UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

ESP 411

Computer Engineering

ESP 411 - Practical 3 - DSP Programming and Application

Name and Surname	Student Number	Signature	% Contribution
H. van der Westhuizen	18141235		100.0

September 2, 2021

Contents

1 Introduction

The following document aims to discusses Amplitude demodulation and it's implementation on the STM32F4-DISC board. The DSP board aims to demodulate AM signals between 526.5kHz - 1606.5kHz as this is the frequency range of the MF-AM broadcasting band. The MF-AM broadcasting band is broken up into 120 channel with each having a 9kHz bandwidth.

Furthermore, the DSP board emulates channel selection (tuning) be applying a digital band-pass filter centred around the desired carrier frequency (channel) within the MF-AM band. Moreover, after applying the digital filter to the modulated signal the signal is demodulated and outputted to the DAC, which is connected to a audio device such as a speaker.

The practical requirements also state that the FFT of the filtered and unfiltered modulated signal should be displayed on the DSP's LCD screen after the signal has been sent through a Anti-Aliasing Filter.

The students are expected to meet all of the requirements as all stages are required for AM demodulation as well as clear audio output.

2 Theoretical Background

2.1 Amplitude Modulation

Amplitude modulation is achieved by mixing a Radio-frequency (RF) carrier signal with a lower frequency modulated message signal. The output of the mixer yields a modulated RF carrier signal; with it's envelope matching the shape of the original modulating message signal.

Amplitude modulation is mathematically achieved by changing the amplitude of the carrier signal according to the amplitude of the message signal. The message signal as well as the carrier signal is expressed as follows:

$$m(t) = A_m \cos(2\pi f_m t) \tag{1}$$

$$c(t) = A_c \cos(2\pi f_c t) \tag{2}$$

Modulation can only take place if $f_m < f_c$ since the carrier signal must be modulated by the message signal. Mixing the two signals together results in a modulated carrier signal which is expressed in the equation below.

$$s(t) = (A_c + A_m \cos(2\pi f_m t)) \cos(2\pi f_c t) \tag{3}$$

After analysing Equation 3 it is clear that the amplitude of the carrier signal is increased or decreased by the amplitude of the message signal.

Figure 3 graphically demonstrates the modulation of a specified carrier and message signal.

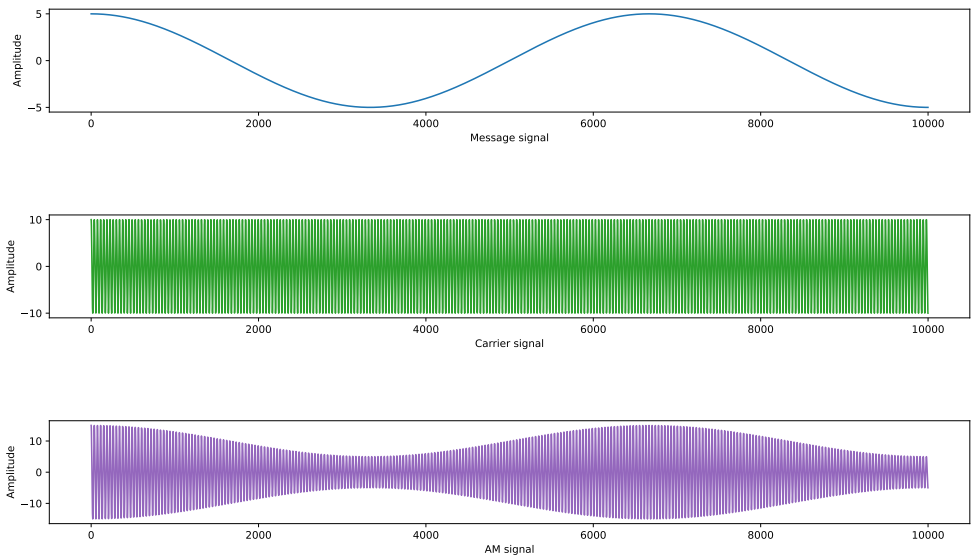


Figure 1: Graphical demonstration of Amplitude Modulation

The main and most important characteristic of an AM signal is its modulation index (m).

$$m = \frac{A_m}{A_c} \tag{4}$$

If the message signal's amplitude is larger than the carrier's amplitude the modulation index > 1 which causes a 180° phase shift whenever the zero point is cut, this is called over-modulation. Ideally the modulation index should be between 0.5 and 1 to ensure the signal is not easily distorted by noise since the amplitude difference between the maximum and minimum amplitude (peak and saddle) should be relatively large.

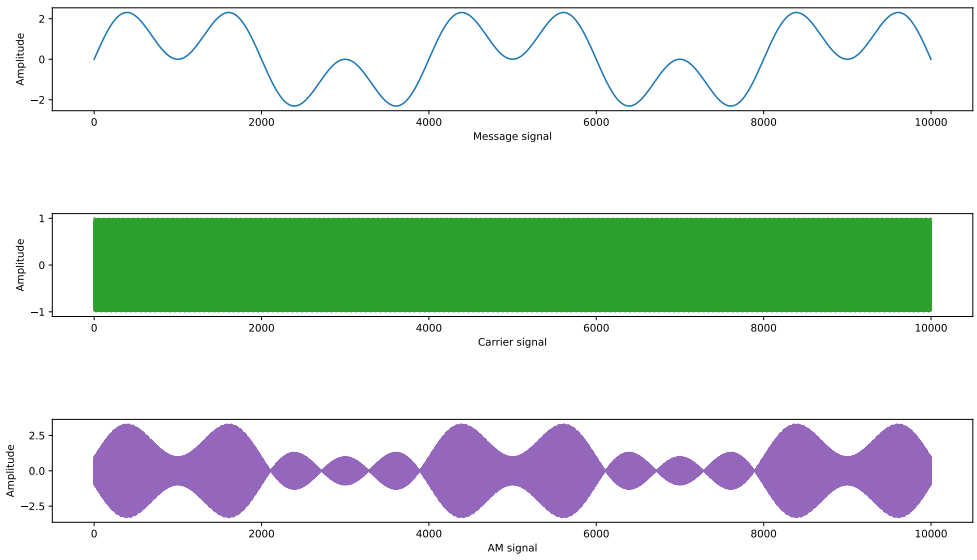


Figure 2: Graphical demonstration of over-modulation

2.2 AM broadcasting

As stated earlier AM broadcasting consists of a specified RF carrier and a modulating message signal. The message signal basically rides on the carrier signal, this allows the transmission of a message over a specified distance. According to the International Telecommunication Union (ITU) medium-wave broadcasting is used to transmit AM signals and operates from 531kHz - 1.602MHz. The frequency band is broken up into 120 channels which allows for 9kHz bandwidth for each channel.

AM signals contain a minimum of three frequency components f_c, $f_c - f_m$ and $f_c + f_m$. If this is not clear one can simply expand Equation 3 into the form below.

$$s(t) = A_c \cos(2\pi f_c t) + \frac{A_c m}{2} \cos[2\pi (f_c + f_m) t] + \frac{A_c m}{2} \cos[2\pi (f_c - f_m) t] \tag{5}$$

Equation 5 shows that the bandwidth equals $2f_m$ and that the amplitude of the edge components are $\frac{A_m}{2}$. Since each channel may only have a bandwidth of 9kHz it is important to attenuate any message signals above 4.5kHz.

2.3 Amplitude Demodulation

Amplitude demodulation aims to extract the message signal from the modulated signal. Analogue circuits preform demodulation by detecting and outputting the envelope of the modulated signal over a continues time interval. One example of an analogue demodulation circuit is represented in Figure 3, it makes us of a diode which only allows the current to flow in one direction which removes the components below 0V.

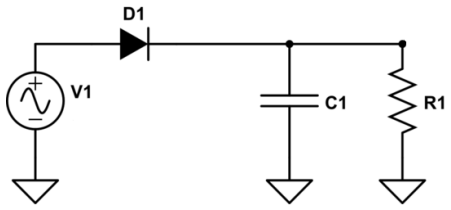


Figure 3: Analogue demodulation

Analogue circuits can easily extract the message signal from the original message since it can sample in the continues time domain whereas digital demodulation implementations must extract the message in the discrete time domain. Digital demodulation becomes more difficult as the carrier frequency increases since the sampling rate of the demodulator must at least be twice the frequency of the carrier.

Due to the fact that demodulation is simple to preform if an analogue circuit is used, digital implementations are not common and are not widely documented, thus the author must design their own to meet the requirements of practical 3.

2.4 Filter specifications

Since the frequency band consists out of 120 channels a digital band-pass filter must be used to select the desired channel which has a specified carrier frequency. For this practical the FIR and IIR digital filters are tested to see if they can adhere to all the requirements.

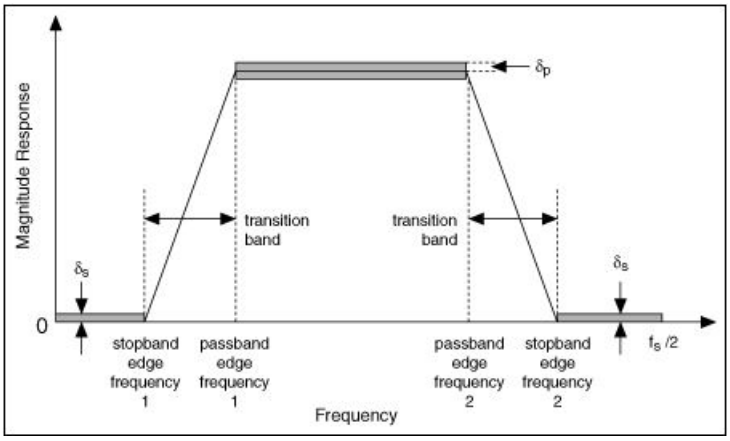


Figure 4: Band-pass Transfer Function.

Relative Frequency (f/F)	Frequency difference (f) from the centre frequency at different channel bandwidths (F) (kHz)				Relative level (dB)
	F = 4,5	F = 5	F = 9	F = 10	
±0,1	0,45	0,5	0,9	1	0
±0,5	2,25	2,5	4,5	5	0
±0,7	3,15	3,50	6,3	7	-35
±1,4	6,3	7	12,6	14	-47
±2,8	12,6	14	25,2	28	-59
≥ ±2,952	13,28	14,76	26,57	29,52	-60

Table 1: ETSI attenuation specifications

The filter specifications can be calculated by making use of Figure 4, Table 1 and the guidelines within [1]. Since each channel has a bandwidth of $9kHz$ the minimum attenuation must be 35dB $6.5kHz$ away from the specified channel's center (carrier frequency). Moreover, a minimum attenuation of 60dB must be achieved 26.57kHz away from the channels center frequency. These specification are required when sending and receiving signals, thus the audio message signal should be attenuated using a low-pass analogue filter and the modulated signal should be digitally filtered.

2.5 Finite Impulse Response Filter

The first FIR filtering method makes use of the window function. There are five window functions Rectangular, Hanning, Hamming, Blackman and Kaiser. Each of these functions have different characteristics as indicated within Table 2 and [2].

Name of window function	Transition width (Hz) (normalized)	Passband ripple (dB)	Main lobe relative to side lobes (dB)	Stopband attenuation (dB) maximum	Window function w(n), $\lvert n\rvert \leq \frac{N-1}{2}$
Rectangular	$\frac{0.9}{N}$	0.7416	13	21	1
Hanning	$\frac{3.1}{N}$	0.0546	31	44	$0.5 + 0.5\cos\left(\frac{2\pi n}{N}\right)$
Hamming	$\frac{3.3}{N}$	0.0194	41	53	$0.54 + 0.46\cos\left(\frac{2\pi n}{N}\right)$
Blackman	$\frac{5.5}{N}$	0.0017	57	75	$0.42 + 0.5\cos\left(\frac{2\pi n}{N-1}\right)$ $+0.08\cos\left(\frac{4\pi n}{N-1}\right)$
Kaiser	$\frac{2.93}{N}(\beta = 4.45)$	0.0274		50	$\frac{I_0\left(\beta\left\{1-[2n/(N-1)]^2\right\}^{1/2}\right)}{I_0(\beta)}$
	$\frac{4.32}{N}(\beta = 6.76)$	0.00275		70	
	$\frac{5.71}{N}(\beta = 8.96)$	0.000275		90	

Table 2: The main window functions to be considered and their respective characteristics

Before continuing, some of the parameters must be discussed. N is the number of coefficients that the filter requires to function as expected, this is normally a digital filter's limiting factor. If N becomes to large the amount of time needed for process each set of coefficients become to much. This causes discontinues output when measuring a continues signal such as a sound wave. N can be calculated using the desired attenuation and the normalised transition frequency as shown below.

$$N \geqslant \frac{A - 7.95}{14.36\Delta f} \tag{6}$$

The Kaiser function makes use of β and the value of β changes as the desired attenuation increases.

$$\begin{array}{ll} \beta = 0 & \text{if } A \leqslant 21 \text{ dB} \\ \beta = 0.5842(A-21)^{0.4} + 0.07886(A-21) & \text{if } 21 \text{ dB} < A < 50 \text{ dB} \\ \beta = 0.1102(A-8.7) & \text{if } A \geqslant 50 \text{ dB} \end{array} \tag{7}$$

Considering the specification stipulated in Filter specifications section it is clear that the rectangular window method can not be attenuated by the required 35dB. Furthermore, the Hanning and Hamming function can also be excluded since they can not reach the required 60db attenuation 26.57kHz from the channels center. This leaves the Blackman and Kaiser window functions. The Blackman and Kaiser functions will be simulated and checked for compliance.

Moving on from the window method, the Optimal method follows. This method acts as a neural network as it tries to minimise the error between a calculated and desired response

by changing the coefficients until the error is small enough. The Remez Exchange algorithm can be used to minimise the error function:

$$E(\omega) = W(\omega)\left[H_0(\omega) - H(\omega)\right] \tag{8}$$

The approximate number of coefficients the Remez Exchange algorithm requires can be calculated by using the equation below.

$$N \simeq \frac{C_\infty\left(\delta_p, \delta_s\right)}{\Delta F} + g\left(\delta_p, \delta_s\right)\Delta f + 1 \tag{9}$$

$$C_\infty\left(\delta_n, \delta_s\right) = \log_{10}\delta_s\left[b_1\left(\log_{10}\delta_\mathrm{p}\right)^2 + b_2\log_{10}\delta_\mathrm{p} + b_3\right] + \left[b_4\left(\log_{10}\delta_\mathrm{p}\right)^2 + b_5\log_{10}\delta_\mathrm{p} + b_6\right] \tag{10}$$

Where:

$$g\left(\delta_\mathrm{p}, \delta_\mathrm{s}\right) = -14.6\log_{10}\left(\frac{\delta_\mathrm{p}}{\delta_\mathrm{s}}\right) - 16.9$$

$$b_1 = 0.01201 \quad b_2 = 0.09664$$

$$b_3 = -0.51325 \quad b_4 = 0.00203$$

$$b_5 = -0.5705 \quad b_6 = -0.44314$$

This method does show promise but it may take too long since the number of coefficients and their calculation will take up valuable processing time.

2.6 Infinite Impulse Response

Before one can explain how the IIR filter works one must first stipulate it's characteristics and how it's recursive output is represented.

$$y(n) = \sum_{k=0}^{\infty} h(k)x(n-k) = \sum_{k=0}^{N} b_k x(n-k) - \sum_{k=1}^{M} a_k y(n-k) \tag{11}$$

The function $h(k)$ is the impulse response and is given in the form below.

$$H(z) = \frac{b_0 + b_1 z^{-1} + \ldots + b_N z^{-N}}{1 + a_1 z^{-1} + \ldots + a_M z^{-M}} \tag{12}$$

When evaluating Equation 12 it is clear that the poles as well as zero values must be calculated. Once this is done the recursive realisation structure can be constructed.

There are many different filter realisation structures such as the Butterworth, Chebyshev I,Chebyshev II, Elliptic and Bessel. Each of these structures rely heavily on their analogue counterparts. Butterworth only consists of poles that are cascaded according to it's order. An n_th order Butterworth filter has the following transfer function.

$$H(s) = \frac{1}{(s - p_n)^n} \tag{13}$$

Using the z transform as well as the sampling frequency f_s one can obtain s.

$$s = 2f_s\frac{1 - z^{-1}}{1 + z^{-1}} \tag{14}$$

The analogue Butterworth structure is calculated as follows:

$$\begin{aligned} p_{ak} &= -\sin(\theta) + j\cos(\theta) \\ \theta &= \frac{(2k-1)\pi}{2N}, \quad k = 1:N \end{aligned} \tag{15}$$

$$F_c = \frac{f_s}{\pi}\tan\left(\frac{\pi f_c}{f_s}\right) \tag{16}$$

Rewrite the realisation structure in polynomial representation using coefficients a_n and b_n.

$$H(z) = K\frac{b_0 + b_1 z^{-1} + \ldots + b_N z^{-N}}{1 + a_1 z^{-1} + \ldots + a_N z^{-N}} \tag{17}$$

The Elliptical Filter is described using the following equations:

$$H(j\omega) = \frac{1}{\sqrt{1 + \varepsilon^2 R_K^2\left(\xi, \frac{\omega}{\omega_\mathrm{p}}\right)}} \tag{18}$$

Where R_k is the n^{th} order elliptic rational function and cd() is the Jacobi elliptic cosine function.

$$R_n(\xi, x) \equiv \mathrm{cd}\left(n\frac{K\left(1/L_n(\xi)\right)}{K(1/\xi)}\mathrm{cd}^{-1}(x, 1/\xi), 1/L_n(\xi)\right) \tag{19}$$

$$u = \int_0^\varphi \frac{\mathrm{d}\theta}{\sqrt{1 - m\cdot\sin^2\theta}} \tag{20}$$

$$\mathrm{cn}\, u = \cos\varphi \tag{21}$$

In each case the process remains the same. One first obtains the z domain which yields the poles. These poles are then used to construct the cascading second order realisation structure by using the following equation as indicated in [3].

$$H(z) = K\frac{(z+1)^N}{(z-p_1)(z-p_2)\ldots(z-p_N)} \tag{22}$$

3 Design

The entire DSP system is designed around the European Telecommunication Standards Institute (ETSI) requirements. Table 1 shows what these requirements entail. If the signal operates within the MF-AM band it is only allowed 9kHz bandwidth. The requirements also state that an attenuation of $35dB$ must be reached within $6.3kHz$ and $60dB$ within $26.57kHz$. These requirements will guide the authors choice in Anti-Aliasing filter, Low-Pass filter and Digital FIR or IIR filter.

The design of the system will be broken up into stages and will be designed in order, starting with the Low-pass filter used to limit the message signals frequency and ending with the reconstruction filter used to smooth out the DAC output.

Figure 5 shows how the entire system interacts. Firstly the message signal is fed into the Low-pass filter, which attenuates any frequency components above $4.5kHz$. The filtered message signal is mixed with the carrier signal, the carrier frequency must be between $531kHz$ and $1606.5kHz$. The mixed signal results in the Amplitude modulated signal. The AM signal is then fed into the DSP's ADC which makes use of the triple interleaved ADC function. Once the ADC has received enough sample the FIR or IIR digital filter filters the ADC input, ensuring a $35dB$ and $60dB$ attenuation at $6.3kHz$ and $26.57kHz$ away from the carrier frequency. The filtered buffer is broken up into sample blocks, each block consists of 16 elements. The maximum amplitude of each block scaled and offset before being outputted through the DAC.

This method is called envelope extraction. Finally the DAC output is fed through a reconstruction filter which is used to smooth out the signal for the audio device.

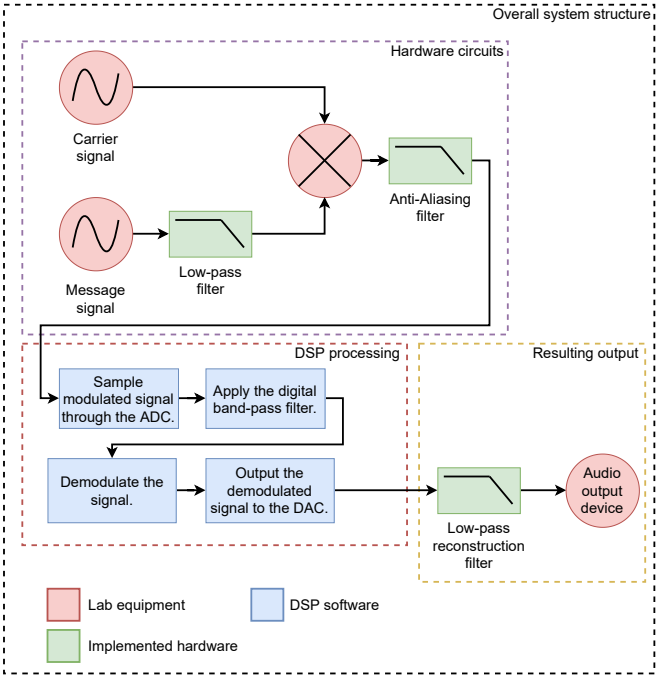


Figure 5: Diagram illustrating the entire DSP system implementation.

3.1 Low-pass filter

The Low-pass butterworth filter needs to attenuated the frequency components above 4.5kHz since any frequency component above this will interfere with other channels. A simple implementation is a 5_{th} order Butterworth filter which has a 30dB attenuation at 8kHz. Figure 7 shows the components used in the circuit's design.

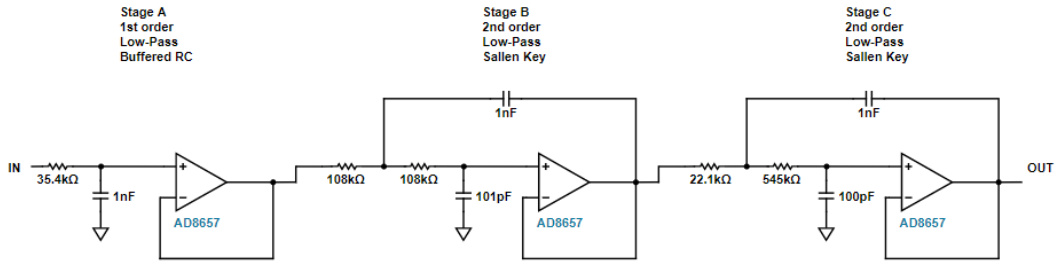


Figure 6: Low-pass filter circuit diagram.

3.2 Amplitude Modulation

Since the students are not required to build a AM circuit the ABC Lab equipment is used to generate the carrier signal and to modulate a audio signal such as a song or sine wave with the carrier signal. There are two important aspects to consider, the modulation index and the amplitude of the modulated signal. The modulation index must be below 1 since phase shifts are undesired, especially when a listening to music. The amplitude of the modulated signal must not saturate the the ADC, thus modulated signal should never have a amplitude above 3.3V.

3.3 Digital Band-pass filter

Choosing the correct digital filter is quite difficult since each filter has unique characteristics and may excel in some circumstances but fail in others. The best digital filter is chosen based off of the amount of coefficients the filter has, the results gather through simulations, speed of execution and most importantly attenuation level.

It is clear that the Rectangular, Hanning and Hamming filters are not a viable option since they can not reach a stop-band attenuation of 60dB. This leaves the Blackman and Kaiser window functions. Figures 14 shows that the Blackman filter reaches the required 35dB and 60dB attenuation if the number of coefficients exceed 15 856, but it does not achieve either of these attenuation goals if the amount of coefficients decrease. The number of coefficients must be low since computation time is limited within the DSP, also the coefficients should be smaller than 256 since a larger buffer exponentially increase computation time. The same holds true for the Kaiser window function none of the attenuation requirements are met if the coefficients are decreased and the ripple caused some components to increase in amplitude as seen in Figure 15. These findings show that the window functions are not applicable within this system.

When analysing IIR filters it is clear that they have one large advantage, they have far less coefficients which speeds up computation considerably.

The Butterworth filter does not reach the required 60dB attenuation as in indicated by Figure 17. It smoothly reduces the amplitude of the components outside the band-pass below 35dB given a $6.3kHz$ bandwidth. The bandwidth is $6.3kHz$ since this complies with ETSI's first requirement of 35dB attenuation at $6.3kHz$.

The ChebyshevII filter works almost too well since it attenuates all the components except for the carrier frequency component as seen in Figure 18. The demodulated signal in Figure 25a shows how the entire message signal attenuates. ChebyshevII is a great option if one desires a single frequency component.
The Bessel filter barley meets specification but this indicates difficult implementation going froward thus it should be considered as the last viable option.

The Elliptical IIR filter meets all the requirements since it easily reaches the required 35dB attenuation 6.3kHz away from the carrier frequency. The reduced ripple of the Elliptic filter ensures that the frequency components outside the pass-band are not pushed up by the ripples within the stop-band.

Finally the Biquad filter implementation. Although the filter does not meet the requirement it does provide linear phase shift which is perfect when sending audio signals. If the parameters are improved the Biquad implementation can easily be the best option although for now the Elliptic filter is used.

The coefficients of the Elliptic filter is calculated before hand and is given to the DSP in the form of lookup table. Moreover, the fourth order 2 stage IIR Elliptical filter has the following realisation structure.

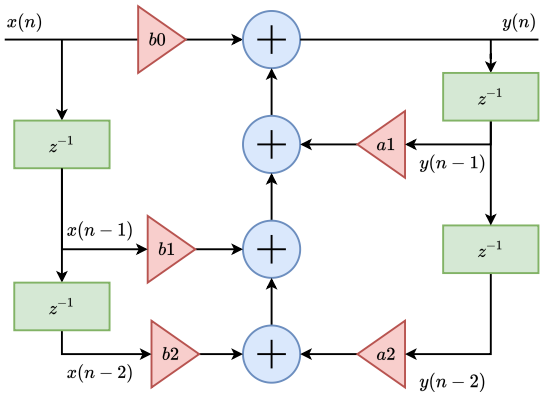


Figure 7: Elliptical filter's realisation structure.

3.4 DSP software design

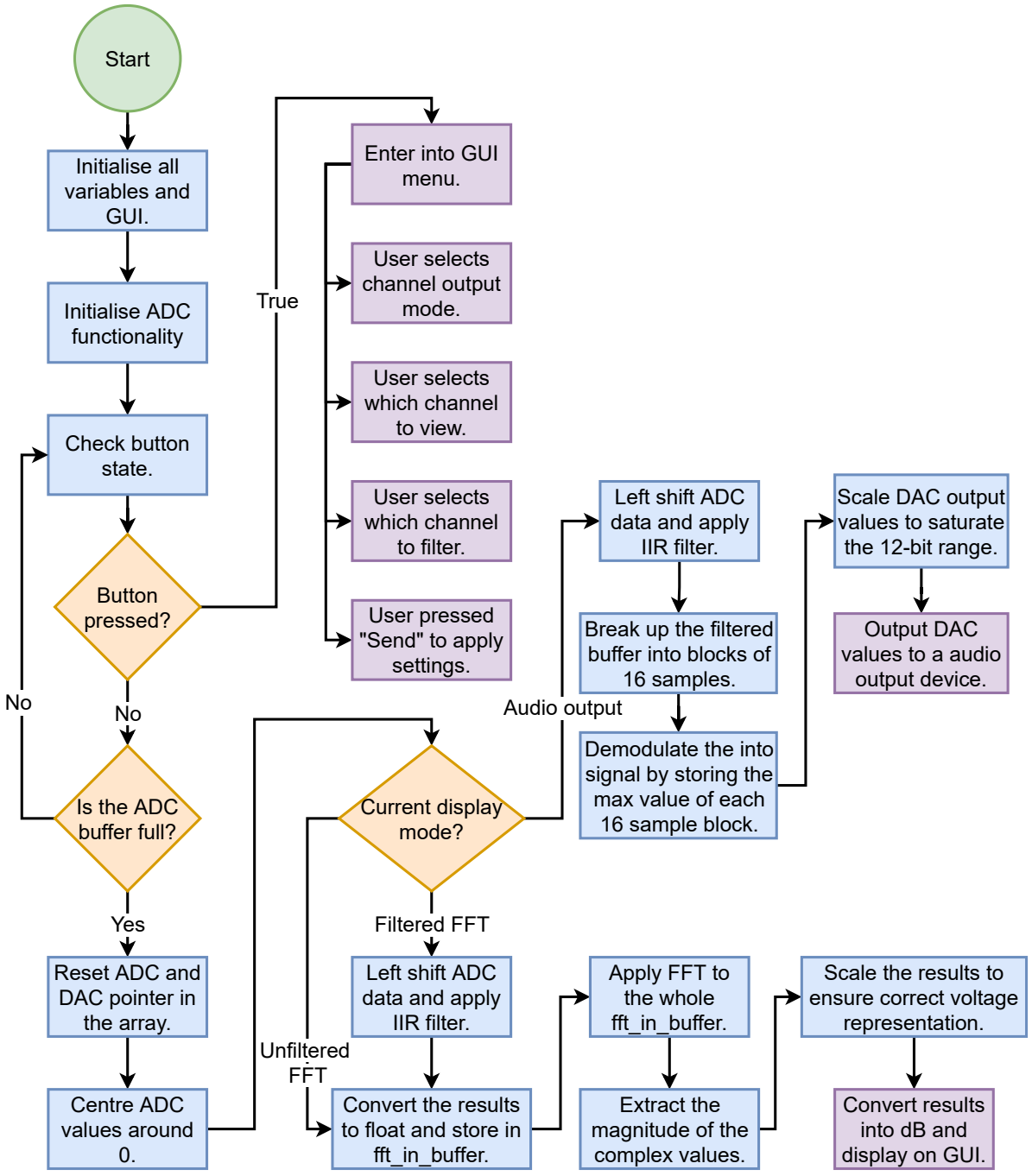


Figure 8: DSP software design.

Figure 8 and [4] shows how the software for the DSP system is designed. One major constraints is the amount of cycles/time the DSP board has to sample the ADC, filter the ADC, Demodulate and finally output to the DAC. The DSP must sample the ADC every $26\mu s$ leaves very little room for error as indicated in [5] and [6]. Thus the CMSIS-DSP libraries were used to optimise the system and to take full advantage of the DMA. The system makes use of Q15, Q31 and float 32 values. Applying these field

operations over a 2048 index array takes $3ms$, $536\mu s$ and $798\mu s$ respectively.

This indicates that q15 is the best implementation since it is much faster. All mathematical operations will be preformed in this field and will be scaled within the $2\hat{1}5$ finite field.

3.5 Demodulation algorithm

As stated before demodulation is simple to preform in the continues time domain since diodes can be used to ensure current only flows in the positive direction. In the discrete domain it becomes slightly more difficult but is still relatively simple since the message frequency is mapped to the amplitude of the resulting modulated signal. Peak detection is the method implemented on the DSP. Peak detection works as follows, the 2048 ADC samples are broken up into groups of 16. The peak value of each group is acquired and stored. To ensure real time DAC output the maximum value within the 16 element block is outputted to the DAC after scaling. The scaling function is also assisted by the CMSIS Arm libraries as they can reduce overhead by 33%. Another advantage is the fact that the DAC uses the DMA which has a separate clock form the CPU which allows for DAC to output while continuing other operations.

3.6 Anti-Aliasing filter

The relative small size of the pass-band makes the requirements of the Anti-Aliasing filter very stringent. In order to meet the required fall off rate a 9_{th} order butterworth filter is used. The filters cutoff frequency must start at the end of the MF-AM band which is $1.62MHz$ and the stop-band will be at $2.1MHz$ with a $-20dB$ attenuation. Unfortunately if the desired attenuation increases the Q value becomes unstable. The op-amps used in Figure 9 have a slew rate of $225V/\mu s$ which is very high.

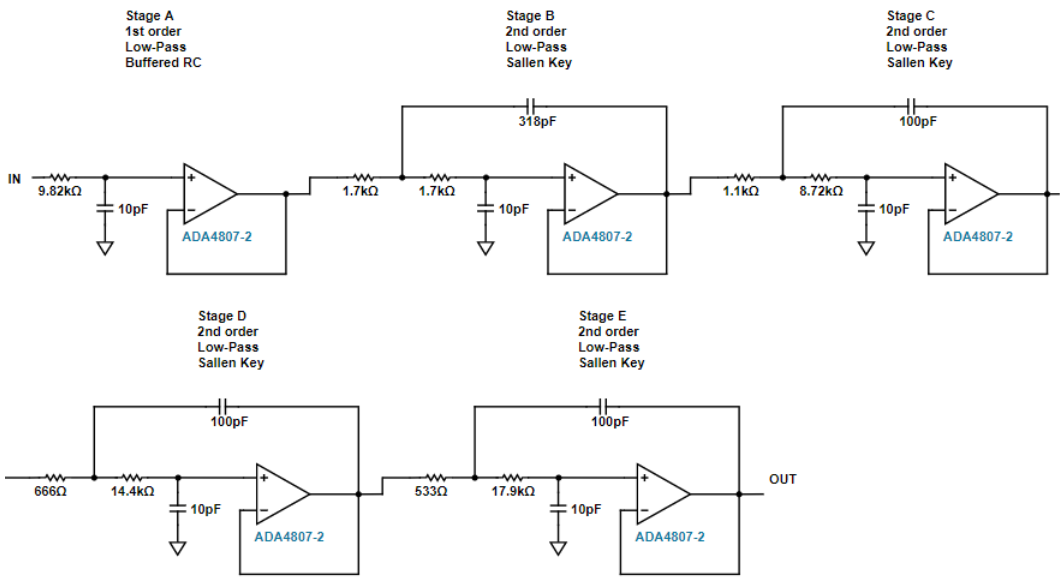


Figure 9: 9_{th} order butterworth Anti-Aliasing filter.

Finally a second order Multiple Feedback reconstruction filter is used to attenuate the harmonics created by the roughly 16 times lower resolution DAC output. The circuit ensures $40dB$ attenuation at the first harmonic of $100kHz$ and is realised within Figure 11

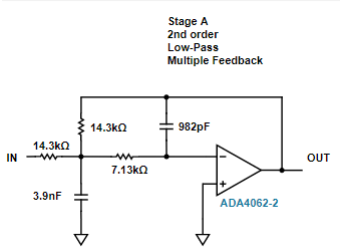


Figure 10: Multiple Feedback reconstruction filter realisation.

The GUI is used to select the desired output method as well as the displayed channel. The GUI menu allows the user to select the correct channel to view while allowing independent filtering as seen by the images within the Results section.

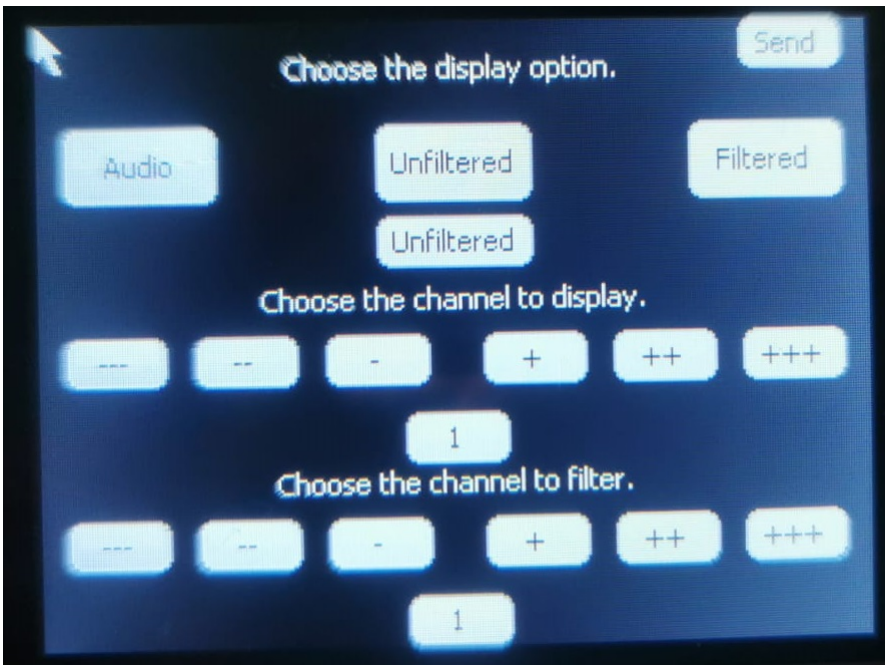


Figure 11: GUI menu screen.

The Various buttons increase or decrease the channel selector or filterer by 1, 10 or 100.

4 Simulations

Each aspect of the system has been simulated to ensure the digital filter, Anti-Aliasing filter, demodulation and low-pass filter meet the requirements under ideal circumstances.

4.1 Digital filter

The following simulations were completed within Python 3.7. The Python script simulates the filtering capabilities of the various FIR and IIR filters at channel 60. Amplitude modulation is preformed using a 1.062MHz carrier frequency which lies exactly in the middle of channel 60. The message signal is a summation of three pure sign waves at 3kHz , 5kHz and 7kHz, this should show the attenuation created by the various filters. Please take note all three sign waves have the same amplitude. Figure 13 is used as the controlled AM signal.

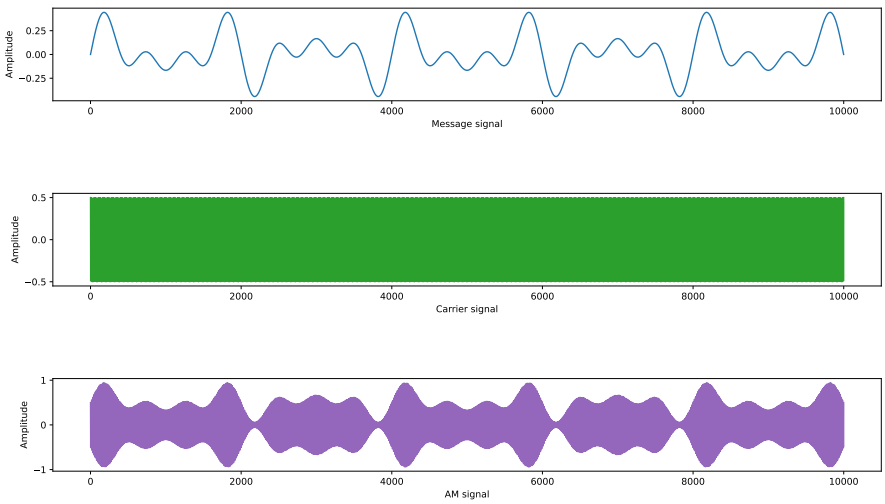


Figure 12: The input AM signal.

The accuracy and validity of the filters are measured according to their ability to meet the ETSI standards. As stated before the Rectangular, Hanning and Hamming filters will not be simulated as they do not meet the minimum requirements.

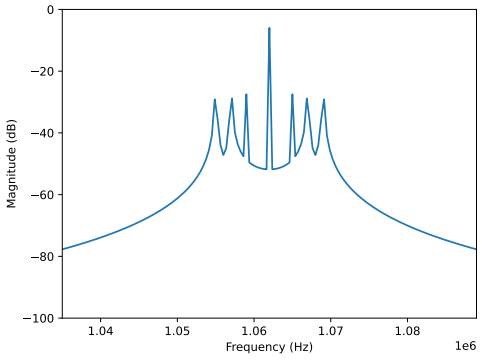
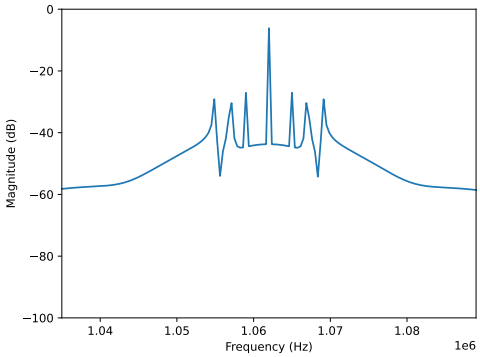
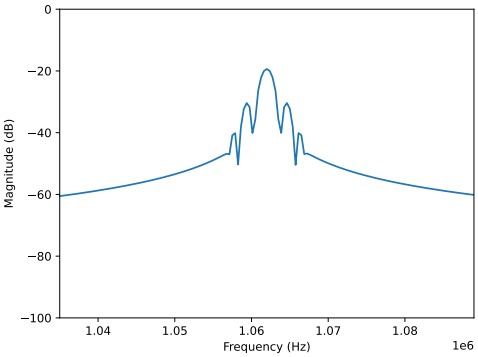


Figure 13: Unfiltered FFT response.

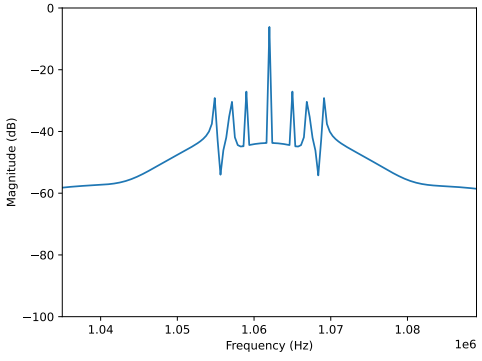
4.1.1 Blackman filter



(a) Blackman FIR filter with 341 coefficients.



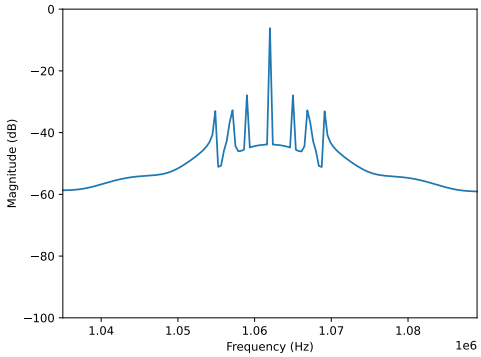
(b) Blackman FIR filter with 15856 coefficients.



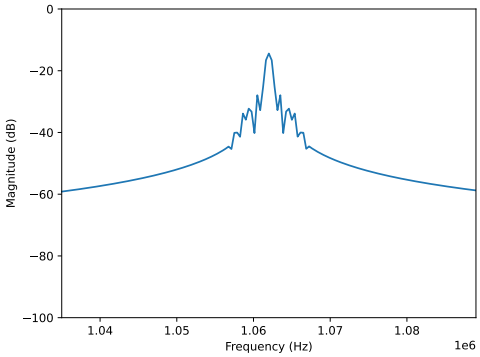
(c) Blackman FIR filter with 341 coefficients and a reduced pass-band.

Figure 14: FFT response after applying the Blackman filter.

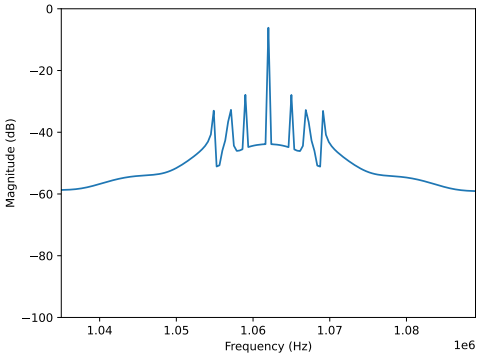
4.1.2 Kaiser filter



(a) Kaiser FIR filter with 341 coefficients.



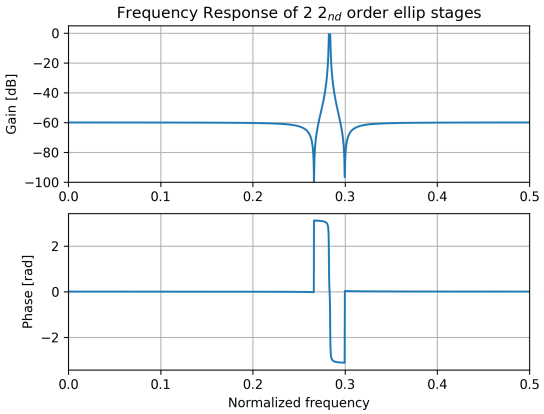
(b) Kaiser FIR filter with 12461 coefficients.



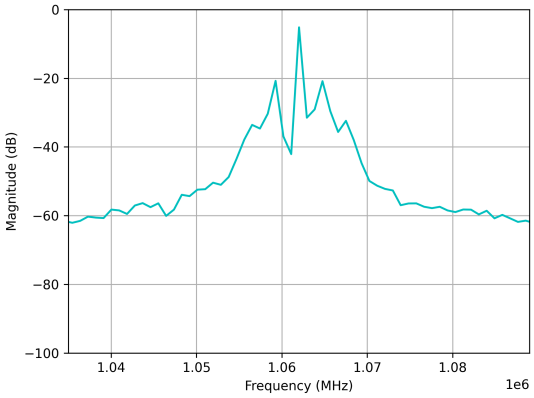
(c) Kaiser FIR filter with 341 coefficients and a reduced pass-band.

Figure 15: FFT response after applying the Kaiser filter.

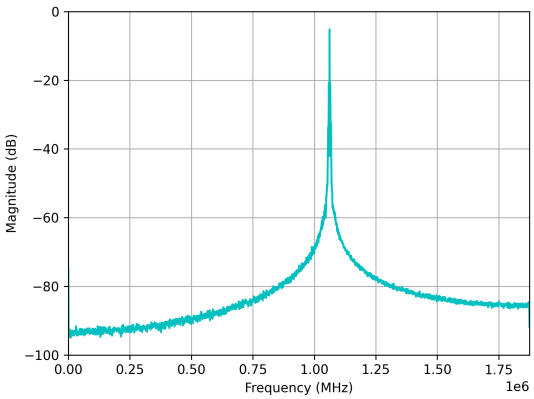
4.1.3 Elliptical IIR filter



(a) Elliptical FFT bode plot with a bandwidth of 6.3kHz.



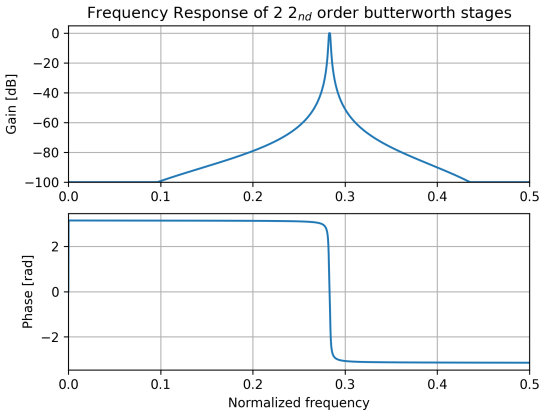
(b) Elliptical FFT response plot with a bandwidth of 6.3kHz.



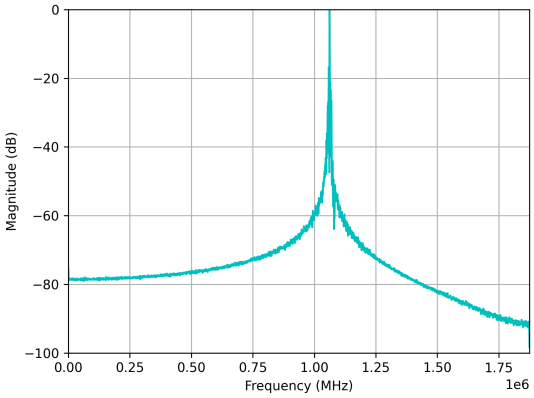
(c) Elliptical FFT response plot with a bandwidth of 6.3kHz.

Figure 16: FFT response after applying the Elliptical filter.

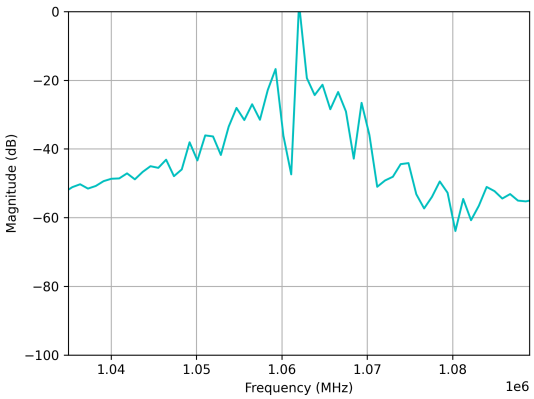
4.1.4 Butterworth IIR filter



(a) Butterworth FFT bode plot with a bandwidth of 6.3kHz.



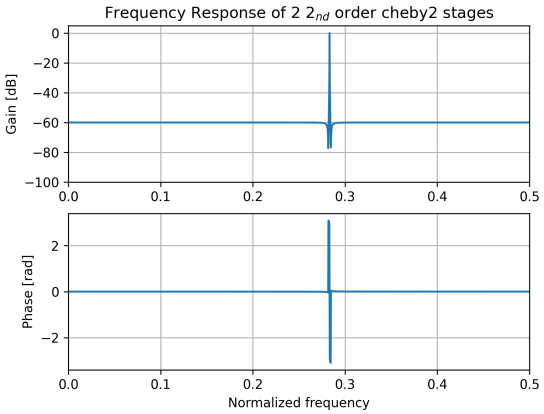
(b) Butterworth FFT response plot with a bandwidth of 6.3kHz.



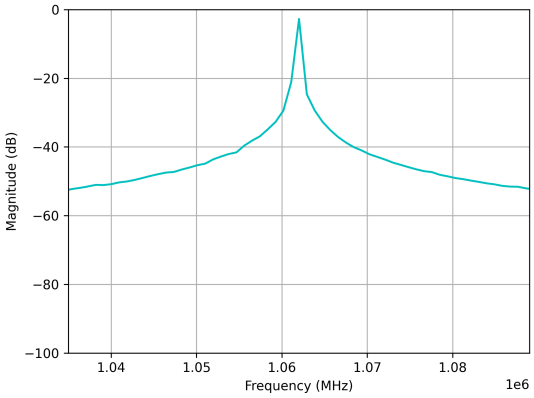
(c) Butterworth FFT response plot with a bandwidth of 6.3kHz.

Figure 17: FFT response after applying the Butterworth filter.

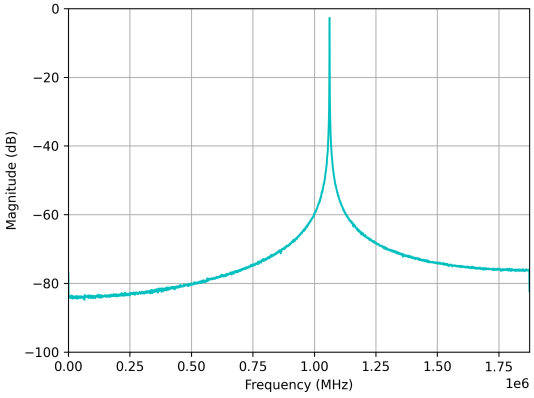
4.1.5 ChebyshevII IIR filter



(a) ChebyshevII FFT bode plot with a bandwidth of 6.3kHz.



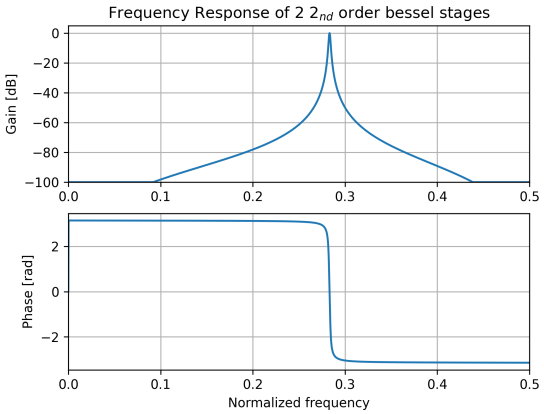
(b) ChebyshevII FFT response plot with a bandwidth of 6.3kHz.



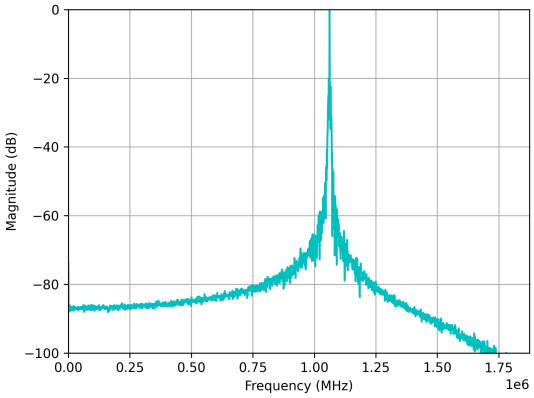
(c) ChebyshevII FFT response plot with a bandwidth of 6.3kHz.

Figure 18: FFT response after applying the ChebyshevII filter.

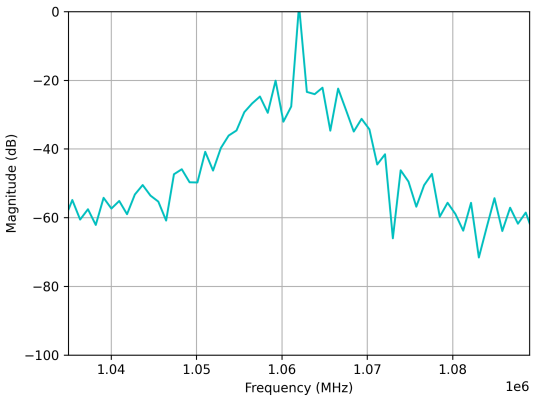
4.1.6 Bessel IIR filter



(a) Bessel FFT bode plot with a bandwidth of 6.3kHz.



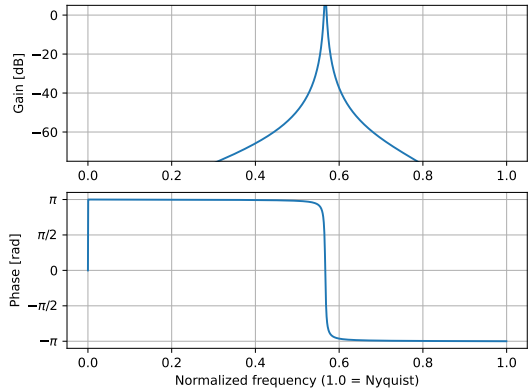
(b) Bessel FFT response plot with a bandwidth of 6.3kHz.



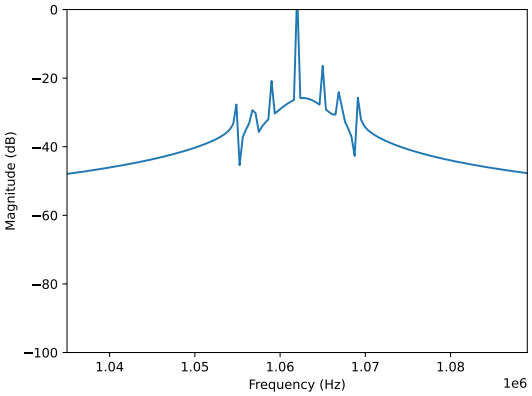
(c) Bessel FFT response plot with a bandwidth of 6.3kHz.

Figure 19: FFT response after applying the Bessel filter.

4.1.7 Biquad IIR filter



(a) Biquad FFT bode plot.

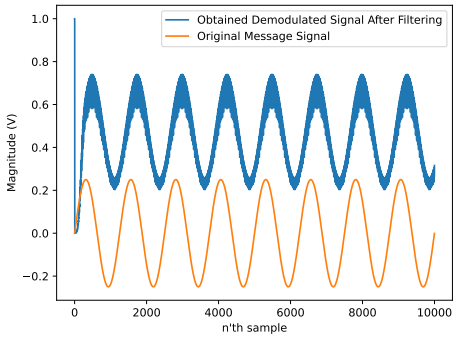


(b) Biquad FFT response.

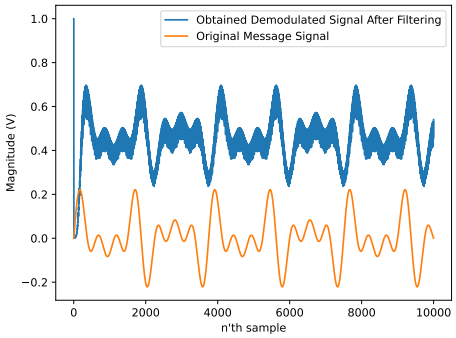
Figure 20: FFT response after applying the Bessel filter.

4.2 Demodulation

Since the filters attenuate any signal outside of the band-pass area the demodulation is simulated using a pure 3kHz sign wave as well as the one used during the filtering simulation stage. The demodulated signal should take the same shape as the message signal, the amplitude can easily be scaled and offset by the DSP board. Although, the demodulated signal will be distorted if there are frequency components above the pass-band as the filters should filter these components out.

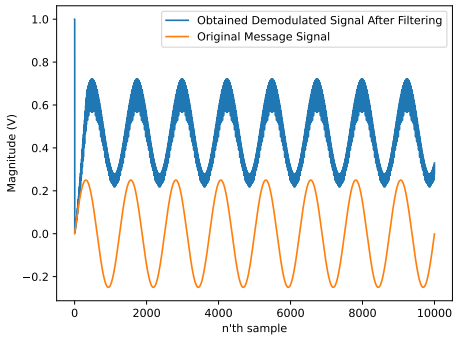


(a) Blackman demodulation, 3kHz message signal.

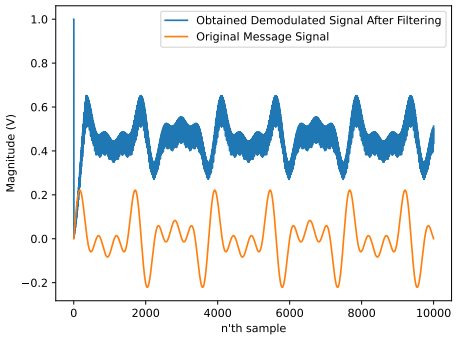


(b) Blackman demodulation of original message signal.

Figure 21: Blackman demodulation.

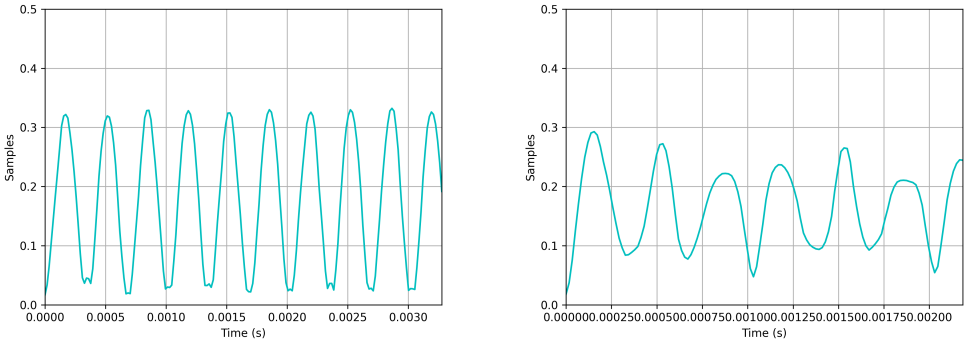


(a) Kaiser demodulation, 3kHz message signal.



(b) Kaiser demodulation of original message signal.

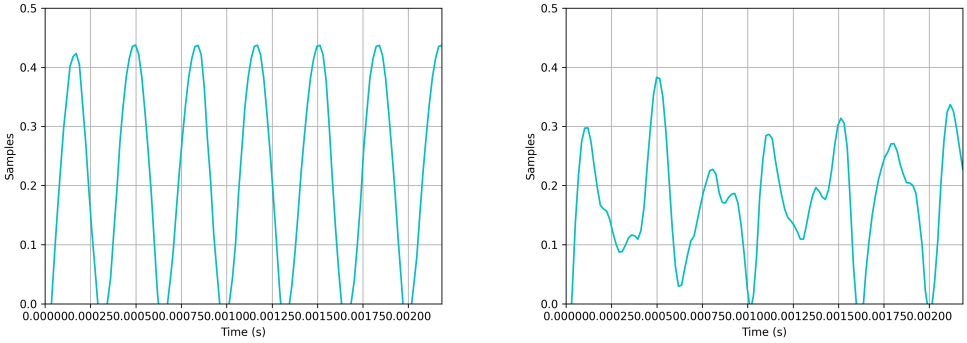
Figure 22: Kaiser demodulation.



(a) Elliptic demodulation, 3kHz message signal.

(b) Elliptic demodulation of original message signal.

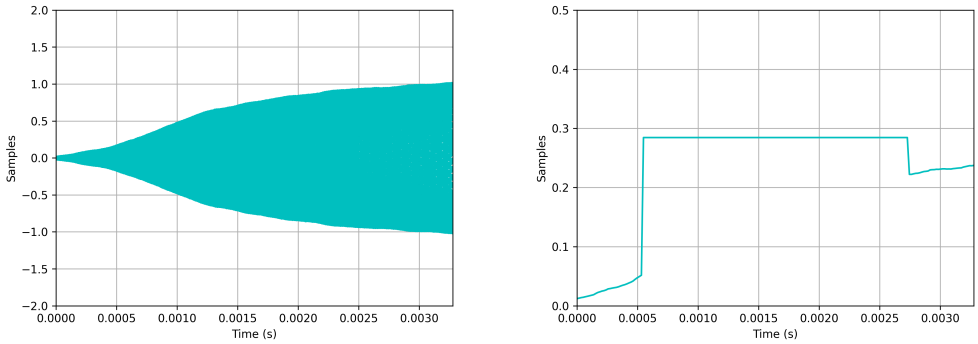
Figure 23: Elliptic demodulation.



(a) Butterworth demodulation, 3kHz message signal.

(b) Butterworth demodulation of original message signal.

Figure 24: Butterworth demodulation.



(a) ChebyshevII demodulation, 3kHz message signal.

(b) ChebyshevII demodulation of original message signal.

Figure 25: ChebyshevII demodulation.

4.3 Anti-Aliasing filter

The Anti-Aliasing filter reduces the amplitude of the harmonics.

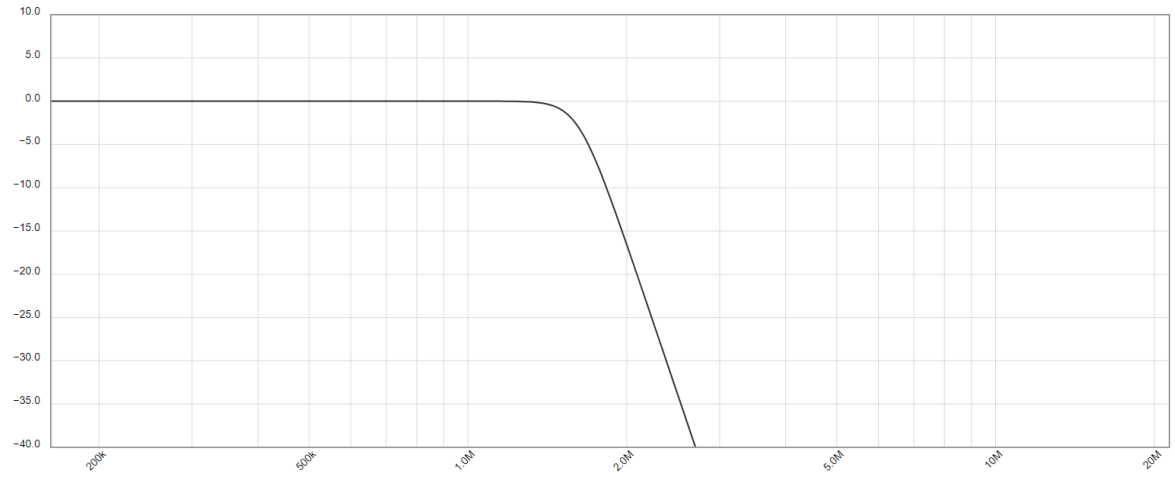


Figure 26: 8_{th} order butterworth Anti-Aliasing filter in dB scale.

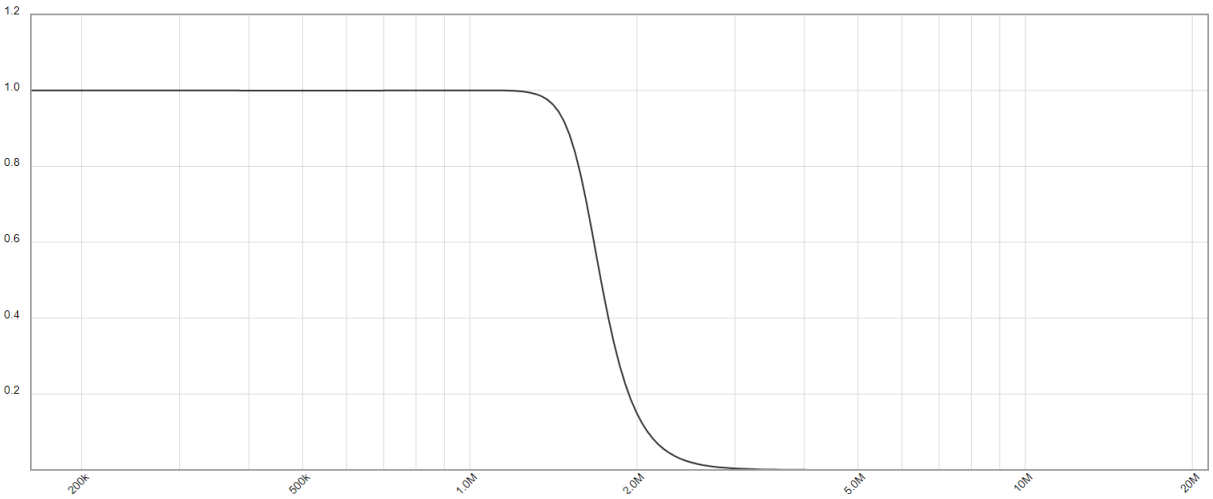
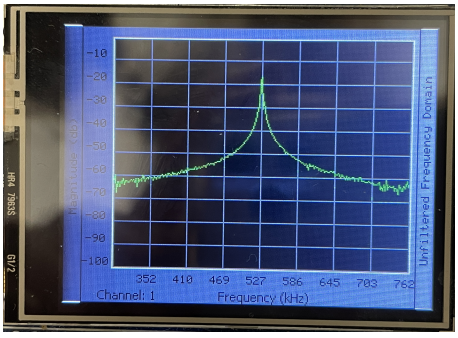


Figure 27: 8_{th} order butterworth Anti-Aliasing filter in V/V scale.

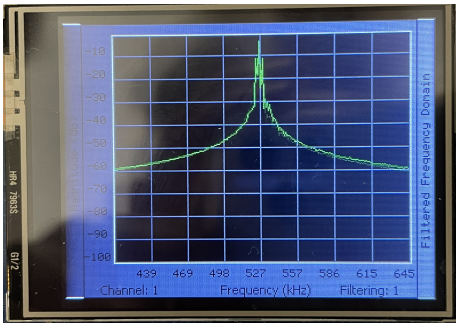
5 Results

The following section tabulates as well as graphically displays the results gathered by the oscilloscopes and LCD screen. The LCD screen is used to show the FFT output of the modulated signal before and after IIR pass-band filtering. The oscilloscope reading are used to display the difference between the modulated and demodulated signal while filtering different channels.

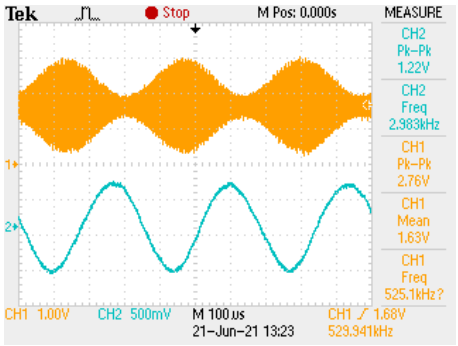
The results are grouped together according to the carrier frequency as well as the filtered channel.



(a) FFT output before filtering.

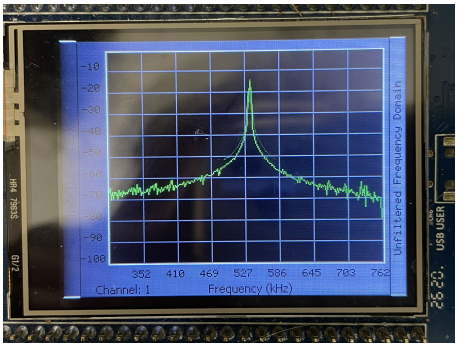


(b) FFT output after filtering.

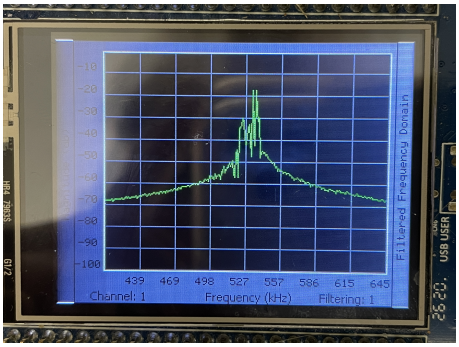


(c) Comparison between the modulated and demodulated signal.

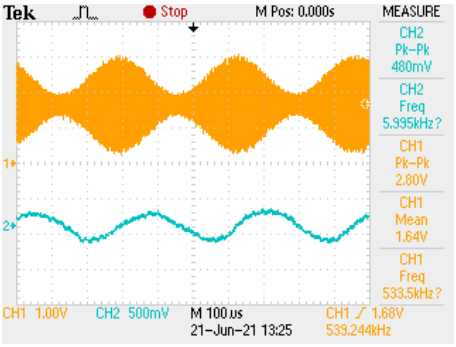
Figure 28: Filtering channel 1 with a carrier signal frequency of 531kHz.



(a) FFT output before filtering.

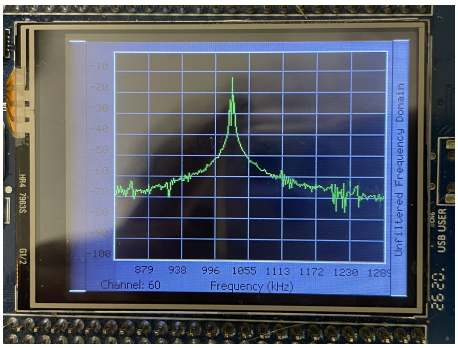


(b) FFT output after filtering.

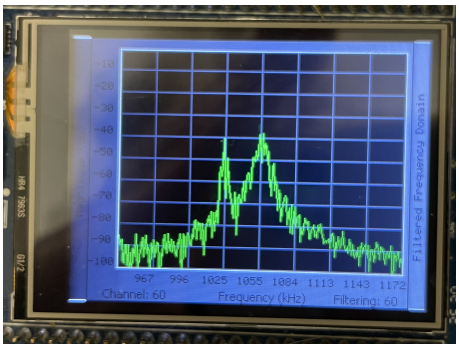


(c) Comparison between the modulated and demodulated signal.

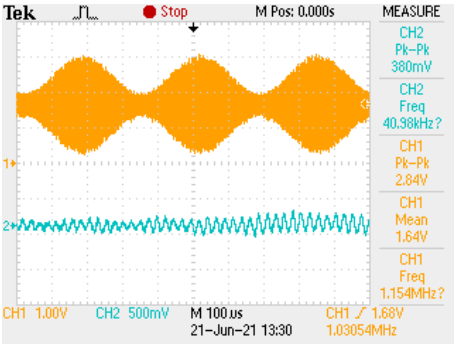
Figure 29: Filtering channel 1 with a carrier signal frequency of 540kHz.



(a) FFT output before filtering.

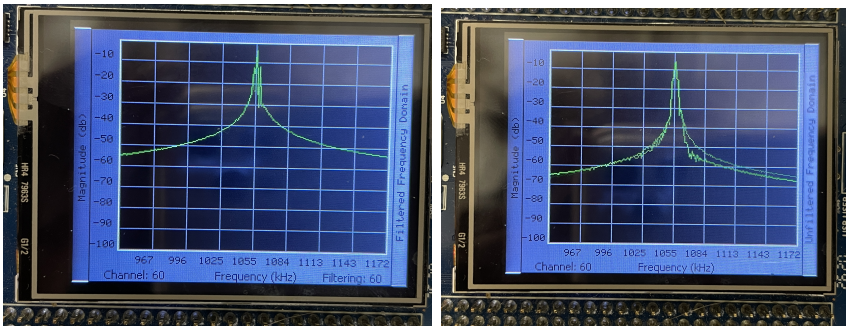


(b) FFT output after filtering.

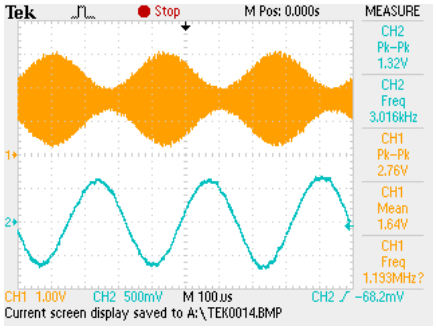


(c) Comparison between the modulated and demodulated signal.

Figure 30: Filtering channel 60 with a carrier signal frequency of 792kHz.

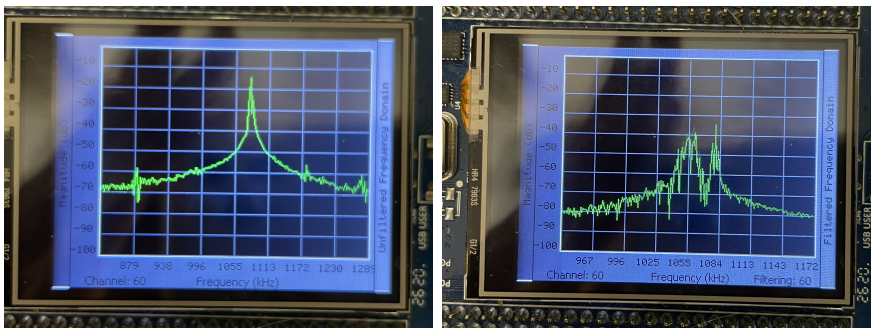


(a) FFT output before filtering. **(b)** FFT output after filtering.

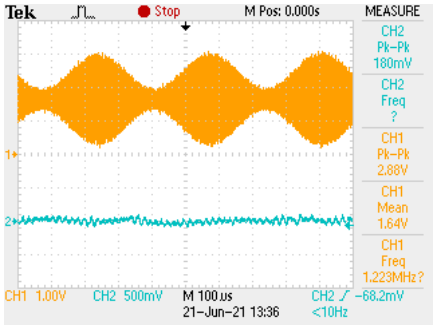


(c) Comparison between the modulated and demodulated signal.

Figure 31: Filtering channel 60 with a carrier signal frequency of 1.062MHz.

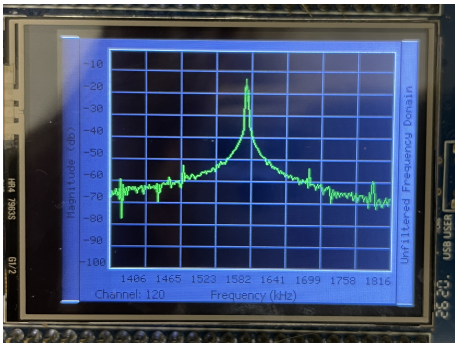


(a) FFT output before filtering. **(b)** FFT output after filtering.

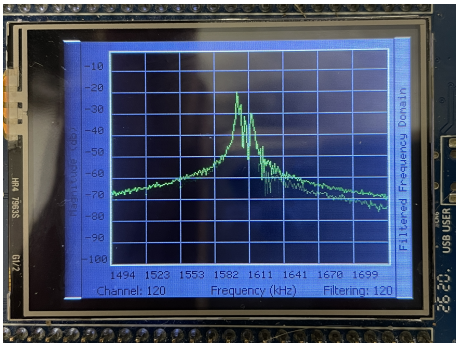


(c) Comparison between the modulated and demodulated signal.

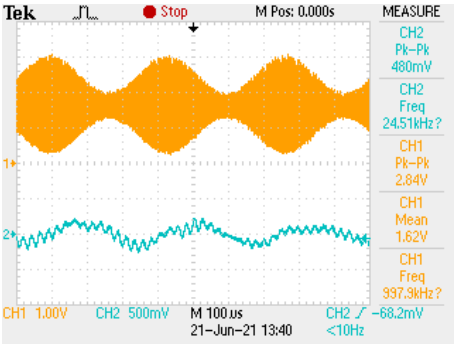
Figure 32: Filtering channel 60 with a carrier signal frequency of 1.332MHz.



(a) FFT output before filtering.

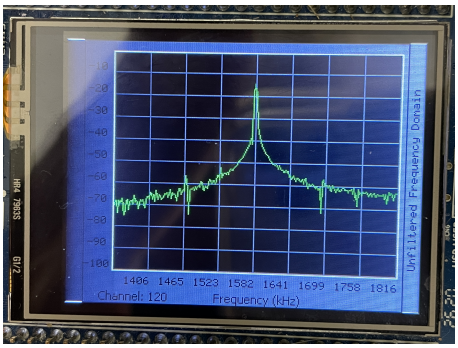


(b) FFT output after filtering.

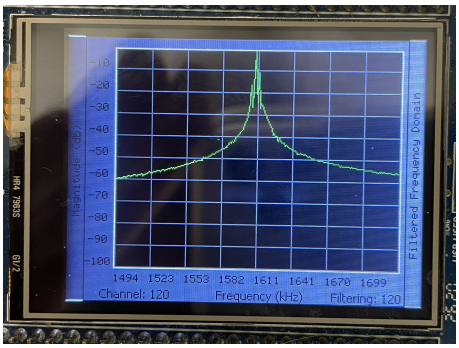


(c) Comparison between the modulated and demodulated signal.

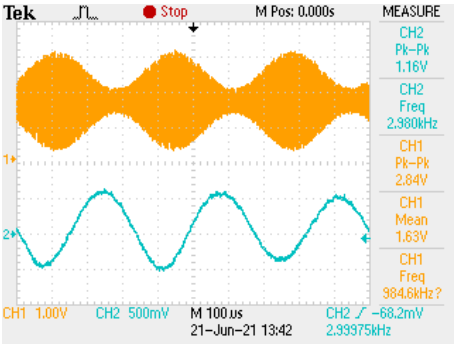
Figure 33: Filtering channel 120 with a carrier signal frequency of 1.593MHz.



(a) FFT output before filtering.



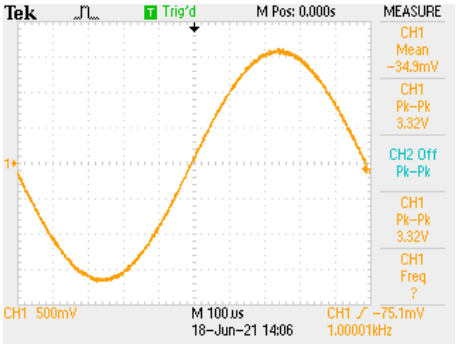
(b) FFT output after filtering.



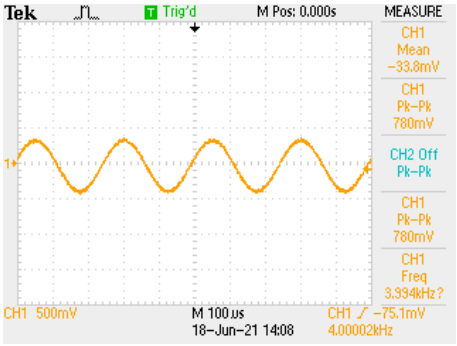
(c) Comparison between the modulated and demodulated signal.

Figure 34: Filtering channel 120 with a carrier signal frequency of 1.602MHz.

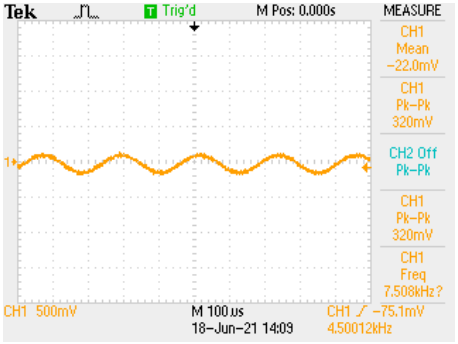
5.1 Low-pass filter



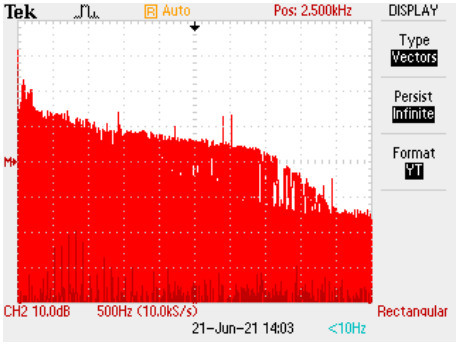
(a) Attenuation caused by the low-pass filter at $1kHz$



(b) Attenuation caused by the low-pass filter at $4kHz$



(c) Attenuation caused by the low-pass filter at $7.5kHz$



(d) FFT response of the low-pass filter.

Figure 35: Low pass filter attenuation.

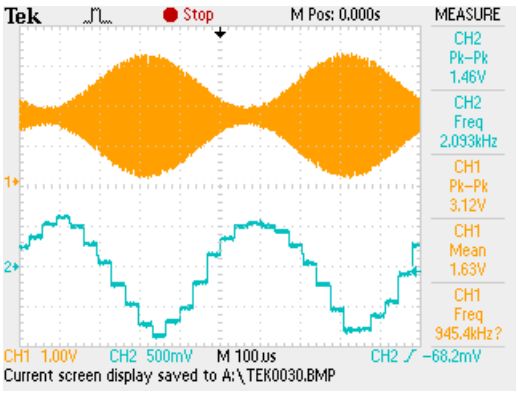


Figure 36: DAC output before reconstruction.

6 Discussion

After implementing the entire demodulation system which consists of various filters, a AM modulator, a demodulator and a reconstruction circuit it is clear that the task was a challenging one. Fortunately the results gathered using the ABC Lab's oscilloscope clearly show that the system operates as indented. Figures 35a - 35d clearly show how the signal attenuates if the frequency crosses the $4.5kHz$ passband. This ensures that attenuation is applied to any frequency components outside the $4.5kHz$ bandwidth of the channel.

If one looks at Figures 30 - 31 it is clear that the filter attenuates any signals outside the bandwidth of the desired channel. When considering Figure 30 it is clear that the IIR filter operates as expected. The channel being filtered is channel 60 but the carrier frequency is centered at channel 57. One can clearly see the attenuated spike next to the filtered channel. In the case of Figure 31 the filtered channel matches the carrier channel, thus they form a summation, this amplifies the message frequency modulated around the carrier. The oscilloscope readings solidify these findings, as the message signal has the same period as the envelope and is attenuated if the carrier frequency does not match the filtered frequency.
The lack of harmonics throughout the system also indicates that the AAF works as expected. The small spikes at the edges are attenuated enough to ensure minimum distortion. Finally it is clear that the required $60dB$ attenuation is reached while applying the filter. Finally Figure 36 shows how the DAC output looks before reconstruction, which implies that the reconstruction filter operates are required.

7 Conclusion

After considering the various simulations, results and DSP timing requirements it is clear that the system preforms well. The Elliptical IIR filter meets the minimum ETSI requirements since the system does manage a $35dB$ attenuation $6.3kHz$ from the carrier frequency. It also manages to reach a $60dB$ attenuation at the stop-band.

The system successfully samples the ADC, applies the filter and demodulates the signal within the tight time constraints. One thing to note is that, without the CMSIS libraries this would not be possible. Nevertheless, the results indicate successful Anti-Aliasing, signal reconstruction and Audio output.

8 References

[1] *Transmitting equipment for the Amplitude Modulated (AM) sound broadcasting service; Harmonised Standard covering the essential requirements*, 2016. [Online]. Available: `https://www.etsi.org/deliver/etsi_en/302000_302099/302017/02.00.03_20/en_302017v020003a.pdf`.

[2] E. C. Ifeachor and B. W. Jervis, *Digital Signal Processing: A Practical Approach*, 2nd ed. Great Britain: Pearson Education, 2002, p. 933.

[3] Matthew Cuthbert Winnan, Hanro van der Westhuizen, and Christiaan Frederik Wagner, "ESP 411 - Practical 1," University Of Pretoria, Tech. Rep. 1, Apr. 2021.

[4] C. F. Wagner, "ESP 411 - Practical 3," University of Pretoria, Pretoria, South Africa, Tech. Rep., 2021-06-22.

[5] *STM32F405/415, STM32F407/417, STM32F427/437 and STM32F429/439 advanced Arm®-based 32-bit MCUs*, RM0090, Rev. 19, ST, Feb. 2021.

[6] Franco, Sergio, *DESIGN WITH OPERATIONAL AMPLIFIERS AND ANALOG INTEGRATED CIRCUITS*. New York: Elisabethl A, Jones, 2002.

Appendices

A Source Code

A.1 GUI Menu

```
1  #include "GUIMenu.h"
2  #include <stdio.h>
3  #include <string.h>
4
5  WM_HWIN hItem;
6  TEXT_Handle optionText;
7  TEXT_Handle title2Text;
8  TEXT_Handle title3Text;
9
10 uint8_t displayOption = 2;
11 int8_t channelDisplayOption = 1;
12 int8_t filterDisplayOption = 1;
13 uint8_t continueCount;
14
15 char channelDisplayOptionD[4] = "1";
16 char filterDisplayOptionD[4] = "1";
17
18 BUTTON_Handle optionTextD;
19 BUTTON_Handle title2TextD;
20 BUTTON_Handle title3TextD;
21 GUI_PID_STATE TS_State_M;
22 BUTTON_Handle audioButton;
23 BUTTON_Handle exitButton;
24 BUTTON_Handle unfilteredButton;
25 BUTTON_Handle filteredButton;
26 BUTTON_Handle channelDisplayMinus3;
27 BUTTON_Handle channelDisplayMinus2;
28 BUTTON_Handle channelDisplayMinus1;
29 BUTTON_Handle channelDisplayAdd3;
30 BUTTON_Handle channelDisplayAdd2;
31 BUTTON_Handle channelDisplayAdd1;
32
33 BUTTON_Handle channelFilterMinus3;
34 BUTTON_Handle channelFilterMinus2;
35 BUTTON_Handle channelFilterMinus1;
36 BUTTON_Handle channelFilterAdd3;
37 BUTTON_Handle channelFilterAdd2;
38 BUTTON_Handle channelFilterAdd1;
39
40 void GUI_Menu() {
41     continueCount = 1;
42 //  channelDisplayOption = 1;
43 //  filterDisplayOption = 1;
44     hItem = WM_GetFirstChild(WM_HBKWIN);
```

```
45      WM_HideWindow(hItem);
46      GUI_Clear();
47      setupMenu();
48      while (continueCount == 1) {
49          GUI_CURSOR_Show();
50          BSP_Pointer_Update();
51          checkButtonPress();
52          GUI_Exec();
53      }
54
55      TS_State_M.Layer = 0;
56      TS_State_M.x = 0;
57      TS_State_M.y = 0;
58      GUI_TOUCH_StoreStateEx(&TS_State_M);
59      GUI_CURSOR_Hide();
60      GUI_Clear();
61      GUI_Exec();
62  }
63
64  void BSP_Pointer_Update(void) {
65      GUI_PID_STATE TS_State;
66      static TS_StateTypeDef prev_state;
67      TS_StateTypeDef ts;
68      uint16_t xDiff, yDiff;
69
70      BSP_TS_GetState(&ts);
71
72      TS_State.Pressed = ts.TouchDetected;
73
74      xDiff = (prev_state.X > ts.X) ?
75              (prev_state.X - ts.X) : (ts.X - prev_state.X);
76      yDiff = (prev_state.Y > ts.Y) ?
77              (prev_state.Y - ts.Y) : (ts.Y - prev_state.Y);
78
79      if (ts.TouchDetected) {
80          if ((prev_state.TouchDetected != ts.TouchDetected) || (
                xDiff > 3)
81                  || (yDiff > 3)) {
82              prev_state = ts;
83
84              TS_State.Layer = 0;
85              TS_State.x = 300 - ts.Y;
86              TS_State.y = 240 - ts.X;
87
88              GUI_TOUCH_StoreStateEx(&TS_State);
89          }
90      }
91  }
92
93  void checkButtonPress() {
```

```
94      if (BUTTON_IsPressed(audioButton)) {
95          displayOption = 1;
96          BUTTON_SetText(optionTextD, "Audio");
97          while (BUTTON_IsPressed(audioButton)) {
98              BSP_Pointer_Update();
99              GUI_Exec();
100         }
101     }
102     if (BUTTON_IsPressed(unfilteredButton)) {
103         displayOption = 2;
104         BUTTON_SetText(optionTextD, "Unfiltered");
105         while (BUTTON_IsPressed(unfilteredButton)) {
106             BSP_Pointer_Update();
107             GUI_Exec();
108         }
109     }
110     if (BUTTON_IsPressed(filteredButton)) {
111         displayOption = 3;
112         BUTTON_SetText(optionTextD, "Filtered");
113         while (BUTTON_IsPressed(filteredButton)) {
114             BSP_Pointer_Update();
115             GUI_Exec();
116         }
117     }
118
119     if (BUTTON_IsPressed(channelDisplayMinus3)) {
120         if (channelDisplayOption - 100 <= 0) {
121             channelDisplayOption = 120 - (100 -
                    channelDisplayOption);
122         } else {
123             channelDisplayOption = channelDisplayOption - 100;
124         }
125         sprintf(channelDisplayOptionD, "%i",
                channelDisplayOption);
126         BUTTON_SetText(title2TextD, channelDisplayOptionD);
127         while (BUTTON_IsPressed(channelDisplayMinus3)) {
128             BSP_Pointer_Update();
129             GUI_Exec();
130         }
131     }
132     if (BUTTON_IsPressed(channelDisplayMinus2)) {
133         if (channelDisplayOption - 10 <= 0) {
134             channelDisplayOption = 120 - (10 -
                    channelDisplayOption);
135         } else {
136             channelDisplayOption = channelDisplayOption - 10;
137         }
138         sprintf(channelDisplayOptionD, "%i",
                channelDisplayOption);
139         BUTTON_SetText(title2TextD, channelDisplayOptionD);
```

```
140             while (BUTTON_IsPressed(channelDisplayMinus2)) {
141                 BSP_Pointer_Update();
142                 GUI_Exec();
143             }
144         }
145 
146         if (BUTTON_IsPressed(channelDisplayMinus1)) {
147             if (channelDisplayOption - 1 <= 0) {
148                 channelDisplayOption = 120 - (1 -
                        channelDisplayOption);
149             } else {
150                 channelDisplayOption = channelDisplayOption - 1;
151             }
152             sprintf(channelDisplayOptionD, "%i",
                    channelDisplayOption);
153             BUTTON_SetText(title2TextD, channelDisplayOptionD);
154             while (BUTTON_IsPressed(channelDisplayMinus1)) {
155                 BSP_Pointer_Update();
156                 GUI_Exec();
157             }
158         }
159         if (BUTTON_IsPressed(channelDisplayAdd1)) {
160             if (channelDisplayOption + 1 > 120) {
161                 channelDisplayOption = 1;
162             } else {
163                 channelDisplayOption = channelDisplayOption + 1;
164             }
165             sprintf(channelDisplayOptionD, "%i",
                    channelDisplayOption);
166             BUTTON_SetText(title2TextD, channelDisplayOptionD);
167             while (BUTTON_IsPressed(channelDisplayAdd1)) {
168                 BSP_Pointer_Update();
169                 GUI_Exec();
170             }
171         }
172         if (BUTTON_IsPressed(channelDisplayAdd2)) {
173             if (channelDisplayOption + 10 > 120) {
174                 channelDisplayOption = channelDisplayOption + 10 -
                        120;
175             } else {
176                 channelDisplayOption = channelDisplayOption + 10;
177             }
178             sprintf(channelDisplayOptionD, "%i",
                    channelDisplayOption);
179             BUTTON_SetText(title2TextD, channelDisplayOptionD);
180             while (BUTTON_IsPressed(channelDisplayAdd2)) {
181                 BSP_Pointer_Update();
182                 GUI_Exec();
183             }
184         }
```

```
185     if (BUTTON_IsPressed(channelDisplayAdd3)) {
186         if (channelDisplayOption + 100 > 120) {
187             channelDisplayOption = channelDisplayOption + 100 -
                    120;
188         } else {
189             channelDisplayOption = channelDisplayOption + 100;
190         }
191         sprintf(channelDisplayOptionD, "%i",
                channelDisplayOption);
192         BUTTON_SetText(title2TextD, channelDisplayOptionD);
193         while (BUTTON_IsPressed(channelDisplayAdd3)) {
194             BSP_Pointer_Update();
195             GUI_Exec();
196         }
197     }
198
199     if (BUTTON_IsPressed(channelFilterMinus3)) {
200         if (filterDisplayOption - 100 <= 0) {
201             filterDisplayOption = 120 - (100 -
                    filterDisplayOption);
202         } else {
203             filterDisplayOption = filterDisplayOption - 100;
204         }
205         sprintf(filterDisplayOptionD, "%i", filterDisplayOption)
                ;
206         BUTTON_SetText(title3TextD, filterDisplayOptionD);
207         while (BUTTON_IsPressed(channelFilterMinus3)) {
208             BSP_Pointer_Update();
209             GUI_Exec();
210         }
211     }
212     if (BUTTON_IsPressed(channelFilterMinus2)) {
213         if (filterDisplayOption - 10 <= 0) {
214             filterDisplayOption = 120 - (10 -
                    filterDisplayOption);
215         } else {
216             filterDisplayOption = filterDisplayOption - 10;
217         }
218         sprintf(filterDisplayOptionD, "%i", filterDisplayOption)
                ;
219         BUTTON_SetText(title3TextD, filterDisplayOptionD);
220         while (BUTTON_IsPressed(channelFilterMinus2)) {
221             BSP_Pointer_Update();
222             GUI_Exec();
223         }
224     }
225
226     if (BUTTON_IsPressed(channelFilterMinus1)) {
227         if (filterDisplayOption - 1 <= 0) {
```

```
228             filterDisplayOption = 120 - (1 - filterDisplayOption
                    );
229         } else {
230             filterDisplayOption = filterDisplayOption - 1;
231         }
232         sprintf(filterDisplayOptionD, "%i", filterDisplayOption)
                ;
233         BUTTON_SetText(title3TextD, filterDisplayOptionD);
234         while (BUTTON_IsPressed(channelFilterMinus1)) {
235             BSP_Pointer_Update();
236             GUI_Exec();
237         }
238     }
239     if (BUTTON_IsPressed(channelFilterAdd1)) {
240         if (filterDisplayOption + 1 > 120) {
241             filterDisplayOption = 1;
242         } else {
243             filterDisplayOption = filterDisplayOption + 1;
244         }
245         sprintf(filterDisplayOptionD, "%i", filterDisplayOption)
                ;
246         BUTTON_SetText(title3TextD, filterDisplayOptionD);
247         while (BUTTON_IsPressed(channelFilterAdd1)) {
248             BSP_Pointer_Update();
249             GUI_Exec();
250         }
251     }
252     if (BUTTON_IsPressed(channelFilterAdd2)) {
253         if (filterDisplayOption + 10 > 120) {
254             filterDisplayOption = filterDisplayOption + 10 -
                    120;
255         } else {
256             filterDisplayOption = filterDisplayOption + 10;
257         }
258         sprintf(filterDisplayOptionD, "%i", filterDisplayOption)
                ;
259         BUTTON_SetText(title3TextD, filterDisplayOptionD);
260         while (BUTTON_IsPressed(channelFilterAdd2)) {
261             BSP_Pointer_Update();
262             GUI_Exec();
263         }
264     }
265     if (BUTTON_IsPressed(channelFilterAdd3)) {
266         if (filterDisplayOption + 100 > 120) {
267             filterDisplayOption = filterDisplayOption + 100 -
                    120;
268         } else {
269             filterDisplayOption = filterDisplayOption + 100;
270         }
```

```
271             sprintf(filterDisplayOptionD, "%i", filterDisplayOption)
                    ;
272             BUTTON_SetText(title3TextD, filterDisplayOptionD);
273             while (BUTTON_IsPressed(channelFilterAdd3)) {
274                 BSP_Pointer_Update();
275                 GUI_Exec();
276             }
277         }
278         if (BUTTON_IsPressed(exitButton)) {
279             WM_DeleteWindow(audioButton);
280             WM_DeleteWindow(unfilteredButton);
281             WM_DeleteWindow(filteredButton);
282             WM_DeleteWindow(channelFilterMinus3);
283             WM_DeleteWindow(channelFilterMinus2);
284             WM_DeleteWindow(channelFilterMinus1);
285             WM_DeleteWindow(channelFilterAdd1);
286             WM_DeleteWindow(channelFilterAdd2);
287             WM_DeleteWindow(channelFilterAdd3);
288             WM_DeleteWindow(channelDisplayMinus3);
289             WM_DeleteWindow(channelDisplayMinus2);
290             WM_DeleteWindow(channelDisplayMinus1);
291             WM_DeleteWindow(channelDisplayAdd3);
292             WM_DeleteWindow(channelDisplayAdd2);
293             WM_DeleteWindow(channelDisplayAdd1);
294             WM_DeleteWindow(title3TextD);
295             WM_DeleteWindow(title2TextD);
296             WM_DeleteWindow(optionText);
297             WM_DeleteWindow(title3Text);
298             WM_DeleteWindow(title2Text);
299             WM_DeleteWindow(optionTextD);
300             WM_DeleteWindow(exitButton);
301             continueCount = 0;
302         }
303     }
304
305     void setupMenu() {
306         GUI_SetBkColor(GUI_WHITE);
307         optionText = TEXT_CreateEx(0, 12, 320, 20, WM_HBKWIN,
                WM_CF_SHOW,
308         TEXT_CF_VCENTER, GUI_ID_TEXT6, "Choose the display option.")
                ;
309         TEXT_SetTextColor(optionText, GUI_WHITE);
310         TEXT_SetTextAlign(optionText, GUI_TA_HCENTER);
311         audioButton = BUTTON_CreateEx(10, 40, 60, 30, WM_HBKWIN,
                WM_CF_SHOW, 0,
312         GUI_ID_BUTTON0);
313         BUTTON_SetText(audioButton, "Audio");
314         unfilteredButton = BUTTON_CreateEx(130, 40, 60, 30,
                WM_HBKWIN, WM_CF_SHOW,
315                 0, GUI_ID_BUTTON1);
```

```
316     BUTTON_SetText(unfilteredButton, "Unfiltered");
317     filteredButton = BUTTON_CreateEx(250, 40, 60, 30, WM_HBKWIN,
            WM_CF_SHOW, 0,
318     GUI_ID_BUTTON2);
319     BUTTON_SetText(filteredButton, "Filtered");
320
321     optionTextD = BUTTON_CreateEx(130, 75, 60, 20, WM_HBKWIN,
            WM_CF_SHOW, 0,
322     GUI_ID_BUTTON0);
323     if (displayOption == 1)
324         BUTTON_SetText(optionTextD, "Audio");
325     else if(displayOption == 2)
326         BUTTON_SetText(optionTextD, "Unfiltered");
327     else if(displayOption == 3)
328             BUTTON_SetText(optionTextD, "Filtered");
329     title2Text = TEXT_CreateEx(0, 100, 320, 20, WM_HBKWIN,
            WM_CF_SHOW,
330     TEXT_CF_VCENTER, GUI_ID_TEXT7, "Choose the channel to
            display.");
331     TEXT_SetTextColor(title2Text, GUI_WHITE);
332     TEXT_SetTextAlign(title2Text, GUI_TA_HCENTER);
333     channelDisplayMinus3 = BUTTON_CreateEx(10, 120, 40, 20,
            WM_HBKWIN,
334     WM_CF_SHOW, 0, GUI_ID_BUTTON3);
335     BUTTON_SetText(channelDisplayMinus3, "---");
336     channelDisplayMinus2 = BUTTON_CreateEx(60, 120, 40, 20,
            WM_HBKWIN,
337     WM_CF_SHOW, 0, GUI_ID_BUTTON4);
338     BUTTON_SetText(channelDisplayMinus2, "--");
339     channelDisplayMinus1 = BUTTON_CreateEx(110, 120, 40, 20,
            WM_HBKWIN,
340     WM_CF_SHOW, 0, GUI_ID_BUTTON5);
341     BUTTON_SetText(channelDisplayMinus1, "-");
342
343     title2TextD = BUTTON_CreateEx(140, 150, 40, 20, WM_HBKWIN,
            WM_CF_SHOW, 0,
344     GUI_ID_BUTTON0);
345     BUTTON_SetText(title2TextD, channelDisplayOptionD);
346
347     channelDisplayAdd1 = BUTTON_CreateEx(170, 120, 40, 20,
            WM_HBKWIN,
348     WM_CF_SHOW, 0, GUI_ID_BUTTON6);
349     BUTTON_SetText(channelDisplayAdd1, "+");
350     channelDisplayAdd2 = BUTTON_CreateEx(220, 120, 40, 20,
            WM_HBKWIN,
351     WM_CF_SHOW, 0, GUI_ID_BUTTON7);
352     BUTTON_SetText(channelDisplayAdd2, "++");
353     channelDisplayAdd3 = BUTTON_CreateEx(270, 120, 40, 20,
            WM_HBKWIN,
354     WM_CF_SHOW, 0, GUI_ID_BUTTON8);
```

```
355       BUTTON_SetText(channelDisplayAdd3, "+++");
356
357       title3Text = TEXT_CreateEx(0, 170, 320, 20, WM_HBKWIN,
              WM_CF_SHOW,
358       TEXT_CF_VCENTER, GUI_ID_TEXT7, "Choose the channel to filter
              .");
359       TEXT_SetTextColor(title3Text, GUI_WHITE);
360       TEXT_SetTextAlign(title3Text, GUI_TA_HCENTER);
361       channelFilterMinus3 = BUTTON_CreateEx(10, 190, 40, 20,
              WM_HBKWIN,
362       WM_CF_SHOW, 0, GUI_ID_BUTTON3);
363       BUTTON_SetText(channelFilterMinus3, "---");
364       channelFilterMinus2 = BUTTON_CreateEx(60, 190, 40, 20,
              WM_HBKWIN,
365       WM_CF_SHOW, 0, GUI_ID_BUTTON4);
366       BUTTON_SetText(channelFilterMinus2, "--");
367       channelFilterMinus1 = BUTTON_CreateEx(110, 190, 40, 20,
              WM_HBKWIN,
368       WM_CF_SHOW, 0, GUI_ID_BUTTON5);
369       BUTTON_SetText(channelFilterMinus1, "-");
370
371       channelFilterAdd1 = BUTTON_CreateEx(170, 190, 40, 20,
              WM_HBKWIN, WM_CF_SHOW,
372             0, GUI_ID_BUTTON6);
373       BUTTON_SetText(channelFilterAdd1, "+");
374       channelFilterAdd2 = BUTTON_CreateEx(220, 190, 40, 20,
              WM_HBKWIN, WM_CF_SHOW,
375             0, GUI_ID_BUTTON7);
376       BUTTON_SetText(channelFilterAdd2, "++");
377       channelFilterAdd3 = BUTTON_CreateEx(270, 190, 40, 20,
              WM_HBKWIN, WM_CF_SHOW,
378             0, GUI_ID_BUTTON8);
379       BUTTON_SetText(channelFilterAdd3, "+++");
380
381       title3TextD = BUTTON_CreateEx(140, 220, 40, 20, WM_HBKWIN,
              WM_CF_SHOW, 0,
382       GUI_ID_BUTTON0);
383       BUTTON_SetText(title3TextD, filterDisplayOptionD);
384       exitButton = BUTTON_CreateEx(270, 0, 40, 20, WM_HBKWIN,
              WM_CF_SHOW, 0,
385       GUI_ID_BUTTON0);
386       BUTTON_SetText(exitButton, "Send");
387
388       GUI_Exec();
389  }
```

A.2 GUI Menu

```
1  #include "GUISetup.h"
2  #include <stdio.h>
3  #include <string.h>
4
5  GUI_POINT _aPoints[NUM_GUI_DISPLAY_BINS];
6  GRAPH_DATA_Handle _hDataXY;
7  GRAPH_DATA_Handle _hDataYT;
8  TEXT_Handle freqText;
9  TEXT_Handle showChannel;
10 TEXT_Handle showChannel2;
11 TEXT_Handle ampltudeText;
12 TEXT_Handle xLabel;
13 TEXT_Handle yLabel;
14 TEXT_Handle audioD;
15 TEXT_Handle audioFilter;
16 float32_t highest;
17 int value;
18 int value2;
19 char result[20];
20 char result2[20];
21 char channelShowContent[20];
22 char channelShowFiltered[20];
23 GRAPH_SCALE_Handle hScale;
24 GRAPH_SCALE_Handle hScale2;
25 WM_HWIN hGraph;
26 char dbText[] = "Magnitude (db)";
27 char vText[] = "Magnitude (V)";
28 char d1[] = "Unfiltered Frequency Domain";
29 char d2[] = "Unfiltered Time Domain";
30 char d3[] = "Filtered Frequency Domain";
31 char d4[] = "Filtered Time Domain";
32 GUI_RECT Rect = { 0, 1, 15, 238 };
33 GUI_RECT Rect2 = { 305, 1, 320, 238 };
34 WM_HWIN hItem;
35 float scaleXF;
36
37 void Graph_Setup() {
38     scaleXF = SAMPLE_FREQ / ((float) NUM_SAMPLES);
39
40     // GUI setup the x axis pixels for both freq and time domain
41     for (int i = 0; i < NUM_GUI_DISPLAY_BINS; i++) {
42         _aPoints[i].x = i;
43     }
44
45     //MULTIBUF is needed to ensure the screen is painted quickly
       , and fits within the 180MHz tick.
46     WM_MULTIBUF_Enable(1);
47
48     // Graph placement remains the same.
```

```
49      //The graph box is created and the parent is set as the
            window WIDGET.
50      hGraph = GRAPH_CreateEx(15, 0, LCD_GetXSize() - 29,
            LCD_GetYSize(),
51      WM_HBKWIN, WM_CF_HIDE, 0, GUI_ID_GRAPH0);
52      //The graph grid is 30 pixels large (x axis).
53      GRAPH_SetGridDistX(hGraph, 32);
54      //The graph grid is 20 pixels large (x axis).
55      GRAPH_SetGridDistY(hGraph, 20);
56      //The grid starts at 0
57      GRAPH_SetGridOffY(hGraph, 0);
58      //Enable grid
59      GRAPH_SetGridVis(hGraph, 1);
60
61      //Border around the graph to hold labels.
62      GRAPH_SetBorder(hGraph, 27, 10, 5, 30);
63      //Color of the border.
64      GRAPH_SetColor(hGraph, GUI_DARKGRAY, GRAPH_CI_BORDER);
65
66      //Scale for the dB/ Volt axis
67      hScale = GRAPH_SCALE_Create(12, GUI_TA_HCENTER |
            GUI_TA_VCENTER,
68      GRAPH_SCALE_CF_VERTICAL, 1);
69      GRAPH_AttachScale(hGraph, hScale);
70      //Scale for the kHz axis
71      hScale2 = GRAPH_SCALE_Create(220, GUI_TA_VCENTER |
            GUI_TA_HCENTER,
72      GRAPH_SCALE_CF_HORIZONTAL, 1);
73      GRAPH_AttachScale(hGraph, hScale2);
74      // Create TEXT labels for the dB and KHz scale
75      yLabel = TEXT_CreateEx(136, 223, 80, 20, WM_HBKWIN,
            WM_CF_SHOW,
76      TEXT_CF_VCENTER, GUI_ID_TEXT0, "kHz");
77      showChannel = TEXT_CreateEx(30, 223, 80, 20, WM_HBKWIN,
            WM_CF_SHOW,
78      TEXT_CF_VCENTER, GUI_ID_TEXT8, "Channel: ");
79      showChannel2 = TEXT_CreateEx(240, 223, 80, 20, WM_HBKWIN,
            WM_CF_SHOW,
80      TEXT_CF_VCENTER, GUI_ID_TEXT8, "");
81      //Execute all the GUI instructions before moving on to the
            next instruction in main.
82      //Enter into the GUI initializer for the graph.
83      freqText = TEXT_CreateEx(250, 223, 100, 20, WM_HBKWIN,
            WM_CF_SHOW,
84      TEXT_CF_VCENTER, GUI_ID_TEXT2, "");
85      audioD = TEXT_CreateEx(115, 120, 90, 20, WM_HBKWIN,
            WM_CF_HIDE,
86      TEXT_CF_VCENTER, GUI_ID_TEXT9, "Outputting audio.");
87      audioFilter = TEXT_CreateEx(120, 160, 80, 20, WM_HBKWIN,
            WM_CF_HIDE,
```

```
88      TEXT_CF_VCENTER, GUI_ID_TEXT6, "");
89      ampltudeText = TEXT_CreateEx(20, 223, 100, 20, WM_HBKWIN,
            WM_CF_SHOW,
90      TEXT_CF_VCENTER, GUI_ID_TEXT2, "");
91      _hDataXY = GRAPH_DATA_XY_Create(GUI_GREEN, NUM_SAMPLES,
            _aPoints,
92              GUI_COUNTOF(_aPoints));
93      GUI_SetColor(GUI_DARKGRAY);
94      GUI_SetTextMode(GUI_TM_XOR);
95      TEXT_SetText(audioFilter, "Filtering: 1");
96      GUI_Exec();
97  }
98
99  //Display the XY data in the frequency domain.
100 void GUI_XY_Graph(float32_t *y_data, int arr_length, uint8_t
        displayOption,
101         uint8_t channelDisplayOption) {
102     highest = 0;
103     value2 = highest;
104     // Remove the following if it works every time
105     // Check array length
106     if (arr_length > GUI_COUNTOF(_aPoints)) {
107         while (1)
108             ;
109     }
110
111     if (displayOption == 2) {
112         // Frequency domain - Unfiltered
113         // Input values: 0 to -100 dB
114         // Expected values from 0 to -200, so the dB value must
                be multiplied by 2 before calling this function.
115         // The actual values received (0 to -200) will be
                converted, resulting in 200 to 0.
116
117         // Offset y_data to center around the selected display
                channel.
118         y_data += 162 + (uint32_t) ((4.9152) * (
                channelDisplayOption - 1));
119
120         for (int i = 0; i < arr_length; i += 1) {
121             _aPoints[i].y = ((I16P) y_data[i]) + 200;
122
123 //          //Find highest point
124 //          if (y_data[i] > highest && i > 3) {
125 //              value = _aPoints[i].x * scaleXF;
126 //              highest = y_data[i];
127 //          }
128         }
129     } else {
130         y_data += (290*FFT_FILTERED_ZOOM - 128) + (uint32_t)
```

```
                ((4.9152 * FFT_FILTERED_ZOOM) * (channelDisplayOption
                - 1));
131
132             for (int i = 0; i < arr_length; i += 1) {
133                 _aPoints[i].y = ((I16P) y_data[i]) + 200;
134
135 //              //Find highest point
136 //              if (y_data[i] > highest && i > 3) {
137 //                  value = _aPoints[i].x * scaleXF;
138 //                  highest = y_data[i];
139 //              }
140             }
141         }
142         // Convert value2 to dB
143         value2 = (int) highest;
144
145         // Ignore noise
146         if (value2 < -54) {
147             value = 0;
148             value2 = 0;
149         }
150         sprintf(result, "%i Hz", value);
151         sprintf(result2, "%i dB", value2);
152
153         // Get the GRAPH handle
154         hItem = WM_GetFirstChild(WM_HBKWIN);
155
156         // Detach data handle and delete it to ensure the same
157         // variable is used and the memory is saved.
158         GRAPH_DetachData(hItem, _hDataXY);
159         GRAPH_DATA_XY_Delete(_hDataXY);
160
161         // Create a new XY data object array for the graph.
162         _hDataXY = GRAPH_DATA_XY_Create(GUI_GREEN, arr_length,
                _aPoints,
163                 GUI_COUNTOF(_aPoints));
164         //Attach the data to the graph (prime it).
165         GRAPH_AttachData(hItem, _hDataXY);
166         //Type of line (default).
167         GRAPH_DATA_XY_SetLineStyle(_hDataXY, GUI_LS_SOLID);
168         // Update the pointer if the User button is being pressed.
169         //Pauses the graph for easy showing potential.
170         //Print the Hz of the expected input signal
171 //          sprintf(result,"%i Hz",value);
172 //          sprintf(result2,"%i dB",value2);
173 //          TEXT_SetText(freqText, result);
174 //          TEXT_SetText(ampltudeText,result2);
175         GUI_Exec();
176 }
177
```

```
178 // Display the XY data in the time domain.
179 // domain = 0 is freq 1 is time
180 // PrePost = 0 is pre and 1 is post prosessing
181 void GUI_GraphChange(uint8_t displayOption, int8_t channel,
182         int8_t filterChannel) {
183     sprintf(channelShowFiltered, "Filtering: %i", filterChannel)
        ;
184     WM_HideWindow(audioD);
185     WM_HideWindow(audioFilter);
186     if (displayOption == 1) {
187         hItem = WM_GetFirstChild(WM_HBKWIN);
188         WM_HideWindow(hItem);
189         GUI_SetBkColor(GUI_BLACK);
190         GUI_Clear();
191         WM_ShowWindow(audioD);
192         WM_ShowWindow(audioFilter);
193         TEXT_SetTextColor(audioD, GUI_WHITE);
194         TEXT_SetTextAlign(audioD, GUI_TA_HCENTER);
195         TEXT_SetText(audioFilter, channelShowFiltered);
196         TEXT_SetTextColor(audioFilter, GUI_WHITE);
197         TEXT_SetTextAlign(audioFilter, GUI_TA_HCENTER);
198     } else if (displayOption == 2) {
199         WM_ShowWindow(hGraph);
200         GUI_FillRectEx(&Rect);
201         GUI_DispStringInRectEx(dbText, &Rect, GUI_TA_HCENTER |
            GUI_TA_VCENTER,
202                 strlen(dbText), GUI_ROTATE_CCW);
203
204         //The handlers for the x and y scale. The parent window
            is the WM_HWIN.
205         //The graph needs a handle to ensure data can be changed
            .
206         //No decimals are needed
207         GRAPH_SCALE_SetNumDecs(hScale, 0);
208         //Show from 0 till -100 dB thus the scale must be
            divided by 2.
209         GRAPH_SCALE_SetFactor(hScale, 0.5);
210         //20 pixels between dB labels.
211         GRAPH_SCALE_SetTickDist(hScale, 20);
212         //Graph shows from 0 till -100 dB
213         GRAPH_SCALE_SetOff(hScale, 200);
214         //Change colour and attach scale.
215         GRAPH_SCALE_SetTextColor(hScale, GUI_BLACK);
216
217         //The labels show till the closest 100Hz
218         GRAPH_SCALE_SetNumDecs(hScale2, 0);
219         //Each bin is 93,75 Hz thus each pixel must be
            multiplied by this factor.
220         GRAPH_SCALE_SetFactor(hScale2, (float) (scaleXF /
            1000.0));
```

```
221             //30 pixels between kHz labels.
222             GRAPH_SCALE_SetTickDist(hScale2, 32);
223             //Graph starts at 0 and fits the 257 bins.
224             GRAPH_SCALE_SetOff(hScale2, -(float) (162 + (4.9152 * (
                    channel - 1))));
225             //Attach the y scale
226             GRAPH_SCALE_SetTextColor(hScale2, GUI_BLACK);
227             TEXT_SetText(yLabel, "Frequency (kHz)");
228             sprintf(channelShowContent, "Channel: %i", channel);
229             TEXT_SetText(showChannel, channelShowContent);
230             TEXT_SetText(showChannel2, "");
231             GUI_FillRectEx(&Rect2);
232             GUI_DispStringInRectEx(d1, &Rect2, GUI_TA_HCENTER |
                    GUI_TA_VCENTER,
233                     strlen(d1), GUI_ROTATE_CCW);
234         } else if (displayOption == 3) {
235             WM_ShowWindow(hGraph);
236             GUI_FillRectEx(&Rect);
237             GUI_DispStringInRectEx(dbText, &Rect, GUI_TA_HCENTER |
                    GUI_TA_VCENTER,
238                     strlen(dbText), GUI_ROTATE_CCW);
239
240             //The handlers for the x and y scale. The parent window
                    is the WM_HWIN.
241             //The graph needs a handle to ensure data can be changed
                    .
242             //No decimals are needed
243             GRAPH_SCALE_SetNumDecs(hScale, 0);
244             //Show from 0 till -100 dB thus the scale must be
                    divided by 2.
245             GRAPH_SCALE_SetFactor(hScale, 0.5);
246             //20 pixels between dB labels.
247             GRAPH_SCALE_SetTickDist(hScale, 20);
248             //Graph shows from 0 till -100 dB
249             GRAPH_SCALE_SetOff(hScale, 200);
250             //Change colour and attach scale.
251             GRAPH_SCALE_SetTextColor(hScale, GUI_BLACK);
252
253             //The labels show till the closest 100Hz
254             GRAPH_SCALE_SetNumDecs(hScale2, 0);
255             //Each bin is 93,75 Hz thus each pixel must be
                    multiplied by this factor.
256             GRAPH_SCALE_SetFactor(hScale2, scaleXF / (1000.0 *
                    FFT_FILTERED_ZOOM));
257             //30 pixels between kHz labels.
258             GRAPH_SCALE_SetTickDist(hScale2, 32);
259             //Graph starts at 0 and fits the 257 bins.
260             GRAPH_SCALE_SetOff(hScale2, -((290*FFT_FILTERED_ZOOM -
                    128) + ((4.9152*FFT_FILTERED_ZOOM) * (channel - 1))))
                    ;
```

```
261             //Attach the y scale
262             GRAPH_SCALE_SetTextColor(hScale2, GUI_BLACK);
263             TEXT_SetText(yLabel, "Frequency (kHz)");
264
265             GUI_FillRectEx(&Rect2);
266             GUI_DispStringInRectEx(d3, &Rect2, GUI_TA_HCENTER |
                    GUI_TA_VCENTER,
267                     strlen(d3), GUI_ROTATE_CCW);
268             sprintf(channelShowContent, "Channel: %i", channel);
269             TEXT_SetText(showChannel, channelShowContent);
270             TEXT_SetText(showChannel2, channelShowFiltered);
271         }
272         GUI_Exec();
273     }
```

A.3 Elliptic coefficients

```
 1 {17, 0, -18, 17, 20498, -16341, 16384, 0, -23079, 16384, 20713,
      -16342},
 2 {17, 0, -17, 17, 20111, -16341, 16384, 0, -22729, 16384, 20329,
      -16342},
 3 {17, 0, -17, 17, 19719, -16341, 16384, 0, -22373, 16384, 19940,
      -16342},
 4 {17, 0, -17, 17, 19323, -16341, 16384, 0, -22013, 16384, 19546,
      -16342},
 5 {17, 0, -16, 17, 18922, -16341, 16384, 0, -21648, 16384, 19148,
      -16342},
 6 {17, 0, -16, 17, 18518, -16341, 16384, 0, -21277, 16384, 18746,
      -16342},
 7 {17, 0, -15, 17, 18109, -16341, 16384, 0, -20902, 16384, 18339,
      -16342},
 8 {17, 0, -15, 17, 17696, -16341, 16384, 0, -20522, 16384, 17928,
      -16342},
 9 {17, 0, -14, 17, 17278, -16341, 16384, 0, -20138, 16384, 17514,
      -16342},
10 {17, 0, -14, 17, 16857, -16341, 16384, 0, -19749, 16384, 17095,
      -16342},
11 {17, 0, -14, 17, 16433, -16341, 16384, 0, -19355, 16384, 16672,
      -16341},
12 {17, 0, -13, 17, 16004, -16341, 16384, 0, -18957, 16384, 16245,
      -16341},
13 {17, 0, -13, 17, 15572, -16341, 16384, 0, -18555, 16384, 15815,
      -16341},
14 {17, 0, -12, 17, 15136, -16341, 16384, 0, -18148, 16384, 15381,
      -16341},
15 {17, 0, -12, 17, 14697, -16341, 16384, 0, -17738, 16384, 14944,
      -16341},
16 {17, 0, -11, 17, 14254, -16341, 16384, 0, -17323, 16384, 14503,
      -16341},
17 {17, 0, -11, 17, 13808, -16341, 16384, 0, -16904, 16384, 14059,
      -16341},
18 {17, 0, -10, 17, 13359, -16341, 16384, 0, -16482, 16384, 13612,
      -16341},
19 {17, 0, -10, 17, 12907, -16341, 16384, 0, -16056, 16384, 13162,
      -16341},
20 {17, 0, -9, 17, 12452, -16341, 16384, 0, -15626, 16384, 12708,
      -16341},
21 {17, 0, -9, 17, 11995, -16341, 16384, 0, -15193, 16384, 12252,
      -16341},
22 {17, 0, -8, 17, 11534, -16341, 16384, 0, -14756, 16384, 11793,
      -16341},
23 {17, 0, -8, 17, 11071, -16341, 16384, 0, -14316, 16384, 11332,
      -16341},
24 {17, 0, -7, 17, 10605, -16341, 16384, 0, -13873, 16384, 10867,
      -16341},
```

```
25 {17, 0, -7, 17, 10137, -16341, 16384, 0, -13426, 16384, 10401,
     -16341},
26 {17, 0, -6, 17, 9667, -16341, 16384, 0, -12977, 16384, 9932,
     -16341},
27 {17, 0, -6, 17, 9194, -16341, 16384, 0, -12524, 16384, 9460,
     -16341},
28 {17, 0, -6, 17, 8720, -16341, 16384, 0, -12069, 16384, 8987,
     -16341},
29 {17, 0, -5, 17, 8243, -16341, 16384, 0, -11611, 16384, 8511,
     -16341},
30 {17, 0, -5, 17, 7765, -16341, 16384, 0, -11150, 16384, 8034,
     -16341},
31 {17, 0, -4, 17, 7284, -16341, 16384, 0, -10687, 16384, 7555,
     -16341},
32 {17, 0, -4, 17, 6802, -16341, 16384, 0, -10221, 16384, 7074,
     -16341},
33 {17, 0, -3, 17, 6319, -16341, 16384, 0, -9754, 16384, 6591,
     -16341},
34 {17, 0, -3, 17, 5834, -16341, 16384, 0, -9283, 16384, 6107,
     -16341},
35 {17, 0, -2, 17, 5348, -16341, 16384, 0, -8811, 16384, 5621,
     -16341},
36 {17, 0, -2, 17, 4860, -16341, 16384, 0, -8337, 16384, 5135,
     -16341},
37 {17, 0, -1, 17, 4372, -16341, 16384, 0, -7861, 16384, 4647,
     -16341},
38 {17, 0, -1, 17, 3882, -16341, 16384, 0, -7383, 16384, 4158,
     -16341},
39 {17, 0, 0, 17, 3392, -16341, 16384, 0, -6903, 16384, 3668,
     -16341},
40 {17, 0, 0, 17, 2901, -16341, 16384, 0, -6422, 16384, 3177,
     -16341},
41 {17, 0, 1, 17, 2409, -16341, 16384, 0, -5940, 16384, 2685,
     -16341},
42 {17, 0, 1, 17, 1916, -16341, 16384, 0, -5456, 16384, 2193,
     -16341},
43 {17, 0, 2, 17, 1424, -16341, 16384, 0, -4971, 16384, 1701,
     -16341},
44 {17, 0, 2, 17, 930, -16341, 16384, 0, -4484, 16384, 1208,
     -16341},
45 {17, 0, 3, 17, 437, -16341, 16384, 0, -3997, 16384, 714,
     -16341},
46 {17, 0, 3, 17, -56, -16341, 16384, 0, -3509, 16384, 221,
     -16341},
47 {17, 0, -3, 17, -273, -16341, 16384, 0, 3834, 16384, -550,
     -16341},
48 {17, 0, -3, 17, -766, -16341, 16384, 0, 4322, 16384, -1043,
     -16341},
49 {17, 0, -2, 17, -1259, -16341, 16384, 0, 4809, 16384, -1536,
     -16341},
```

```
50 {17, 0, -2, 17, -1752, -16341, 16384, 0, 5294, 16384, -2029,
    -16341},
51 {17, 0, -1, 17, -2245, -16341, 16384, 0, 5779, 16384, -2521,
    -16341},
52 {17, 0, -1, 17, -2737, -16341, 16384, 0, 6262, 16384, -3013,
    -16341},
53 {17, 0, 0, 17, -3228, -16341, 16384, 0, 6743, 16384, -3504,
    -16341},
54 {17, 0, 0, 17, -3719, -16341, 16384, 0, 7223, 16384, -3994,
    -16341},
55 {17, 0, 1, 17, -4209, -16341, 16384, 0, 7702, 16384, -4484,
    -16341},
56 {17, 0, 1, 17, -4698, -16341, 16384, 0, 8178, 16384, -4972,
    -16341},
57 {17, 0, 2, 17, -5186, -16341, 16384, 0, 8653, 16384, -5459,
    -16341},
58 {17, 0, 2, 17, -5672, -16341, 16384, 0, 9126, 16384, -5945,
    -16341},
59 {17, 0, 3, 17, -6158, -16341, 16384, 0, 9597, 16384, -6430,
    -16341},
60 {17, 0, 3, 17, -6641, -16341, 16384, 0, 10066, 16384, -6913,
    -16341},
61 {17, 0, 4, 17, -7124, -16341, 16384, 0, 10532, 16384, -7394,
    -16341},
62 {17, 0, 4, 17, -7605, -16341, 16384, 0, 10996, 16384, -7874,
    -16341},
63 {17, 0, 5, 17, -8084, -16341, 16384, 0, 11458, 16384, -8352,
    -16341},
64 {17, 0, 5, 17, -8561, -16341, 16384, 0, 11917, 16384, -8829,
    -16341},
65 {17, 0, 6, 17, -9036, -16341, 16384, 0, 12373, 16384, -9303,
    -16341},
66 {17, 0, 6, 17, -9510, -16341, 16384, 0, 12826, 16384, -9775,
    -16341},
67 {17, 0, 7, 17, -9981, -16341, 16384, 0, 13277, 16384, -10245,
    -16341},
68 {17, 0, 7, 17, -10450, -16341, 16384, 0, 13724, 16384, -10712,
    -16341},
69 {17, 0, 8, 17, -10916, -16341, 16384, 0, 14169, 16384, -11177,
    -16341},
70 {17, 0, 8, 17, -11380, -16341, 16384, 0, 14610, 16384, -11640,
    -16341},
71 {17, 0, 9, 17, -11841, -16341, 16384, 0, 15048, 16384, -12100,
    -16341},
72 {17, 0, 9, 17, -12300, -16341, 16384, 0, 15482, 16384, -12557,
    -16341},
73 {17, 0, 10, 17, -12756, -16341, 16384, 0, 15913, 16384, -13011,
    -16341},
74 {17, 0, 10, 17, -13209, -16341, 16384, 0, 16340, 16384, -13462,
    -16341},
```

```
75 {17, 0, 11, 17, -13659, -16341, 16384, 0, 16764, 16384, -13911,
      -16341},
76 {17, 0, 11, 17, -14106, -16341, 16384, 0, 17184, 16384, -14356,
      -16341},
77 {17, 0, 12, 17, -14549, -16341, 16384, 0, 17600, 16384, -14797,
      -16341},
78 {17, 0, 12, 17, -14990, -16341, 16384, 0, 18012, 16384, -15236,
      -16341},
79 {17, 0, 12, 17, -15427, -16341, 16384, 0, 18420, 16384, -15671,
      -16341},
80 {17, 0, 13, 17, -15860, -16341, 16384, 0, 18823, 16384, -16102,
      -16341},
81 {17, 0, 13, 17, -16290, -16341, 16384, 0, 19223, 16384, -16530,
      -16341},
82 {17, 0, 14, 17, -16716, -16341, 16384, 0, 19618, 16384, -16954,
      -16341},
83 {17, 0, 14, 17, -17139, -16341, 16384, 0, 20008, 16384, -17374,
      -16342},
84 {17, 0, 15, 17, -17557, -16341, 16384, 0, 20395, 16384, -17791,
      -16342},
85 {17, 0, 15, 17, -17971, -16341, 16384, 0, 20776, 16384, -18203,
      -16342},
86 {17, 0, 16, 17, -18382, -16341, 16384, 0, 21153, 16384, -18611,
      -16342},
87 {17, 0, 16, 17, -18788, -16341, 16384, 0, 21525, 16384, -19015,
      -16342},
88 {17, 0, 16, 17, -19190, -16341, 16384, 0, 21892, 16384, -19414,
      -16342},
89 {17, 0, 17, 17, -19587, -16341, 16384, 0, 22254, 16384, -19809,
      -16342},
90 {17, 0, 17, 17, -19981, -16341, 16384, 0, 22611, 16384, -20200,
      -16342},
91 {17, 0, 18, 17, -20369, -16341, 16384, 0, 22963, 16384, -20586,
      -16342},
92 {17, 0, 18, 17, -20753, -16341, 16384, 0, 23310, 16384, -20967,
      -16342},
93 {17, 0, 19, 17, -21132, -16341, 16384, 0, 23651, 16384, -21343,
      -16342},
94 {17, 0, 19, 17, -21507, -16341, 16384, 0, 23987, 16384, -21715,
      -16342},
95 {17, 0, 19, 17, -21876, -16341, 16384, 0, 24318, 16384, -22082,
      -16342},
96 {17, 0, 20, 17, -22241, -16341, 16384, 0, 24643, 16384, -22443,
      -16342},
97 {17, 0, 20, 17, -22600, -16341, 16384, 0, 24963, 16384, -22800,
      -16342},
98 {17, 0, 21, 17, -22954, -16341, 16384, 0, 25276, 16384, -23151,
      -16342},
99 {17, 0, 21, 17, -23303, -16341, 16384, 0, 25585, 16384, -23498,
      -16342},
```

```
100 {17, 0, 21, 17, -23647, -16341, 16384, 0, 25887, 16384, -23838,
       -16342},
101 {17, 0, 22, 17, -23986, -16341, 16384, 0, 26184, 16384, -24174,
       -16342},
102 {17, 0, 22, 17, -24319, -16341, 16384, 0, 26474, 16384, -24504,
       -16342},
103 {17, 0, 22, 17, -24646, -16341, 16384, 0, 26759, 16384, -24828,
       -16342},
104 {17, 0, 23, 17, -24968, -16341, 16384, 0, 27037, 16384, -25147,
       -16342},
105 {17, 0, 23, 17, -25284, -16341, 16384, 0, 27310, 16384, -25460,
       -16342},
106 {17, 0, 23, 17, -25595, -16341, 16384, 0, 27576, 16384, -25767,
       -16342},
107 {17, 0, 24, 17, -25899, -16341, 16384, 0, 27836, 16384, -26068,
       -16342},
108 {17, 0, 24, 17, -26198, -16341, 16384, 0, 28090, 16384, -26363,
       -16342},
109 {17, 0, 24, 17, -26491, -16341, 16384, 0, 28337, 16384, -26653,
       -16342},
110 {17, 0, 25, 17, -26777, -16341, 16384, 0, 28578, 16384, -26936,
       -16342},
111 {17, 0, 25, 17, -27058, -16341, 16384, 0, 28813, 16384, -27213,
       -16342},
112 {17, 0, 25, 17, -27332, -16341, 16384, 0, 29041, 16384, -27484,
       -16342},
113 {17, 0, 26, 17, -27601, -16341, 16384, 0, 29262, 16384, -27749,
       -16342},
114 {17, 0, 26, 17, -27863, -16341, 16384, 0, 29477, 16384, -28008,
       -16342},
115 {17, 0, 26, 17, -28118, -16341, 16384, 0, 29685, 16384, -28260,
       -16342},
116 {17, 0, 27, 17, -28368, -16341, 16384, 0, 29887, 16384, -28505,
       -16342},
117 {17, 0, 27, 17, -28610, -16341, 16384, 0, 30082, 16384, -28745,
       -16342},
118 {17, 0, 27, 17, -28847, -16341, 16384, 0, 30270, 16384, -28977,
       -16342},
119 {17, 0, 27, 17, -29076, -16341, 16384, 0, 30451, 16384, -29203,
       -16342},
120 {17, 0, 28, 17, -29299, -16341, 16384, 0, 30626, 16384, -29423,
       -16342}
```