



ELEC 305

Digital System Design Lab

Fall 2024

Lecture 6:

DSP Algorithms on FPGAs

Term projects

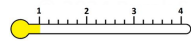


- The project topic is not set, students prepare their own proposals and negotiate with the instructor
- There are many great project examples on the MIT 6.111 course website that you can check out: <https://web.mit.edu/6.111/volume2/www/f2019/> (see “Past projects - all”)
- These are typically challenging signal (audio or image) processing and/or game-based projects
- Since this is the 2nd semester of our course, we don’t have as many interface modules and infrastructure capabilities, but we can purchase modules if you have exciting proposals for them!
 - I can help you with building custom modules / boards too, but these things take time, and you’ll have to factor that into your time plan in your project proposal
- We already have a few audio IO modules and accelerometers as well as FPGA boards that you can use for the projects, I’ll provide more details on them later

Term projects



- I will accept individual submissions, I believe we can arrange board access. We'll scale CPs accordingly.
- You'll get 1 chance at a proposal revision (until **12.12.2024**) and you have to have a proposal accepted by the second deadline (**22.12.2024**) or you unfortunately lose 35% of the course grade. Earlier is OK.
- Challenging projects will receive higher credit. To facilitate this, each project will be issued a “challenge point” (CP) between 1 and 4. The CP of your project will modulate your 35% project grade as follows:



- CP = 1 → your project can get max. 28.5% out of the 35%



- CP = 2 → your project can get max. 30.5% out of the 35%



- CP = 3 → your project can get max. 32.5% out of the 35%



- CP = 4 → your project can get max. 35.0% out of the 35%

- I will provide suggestions during the proposal phase to push all projects up to CP=4, but it's your choice



- I'm open to questions about project ideas
(send me an e-mail, we can arrange a meeting too).
- I know it's not easy to choose one, so please ask, we'll find a project that motivates you



- Short review for Part 1
- Motivation for DSP on FPGAs
- Breaking down the FIR filter algorithm
- Optimizing the FIR filter

Short Review for Part 1



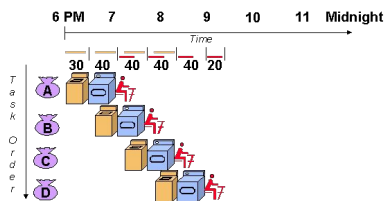
- Part 1 consisted of ...
 - **describing** a circuit using (“DUT”) VHDL based on a given set of specifications
 - using Vivado to automatically **synthesize** the DUT and **implement** it on the FPGA
 - **simulate** DUT **behavior** in VHDL, characterize its accuracy
 - **constrain** and **analyze** its **timing** characteristics
 - consequently **optimize** it if it fails

```

1.  library IEEE;
2.  use IEEE.STD_LOGIC_1164.ALL;
3.
4.  entity coffeemaker is
5.      Port ( clk : in  STD_LOGIC;
6.            led : out STD_LOGIC;
7.            sw  : in  STD_LOGIC
8.            );
9.  end coffeemaker;
10.
11. architecture Behavioral of coffeemaker is
12.     signal pulse : std_logic := '0';
13.     signal count : integer range 0 to 199999999 := 0;
14. begin
15.     process(clk, sw)
16.     begin
17.         ...
18.     end process;
19.
20.     led <= pulse;
21. end Behavioral;
    
```

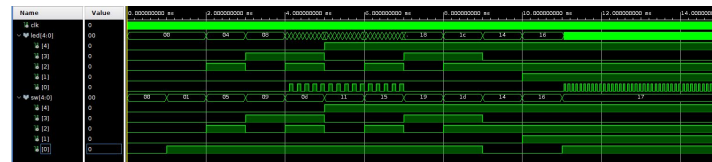
Failed Timing! -8.808 -8.801

Laundry pipeline diagram



Arithmetic Functions

Design Element	
DSP48E1	Primitive: 48-bit Multi-Functional Arithmetic Block

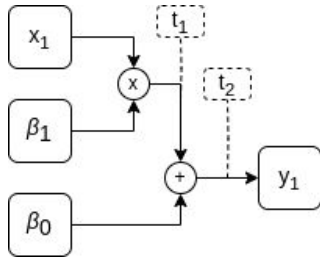


Short Review for Part 1



- Recently, we talked about fixed point number representations, and doing arithmetic operations with them

$$y_1 = \beta_0 + x_1 * \beta_1$$



© 2024 Burak Soner

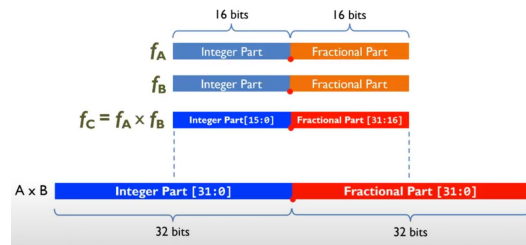
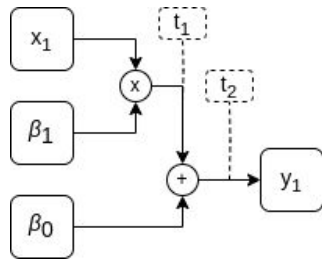
© 2024 Burak Soner

Short Review for Part 1



- Recently, we talked about fixed point number representations, and doing arithmetic operations with them

$$y_1 = \beta_0 + x_1 * \beta_1$$



img src <https://www.youtube.com/watch?v=YXKDjvCjWwE&t=1s>

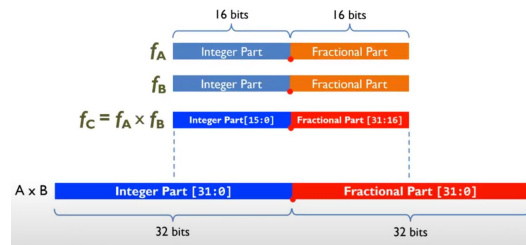
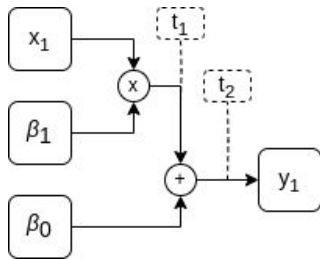
- We also mentioned two important classes of optimizations:

Short Review for Part 1



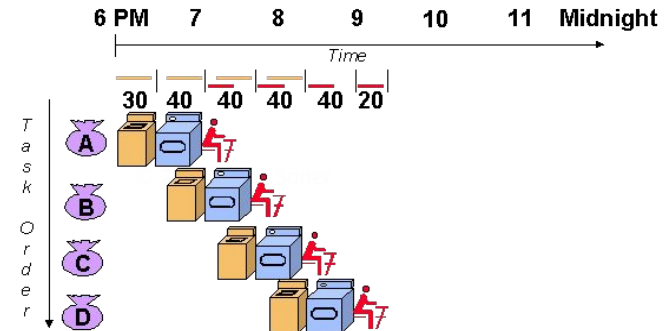
- Recently, we talked about fixed point number representations, and doing arithmetic operations with them

$$y_1 = \beta_0 + x_1 * \beta_1$$



img src <https://www.youtube.com/watch?v=YXKDjVcQWfE&t=1s>

Laundry pipeline diagram



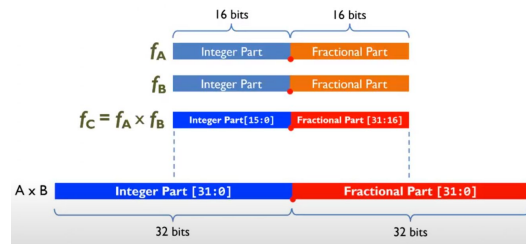
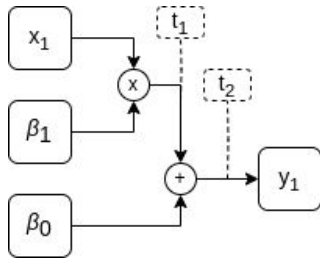
- We also mentioned two important classes of optimizations:
 1. RTL-level, pipelining and parallelism

Short Review for Part 1



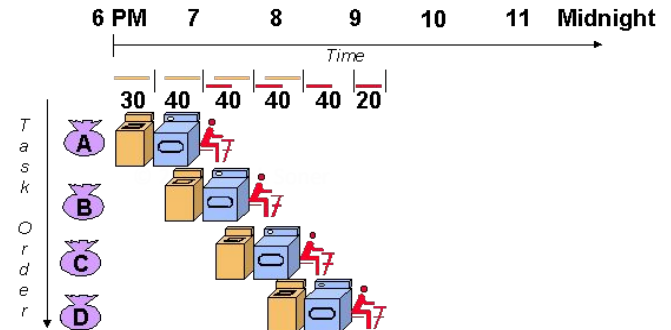
- Recently, we talked about fixed point number representations, and doing arithmetic operations with them

$$y_1 = \beta_0 + x_1 * \beta_1$$



img src <https://www.youtube.com/watch?v=YXKDjvCjWvE&t=1s>

Laundry pipeline diagram



- We also mentioned two important classes of optimizations:
 1. RTL-level, pipelining and parallelism
 2. Using device primitives (e.g., DSP48E1 for our Artix-7)

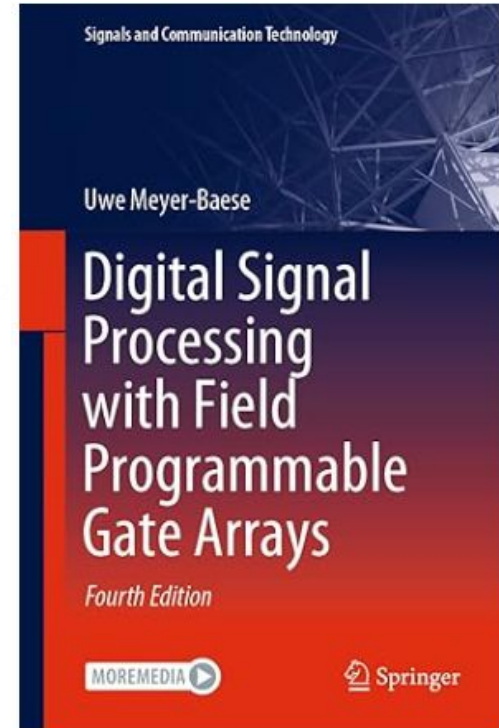
Arithmetic Functions

Design Element	
DSP48E1	Primitive: 48-bit Multi-Functional Arithmetic Block

Motivation: DSP on FPGAs



- Signal processing circuits are heavily used in telecom / networking
 - Filters, equalizers, OFDMs, PLLs, interpolators, DDS, ...
- The prototyping cycle usually goes like this:
 1. Very early PoC → CPUs / MCUs (“just testing”)
 2. Hardware-in-the-loop / custom application → FPGAs
 3. Proven design, let’s make it scalable → ASIC
- We’re investigating 2. There are books dedicated to it →→
- The lifecycle stops at 2 if there isn’t a market for the ASIC



Motivation: DSP on FPGAs



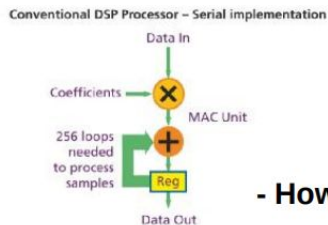
- FPGA clock speeds are not faster than CPU/GPUs or ASICs!

- but the two optimizations we mentioned make FPGAs very attractive for DSP:

1. arbitrary parallelization / pipelining schemes
2. coupling that with efficient primitives such as DSP IP cores

Why FPGA for Signal Processing?

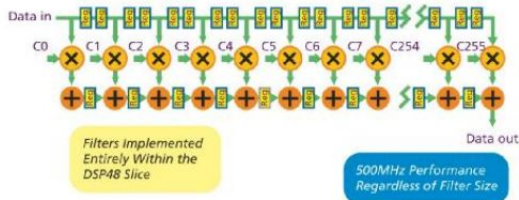
256-tap Filter Example



$$\frac{1 \text{ GHz}}{256 \text{ clock cycles}} = 4 \text{ MSPS}$$

- How much computational power do you need?

Virtex-4 Parallel Implementation Consumes Zero Logic Resources

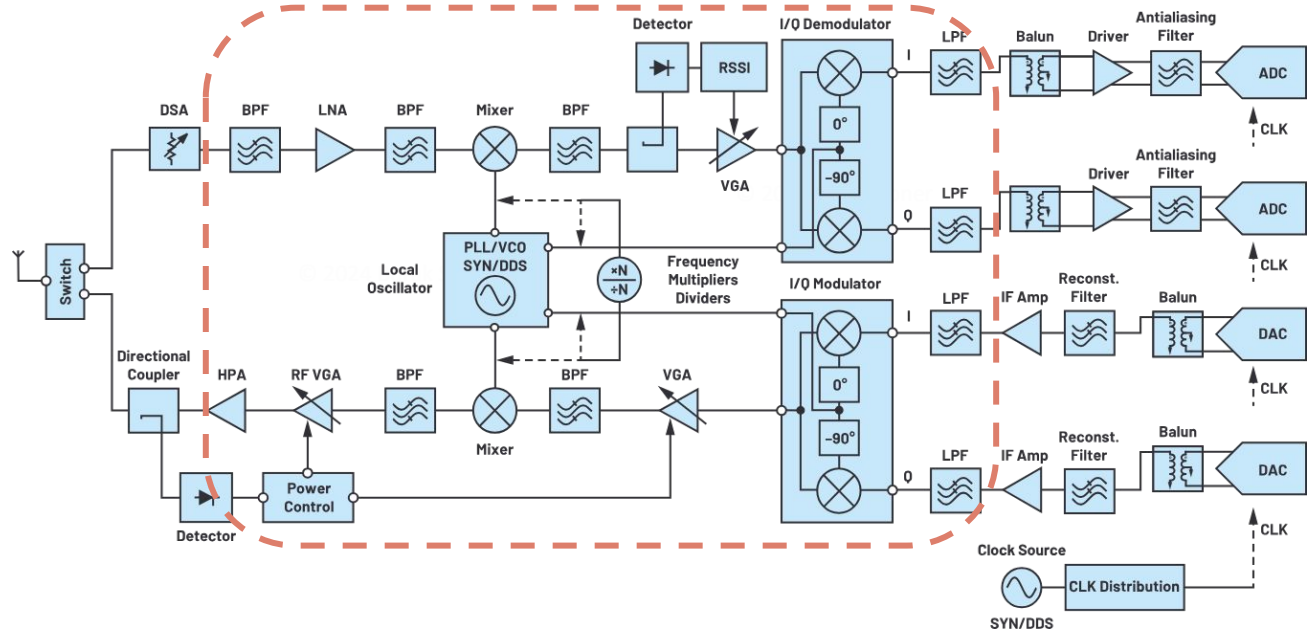


$$\frac{500 \text{ MHz}}{1 \text{ clock cycle}} = 500 \text{ MSPS}$$

Motivation: DSP on FPGAs



- One of the most exciting FPGA-DSP applications is a software-defined radio (SDR)
- Radio chains were “hardware-defined” →
- Main reason: HUGE bandwidth (throughput!) requirements
- SDR: just use super-fast ADC / DACs and implement **the rest of the radio chain** on FPGA



img src: <https://www.analog.com/en/resources/analog-dialogue/articles/rf-signal-chain-discourse-part-2-essential-building-blocks.html>

Motivation: DSP on FPGAs



- Being able to use an SDR & designing DSP algorithms for its on-board FPGA is an important skill for almost all defense and telecommunications jobs out there!



img src: <https://www.testdynamics.co.za/Product/Eltus.html>

Senior Digital Design Engineer

ERA RF Technologies · Pendik, İstanbul, Türkiy...

Easy Apply

Save



Additional Skillsets for Distinction:

- Experience on Software Defined Radios (SDRs)
- Experience in defense, avionics or space projects
- Familiarity with RF, analog and mixed-signal design and their operational principles
- Familiarity with high-frequency circuits

SENIOR DSP/ALGORITHM DESIGN EN...

ERA RF Technologies · Pendik, İstanbul, Türkiy...

Easy Apply

Save



solutions and prototypes, if/when required

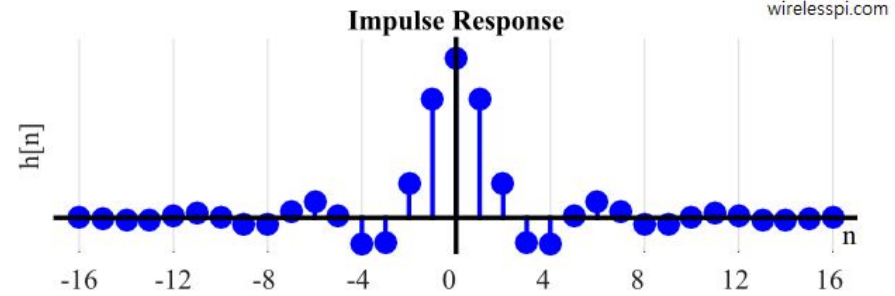
Additional Skillsets for Distinction:

- Experience on Software Defined Radios (SDRs)
- Experience in defense, avionics or space projects
- Experience on digital design, hardware description languages and FPGA architectures
- Familiarity with RF, analog and mixed-signal design and their operational principles
- Familiarity with high-frequency circuits

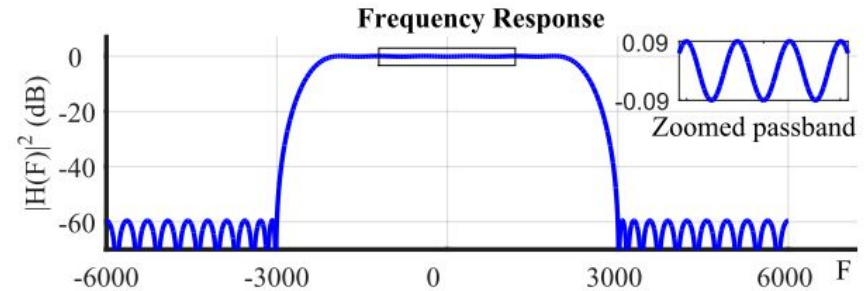
FIR filter



- Let's start simple → one important building block used in most DSP applications is the **FIR filter**



(a) Filter coefficients in time domain



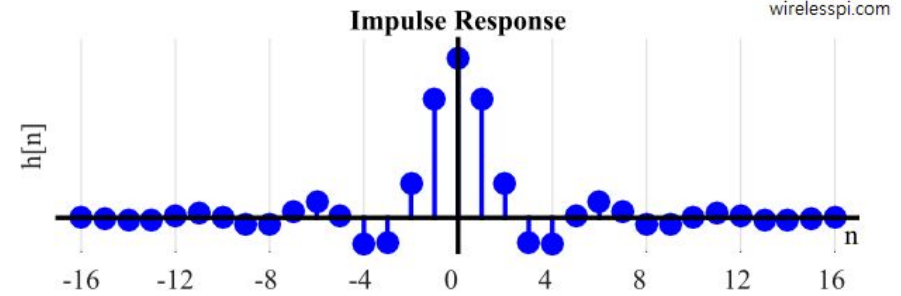
(b) Note the stopband attenuation of 60 dB and passband ripple within 0.1 dB

Img src: <https://wirelesspi.com/finite-impulse-response-fir-filters/>

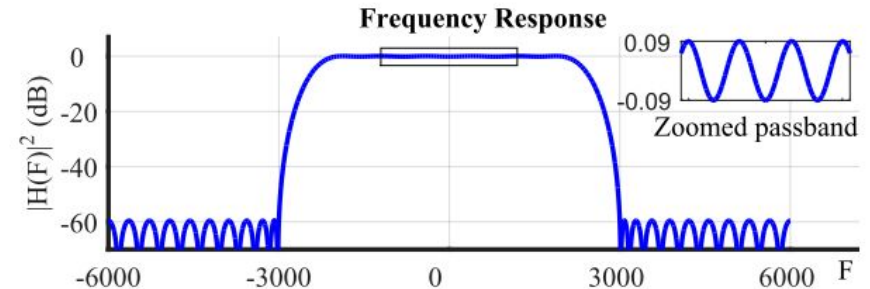
FIR filter



- Let's start simple → one important building block used in most DSP applications is the FIR filter
- FIR: Finite Impulse Response



(a) Filter coefficients in time domain



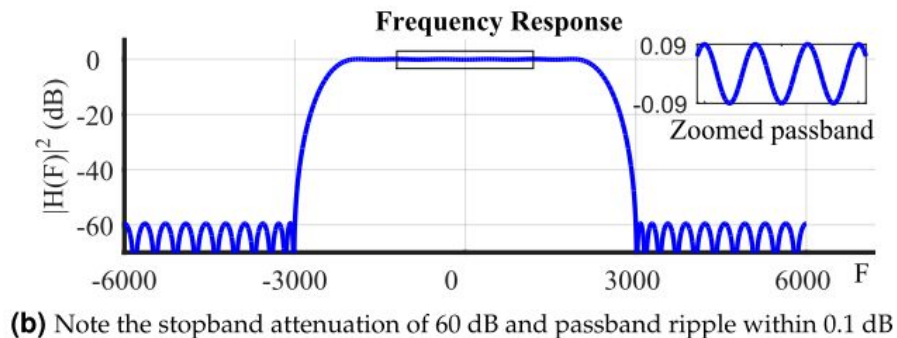
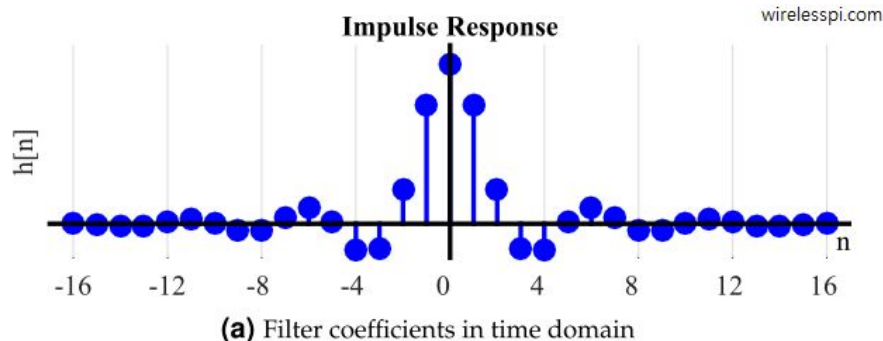
(b) Note the stopband attenuation of 60 dB and passband ripple within 0.1 dB

Img src: <https://wirelesspi.com/finite-impulse-response-fir-filters/>

FIR filter



- Let's start simple → one important building block used in most DSP applications is the FIR filter
- FIR: Finite Impulse Response
 - meaning the impulse response of the filter is “zero after some point”
- This makes the filter stable
 - You might remember IIRs as being (potentially) unstable, that's because they have feedback, their impulse may never decay to 0 (depending on how the fed-back output turns out)



img src: <https://wirelesspi.com/finite-impulse-response-fir-filters/>



- We mentioned SDRs, so let's cover a telecommunication application where the FIR filter takes a role
- The Amplitude Modulation (AM) demodulator is one such appl. where the envelope of a signal is recovered from an RF signal in noise
- **What is the mathematical form of the AM RF signal?**



- We mentioned SDRs, so let's cover a telecommunication application where the FIR filter takes a role
 - The Amplitude Modulation (AM) demodulator is one such appl. where the envelope of a signal is recovered from an RF signal in noise
 - The RF signal is basically a carrier (ω) multiplied with the “message” signal ($A(t)$ below) contaminated with noise:
- **How do we demodulate this at the receiver side?**

$$m(t) = A(t) \cdot \cos(\omega t + \phi(t)) + \text{noise}$$

FIR filter

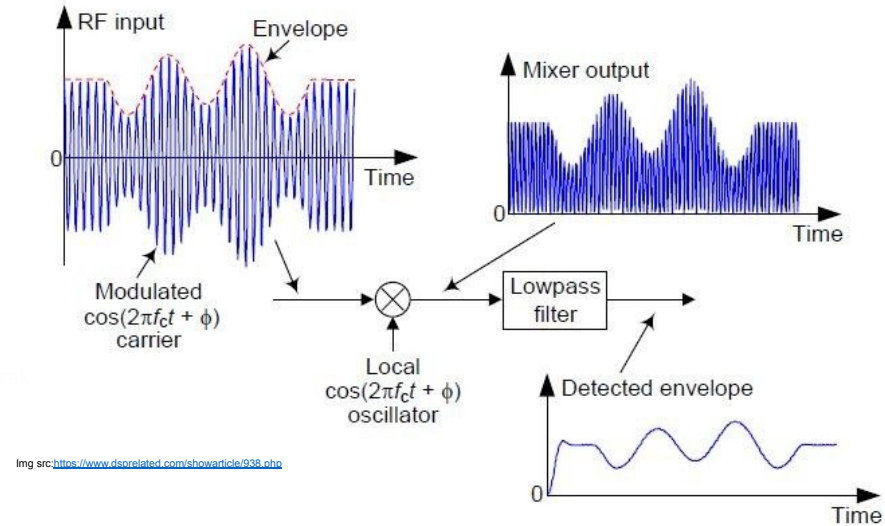


- We mentioned SDRs, so let's cover a telecommunication application where the FIR filter takes a role

- The Amplitude Modulation (AM) demodulator is one such appl. where the envelope of a signal is recovered from an RF signal in noise

- The RF signal is basically a carrier (ω) multiplied with the “message” signal ($A(t)$ below) contaminated with noise:

$$m(t) = A(t) \cdot \cos(\omega t + \phi(t)) + \text{noise}$$

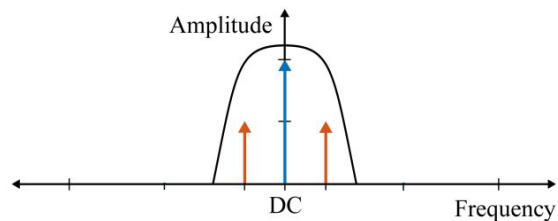


- On the receiver side we first run the RF through a “mixer” (multiplication with a local oscillator at carrier freq.), and then **lowpass-filter** the output to recover $A(t)$

FIR filter

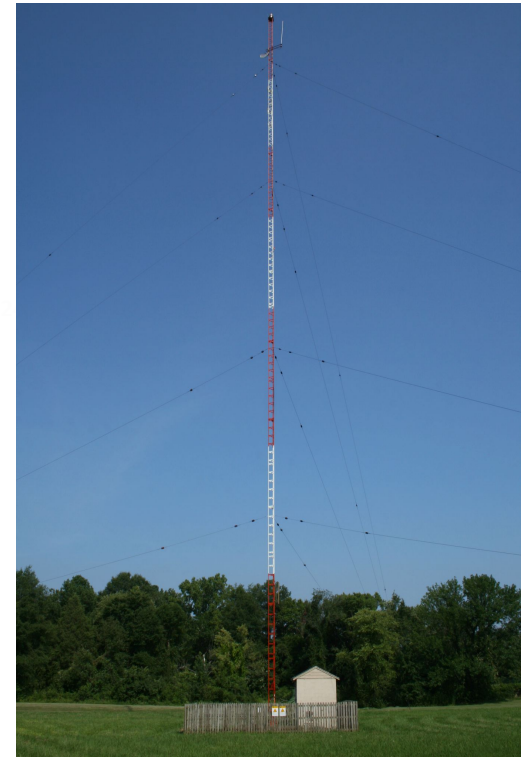


- The LPF in this AM demod can be implemented with an FIR filter
 - It doesn't have to be an FIR by the way, don't get confused, you could use an IIR, but designing a stable IIR filter for a simple task like this is typically more complicated than necessary, and outside the scope of this course
- Let's assume a 1 MHz carrier and $A(t)$ bandwidth of 10 kHz
 - Tidbit: AM is 535 to 1605 kHz, $1/2 \lambda$ antennas are basically towers $\rightarrow\rightarrow\rightarrow\rightarrow$



img src: <https://europepmc.org/article/MED/28900596>

- Assuming a perfect mixer, we'd be operating at baseband, and want the FIR LPF to have a cut-off at 15 kHz to be safe



img src: https://en.wikipedia.org/wiki/Mast_radiator

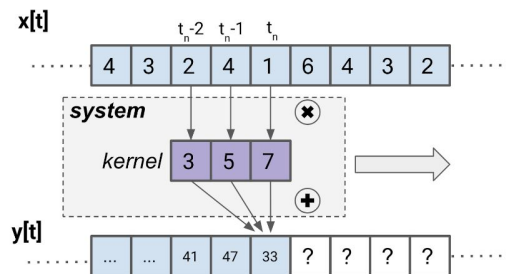


- We want a digital implementation, we'll use an ADC, **how fast should we sample the signal?**

FIR filter



- We want a digital implementation, so we need to digitize the incoming baseband signals first
 - In an SDR, the mixing part could be achieved with what's called a “digital downconversion” (DDC), so we would be working on the FIR filter with digitized inputs directly. This is an advanced method though, see [this ADI article](#) on IQ sampling and DDC for more info. Let's assume we're digitizing the incoming baseband for this FIR filter analysis.
- Let's pass it through an ADC, sampling the incoming signal at 100 kHz
 - Note that this is much higher than the necessary 20 kHz @Nyquist, no good reason, just for easier visualization
- To apply the filter, we'll **convolve** the resulting digitized input signal with the filter “kernel”
- The kernel is what computer scientists call a stripped version of the impulse response of the filter (cut the zeros out)



img src: <https://cyclino74.com/tutorials/demystifying-digital-filters-part-1>

FIR filter



- We usually need to go through FIR filter design steps to get the kernel, but that's not really a part of this course
- Therefore let's use an online calculator: <https://fiiir.com/>
- Given a few desired parameters, FIR filter design is a pretty straightforward optimization problem
- This is an automated designer that solves that problem to some degree of tolerance, taking as input...
 - sampling frequency
 - windowing method to be used
 - desired cutoff frequency
 - desired transition bandwidth

Filter Settings

Filter type
Low pass (windowed-sinc FIR)

Low-pass windowed-sinc FIR filter (more info in [How to Create a Simple Low-Pass Filter](#)). The ideal response is updated on the fly, but you still have to push the Compute Filter button below to recompute the filter.

Ideal response

Sampling rate [Hz]
1

Cutoff frequency f_L [Hz]
0.1

Transition bandwidth b_L [Hz]
0.08

Window type
Blackman

Compute Filter Reset

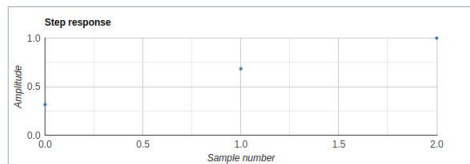
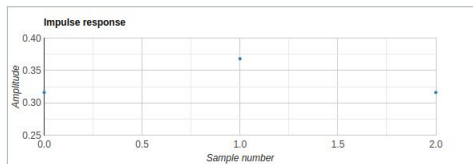
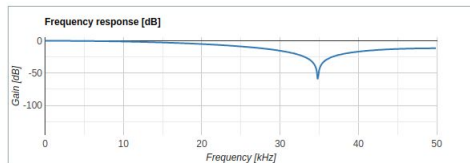
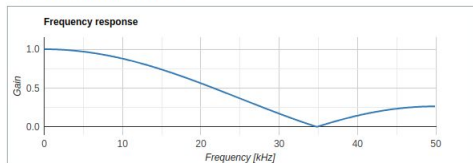
Disclaimer: This tool is made available in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

FIR filter



- The simplest FIR filter we can think of for this application is one with a rectangular window and no constraint on the transition bandwidth →→→
- This results in a 3-coefficient approximation of the sinc:

Filter Characteristics



Filter code

```
0.315959998165989776
0.368081803669838583
0.315959998165989776
```

Status

Filter is up to date. Filter has 3 coefficients.

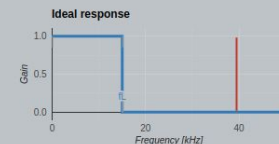
FIIR! - Design FIR & IIR Filters - From TomRo

Filter Settings

Filter type

Low pass (windowed-sinc FIR)

Low-pass windowed-sinc FIR filter (more info in [How to Create a Simple Low-Pass Filter](#)). The ideal response is updated on the fly, but you still have to push the Compute Filter button below to recompute the filter.



Sampling rate [Hz]

100000

Cutoff frequency f_c [Hz]

15000

Transition bandwidth b_L [Hz]

49000

Window type

Rectangular

Compute Filter

Reset

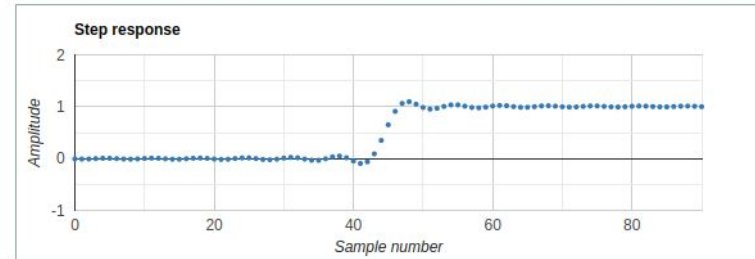
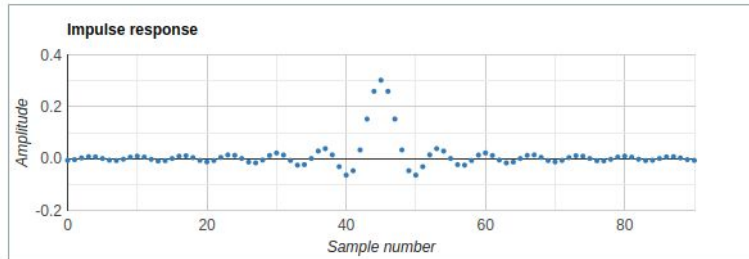
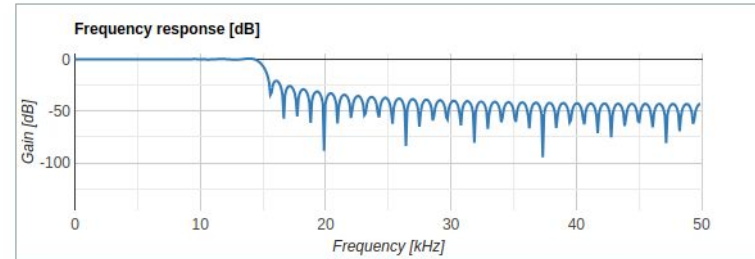
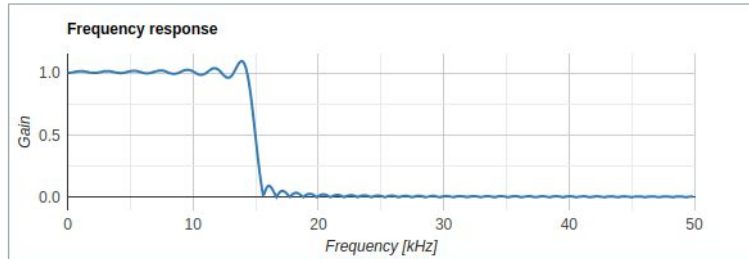
Disclaimer: This tool is made available in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

FIR filter



- The tightest transition band (1kHz) creates the best approx. to the ideal FIR LPF with a rect window (rect pulse in the freq domain), but this one has 91 coefs (higher complexity):

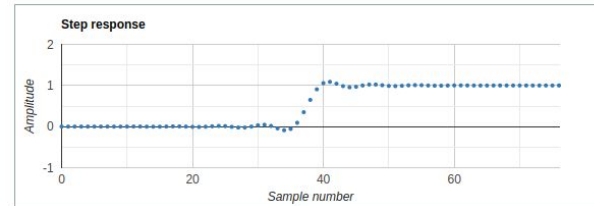
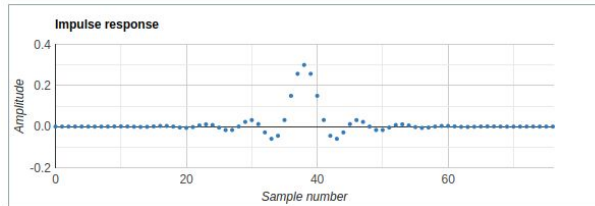
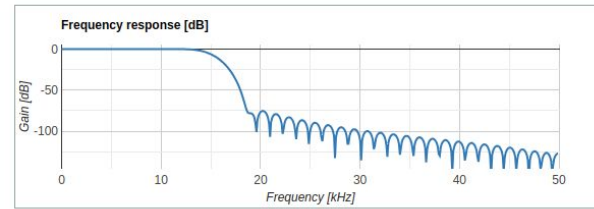
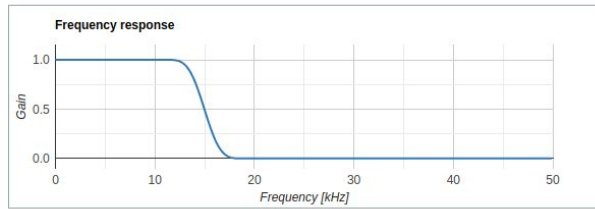
Filter Characteristics





- If we need smoother functions at the output we could opt for Blackman windows too (with even higher complexity though):

Filter Characteristics



- Nevertheless, these are more advanced DSP topics, not our concern for now

FIR filter

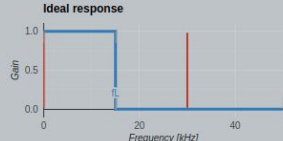


- Let's settle for a simpler 5-coef filter with a relaxed transition bandwidth constraint

Filter Settings

Filter type
Low pass (windowed-sinc FIR)

Low-pass windowed-sinc FIR filter (more info in [How to Create a Simple Low-Pass Filter](#)). The ideal response is updated on the fly, but you still have to push the Compute Filter button below to recompute the filter.



Sampling rate [Hz]
100000

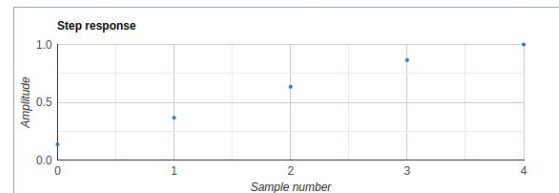
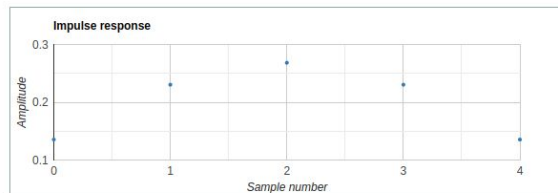
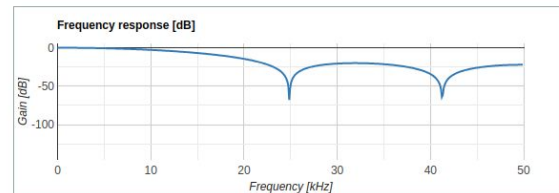
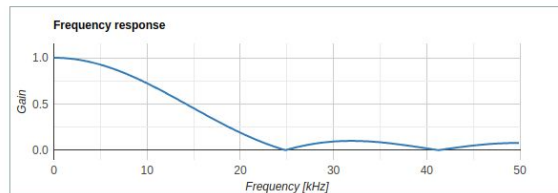
Cutoff frequency f_c [Hz]
15000

Transition bandwidth b_L [Hz]
30000

Window type
Rectangular

Compute Filter Reset

Filter Characteristics



Filter code

```
0.135417630349277313  
0.230386233443460930  
0.268392272414523625  
0.230386233443460930  
0.135417630349277313
```

Coefficients as list of numbers

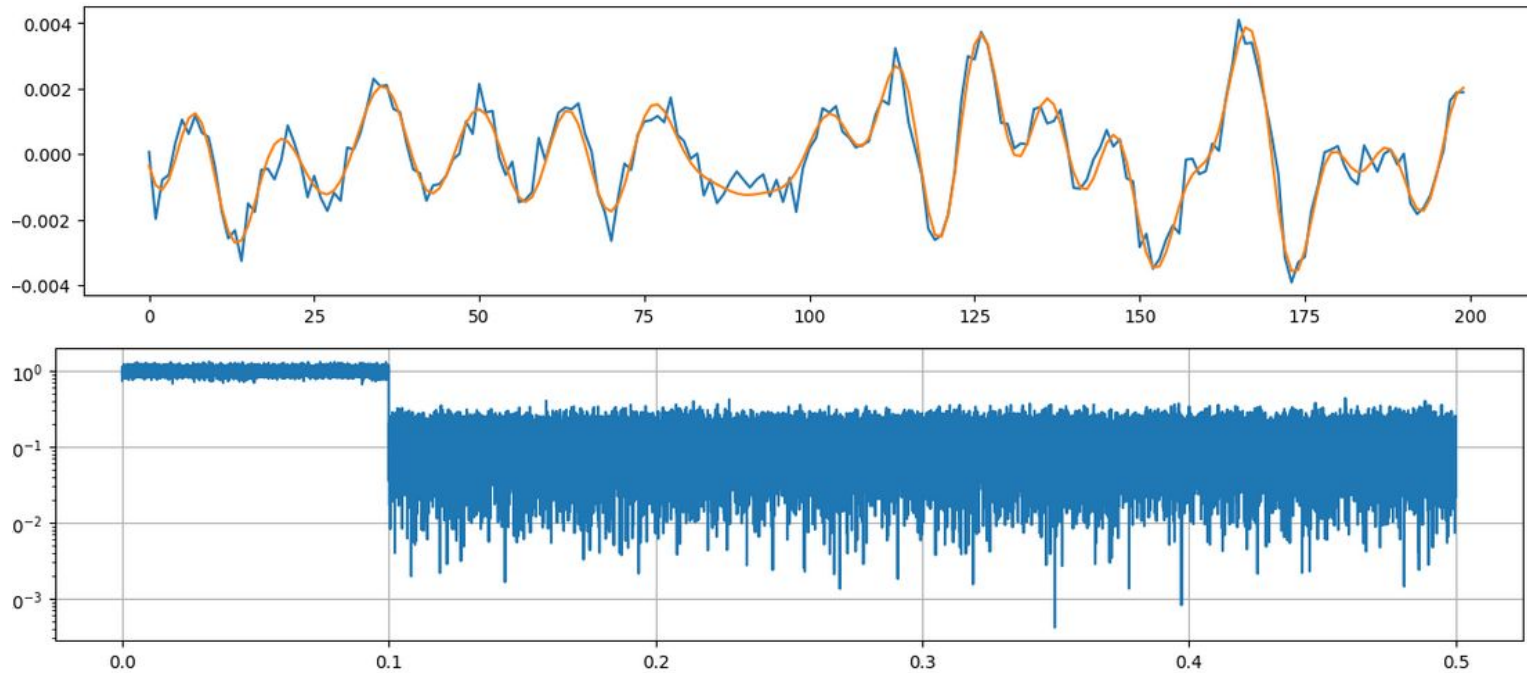
Status

Filter is up to date. Filter has 5 coefficients.

FIR filter



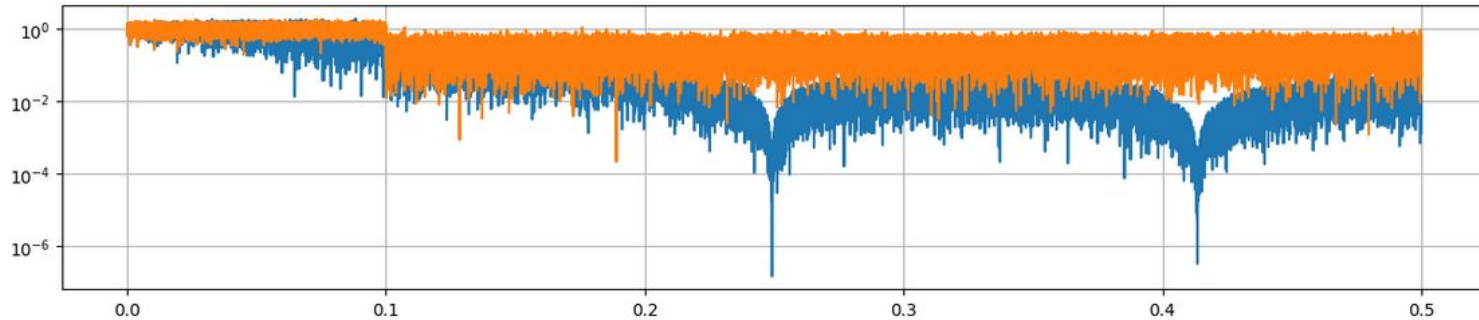
- Let's simulate this, a random baseband signal of 10 kHz bandwidth, contaminated by noise:



FIR filter



- When we apply the 5-coef filter we just designed, the high-band noise power decreases



- OK we have the software verification, the LPF works
- **how can we deploy this on an FPGA, what do we need to consider?** How did we transition from an algorithm on paper to a digital design earlier (hint: linear regression)?



- Considerations:
 - Lay out the computational graph like we did earlier for the linear regressor
 - The FIR is already in discrete time, so sampling is already done
 - Quantize the values in the computational graph (i.e., convert floats to fixed point)
- Let's start by drawing the computational graph to motivate STA and optimizations
- we'll translate the numbers to fixed point representations and have a look at accuracy against the software implementation in the pre-lab session



- The FIR filter is applied by a convolution operation

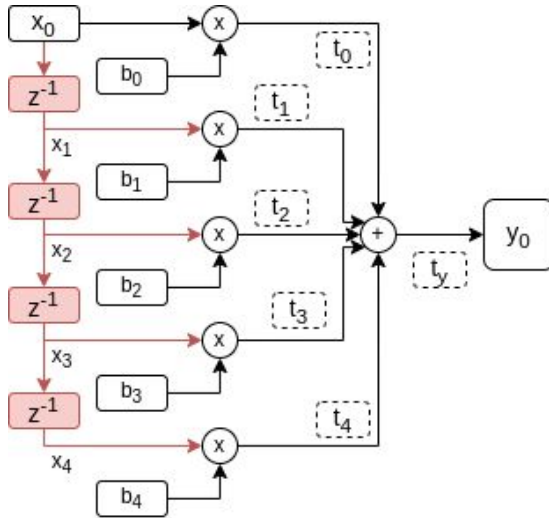
$$\begin{aligned}y[n] &= b_0x[n] + b_1x[n - 1] + \dots + b_Nx[n - N] \\ &= \sum_{i=0}^N b_i \cdot x[n - i],\end{aligned}$$

- where $x[n-N]$ are input samples, and b_0, b_1, \dots, b_N are kernel elements (i.e., impulse response samples)
 - note however that the kernel is inverted in time, so b_0 is the last element of the impulse response
- digital designers typically call the result of the multiplication with a kernel element and its corresponding input sample, e.g., $b_0 \cdot x[n]$, a [“filter tap”](#)
- the # of taps is an indication how complex the implementation is (our example is a 5-tap filter)

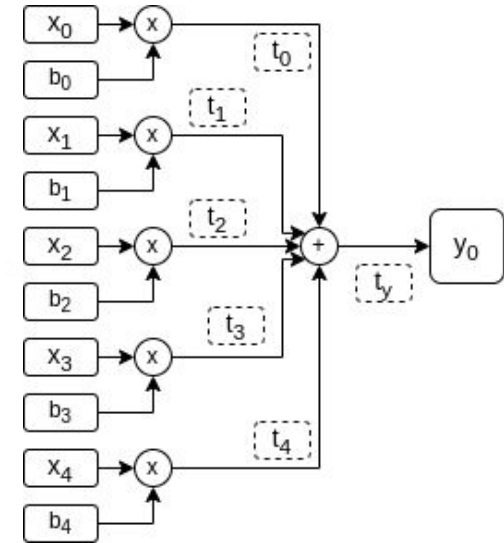
FIR filter



- The computational graph therefore looks like this →→→→→
- However we already know $x_{n-1} = (x_n \text{ delayed by 1 clock cycle})$, so this is an equivalent graph:



- The red line is called a “delay line”
- It’s typically implemented with a series of flip-flops delaying the signal by 1 clock cycle each

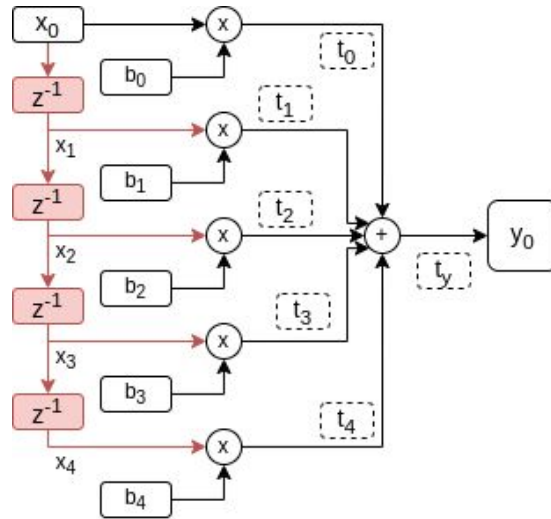


- This graph repeats for every input sample to compute the corresponding output sample

FIR filter



- Now let's forget about Vivado for a sec and think of how we would implement this ourselves



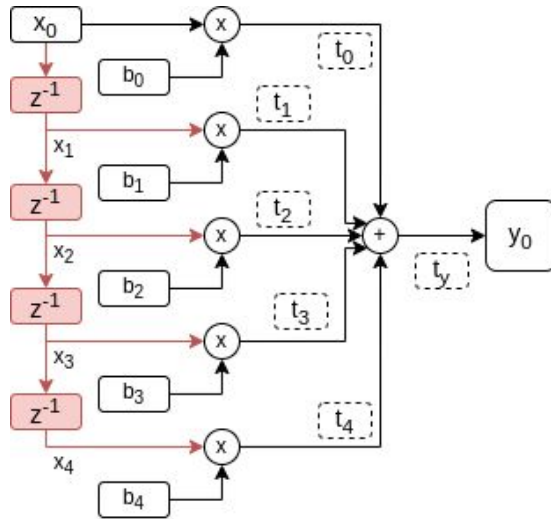
© 2024 Burak Soner

© 2024 Burak Soner

FIR filter: Optimizations



- Now let's forget about Vivado for a sec and think of how we would implement this ourselves
- Delays are basically FFs. We need 5 multipliers and 5 adders.



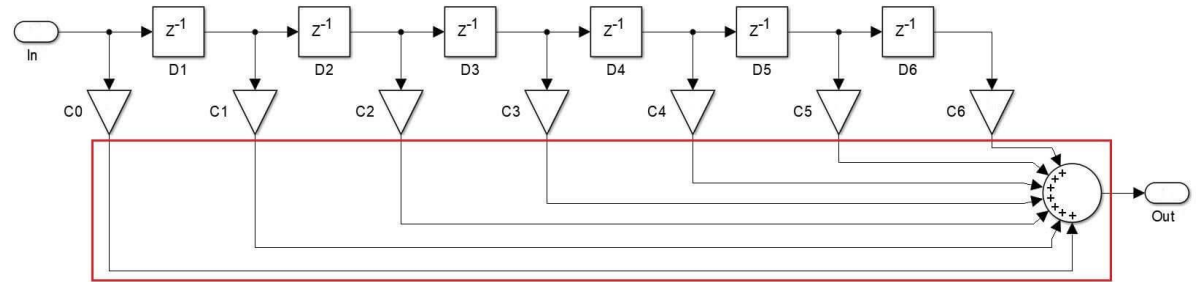
- Now try to picture the timing: Assign virtual flip-flops at circuit inputs and outputs like STA does, and then compute the time that it takes for the combinational circuit in between to run.
- Mults, adders and flip-flops, ..., that's pretty long, this circuit may run into timing problems with fast clocks
- Can you think of some basic optimizations to reduce the path lengths?

FIR filter: Optimizations

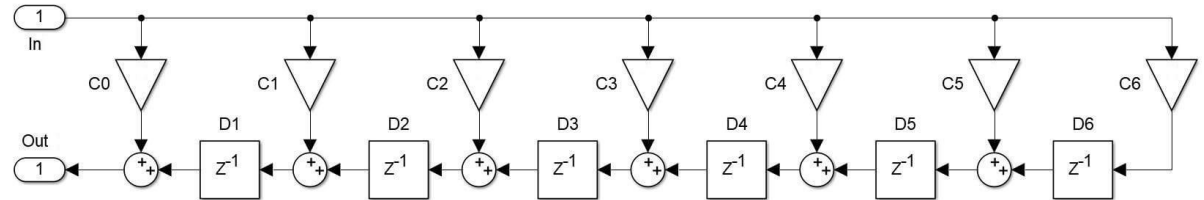


- The one we have is called “direct form”. One well-known optimization is a “transposed” form

- This changes the order in which the ops are applied, so it’s the same math, but less combinational delay for the paths



- Any ideas for increasing clock speeds further?

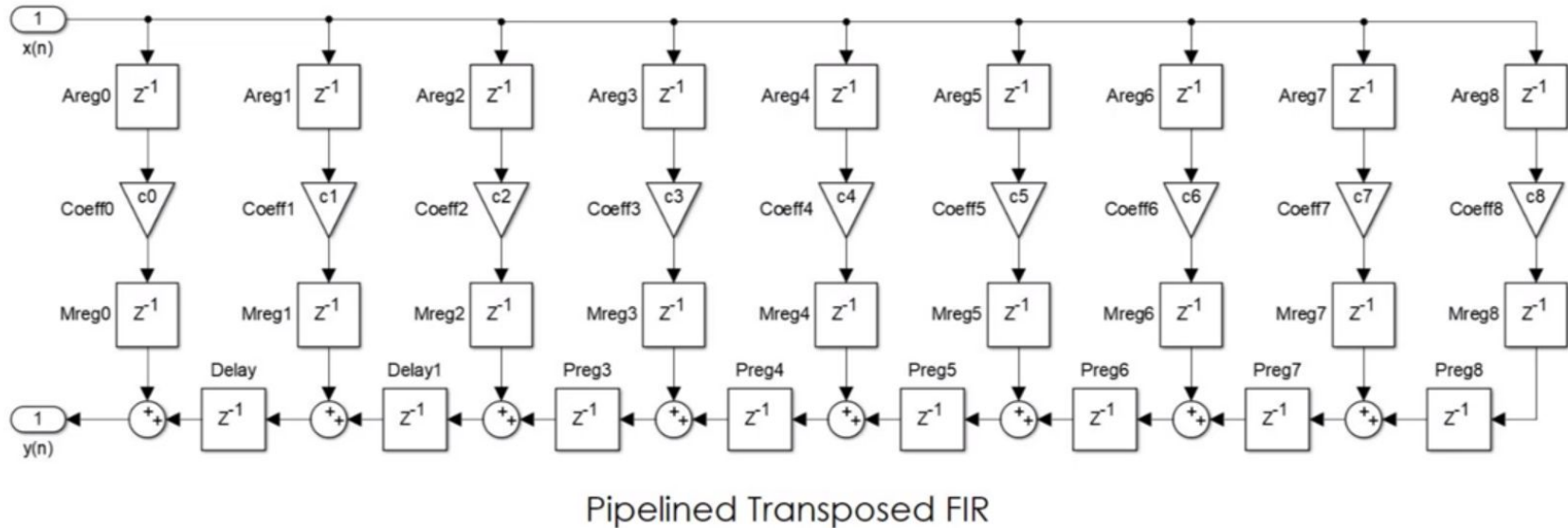


img src: <https://yhdvvhiz.com/part-2-finite-impulse-response-fir-filters/>

FIR filter: Optimizations



- Pipelining this is also possible, creating even less risk of setup-hold time violations, allowing for faster clocks in expense of more clock cycles of latency

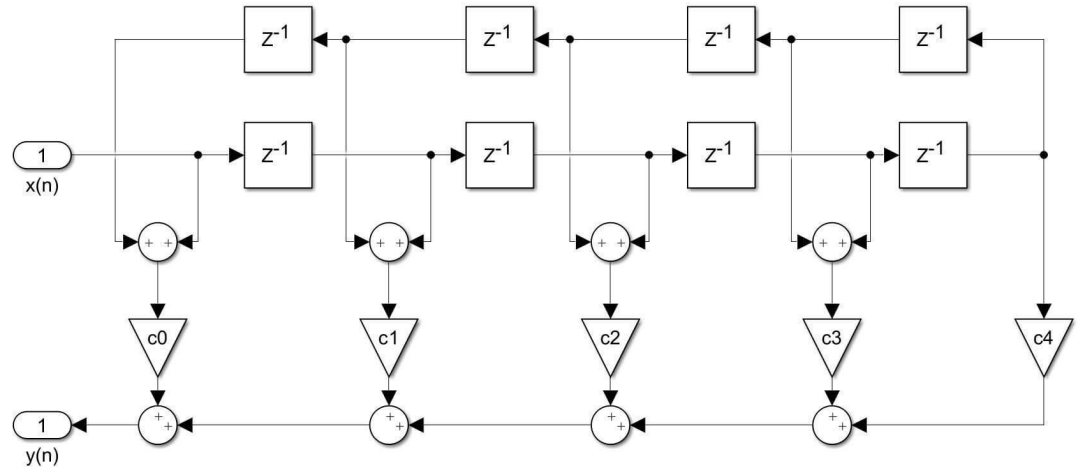


img src: <https://www.youtube.com/watch?v=1LUX-V5yCA&t=158s>

FIR filter: Optimizations



- More complex optimizations are possible, especially depending on the values of the coeffs
- For instance for symmetric coeffs, the computations can be “folded” to reduce the number of multiplications by a factor of 2:
- Or half-band filters allow for skipping computations for zero-valued coefficients etc.
- The list of optimizations for specific FIR filter configurations can keep going, this is a “fruitful” engineering problem





next → lab 3

I'll provide you Python-based software tools to help your testbench

