Polyphase implementation

November 29th, 2017

Oltan Doci

History

Date	Draft	Author	Comment
November 29th, 2017	0	O.D.	Creation

Classical processing

- In this presentation we will focus on the polyphase implementation of FIR decimator filters.

- It’s very important to understand that polypase concept is **equivalent** to the classical principle of FIR decimator, it is an efficient way to save MIPS so to increase throughput for a FPGA implementation for ex where we can use parallel structure.

- In theory a classical FIR decimator is done following 2 classical steps:

  - FIR filtering
  - Decimation

- We see here that we filter all input samples and et the end we will keep only the decimated filtered samples.

- By doing this we overload the processor MIPS by computing unused samples so we have a poor throughput.

Classical processing

- We are going to reduce the processor burden thanks to the decimation factor.
- For a better understanding of the polyphase principle we are going to use the same example on this presentation.
- Consider that we have to filter with a N = 16 taps FIR and then to decimate by a factor M = 8 a buffer of 32 samples.
- FIR coefficients:

$$h_0\ h_1\ h_2\ h_3\ h_4\ h_5\ h_6\ h_7\ h_8\ h_9\ h_{10}\ h_{11}\ h_{12}\ h_{13}\ h_{14}\ h_{15}$$

- Input samples (coming from an ADC for ex):

$$x_0\ x_1\ x_2\ x_3\ x_4\ x_5\ x_6\ x_7\ x_8\ x_9\ x_{10}\ x_{11}\ x_{12}\ x_{13}\ x_{14}\ x_{15}\ x_{16}\ x_{17}\ x_{18}\ x_{19}\ x_{20}\ x_{21}\ x_{22}\ x_{23}\ x_{24}\ x_{25}\ x_{26}\ x_{27}\ x_{28}\ x_{29}\ x_{30}\ x_{31}$$

- The filtering computing will be:

$$y(n) = \sum_{k=0}^{N-1} h(k)x(n-k)$$

FIR decimator: direct form

- The direct form of implementation consists on filtering each sample and then decimate:

  -FIR filtering:

$$y(0) = h(0)x(0) + h(1)x(-1) + h(2)x(-2) + h(3)x(-3) + h(4)x(-4) + h(5)x(-5) + ... + h(15)x(-15)$$
$$y(1) = h(0)x(1) + h(1)x(0) + h(2)x(-1) + h(3)x(-2) + h(4)x(-3) + h(5)x(-4) + ... + h(15)x(-14)$$
$$y(2) = h(0)x(2) + h(1)x(1) + h(2)x(0) + h(3)x(-1) + h(4)x(-2) + h(5)x(-3) + ... + h(15)x(-13)$$
$$...$$
$$y(8) = h(0)x(8) + h(1)x(7) + h(2)x(6) + h(3)x(5) + h(4)x(4) + h(5)x(3) + ... + h(15)x(-7)$$
$$...$$
$$y(16) = h(0)x(16) + h(1)x(15) + h(2)x(14) + h(3)x(13) + h(4)x(12) + h(5)x(11) + ... + h(15)x(1)$$
$$...$$
$$y(24) = h(0)x(24) + h(1)x(23) + h(2)x(22) + h(3)x(21) + h(4)x(20) + h(5)x(19) + ... + h(15)x(9)$$
$$...$$
$$y(31) = h(0)x(31) + h(1)x(30) + h(2)x(29) + h(3)x(28) + h(4)x(27) + h(5)x(26) + ... + h(15)x(16)$$

  - Decimation:

$$output: \ y_{dec}(0) = y(0), y_{dec}(1) = y(8), y_{dec}(2) = y(16), y_{dec}(3) = y(24)$$

FIR decimator: direct form

- We notice that at the end, after decimation by M, we keep only 4 filtered samples over 32

- So it means that we have computed enough filtered samples for nothing because they are unusual !

- We did 32 computing of N multiplications

- We are going to reduce MIPS by thinking a better way on how to compute FIR decimator

FIR decimator: efficient form

- To be more efficient, we are going to compute only the useful samples:

$$y_{dec}(0) = y(0) = h(0)x(0) + h(1)x(-1) + h(2)x(-2) + h(3)x(-3) + h(4)x(-4) + h(5)x(-5) + ... + h(15)x(-15)$$
$$y_{dec}(1) = y(8) = h(0)x(8) + h(1)x(7) + h(2)x(6) + h(3)x(5) + h(4)x(4) + h(5)x(3) + ... + h(15)x(-7)$$
$$y_{dec}(2) = y(16) = h(0)x(16) + h(1)x(15) + h(2)x(14) + h(3)x(13) + h(4)x(12) + h(5)x(11) + ... + h(15)x(1)$$
$$y_{dec}(3) = y(24) = h(0)x(24) + h(1)x(23) + h(2)x(22) + h(3)x(21) + h(4)x(20) + h(5)x(19) + ... + h(15)x(9)$$

- We see that now we did only 4 computing of N multiplications which is better than before

- Here we have directly the output samples decimated so no need to keep one over M samples

FIR decimator: efficient form

- Conclusion of this:

  - FIR filtering on the useful samples at Fs
  - Output samples at Fs/M

- In SW for ex we can easily implement the efficient form by incrementing the pointer offset of the buffer by M, that's all !

- At the beginning the delay line is initialized with 0 of course:

$$x(-1) = x(-2) = x(-3) = x(-4) = \ldots = x(-15) = 0$$

- For a better implementation the delay line of N – 1 values needs to be part of the buffer (at the beginning) and each time that a new buffer needs to be filtered.

- By this way the 1st sample starts at index N, from 0 to N-1 there is the previous delay line!

- The number of samples at the output is divided by M compared to the input!

FIR decimator: polyphase form

- If we have a parallel architecture for ex FPGA, we can implement the FIR decimator into a polyphase form.
- Let’s rewrite the previous equations in parallel:

$$\begin{aligned}
y_{dec}(0) = {} & h(0)x(0) + h(8)x(-8) \\
& + h(1)x(-1) + h(9)x(-9) \\
& + h(2)x(-2) + h(10)x(-10) \\
& + h(3)x(-3) + h(11)x(-11) \\
& + h(4)x(-4) + h(12)x(-12) \\
& + h(5)x(-5) + h(13)x(-13) \\
& + h(6)x(-6) + h(14)x(-14) \\
& + h(7)x(-7) + h(15)x(-15)
\end{aligned}
\qquad
\begin{aligned}
y_{dec}(1) = {} & h(0)x(8) + h(8)x(0) \\
& + h(1)x(7) + h(9)x(-1) \\
& + h(2)x(6) + h(10)x(-2) \\
& + h(3)x(5) + h(11)x(-3) \\
& + h(4)x(4) + h(12)x(-4) \\
& + h(5)x(3) + h(13)x(-5) \\
& + h(6)x(2) + h(14)x(-6) \\
& + h(7)x(1) + h(15)x(-7)
\end{aligned}$$

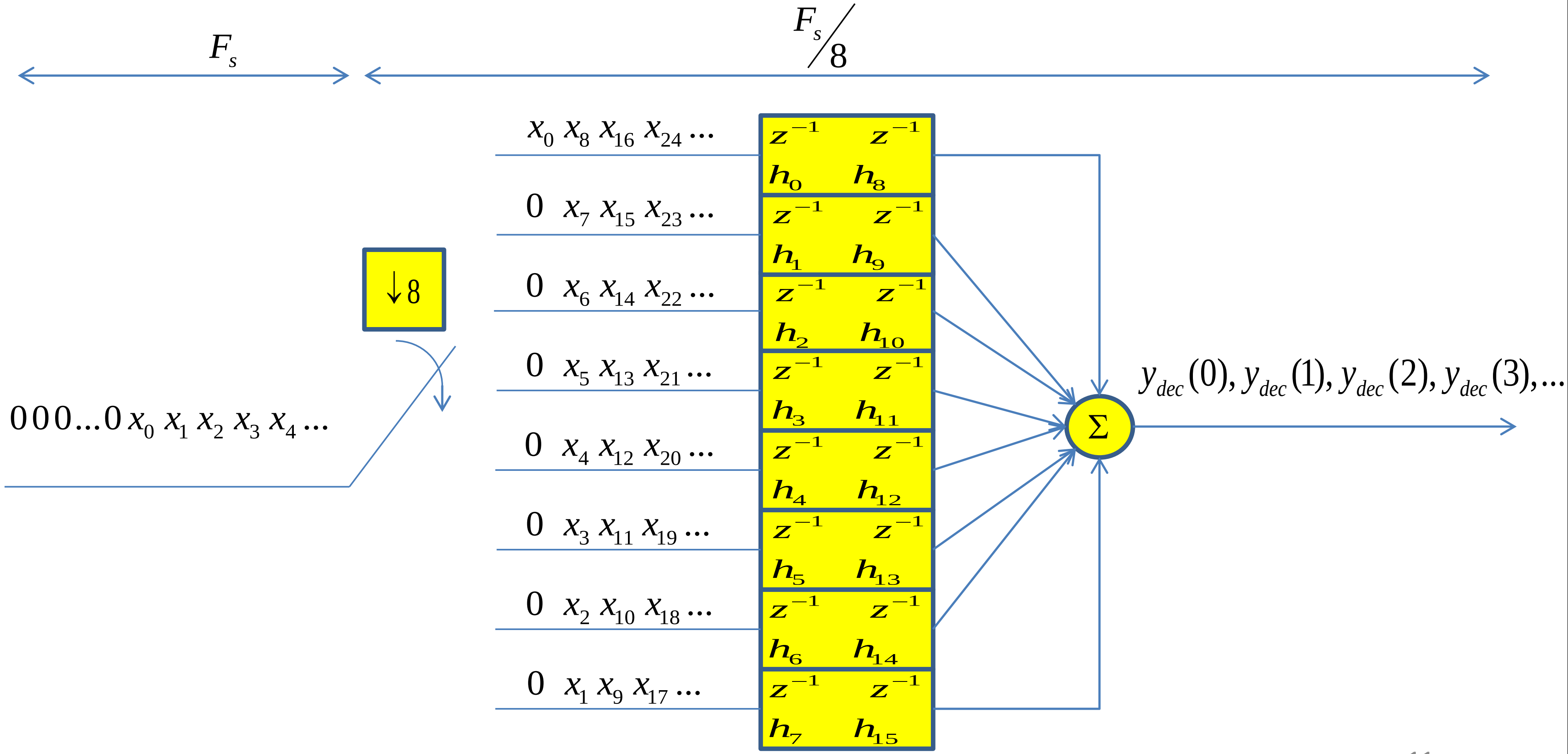
$$\begin{aligned}
y_{dec}(2) = {} & h(0)x(16) + h(8)x(8) \\
& + h(1)x(15) + h(9)x(7) \\
& + h(2)x(14) + h(10)x(6) \\
& + h(3)x(13) + h(11)x(5) \\
& + h(4)x(12) + h(12)x(4) \\
& + h(5)x(11) + h(13)x(3) \\
& + h(6)x(10) + h(14)x(2) \\
& + h(7)x(9) + h(15)x(1)
\end{aligned}
\qquad
\begin{aligned}
y_{dec}(3) = {} & h(0)x(24) + h(8)x(16) \\
& + h(1)x(23) + h(9)x(15) \\
& + h(2)x(22) + h(10)x(14) \\
& + h(3)x(21) + h(11)x(13) \\
& + h(4)x(20) + h(12)x(12) \\
& + h(5)x(19) + h(13)x(11) \\
& + h(6)x(18) + h(14)x(10) \\
& + h(7)x(17) + h(15)x(9)
\end{aligned}$$

FIR decimator: polyphase form

- We notice here that we have 8 parallel ‘small’ FIR structures called bancs.
- This is a simple representation of a matrix with M lines and K columns.
- M is the decimation factor and K is the number of coefficients of each banc.
- The prototype FIR from which are deduced the banks, has a number of coefficients N = M x K (for our ex N = 16, so for each banc K = N/M = 2)
- We can easily see that the values of M x K matrix are rearranged in the following way:

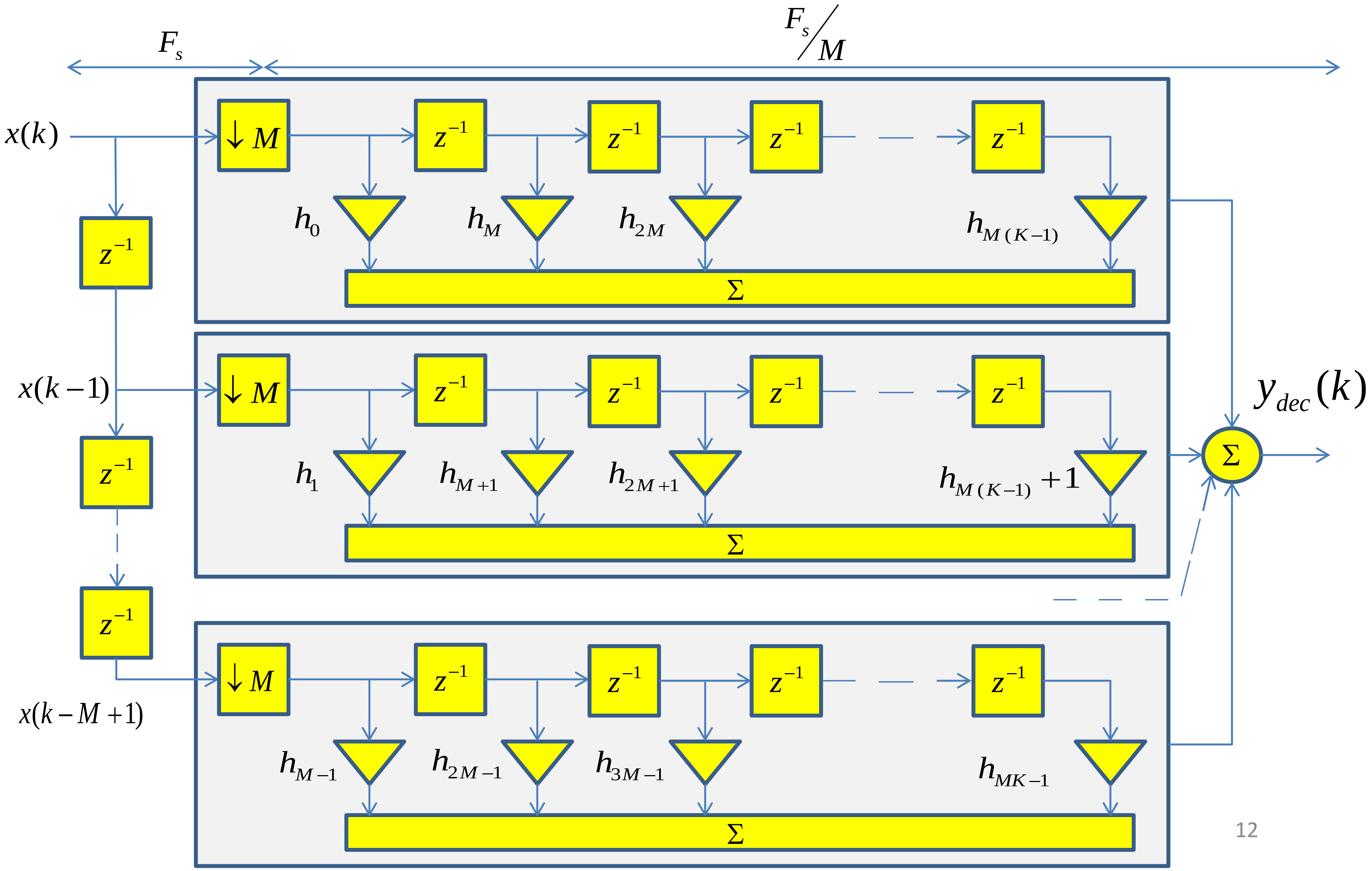
  - on each banc the taps are M multiples of the prototype FIR and are always the same
  - on each line the input samples are delayed by M and on each column they are delayed only by 1
- This is a very important information on a FPGA because of parallelism: each banc is computed in parallel and with a rhythm of Fs/M !
- Let’s represent this parallel computing with input and output sampling rates applied to our example and to the generic case

FIR decimator: polyphase form



F_s
$F_s/8$
$\downarrow 8$
$0\,0\,0 \ldots 0\; x_0\; x_1\; x_2\; x_3\; x_4 \ldots$
$x_0\; x_8\; x_{16}\; x_{24} \ldots$
$0\;\; x_7\; x_{15}\; x_{23} \ldots$
$0\;\; x_6\; x_{14}\; x_{22} \ldots$
$0\;\; x_5\; x_{13}\; x_{21} \ldots$
$0\;\; x_4\; x_{12}\; x_{20} \ldots$
$0\;\; x_3\; x_{11}\; x_{19} \ldots$
$0\;\; x_2\; x_{10}\; x_{18} \ldots$
$0\;\; x_1\; x_9\; x_{17} \ldots$
z^{-1} z^{-1} h_0 h_8
z^{-1} z^{-1} h_1 h_9
z^{-1} z^{-1} h_2 h_{10}
z^{-1} z^{-1} h_3 h_{11}
z^{-1} z^{-1} h_4 h_{12}
z^{-1} z^{-1} h_5 h_{13}
z^{-1} z^{-1} h_6 h_{14}
z^{-1} z^{-1} h_7 h_{15}
Σ
$y_{dec}(0), y_{dec}(1), y_{dec}(2), y_{dec}(3), \ldots$

FIR decimator: polyphase form



Multichannel

- Suppose now that we need to filter several channels where the number of channels is M, same as the decimation factor.

- Each channel is shifted in frequency by an intermediate frequency (IF) so it means that the M channels are in a wideband.

- In order to process this signal in baseband, we follow these 3 classical steps:

  - Baseband translation
  - FIR filtering
  - Decimation

- If we consider M channels in wideband, we are going to do use a lot of MIPS in order to do baseband processing because these 3 classical steps must be applied to each channel.

- It’s a huge computing and maybe cannot be realized with an FPGA if the number of FIR taps and/or the sampling frequency are too big!

- Now we will demonstrate that **these 3 classical steps are exactly equivalent to a polyphase followed by FFT implementation** inside a FPGA for ex and thanks to the polyphase principle coupled with DFT, we can do this computing in a very simple way!

Multichannel

- For baseband translation we multiply each sample by the complex exponential

- We are going to find another important information here if we choose a smart sampling frequency that is a multiple of the channel BW

- By this way we are going to set IF frequencies in the following way:

$$f_n = n \times \frac{F_s}{M}$$

- So:

$$e^{j2\pi k T_s f_n} = e^{j2\pi k \frac{f_n}{F_s}}$$

$$= e^{j2\pi k \frac{n \times \frac{F_s}{M}}{F_s}}$$

$$= e^{j2\pi \frac{kn}{M}}$$

Multichannel

- The beauty of this is that we already feel that DFT is coming soon and also we have the following periodicity:

$$\begin{aligned} e^{j2\pi\frac{(k+M)n}{M}} &= e^{j2\pi\frac{kn}{M}} \times e^{j2\pi\frac{Mn}{M}} \\ &= e^{j2\pi\frac{kn}{M}} \times e^{j2\pi n} \\ &= e^{j2\pi\frac{kn}{M}} \end{aligned}$$

- This is very interesting because if we apply this property to our example we have only M-1 exponentials (stored in a ROM for ex), so no need to implement a phase accumulator with different steps:

$$x_{bb}^{n}(k) = x_{wb}^{n}(k) \times e^{j2\pi\frac{kn}{M}},\ k = 0,1,2,\ldots,M-1$$

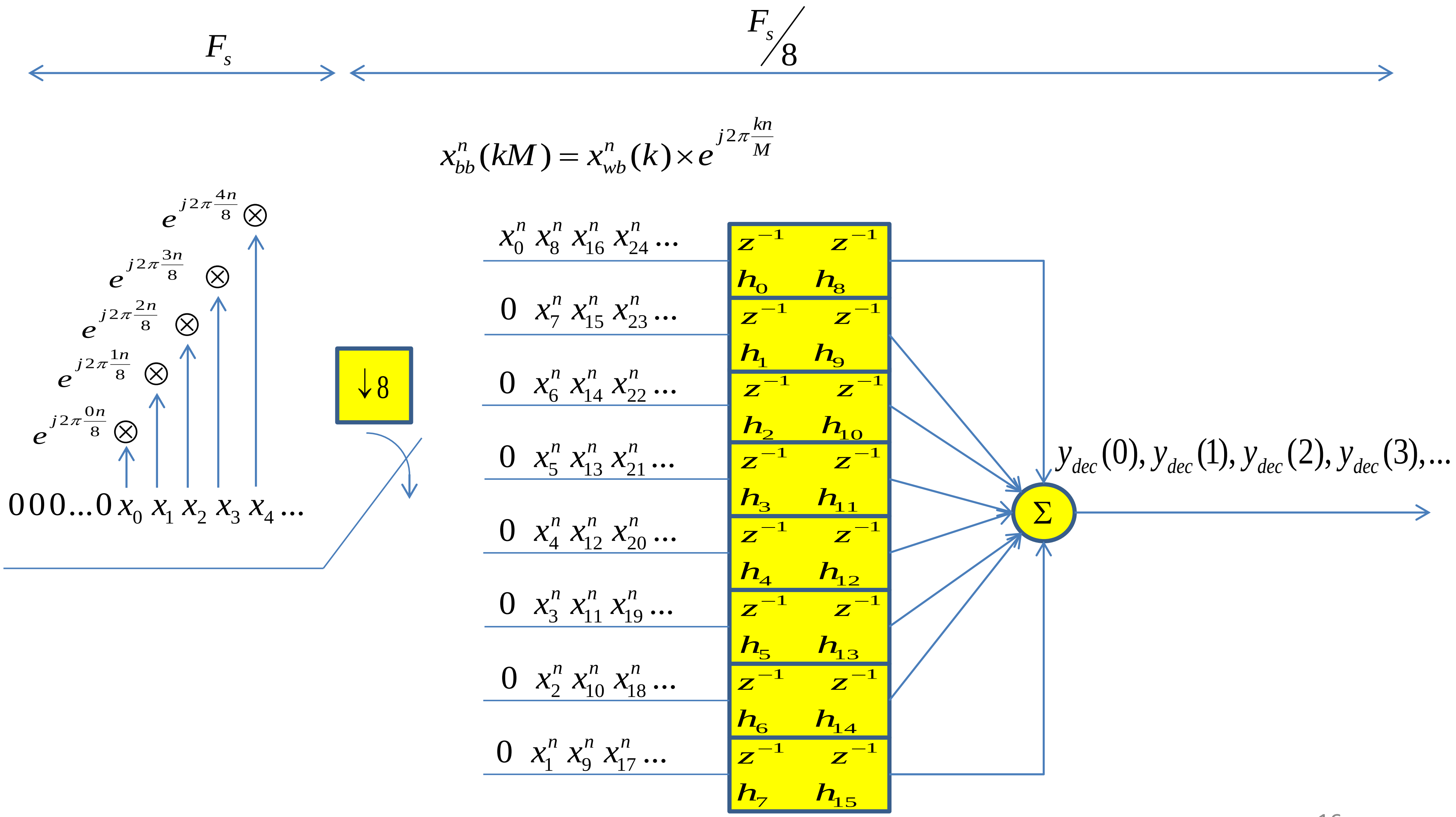
- So

$$x_{bb}^{n}(k+M) = x_{wb}^{n}(k+M) \times e^{j2\pi\frac{kn}{M}}$$

- So here we deduce directly that each input sample at each bank is multiplied by the same exponential because these samples are multiple of M!

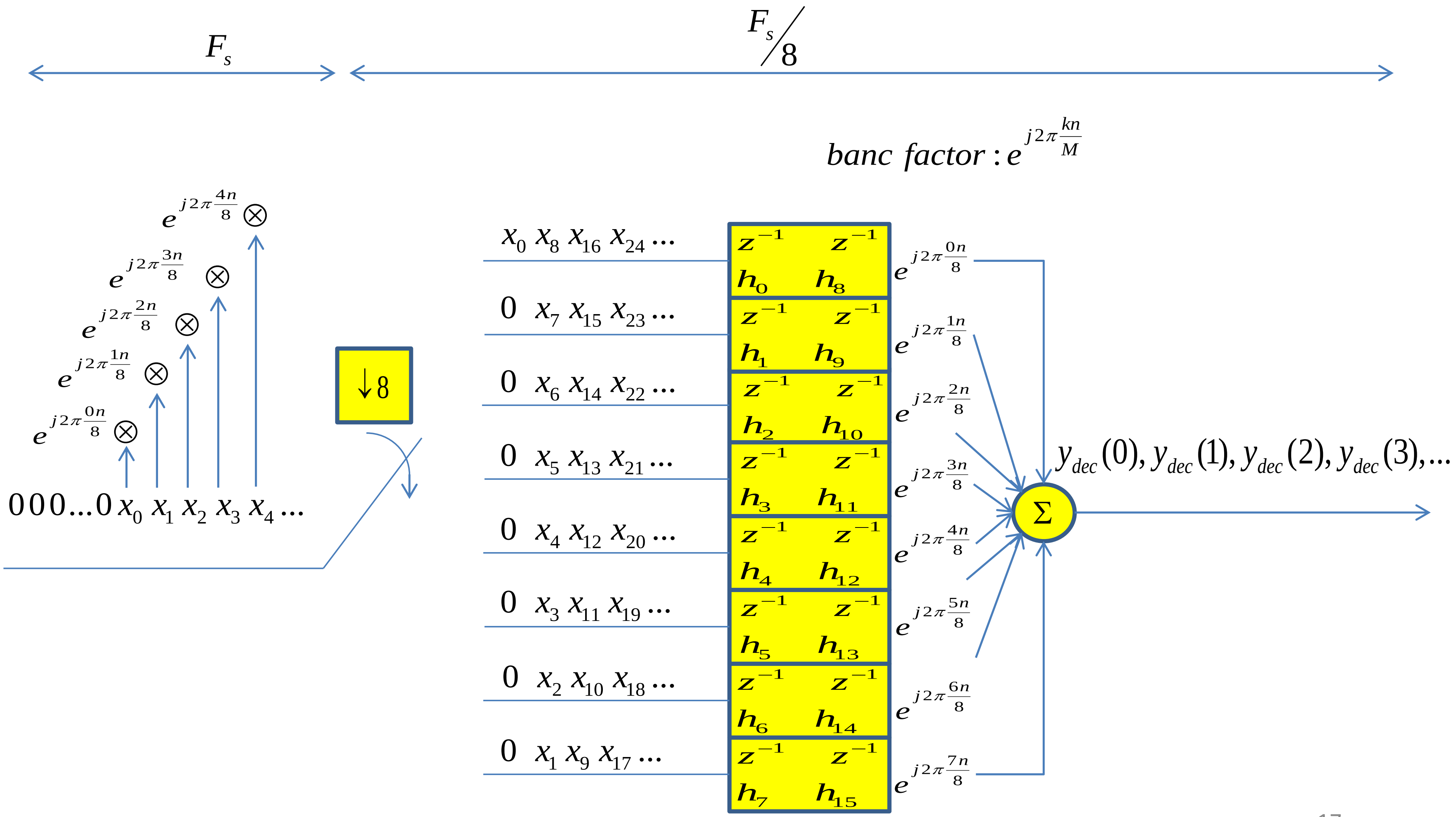
Multichannel

$$x_{bb}^{n}(kM) = x_{wb}^{n}(k) \times e^{j2\pi\frac{kn}{M}}$$



F_s
$F_s/8$
$e^{j2\pi\frac{4n}{8}} \otimes$
$e^{j2\pi\frac{3n}{8}} \otimes$
$e^{j2\pi\frac{2n}{8}} \otimes$
$e^{j2\pi\frac{1n}{8}} \otimes$
$e^{j2\pi\frac{0n}{8}} \otimes$
$000\ldots0\, x_0\, x_1\, x_2\, x_3\, x_4 \ldots$
$\downarrow 8$
$x_0^n\ x_8^n\ x_{16}^n\ x_{24}^n \ldots$
$0\ x_7^n\ x_{15}^n\ x_{23}^n \ldots$
$0\ x_6^n\ x_{14}^n\ x_{22}^n \ldots$
$0\ x_5^n\ x_{13}^n\ x_{21}^n \ldots$
$0\ x_4^n\ x_{12}^n\ x_{20}^n \ldots$
$0\ x_3^n\ x_{11}^n\ x_{19}^n \ldots$
$0\ x_2^n\ x_{10}^n\ x_{18}^n \ldots$
$0\ x_1^n\ x_9^n\ x_{17}^n \ldots$
$z^{-1}\ z^{-1}\ h_0\ h_8$
$z^{-1}\ z^{-1}\ h_1\ h_9$
$z^{-1}\ z^{-1}\ h_2\ h_{10}$
$z^{-1}\ z^{-1}\ h_3\ h_{11}$
$z^{-1}\ z^{-1}\ h_4\ h_{12}$
$z^{-1}\ z^{-1}\ h_5\ h_{13}$
$z^{-1}\ z^{-1}\ h_6\ h_{14}$
$z^{-1}\ z^{-1}\ h_7\ h_{15}$
Σ
$y_{dec}(0), y_{dec}(1), y_{dec}(2), y_{dec}(3), \ldots$

Multichannel



F_s
$F_s/8$
$banc\ factor: e^{j2\pi\frac{kn}{M}}$
$e^{j2\pi\frac{0n}{8}}$ $e^{j2\pi\frac{1n}{8}}$ $e^{j2\pi\frac{2n}{8}}$ $e^{j2\pi\frac{3n}{8}}$ $e^{j2\pi\frac{4n}{8}}$
$000\ldots0\ x_0\ x_1\ x_2\ x_3\ x_4 \ldots$
$\downarrow 8$
$x_0\ x_8\ x_{16}\ x_{24} \ldots$
$0\ x_7\ x_{15}\ x_{23} \ldots$
$0\ x_6\ x_{14}\ x_{22} \ldots$
$0\ x_5\ x_{13}\ x_{21} \ldots$
$0\ x_4\ x_{12}\ x_{20} \ldots$
$0\ x_3\ x_{11}\ x_{19} \ldots$
$0\ x_2\ x_{10}\ x_{18} \ldots$
$0\ x_1\ x_9\ x_{17} \ldots$
z^{-1} h_0 h_8
z^{-1} h_1 h_9
z^{-1} h_2 h_{10}
z^{-1} h_3 h_{11}
z^{-1} h_4 h_{12}
z^{-1} h_5 h_{13}
z^{-1} h_6 h_{14}
z^{-1} h_7 h_{15}
$e^{j2\pi\frac{0n}{8}}$ $e^{j2\pi\frac{1n}{8}}$ $e^{j2\pi\frac{2n}{8}}$ $e^{j2\pi\frac{3n}{8}}$ $e^{j2\pi\frac{4n}{8}}$ $e^{j2\pi\frac{5n}{8}}$ $e^{j2\pi\frac{6n}{8}}$ $e^{j2\pi\frac{7n}{8}}$
Σ
$y_{dec}(0), y_{dec}(1), y_{dec}(2), y_{dec}(3), \ldots$

Multichannel

- So for one channel, at index frequency n, we clearly see a DFT value computed on the M bancs u(k):

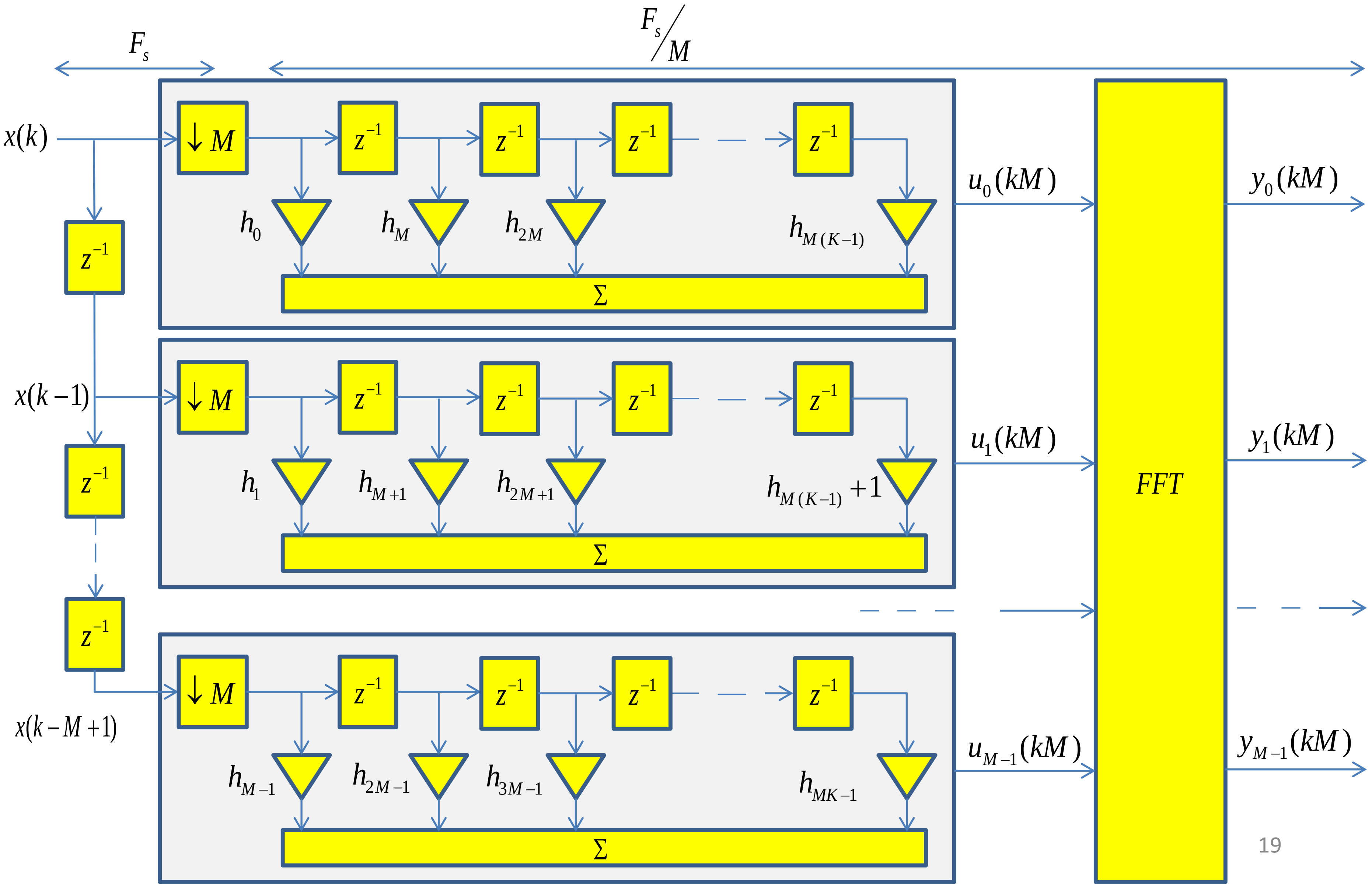
$$y_{dec}^{n}(k) = \sum_{p=0}^{M-1} u(p) e^{j2\pi \frac{pn}{M}}$$

- So, if we do the same computing for M channels we just need a DFT, that’s all!

- For implementation, of course we use an FFT if M is a power of 2, or a prime number multiplied with a power of 2, ….

- It’s a very efficient technique on implementation based on FPGA for ex, by this way the throughput is top!

Multichannel



Conclusions

- Polyphase FIR is equivalent to a FIR decimator.
- Polyphase FIR coupled with DFT is equivalent to do shift then FIR decimator.
- We can interpolate to higher data rates each channel at FFT output if it is needed for a given demodulator because what is important is the baseband filtering that filters with the channel bandwidth (a maximum of noise will be filtered).
- Each banc filter is deduced by the prototype FIR with N taps, so the number of taps per bac would be: K = N / M.
- When we see the diagram we can misunderstand thinking at the inverse order:

  - decimation
  - FIR filtering
  - baseband shift
- This is not true, what we must understand on viewing this diagram is that each banc of FIR decimates the input samples and then each DFT n component will do the shift with the complex exponentials and sum all bancs output: It is exactly the classical way:

  - baseband shift
  - FIR filtering
  - decimation
- All the power of the polyphase implementation is at the fact that by dividing the prototype FIR in parallel to bancs FIR we do the computing at lower sampling frequency, i.e. at Fs/M: this is a maximum gain of MIPS, throughput and resources inside a FPGA for ex.
- The user that has still doubts on polyphase principle, can read again this presentation and it is better to write a matlab simulation in order to understand perfectly, thanks!