University of Cape Town

Department of Electrical Engineering

EEE4114F - Digital Signal Processing

Hardware Implementation of Online Filters Project

Caide Spriestersbach & Sebastian Haug

May 2023

Introduction **2**
Background **3**
Filters 3
Finite Impulse Response Filters: 3
Infinite Impulse Response Filters: 3
Discrete Fourier Transform Filters: 4
Method **4**
FIR Filter Implementation: 4
IIR Filter Implementation: 5
DFT filter implementation: 6
Results **7**
FIR filter 11
IIR filter 12
DFT filter 13
Conclusion **15**
Bibliography **16**

Introduction

The following report will investigate the implementation of three different types of filters on an ESP32 Cam board, to investigate which of the filters performs the best on a floating point hardware. The three types of filters tested were the Infinite Impulse Response (IIR) filters, Finite Impulse Response filters, and lastly the Discrete Fourier Transform (DFT) was performed on the signal, low pass filters were then applied in the frequency domain and the result was Inverse Discrete Fourier Transformed (IDFT) to provide a result. In each of the following cases, the low pass filters which were applied for each type of filter had a cutoff frequency of half the sampling frequency. All of the tests made use of the onboard 11-bit ADC on the ESP32 Cam board.

In real world applications, filters are important since they have the ability to remove unwanted noise, enhance signal quality, and have the capacity to improve the accuracy and reliability of data collected from sensors or other sources. Some possible applications for real-time filtering on an ESP32 Cam board is image processing, as real-time filtering can be used for enhancing video and image quality, this can be done by reducing noise and is important in surveillance. Sensor data collection is another application and filtering can improve the accuracy of the readings as well as remove noise. In audio processing, real-time filtering can be used to reduce noise and is used in equalisation.

The ESP32 Cam board was chosen because it is low-cost, can take images as well contains General Purpose Input Output (GPIO) pins, and is able to read sensor data. It possesses onboard temperature and Hall sensors which can be used for real-time filtering applications as described above. It has 2 high performance 32-bit LX6 CPUs with its main frequency adjustment ranges from 80MHz to 240MHz and has up to 160MHz clock speed. The high performance CPUs are important for filtering applications since these can be computationally intensive and these CPUs can handle these tasks more efficiently allowing for faster and more accurate processing. Time delay is important and high clock speeds ensure that the filtering tasks are executed quickly enough resulting in better responsiveness and more accurate data processing.

Background

Filters

A filter is a frequency selective linear time invariant (LTI) system, which is a system which allows certain frequencies to be passed by the system while others are rejected. In this application, the LTI systems which have a Linear Constant Coefficient Differential Equation representation and are causal are of importance.

Finite Impulse Response Filters:

A finite impulse response (FIR) filter is a type of digital filter that operates by convolving the input signal with a finite impulse response function, which is a sequence of finite duration [1]. The impulse response function is usually designed as a series of weighted coefficients that determine how the input signal is filtered. FIR filters have several desirable properties, such as linear phase response, are always stable – system function contains no poles, and the ability to achieve a sharp cutoff in the frequency domain. These filters are very simple to implement, and all digital signal processors have architectures that are suited to FIR filtering. These types of filters are commonly used in hardware applications where linear phase and low computational complexity are required such as audio and video processing, speech recognition, medical signal processing, and control systems.

Infinite Impulse Response Filters:

An Infinite Impulse Response (IIR) filter is a type of digital filter used for processing digital signals. It's called "infinite impulse response" because its impulse response lasts indefinitely. In other words, when an impulse is applied to the input of an IIR filter, the output will continue to oscillate forever. To implement such a filter using a FIR structure therefore requires an infinite number of calculations. An IIR filter uses feedback to achieve its filtering characteristics. This means that the output of the filter is fed back into the input, creating a feedback loop. The feedback loop allows an IIR filter to achieve high levels of attenuation reduction in a narrow frequency range, making it useful for applications such as audio equalisation, noise reduction, and signal conditioning. IIR filters are commonly used in audio and digital signal processing applications because of their ability to provide high performance filtering with fewer computational resources compared to other filter types, such as finite impulse response (FIR) filters. However, IIR filters can also be more complex to design and implement than FIR filters, as they require careful consideration of stability and

causality. These types of filters are commonly used in audio processing, image processing, and communication systems.

Discrete Fourier Transform Filters:

A crucial application of the Discrete Fourier Transform (DFT) is computing the convolution of signals in the time domain [2]. This involves multiplying the DFT representations of the two signals and then calculating the inverse DFT of the resulting product. This can be used in filtering as we will DFT the signal which will be filtered, we will then multiply it by the DFT of the frequency response function of the low pass filter. This process is effectively low passing the signal, we will then Inverse Discrete Fourier Transform (IDFT) to return the signal to the time domain. This is a naive approach and in order to compute the DFT, a buffer of samples is required. If this buffer were to be infinite, the filter would exhibit ideal filter properties. However due to the real-time application of this filter a buffer of finite length was used in testing as an infinite buffer size is not realisable, this in turn degrades the frequency response of the filter.

Method

For all of the following filters, the inbuilt clock and timer was used to determine the time taken to sample the signal and the time for computation to be completed, this was averaged over a few iterations. This was done for different sample sizes for the FIR filter and different buffer sizes for all the filters. The total time taken from input to output was also recorded and averaged as another measurement to be used for comparison.

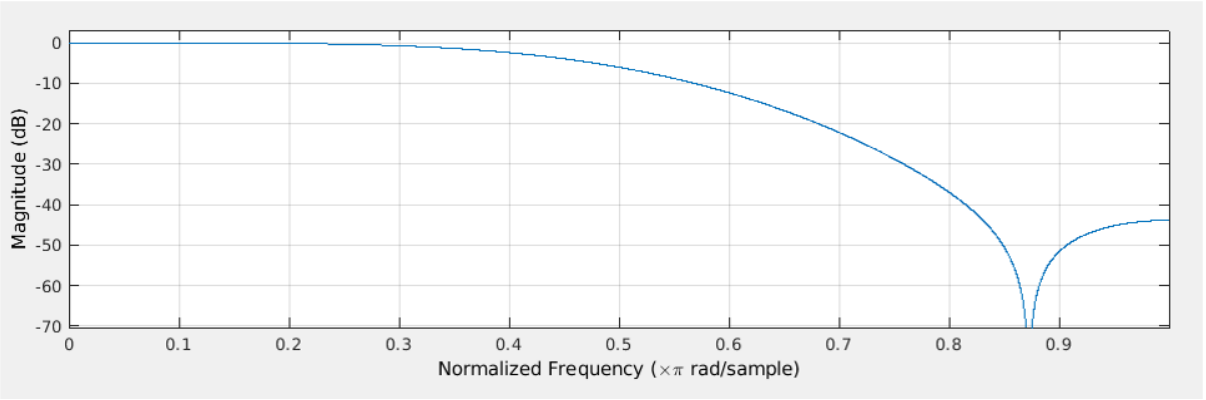
FIR Filter Implementation:

The MATLAB digital filter design tool was used to determine the exact characteristics of the low pass filter to be used on the ESP32 Cam board. The coefficients for the system function were determined using MATLAB and implemented using arduino code to be uploaded to the ESP32 board. The code was initially tested on the arduino IDE before it was implemented on the ESP32 board.

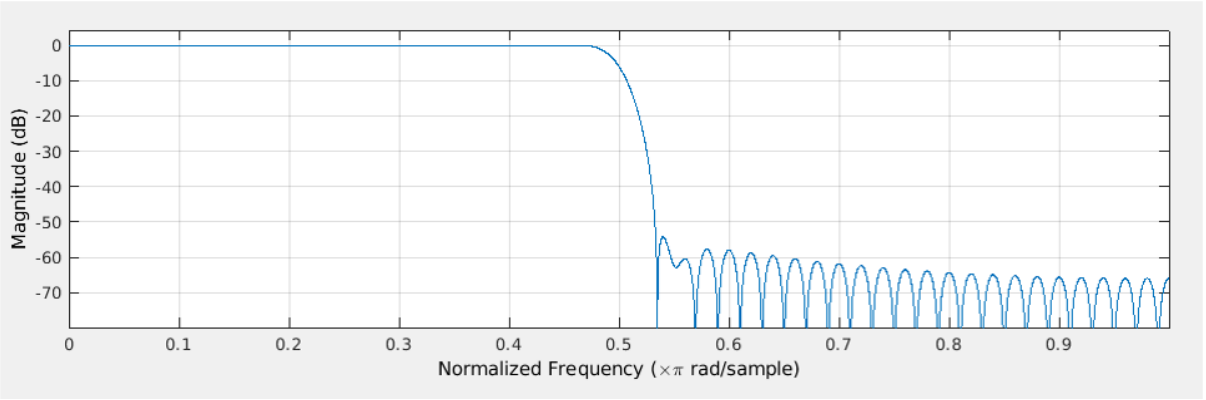
The impulse response characteristics were stored in memory on board the ESP32 board. The discrete time convolution was then performed to apply the filter to the signal. 10 different FIR filters were implemented, each with impulse responses ranging from 10 samples to 100 samples. In order to implement the FIR filter effectively, the windowing method was used.

Since the impulse response in the time domain requires an infinite number of input samples to perform filtering and the system is not causal [3]. We will truncate the impulse response to make it realisable.

The following shows the FIR filter using 10 samples and a Hamming window:



The following shows the FIR filter using 100 samples and a Hamming window:



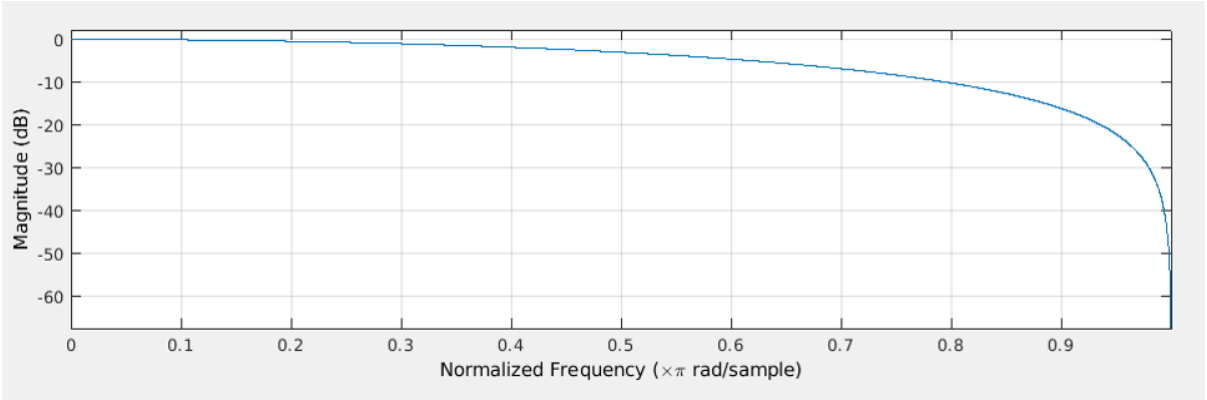
IIR Filter Implementation:

Same as the FIR filter, the MATLAB digital filter design tool was used to determine the exact characteristics of the low pass filter to be used on the ESP32 Cam board. The coefficients for the system function were determined using MATLAB and implemented using arduino code to be uploaded to the ESP32 board. The code was initially tested on the arduino IDE before it was implemented on the ESP32 board.

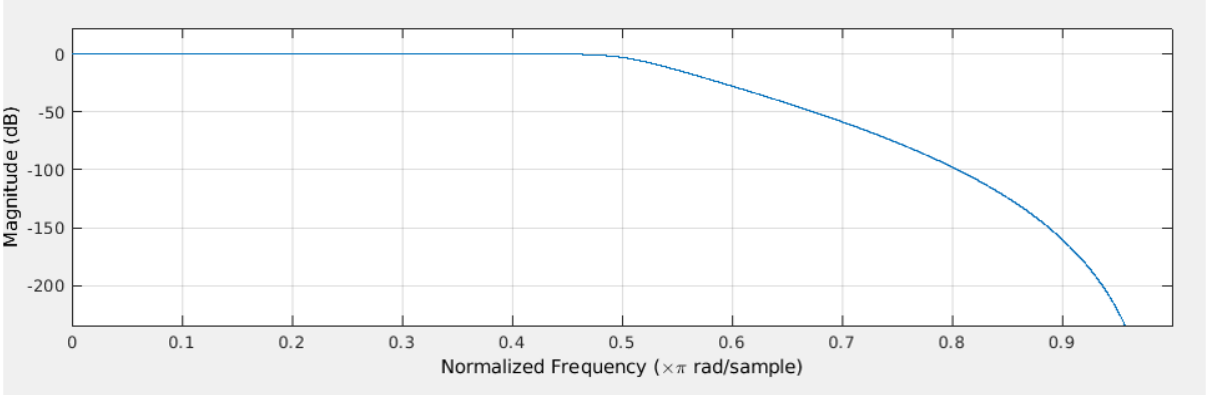
However, since the IIR filter does not require a buffer it is fully realisable using arduino code. The filter chosen was a Butterworth filter as it produces the best output response due to its

flat response with no ripple in the passband. Different orders of the filter were tested, ranging from first order to 10th order. Each pole adds -20dB per decade to the roll off rate.

The following image is the 1st order Butterworth filter implemented on the ESP32 board:



The following image is the 10th order Butterworth filter implemented on the ESP32 board:



DFT filter implementation:

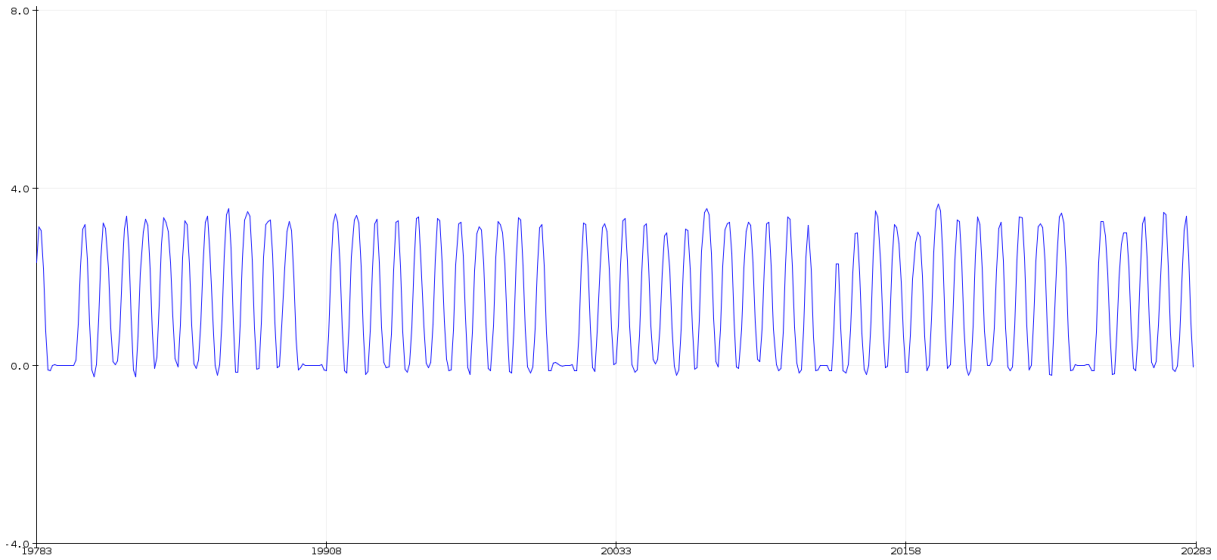
As mentioned above, the filtering in this approach is naive was performed by convolving the impulse response of the filter with the signal. This was realisable by using the DFT of both the signal and the impulse response and multiplying them together. The combined function was then inverse DFT to produce a filtered signal. The impulse response of the low pass filter was calculated using the MATLAB filter design tool and the values provided were stored in an array in the arduino code to be applied to the signals.

Results

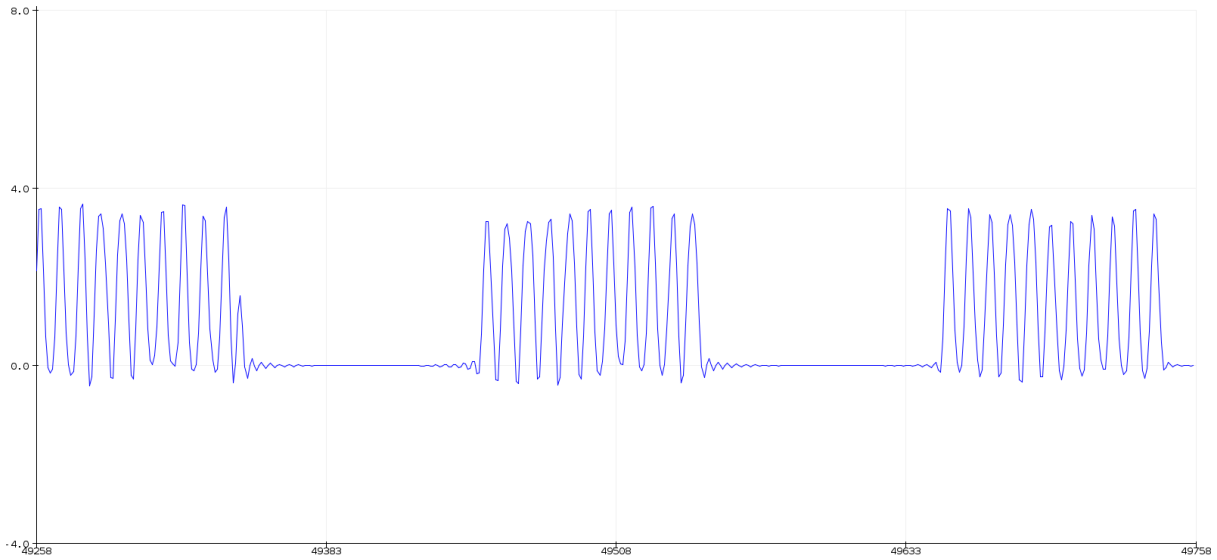
To verify that the implemented filters work as expected, they were tested with a square wave input:

FIR:

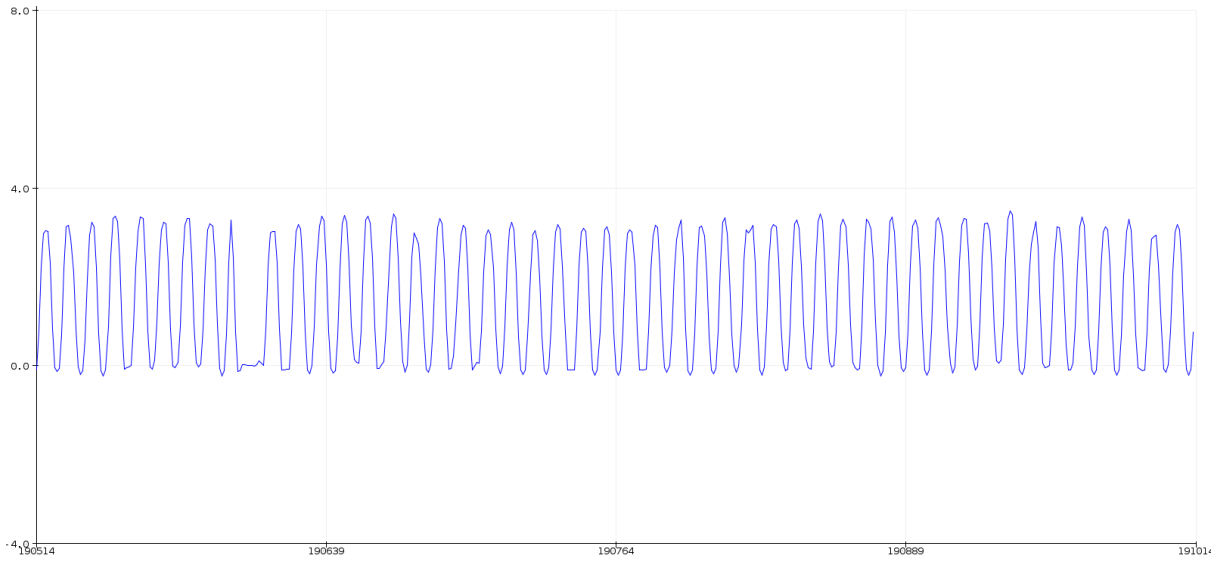
11 Impulse response samples, buffer size 100 at 1.5kHz



101 Impulse response samples, buffer size 100 at 1.5kHz



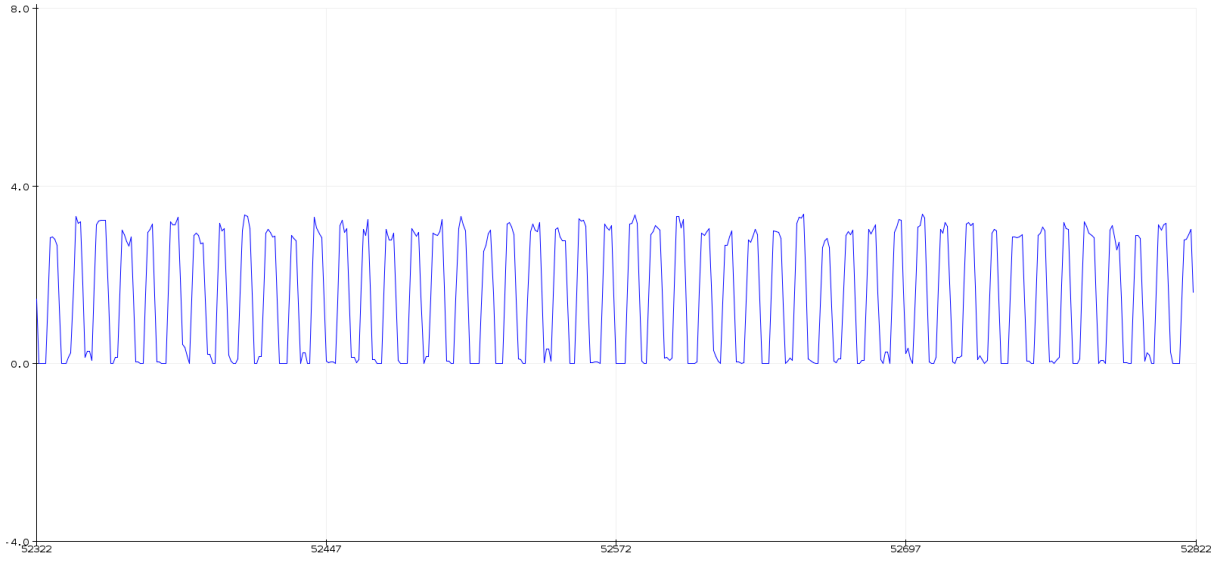
11 Impulse response samples, buffer size 1000, 1.5kHz



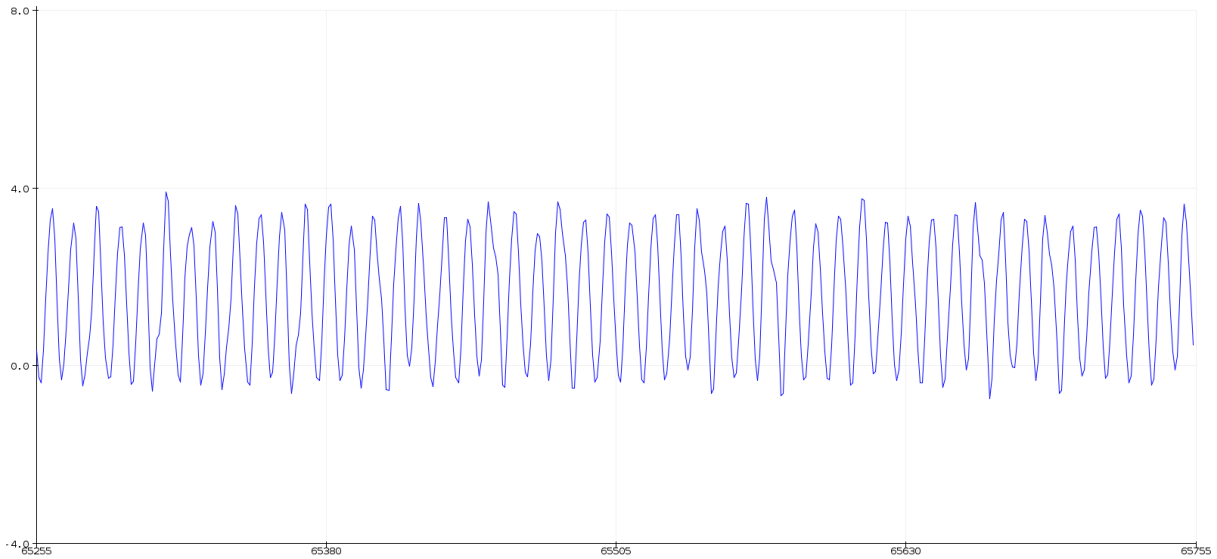
The above plots show that the various filters are working as expected. The square wave is converted to a relatively smooth sin wave. However, since the FIR filter requires a buffer full of samples, some interesting effects arise from the windowing.

IIR:

1 pole at 500Hz

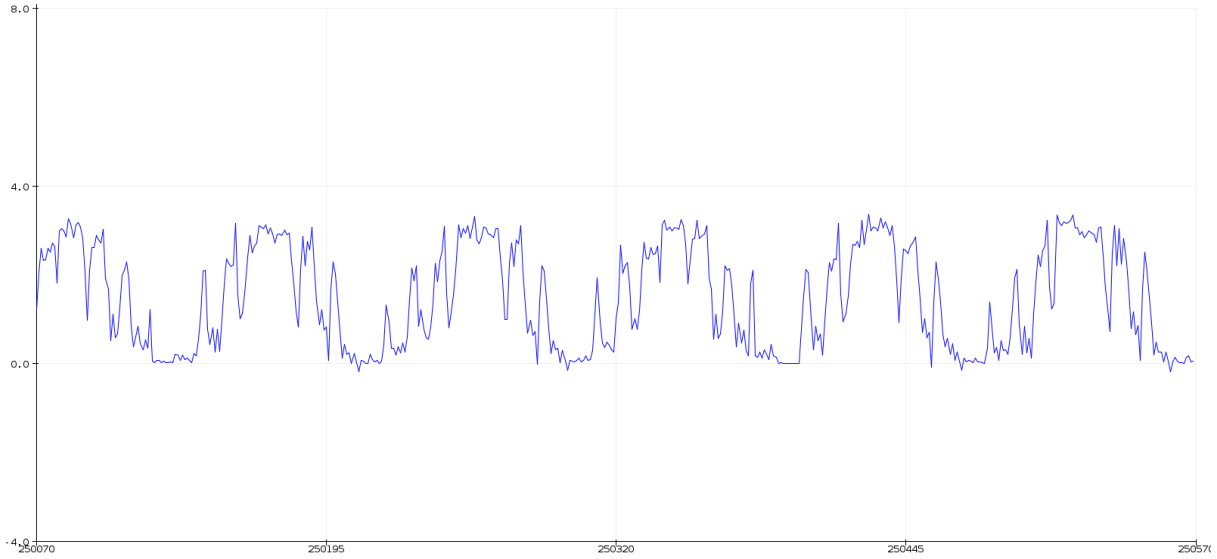


10 poles at 500Hz

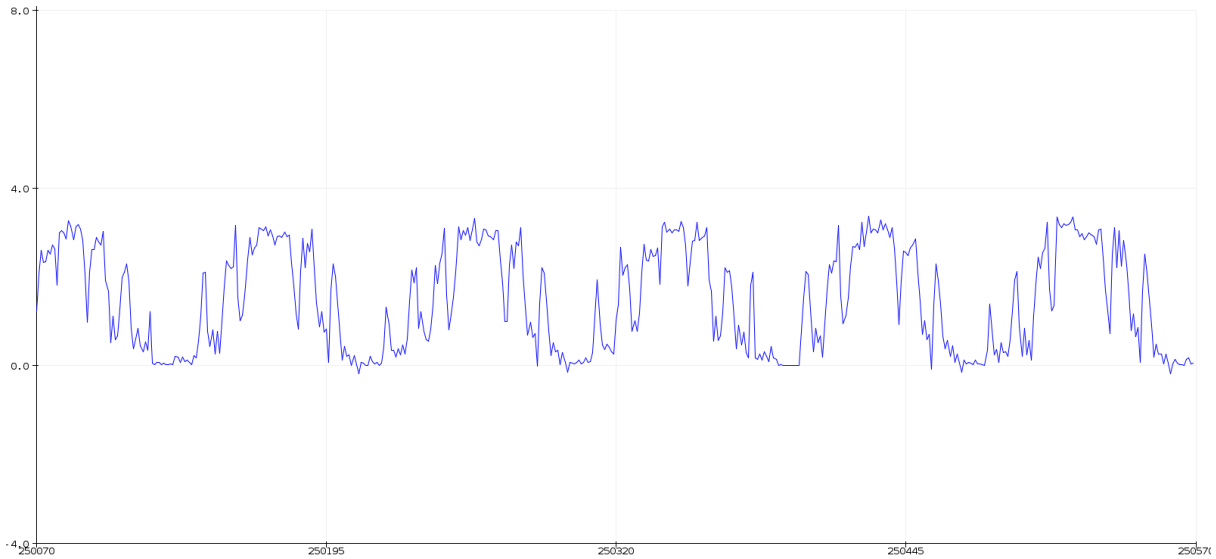


DFT filter:

Buffer size 10, 500Hz



Buffer size 100, 500Hz



This is an incredibly poor filter. While it appears to have a sinusoidal component, the amount of noise added makes it essentially unusable. This is likely due to the fact that the data has a tiny buffer, while the DFT requires an infinite number of samples to produce an accurate result. The DFT and IDFT involves many operations such as applying sin and cos functions, as well as floating point divisions. Since the microcontroller only has a finite floating point resolution, small errors may accumulate throughout the transforms, worsening the result even further.

Interestingly, the filters appear to have significantly different cutoff frequencies, even though they were designed to have the same cutoff. This is likely due to the IIR filter performing calculations per sample, while the FIR performs calculations per buffer. This allows for the FIR filter to sample at the fastest possible sampling rate until the buffer is full, while the IIR filter's sample rate is effectively reduced due to the calculations performed per sample.
Since the cutoff frequency of these filters depends on the sampling frequency, the IIR filter will thus also have a lower frequency stop-band.

Having shown that the above filters mostly function as expected, their performance is evaluated by considering factors such as computation time vs. sampling time, as well as signal propagation delay.
The following is the results of multiple testing rounds for each of the different types of filters as well as the different characteristics which make them up and determine their behaviour.

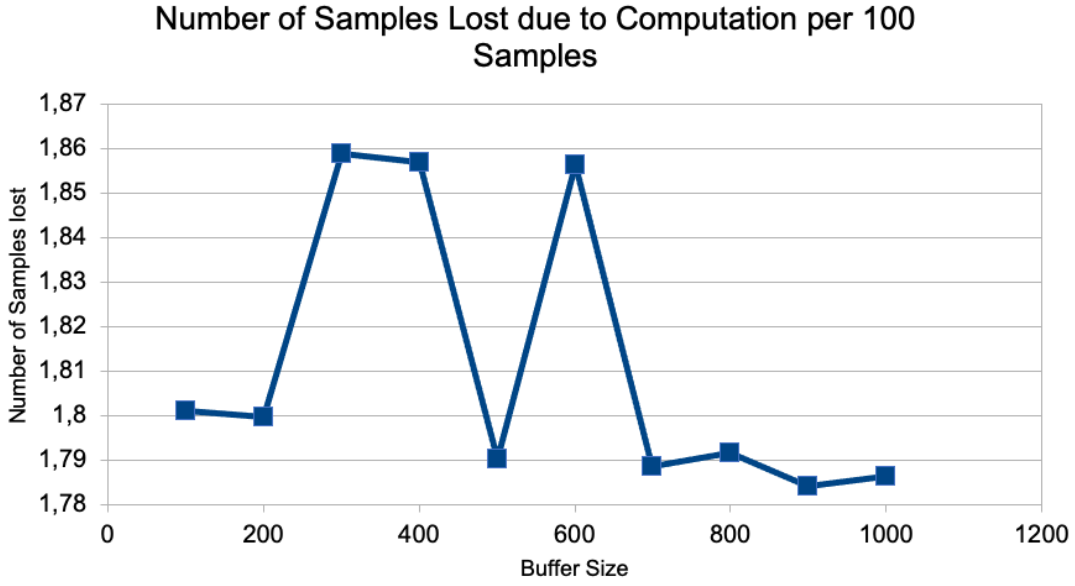
FIR filter

The following results were obtained from the ESP32 Cam board using 11 impulse response samples. The following table shows the different results for different tests performed on the board.

Table showing the various results obtained:

FIR buffer size	Measure time (M)	Compute time (C)	(C / M) * 100	Time Delay
100	7662	138	1.80109631949883	7800
200	15335	276	1.79980436909032	15611
300	22218	413	1.8588531821046	22631
400	29615	550	1.8571669761945	30165
500	38373	687	1.7903213196779	39060
600	44438	825	1.85651919528332	45263
700	53782	962	1.78870253988323	54744
800	61337	1099	1.79174071115314	62436
900	69223	1235	1.78408910333271	70458
1000	76913	1374	1.78643402285699	78287

The following graph displays the number of samples which were lost per each 100 samples of the signal compared to buffer size:



The number of samples lost per 100 samples is relatively low and is contained in a small range between 1,8 samples and 1,86 samples. It can be seen that as the buffer size increases, it increases the time for computation, measuring, and this in turn affects the total overall time between the signal being supplied to the filter and the output being read, known as the time delay. This time delay is an important feature as with real time filtering it is important to have a very small time delay as well as a low loss. It is also apparent that the time taken to compute is significantly less than the measuring time, which means the filter spends most of its total time collecting the samples before it can perform the filtering on them. This is not a good characteristic to have for a real time filter being implemented on hardware, the total time delay is also not good enough for real time filtering.

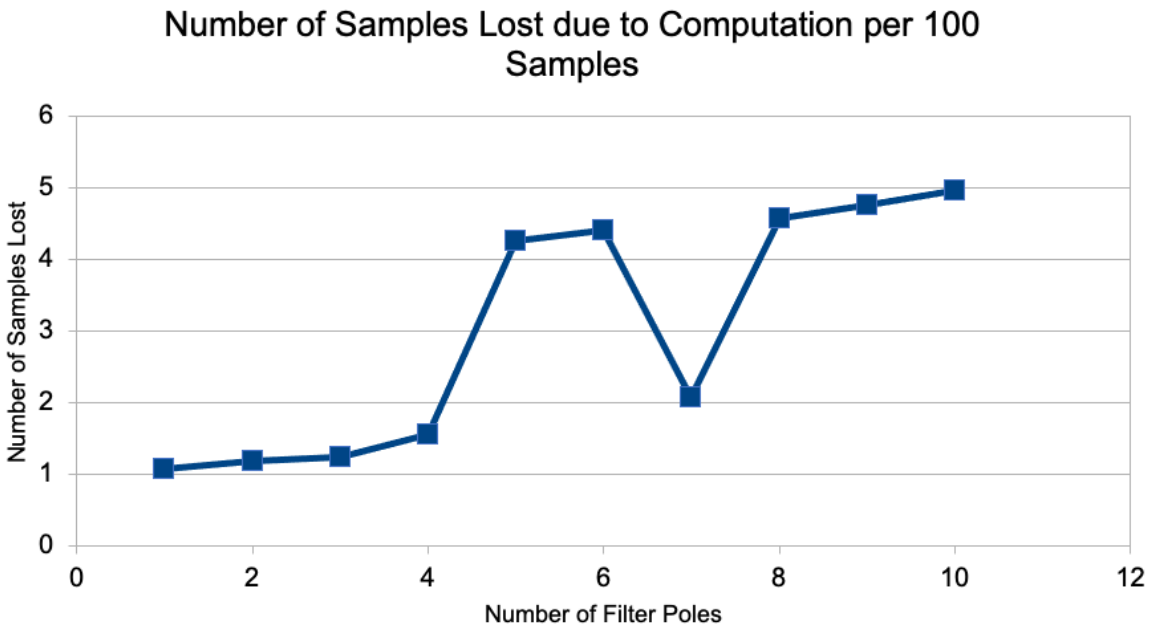
IIR filter

The following test made use of changing the number of poles of the filter to test the different measure time, computation, and total time delay. The following results were obtained by passing a known signal to the ESP32 board and measuring critical data using timing functions inside the code making use of the onboard clock.

Table showing the various results obtained:

Number of Poles	Measure time (M)	Compute time (C)	C / M * 100	Time Delay
1	76.65	0.83	1.08284409654273	77.48
2	76.56	0.91	1.18861024033438	77.47
3	74.34	0.92	1.23755716976056	75.26
4	76.69	1.19	1.55170165601773	77.88
5	80.6	3.43	4.25558312655087	84.03
6	80.54	3.56	4.42016389371741	84.1
7	76.77	1.6	2.08414745343233	78.37
8	81.67	3.74	4.57940492224807	85.41
9	81.35	3.87	4.75722188076214	85.22
10	78.77	3.91	4.96381871270788	82.68

The following graph shows the number of samples lost compared to the number of poles:



It can be seen from the above table, an increase in the number of poles did not significantly impact the measure time as it did not have to sample the signal using a buffer. However, the computation time increased as the number of poles increased due to an increase of the roll off rate of the filter. This increase resulted in a larger time delay when the poles were increased. The number of samples lost per 100 samples increased when the number of poles was increased. The time delay is relatively small and does not fluctuate significantly between the number of poles since most of the time is spent measuring and the difference between computation and measure is negligible. In a real-time filter implementation on hardware it is vital to have a low time delay. The number of samples lost for the lower number of poles is very small and almost negligible.

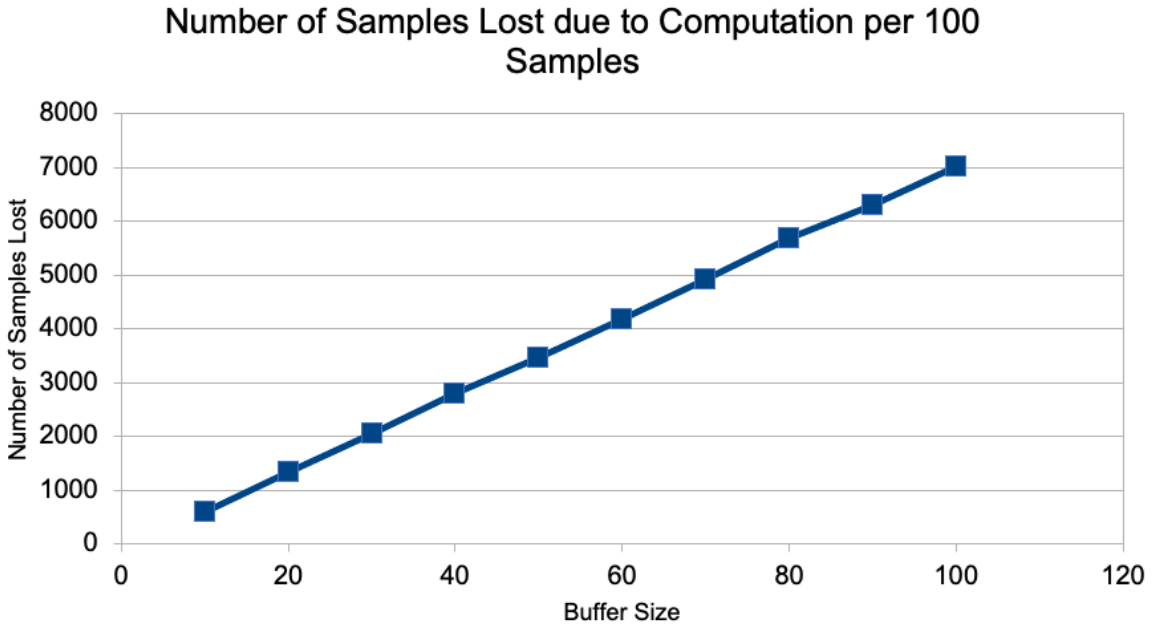
DFT filter

Since the DFT requires a buffer size, different buffer sizes were used to determine the various quantities used to evaluate the performance of the filter being implemented on the ESP32 board.

DFT buffer size	Measure time (M)	Compute time (C)	(C / M) * 100	Time Delay
10	734	4383	597.138964577657	5117

20	1464	19437	1327.66393442623	20901
30	2195	44801	2041.04783599089	46996
40	2928	81554	2785.31420765027	84482
50	3660	127122	3473.27868852459	130782
60	4562	190727	4180.77597544936	195289
70	5124	251949	4917.037470726	257073
80	5855	332908	5685.87532023911	338763
90	6840	430486	6293.65497076023	437326
100	7602	533142	7013.18074191002	540744

The following graph shows how the number of samples lost per 100 samples is affected by the buffer size:



As seen by the above table, both the measuring time and computation time for this filter is extremely long, this is due to the fact that it makes use of a buffer and computing both the DFT and the IDFT is intensive and thus will take longer than the other approaches. As seen by the graph, the relationship between samples lost and buffer size is linear, therefore a

smaller buffer size would be preferable. However for real-time processing applications, the time delay is too long, this is due to the long measuring time and computation time. The number of samples lost is also too much for this filter to be used. The computation time is high relative to the measuring time – takes longer to compute than to sample, this is the reason why a lot of samples are lost.

Conclusion

In the case of the IIR filter, it performs the best in terms of time delay, it has a small sample loss, and a good roll off rate which makes it ideal for the real-time filtering application. However, in the closed loop feedback system, time delays move the system closer to instability. The increase in the delay means the more unstable the system is. However, since the IIR filter does computation on each sample, the sampling rate is more predictable and consistent. This is an important feature in digital filters as the cutoff frequency depends on the sampling frequency, so having a more consistent and predictable sampling rate is more beneficial. Filters which require a buffer may not have a consistent sampling rate and hence the cutoff frequency might not be at the desired frequency, hence the FIR and DFT filters are not desirable.

The DFT filter is the worst choice for a real-time filter as its computation and measuring time are both long resulting in a large delay which is undesirable. The extremely high loss rate of samples makes this form of filtering undesirable for all applications and in particular real-time filtering. As seen in the filter testing section, the DFT filter performs very poorly. The input signal is distorted severely, and little filtering action can be observed. This is the worst choice when it comes to real-time filtering.

The FIR filter has a trade off between time delay and filter roll-off, many impulse response samples cause a larger time delay which is undesirable in real-time filtering. A larger buffer size also increases the time delay, therefore if a FIR filter is to be used a smaller buffer size would be preferable but smaller buffer sizes have shorter windows and the windows edges affect the output, thus resulting in more inaccuracies in the output. The sample loss is small, this is desirable in all filters, thus if this filter is to be used the roll off rate and buffer size must be selected carefully.

Bibliography

[1]B. Meddins, “The design of FIR filters,” *Introduction to Digital Signal Processing*, pp. 102–136, 2000, doi: https://doi.org/10.1016/b978-075065048-9/50007-6.

[2]S. Arar, “Linear Filtering Based on the Discrete Fourier Transform,” *All About Circuits*, Oct. 19, 2017. https://www.allaboutcircuits.com/technical-articles/linear-filtering-based-on-the-discrete-fourier-transform/ (accessed May 15, 2023).

[3]S. Arar, “FIR Filter Design by Windowing: Concepts and the Rectangular Window - Technical Articles,” *www.allaboutcircuits.com*, May 12, 2016. https://www.allaboutcircuits.com/technical-articles/finite-impulse-response-filter-design-by-windowing-part-i-concepts-and-rect/