Proposed Answers

# Generate PRBS and apply filter

1. Generate PRBS 13 signal in PAM4 using python with Vpeak = 0.4V and Freq=10GHz
2. Apply filter to the signal using following transfer function and plot the output signal in time domain:

H(s) = 1/(1+2\*ξ\*jω/ω\_cut+(jω/ω\_cut)^2)

Where:

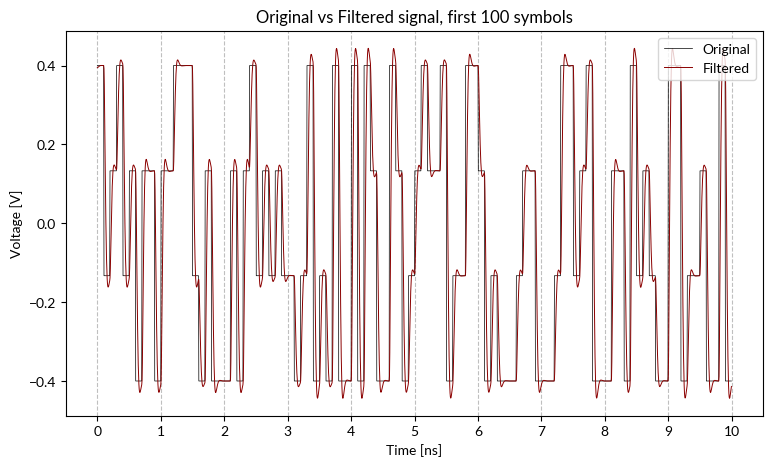
ξ= 0.68

f = 10GHz

ω\_cut = 2\*π\*f

**Folder: EX1 – PRBS & Filter  
Content: PRBS.py (PRBS generator class), Script.py (script used to generate the image)**

Sequence was generated using Wikipedia definition (sources [1] [2] [3]; generated sequence correctness was checked using site [4]). In order to accommodate for PAM4, sequence used is a PRBS13Q. Sequence was then simulated at the required frequency & voltages and sampled at 10 times the frequency for quality, assuming behavior approximate to square wave. Low-pass filter was simulated and applied using the ‘control’ python library (tutorial used [5]). Following is an image of the first 100 symbols (full PRBS13 would show as a filled rectangle). Use the original script in order to generate and visualize the full sequence.



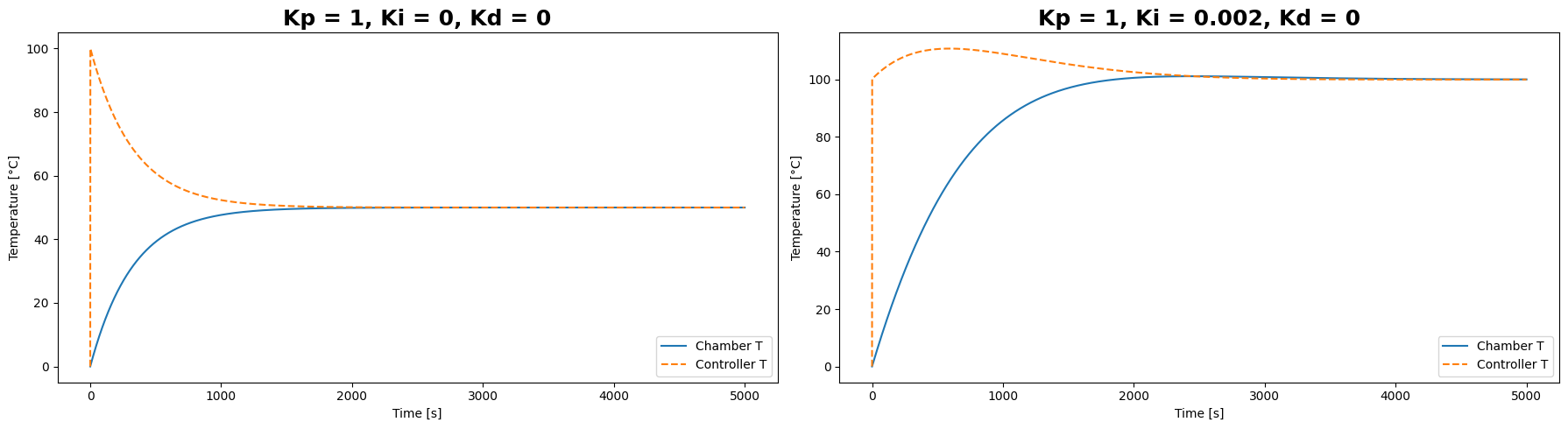
# Feedback theory

1. Simulate a control loop in python for a temperature chamber where t\_initial = 0C and t\_desired = 100C. Use proportional control only.
2. Explain the problems that the loop has.
3. How can the loop control can be improved?

**Folder: EX2 – Feedback Theory  
Content: ClimaticChamber.py (system behavior), PID.py (simple PID script), Script.py (system simulation and image generation)**

The climatic chamber heating system behavior was approximated to that of a resistor, transmitting heat through convection. So: where m is the mass of air in the chamber, c is the air’s specific heat capacity and is the heat flux; k is a constant, taking into account resistor area and air’s forced heat convection [7]. System behavior during cooling was assumed to be the same for ease of calculation.

PID system was according to [6]. First simulation was done with only Kp != 0, second with Ki too.

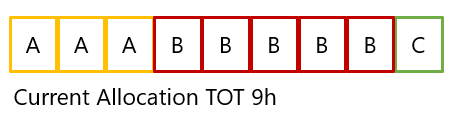


In this kind of system, using Kp only will yield a wrong constant value as the end state. Since the heater setpoint (manipulated variable) depends only proportionately to error, the system will reach a point for which the error (calculated as setpoint – monitored variable current value) will stop changing. IE, at 50°C, the error will be 50°C, if Kp = 1 then the heater will be set at 50°C too and the system’s temperature will not change anymore.

Adding a non-zero integrative term (Ki) will prevent this behavior: even when error is fixed, the total error (and so the heater’s temperature, in this case) will keep increasing. Since it’s an integrative term, the error will quickly diverge for high Ki values, resulting in overshoots of the observed variable. For this system, the Ki and Kp constant were manually calibrated to reduce overshooting as much as possible.

# Decision making

Company ACME Inc has 3 work centers A B C. Each work center costs 30Euro/h. The company produce 1 product called <<Blue>> with the current resource allocation profile. Work centers are paid with standard contract yearly wage.



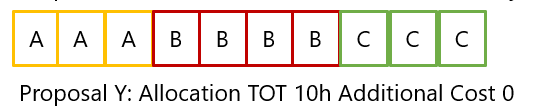
Contribution margin [[1]](#footnote-1) for 1 product is 25Euro and the average demand from market is 40Units per month. (assume 160h of work in a month) (\*\*Mail clarification: selling a product contributes 25€ to the bottom line\*\*)

Company has 2 proposals from 2 engineers X and Y to improve current production rate.

X Improve resource A sparing 2 h and reducing the overall time to 7H.   
Implementation cost is 5000 Euro  
The cost per unit is reduced from 270Euro to 210Euro, computed as operational cost divided per unit.



Y use resource C to do part of the job of B, reducing B time by 1h, increasing total time to 10H. Implementation cost is 0K  
The cost per unit is increased to 300Euro



Which solution will produce the best overall results long term for the company? Why?

**Answer:** Assuming each work center can work on a single product at the time each, the number of units produced per month at the start is equal to 160h / maxTime(A,B,C) = 160h / time(B) = 160h / 5h = 32. So, initial margin per month is 25€ \* 32 = 800€.

Implementing proposal X will result in an increased yield of 60€/unit, for a total of 85€ / unit. Margin per month is roughly 85€ \* 32 = 2720€.

Implementing proposal Y will result in increased unit yield per month: 160h / 4h = 40, which is equal to market’s demand, but would reduce margin by 30€/unit, for a total of -5€ (a loss) for each unit sold.

Implementing both proposals would result in a yield of 85€ - 30€ = 55€ per unit, and a total number of 40 units produced per month, for a monthly margin of roughly 55€ \* 40 = 2200€.

If the company does have the funds required to do so, solution X should be implemented as soon as possible; the investment would be covered in less than three months ( 3 \* (2720€-800€) = 5760€).

Proposal Y should only be implemented along proposal X if:

* An increased number of units produced per month could help remove competitors from the market, or improve the company’s stock value.
* The product selling price was increased:
  + Increasing the selling price by at least 65 dollars would make implementing both solutions more or as profitable. Margin per month would become:   
    Only X: 4800€. Only Y: 2400€. Both: 4800€  
    With the margin for X+Y rising faster as the price increase.

# Digital Signal Processing

What are the 2 fundamental digital blocks for a digital downconversion?

Using a tool of choice (Matlab, Python, LabVIEW, Online) design an efficient filter for a digital down conversion from 100MSPS to 2MSPS. Provide a description of the design choices, for example FIR, IIR, Number of stages, architecture with or without CIC.

What is the reason why the bitness (i.e. 16Bit) from input to output can increase ( i.e. 18 Bits) ?

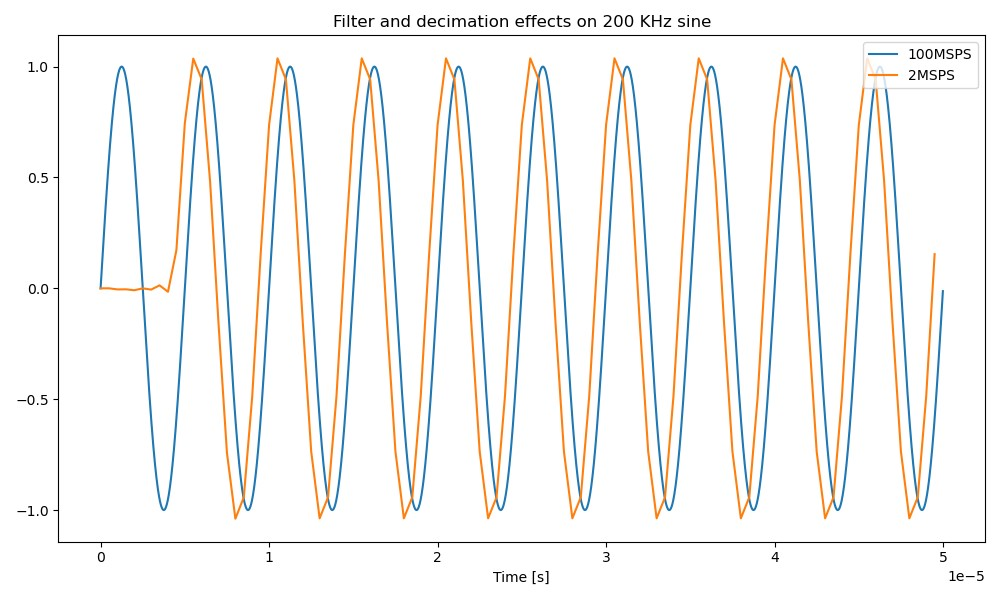
**Folder: EX4 – Digital Signal Processing  
Content: Filter\_Classes.py (classes for CIC and digital FIR), Filter\_Simulation.py (simulation of the filters frequency response), Main (filter + decimators implementation)**

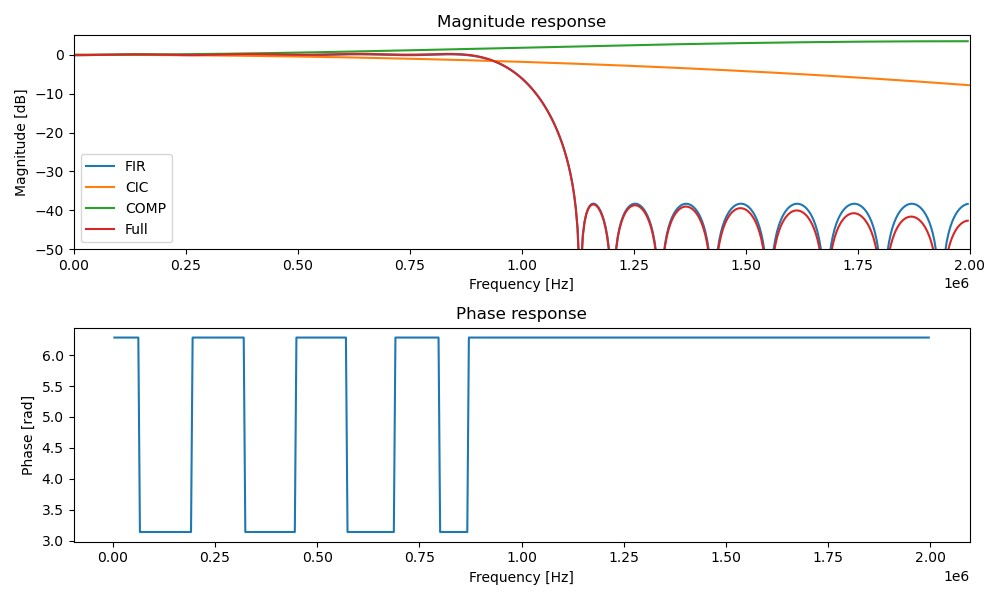
The fundamental digital blocks for a digital downconversion are filter and decimator. The decimator removes samples in order to obtain the required sampling rate, while the filters reduce aliasing effects of this operation.

The provided filter, implemented in python, makes use of three stages. The first is a CIC filter, composed of an integrator, a factor 25 decimator and a comb with delay = 1; the second a FIR CIC compensator to improve pass-band; the third an half band FIR low-pass filter, which is followed by a second, factor 2 decimation.  
Since the conversion requires a factor 50 decimation, a simple FIR would not be sufficient to efficiently filter the data. Because of this, the signal is first decimated using and efficient, low order CIC filter, then compensated for better passband and finally filtered with a half-band FIR.

See the following page for filter response and a filter + decimation example.

The reason bitness required to preserve data can increase, is due to the presence of the CIC filters. The integrator used before decimation and combing have unity feedback, as such output data can have a maximum gain factor equal to (N)M, where M is the CIC’s order and N is the combs’ delay. Since combs shall be implemented after the decimator, N = R\*n, where R is the CIC’s decimation factor and n is the comb delay (usually, for big R, this number should be equal to 1 or 2).  
Because of that, the original data bitness can increase by up to ceiling(Mlog2(Rn)). For example, in this exercise’s filter, since M = 1, R = 25 and n = 1, this value is equal to ceiling(log2(25)) = 5.  
This is only true for data store in a signed, complement of 2 data format (IE int, double…) and not for float.





# Data Analysis and presentation and code

Given the data provide in the stocks\_data.csv use pandas matplotlib and/or plotly to create a chart showing average, min, max price of AAPL over the latest 12 months.

Provide the code to run and create the plot.

If you don’t have python installed, you can leverage colaboratory

<https://colab.research.google.com/notebooks/intro.ipynb?utm_source=scs-index>

**Folder: EX5 – Data Analysis  
Content: AAPL.py (original version of the script, using yfinance to pull AAPL data from the internet), AAPL\_csv.py (main script used to generate the image)**

The plots are attached below. Please refer to the scripts for further info. The exercise was executed twice, once with the provided .csv file and, before receiving it, by downloading data via yfinance package. The second plot is relative to the csv file.

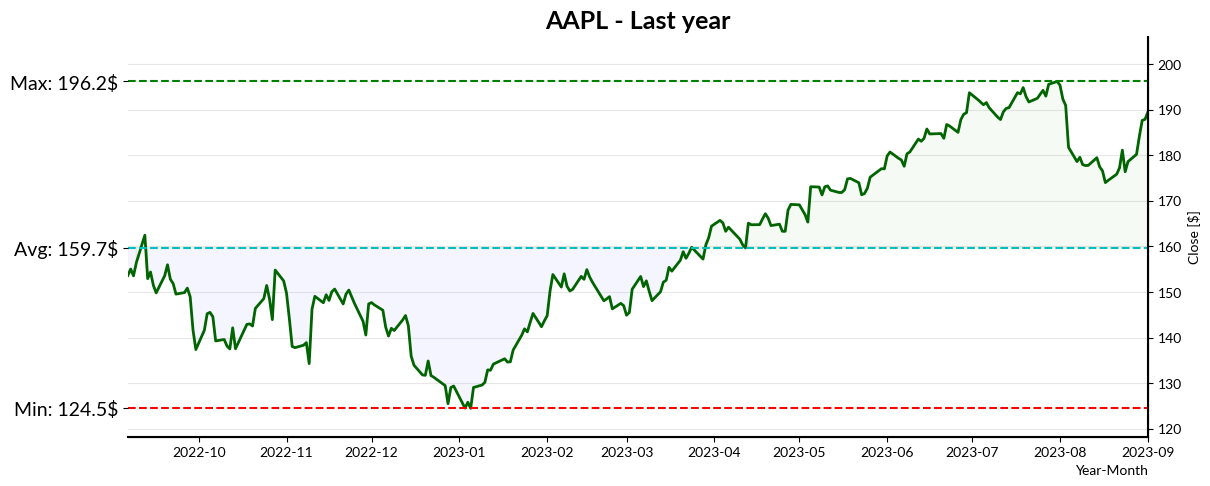


Immagine che contiene testo, linea, Diagramma, Carattere

Descrizione generata automaticamente

# Exercise on EYE diagram an data analysis

Attached is an NRZ trace (Trace\_Y\_After\_FIR.zip) in json format of a pseudo-random electrical signal (PRBS9) with 20 samples per symbol.

It is required:

* Make a plot of the whole trace
* (Eye diagram) Make a plot of the eye of the electrical signal
* Find mean value and variance of crossing times around y=0, using the previously created plot (Eye diagram)
* (Jitter) Plot the 50 bin crossing time distribution on a symbol time

**Folder: EX6 – EYE diagram  
Content: EYE.py (main script)**

Electrical data is read from JSON, then binned in an array of size 42 arrays (2 UI + 2 points for better visualization, padding points are repeated in other traces). Data is plotted (eye diagram) and fitted with a smooth sp-line; crossing points are searched with a simple zero-method function. Data is then replotted on a single UI timescale and crossing point for final analysis: both crossing points are part of the same population, but eye diagram technically requires a 2 UIs long plot. Last two graphs are histogram plots of the crossing point distribution. Average and variance can be found in the last graph.

Execute the script to visualize the data in standard pyplot GUI.

**Immagine che contiene testo, diagramma, schermata, Diagramma

Descrizione generata automaticamente**

# Generate clock and calculate jitter

Generate a clock source f = 100MHz and add sinusoidal phase noise to the clock with f\_noise = 1MHz. Calculate the jitter of the clock affected by noise.

**Folder: EX7 – Clock and Jitter  
Content: Clock generator.py (main script)**

Generated noisy clock with formula , then plotted and analyzed similarly to previous exercise. In this example, A = Anoise = 1. The jitter distribution is the typical “drawbridge” shape for periodic sinusoidal jitter (Sj) [11].

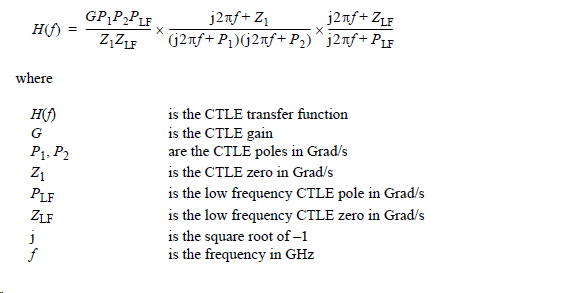
Expected pk-pk jitter is equal to the delay imposed by maximum phase deviation. Let’s take in consideration a time near a crossing for which . The signal would have value: . There will be a crossing when , instead of when . So, in the first case we getand for the second. The difference is thus half the pk-pk jitter, so *.*In this simulation, calculated Sjpp would be ns, which is compatible with the simulation’s value of 3.226 ns.

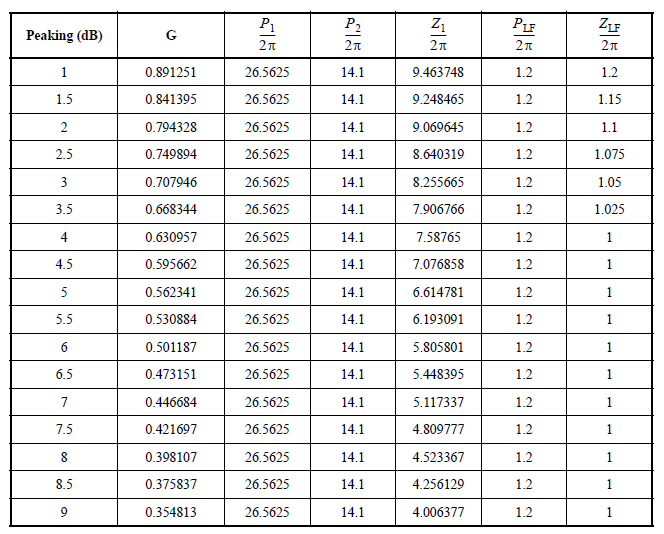
Immagine che contiene testo, schermata, diagramma, Diagramma

Descrizione generata automaticamente

# Data signal Processing with real Channel and equalization

Given the channel representation with S-parameters included in the folder (tx\_test\_ficture.s4p) and the CTLE (Continuous Time Linear Equalizer) equation below. Find the best CTLE configuration among those in the table below that best equalize the channel.





**Folder: EX8 – Real Channel Equalization  
Content: BODE\_FUN.py (method to calculate bode plots, and script to generate the bode plot of the full channel), CTLE.py (main script for real channel simulation), CTLE\_CLASS.py (a class describing the CTLE filter using ‘control’), CTLECHECK.py (a check of the correctness of the simulated filter: generates the magnitude bode plot in a format similar to IEEE 802.3)**

Assuming the attached s-parameter represent a two port differential channels (and not a four port channel), the skrf python package was used to import the s-parameters from the .s4p file, and switching to a mixed mode parameters representation. Finally, assuming purely resistive behavior for the DC component, the s-parameters were padded for DC.

The CTLE filter was implemented using the control python package, using the provided equation. Result was double checked versus the original from IEEE 802.3 by plotting it in the same way it is visualized in the document. This CTLE in particular is used to equalize PAM4 53.125 Gb/s, or 26.5625 GBd. Characteristic frequency is thus equal to the half the baudrate.

For two port systems, when each side’s load is equal to the system’s, the following is true:

where x, y are the port’s indexes [13]. Assuming no voltage is applied on port 2, the system’ transfer function is equal to the s-parameters value for each frequency [12]-

In order to apply the filter, the two transfer functions are multiplied, then Bode plots are drawn. The result is as follow:

Immagine che contiene diagramma, testo, Piano, schematico

Descrizione generata automaticamente

The red vertical lines are the Nyquist frequency of the expected signal. The ideal response would be a flat line until and around the Nyquist frequency, to reduce artifacts, so either peaking 1.5 or 2.0 would work best.

In order to better visualize channel effects, channel + CTLE impulse response was convoluted with a generated PRBS13 PAM4 signal, see below:

Immagine che contiene testo, schermata, Blu elettrico, primavera

Descrizione generata automaticamente

The values inside the plots are the estimated eye heights. Again, this confirms that for a PAM4 26.5625 GBd signal the ideal equalization would be obtained for peaking of 1.5 or 2.0, with 1.5 being more symmetrical and open.

1. Contribution margin defined as (price of product sold – price of raw materials for production). Is basically the pure gross margin on product sold without taking into account operational costs. [↑](#footnote-ref-1)