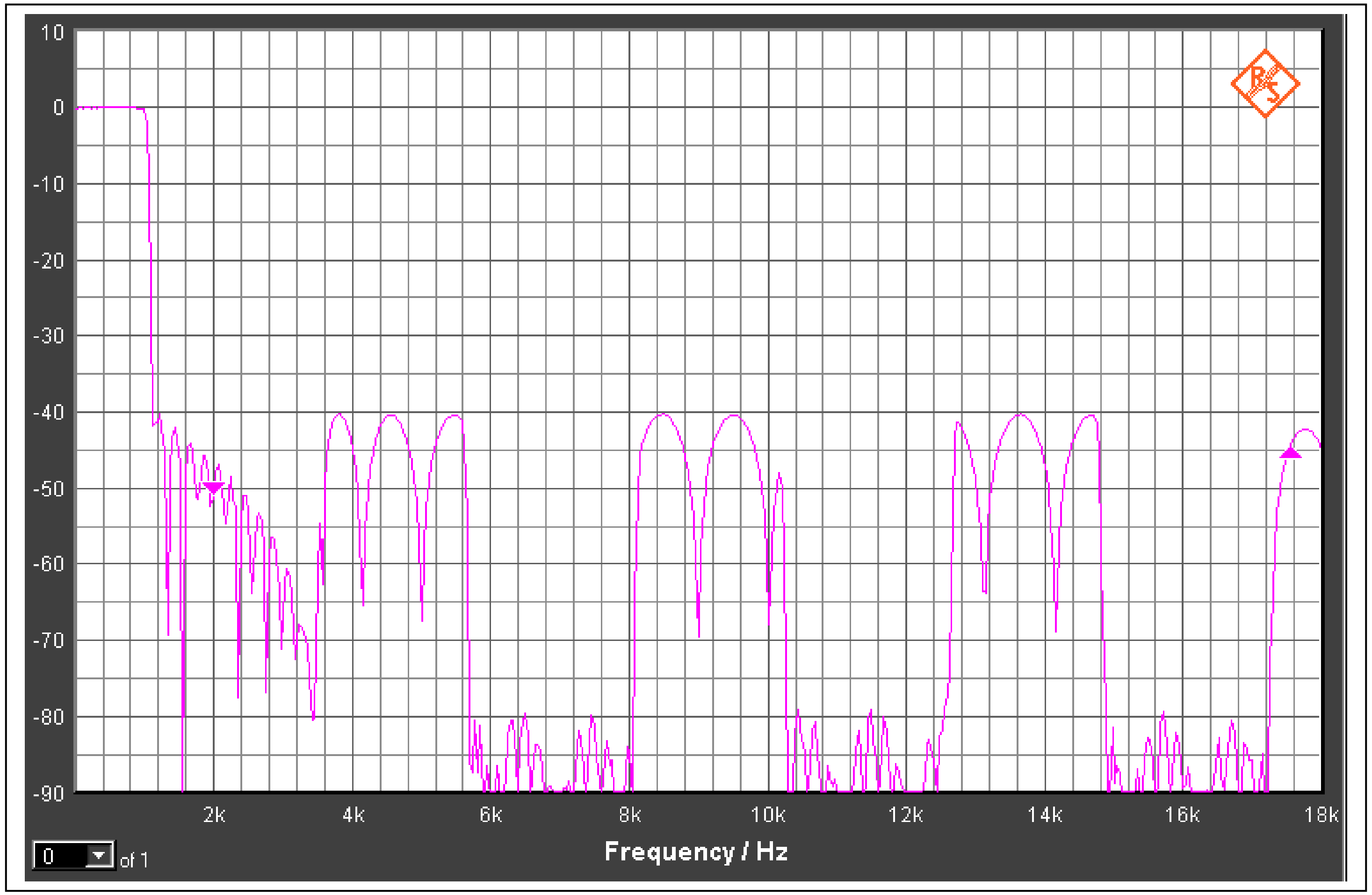
The <u>interpolation</u> filter removes the images

$$X_2(z) = 1/2 \cdot [X(z^{1/2}) \cdot H_{Dec}(z^{1/2}) + X(-z^{1/2}) \cdot H_{Dec}(-z^{1/2})]$$



$$X_4(z)=1/2 \cdot [X(z) \cdot H_{Dec}(z) \cdot H_K(z^2) + X(-z) \cdot H_{Dec}(-z) \cdot H_K(z^2)]$$

Multirate filters, example [6]

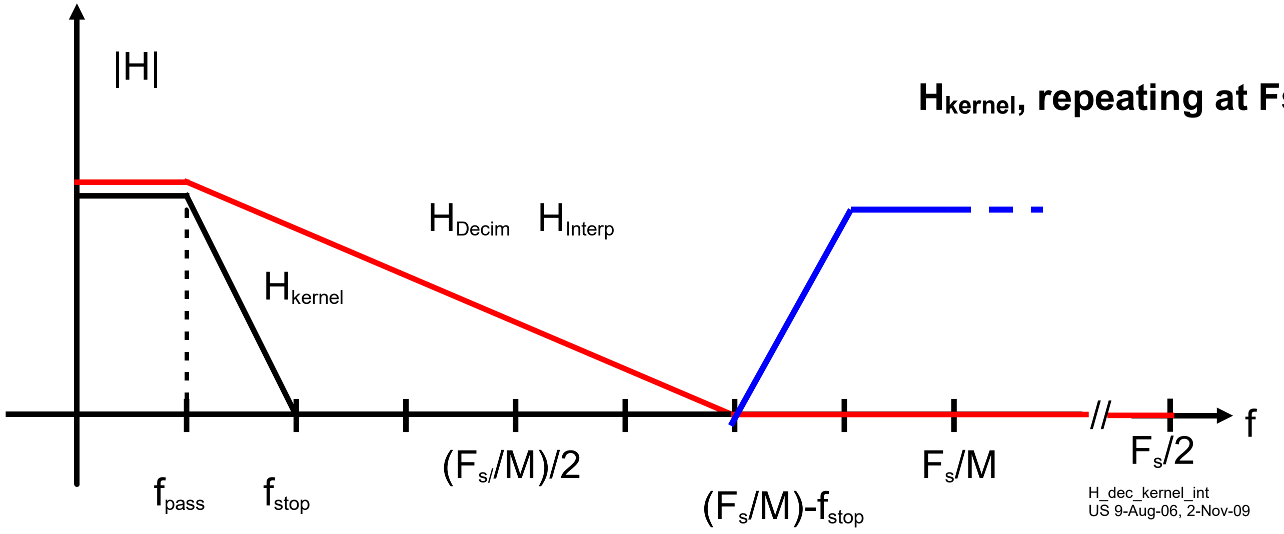


Lab 3+4:
Dec_kernel_int

Multirate filters, again :

Normalized transition band of Dec/Int filters is: $[(F_s/M-f_{stop}) - f_{pass}]/F_s$

All three filters, (decimation, kernel and interpolation filter) operate at F_s/M, i.e. at the lower sampling frequency !!



Multi-rate filters: Computational effort [1]

If b is the (normalized) bandwidth of the transition band, N_{FIR} is :

$$N_{FIR} \approx \frac{2}{3}\cdot\log_{10}\left(\frac{1}{10\cdot\delta_{pass}\cdot\delta_{stop}}\right)\left(\frac{1}{b}\right)$$

$$b_{Dec,Int} = \left(\frac{(F_s/M - f_{stop}) - f_{pass}}{F_s}\right)$$

$$b_{\ker nel} = \left(\frac{f_{stop} - f_{pass}}{(F_s/M)}\right)$$

The effort "E" is equal to the "N_{FIR} * sampling frequency".
Since all filters operate at Fs/M and the ripple in the pass-band is equally distributed across the three filters $\delta_{pass}/3$:

$$\mathit{Effort\ \ for}\ \ H_{Dec}, H_{Int} \approx \frac{2}{3}\cdot\log_{10}\left(\frac{3}{10\cdot\delta_{pass}\cdot\delta_{stop}}\right)\left(\frac{F_s}{(F_s/M - f_{stop}) - f_{pass}}\right)\cdot\left(\frac{F_s}{M}\right)$$

$$\mathit{Effort\ \ for}\ \ H_{\ker nel} \approx \frac{2}{3}\cdot\log_{10}\left(\frac{3}{10\cdot\delta_{pass}\cdot\delta_{stop}}\right)\left(\frac{F_s/M}{f_{stop} - f_{pass}}\right)\cdot\left(\frac{F_s}{M}\right)$$

$$\mathit{Total\ effort\ is}\ \ E_{ges} \approx \frac{2}{3}\cdot\log_{10}\left(\frac{3}{10\cdot\delta_{pass}\cdot\delta_{stop}}\right)\left(\frac{2\cdot F_s}{F_s/M - f_{pass} - f_{stop}} + \frac{F_s/M}{f_{stop} - f_{pass}}\right)\cdot\left(\frac{F_s}{M}\right)$$

Multi-rate filters: Computational effort [2]

Rewrite equation

$$Total\ effort\ E_{ges} \approx \frac{2}{3}\cdot\log_{10}\left(\frac{3}{10\cdot\delta_{pass}\cdot\delta_{stop}}\right)\left(\frac{2\cdot F_s}{F_s - M\cdot(f_{pass}+f_{stop})} + \frac{F_s}{M^2(f_{stop}-f_{pass})}\right)\cdot F_s$$

Find $E_{ges,min}$ by taking 1st derivative, set to 0, get M_{min} via MATLAB roots():

$$M^3{}_{\min}(f_{stop}{}^2 - f_{pass}{}^2) - M^2{}_{\min}(f_{pass}+f_{stop})^2 + M_{\min}\cdot 2\cdot F_s(f_{pass}+f_{stop}) - F_s{}^2 = 0$$

The multi-rate filters can be cascaded, such that the kernel filter requires less and less effort:
K decimation filters, followed by the kernel filter, followed by K interpolation filters, thus : multi-stage decimation/interpolation.

Multirate filters: Computational effort [3]

Example: Let F_s=36621.093750 Hz

Filter specification:

δ_{pass} =0.01,
δ_{stop} = 0.01, thus 40 dB stop-band attenuation,
f_{pass} = 1000 Hz
f_{stop} =1200 Hz
we obtain for a “normal” FIR implementation N_{FIR} = 356

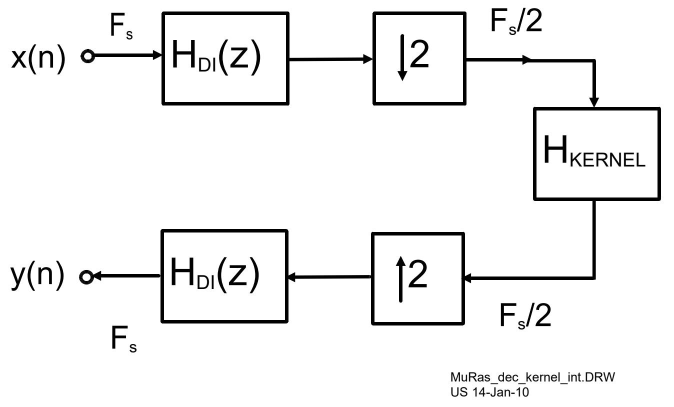
If we use instead of M_{min} = 8.76 (determined via MATLAB from equation above) the integer value M_{min}=8, we obtain:

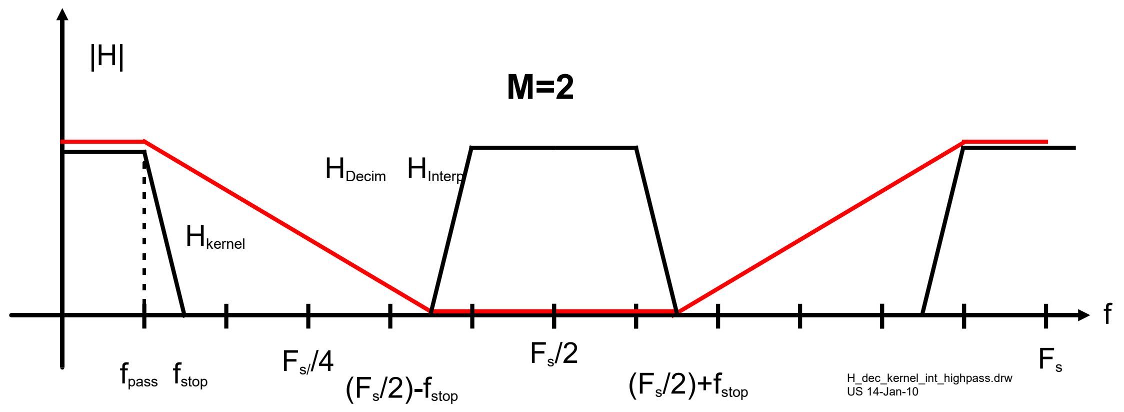
- **1) ”normal” N_{FIR}, : FOPS = 13037109.38,**
- **2) with two stages : FOPS = 569752.13,**
- **overhead of ”normal” FIR-filter is 22.88 (!!)**
- **$N_{FIR,kernel}$=52, N_{FIR,Dec_Int} = 35, $f_{stop,Dec,Int}$ = 3377.64 Hz**

`dec_kernel_int_M8_Bsp_Vorlesung.m`

Multirate Lowpass Filters

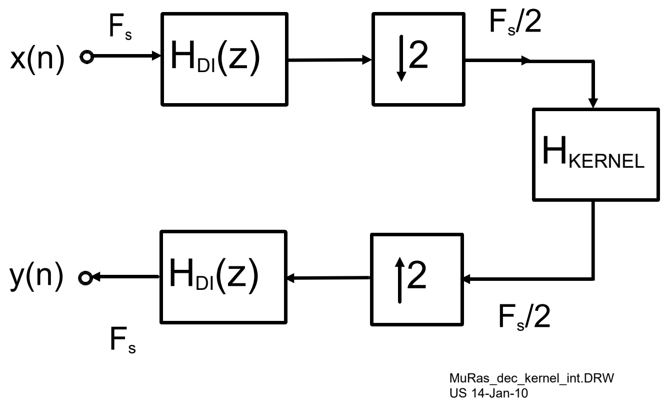
Usually, two identical (decimation and interpolation) low-pass filters and a low-pass kernel filter are used for the decimation and interpolation process.

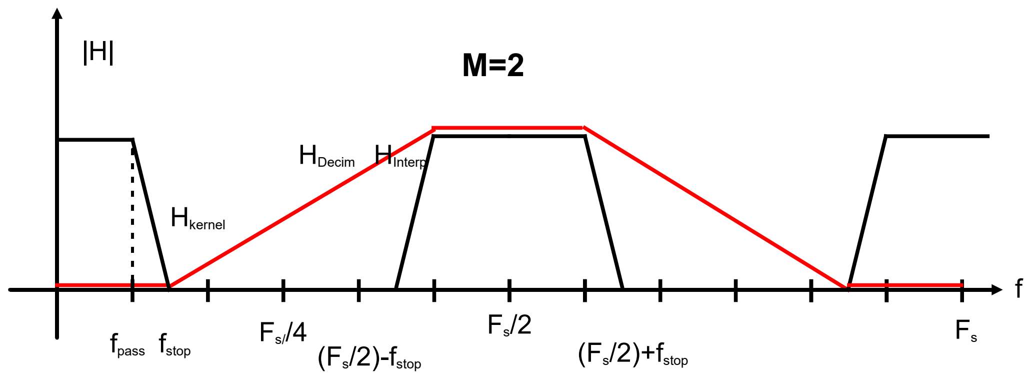




Multirate Highpass Filters

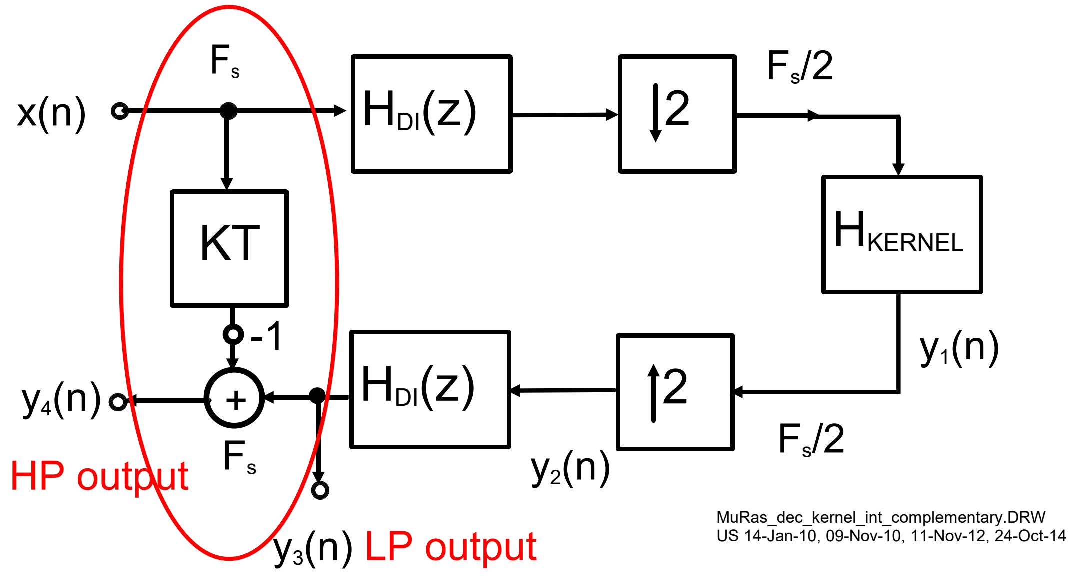
If, instead, a high-pass filter for the decimation and interpolation filters and a low-pass kernel filter is used, a high-pass filter is obtained.





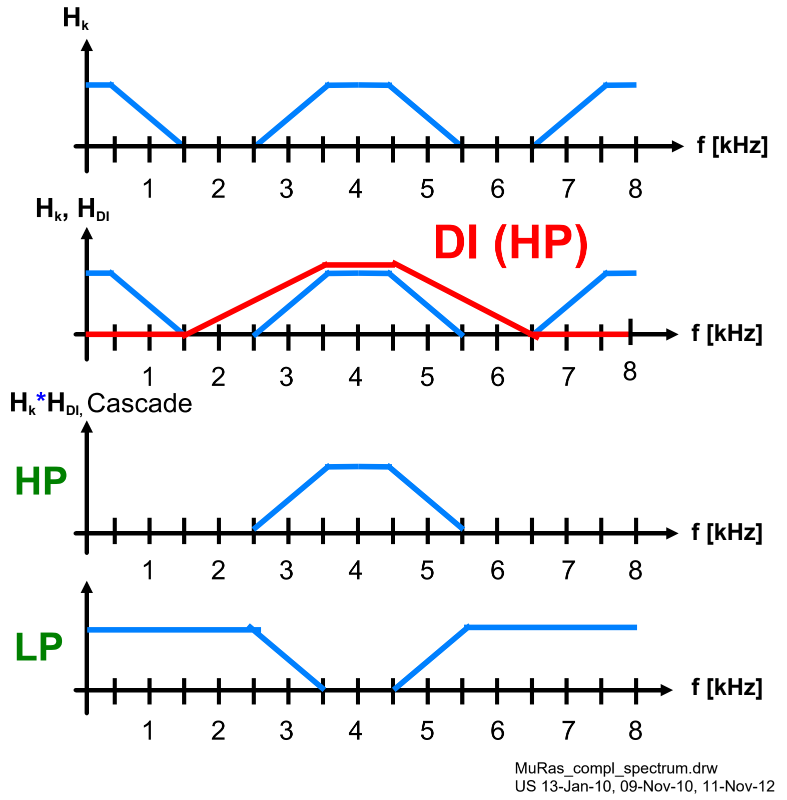
Multirate Complementary Low-pass/High-pass Filters [1]

If the output of the highpass/lowpass filter is subtracted from a cascade of delays which compensates the delay of the filter, a lowpass/highpass branching filter is obtained. Note : $y_4(n)$ is a high-pass filtered output signal, the output of the lower interpolation filter $H_{DI}(z)$ is a low-pass filtered output $y_3(n)$, see next page. Assume F_s= 8 kHz. H_{Kernel} filter operates on F_s= 4 kHz.



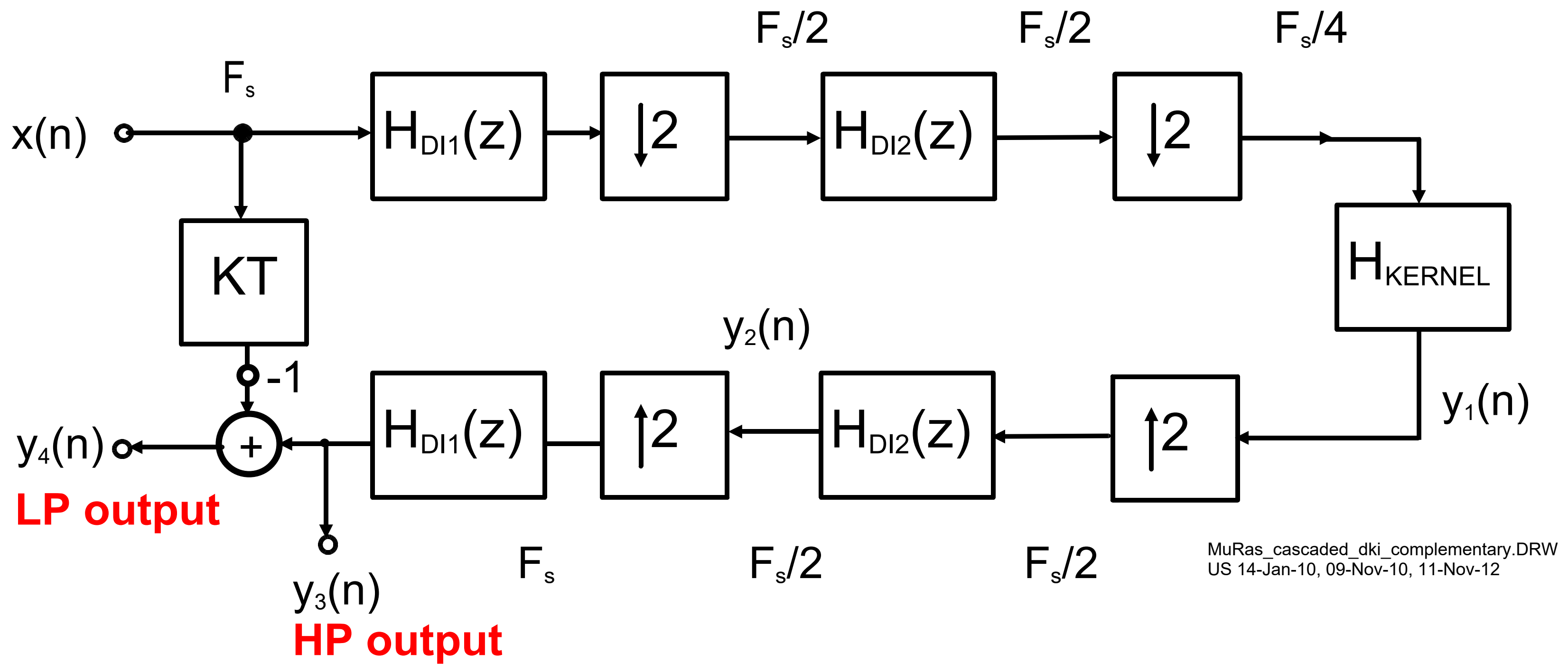
Multirate Complementary Low-pass/High-pass Filters [2]

If the output of the highpass/lowpass filter is subtracted from a delay which compensates the delay of the filter, a lowpass/highpass branching filter is obtained. Assume F_S= 8 kHz. H_{Kernel} filter operates on F_S= 4 kHz.



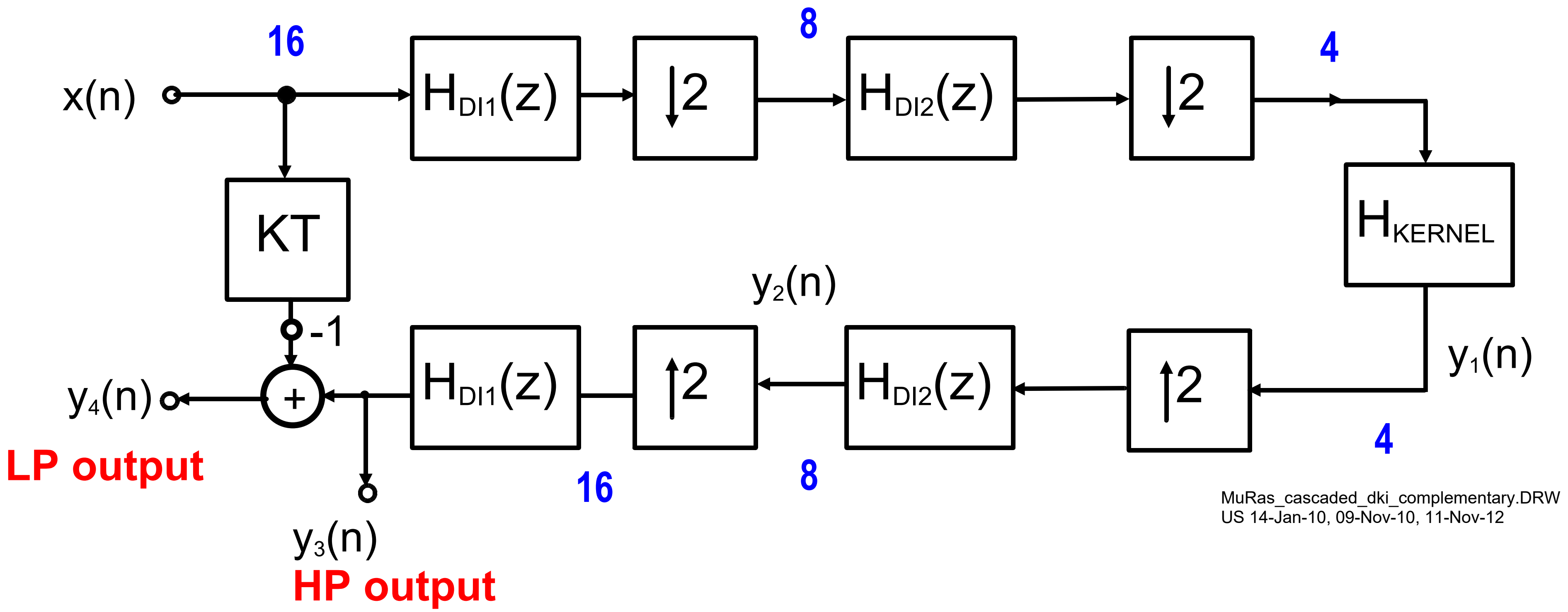
Multirate Cascaded Complementary Filters

If the several decimation filters (highpass/lowpass) are cascaded and the filter output(s) are subtracted from a delay chain which compensates the delay of the filter, a lowpass/highpass branching filter is obtained.
H_{Kernel} filter operates on $F_s/4$, DI2 operates on $F_s/2$ and DI1 operates on F_s !



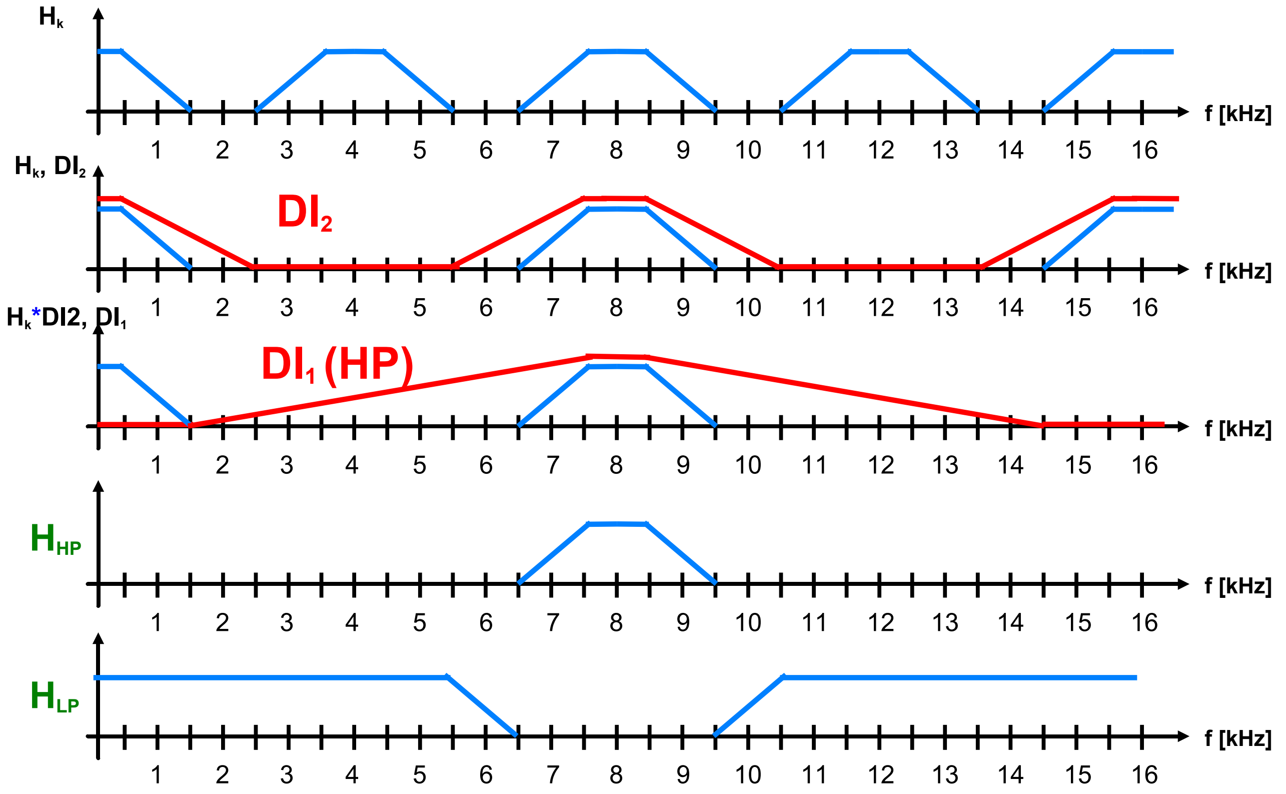
Multirate Cascaded Complementary Filters

Assume that H_{Kernel} and DI2 are lowpass filters and DI1 is a high-pass filter
Assume F_s= 16 kHz. H_{Kernel} filter operates on F_s= 4 kHz.



Multirate Cascaded Complementary Filters

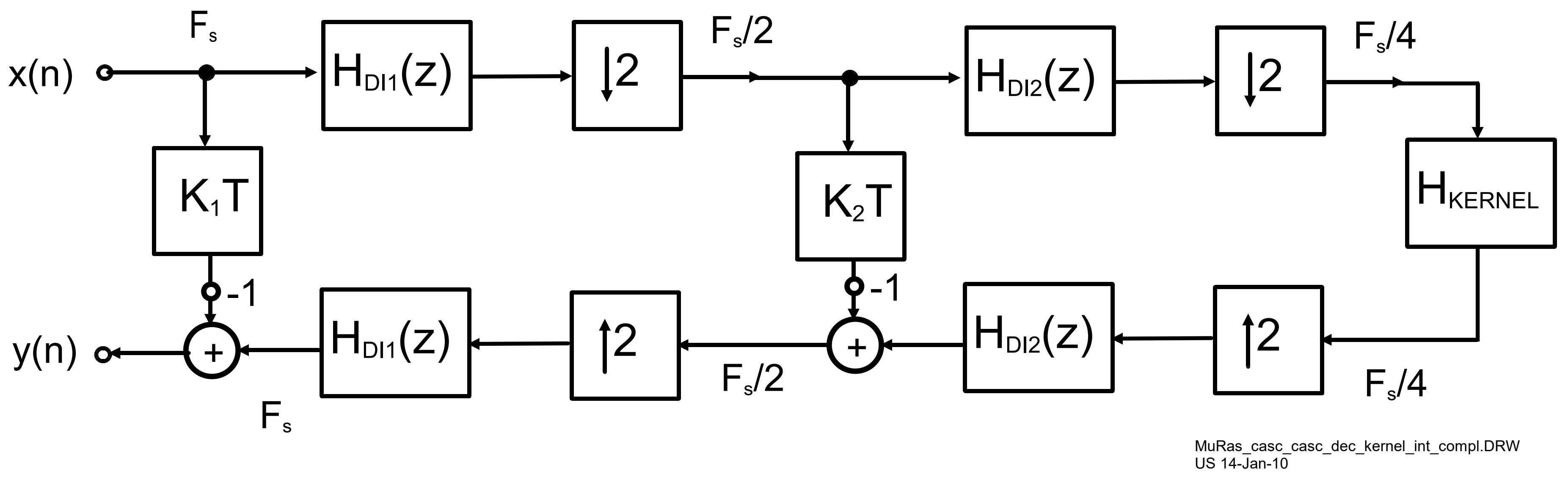
Assume that H_{Kernel} and DI2 are lowpass filters and DI1 is a high-pass filter
Assume F_s= 16 kHz. H_{Kernel} filter operates on F_s= 4 kHz.



MuRas_cascaded_compl_spectrum.drw
US 13-Jan-10, 09-Nov-10, 11-Nov-12

Multirate Cascaded Complementary Filters

This process can be repeated as often as desired. This way, arbitrary filter curves with steep flanks can be obtained.
Depending upon the edge frequencies of the kernel filter, the complementary filter output has to be used (or not).


x(n)
F_s
$H_{DI1}(z)$
↓2
$F_s/2$
$H_{DI2}(z)$
↓2
$F_s/4$
K_1T
K_2T
H_{KERNEL}
-1
-1
y(n)
+
$H_{DI1}(z)$
↑2
$F_s/2$
+
$H_{DI2}(z)$
↑2
$F_s/4$
F_s
MuRas_casc_casc_dec_kernel_int_compl.DRW
US 14-Jan-10