Building Billion Tap Filters

Henry, M. & Owen, J.

DOI 10.1109/MIE.2020.3036040
ISSN 1932-4529
ESSN 1941-0115

Publisher: Institute of Electrical and Electronics Engineers

Building Billion Tap Filters

Abstract

Signal processing engineers recognise two main classes of filter: Finite Impulse Response (FIR) and Infinite Impulse Response (IIR), each with distinct properties. The Prism is a new filter type which combines the best of both: the numerical stability and linear phase of FIR, together with the low-cost calculation of IIR. Here, we show how to build ultra-narrowband FIR filters, containing over one billion samples (taps), using a sequence of Prisms. While designing and operating equivalent conventional FIR filters would require supercomputer resources, these Prism filters are designed instantly and implemented on a single FPGA.

(Main Text Begins)

Open any introductory textbook on signal processing (e.g. [1]), and you'll read that there are two main classes of digital filter: Finite Impulse Response (FIR), and Infinite Impulse Response (IIR) (e.g. Chapters 5 and 6 in [1]). The output of an FIR filter is given by the weighted sum of the input values held in a fixed length data window, i.e. a form of convolution calculation. Each position (or 'tap') in the filter window has its own weight or coefficient, so calculating the filter output with a window of length *n* samples uses *n* multiplications and additions (or accumulations). The tap coefficients are positional: when a new sample arrives, all the old data is shuffled along one position, and the whole calculation has to be repeated in full. The set of positional coefficients is collectively known as the impulse response, as it also describes the filter's response over time to an input consisting of an impulse of unit amplitude and one sample duration. FIR filters are 'finite' because, once the impulse has passed through the filter window, the filter output is zero. Although longer FIR filters can provide better filtering performance (e.g. improved noise suppression and/or a narrower frequency passband), it is uncommon to use an FIR filter with more than a few hundred taps, as the computational burden becomes excessive. This is a major limitation, because FIR filters have many useful properties, including numerical stability, and linear phase.

The alternative form, the IIR filter, can provide powerful filtering with low computational cost, using only a short window of input samples. This is achieved by including additional coefficients for the most recent filter *outputs,* creating a form of feedback. IIR filters are 'infinite' because of this feedback – every sample in the input history has some residual influence on the filter output. This comes at the cost of non-linear phase characteristics and the potential for numerical instability.

The ideal would be to combine the properties of IIR and FIR – to retain the numerical stability and linear phase response of FIR, while having the reduced computational requirement of IIR, so that powerful filters can be constructed. This can be achieved by retaining the basic FIR form – a window of samples with no output feedback, while finding a form of recursive calculation that drastically reduces the computational requirement. This has now been achieved, using a new technique called Prism Signal Processing.

By way of illustration, Figure 1a plots the scaled impulse response of an FIR filter consisting of 1,543,359,520 samples. The individual points are shown without a connecting line, but are packed together so tightly they give the appearance of 11 continuous lines, forming an onion-like pattern. Figure 1b shows the heart of the impulse response in detail, revealing its true structure. Here, a line connects consecutive points for clarity. The impulse response consists of a sinusoid with a period of exactly 20 samples. This behaviour extends across the whole of Figure 1a, the magnitude varying slowly with sample number. Figure 1c shows the corresponding frequency response. Given sample rate f_s, the response consists of a narrow spike at $f_c = 0.05\,f_s$ (corresponding to a period of exactly 20 samples), with extreme attenuation at all other frequencies. Figure 1d shows Figure 1c in detail, revealing a narrow bandpass filter centred on f_c. The passband (taken as gain > -3dB) is the region around f_c within $\pm\, 1.25 \times 10^{-9} f_s$; all frequencies more distant than $3.025 \times 10^{-9} f_s$ from the central frequency f_c have a gain below –80 dB.

To put some perspective on these figures, consider using a sample rate f_s = 20 MHz. In this case the 1.5 Gtap filter window covers 77.2 seconds of data. The central frequency f_c is 1 MHz. The pass band is f_c ± 0.025 Hz. All frequencies outside of f_c ± 0.0605 Hz have a gain of -80 dB or less. Of course, running a 1.5 Gtap filter at 20 MHz is likely to need significant computing power. Using convolution, some 3.09×10^{16} multiply and accumulate operations (MACC) would be required every second. Translated into floating point operations (1 MACC = 2 Flop), and using the preferred unit of the Top 50 supercomputer website [2], this amounts to 61,734 Terraflops/s; any device delivering this performance would be placed just below the top 5 in the June 2020 supercomputing rankings. A further question arising from Figure 1 is to ask how such a large FIR filter could have been designed. To address these issues, we must examine the underlying technique.

Prism Signal Processing

The Prism is a novel type of FIR filter, with two key characteristics which make billion tap filtering possible. Firstly, the Prism doesn't use convolution to perform the filtering calculation. Instead a recursive calculation is employed, so the computational cost is low and fixed irrespective of the filter length – even for billions of samples. Secondly, Prism design is trivial, once parameter values have been selected, so that new, even very large, filters are easily constructed, in real time if required. Reference [3] gives an introduction to Prism Signal Processing in the context of the Internet of Things, while the mathematics of the basic operation, providing an alternative to convolution, is presented more fully in [4]. Here a brief review of its characteristics is given, before explaining how it can be used to construct Gtap filters.

For an input signal x(τ), a Prism calculates double integrals of the form:

$$I^{h}_{[S|C][S|C]} = m^2 \int_{-\frac{1}{m}}^{0} [\sin|\cos](2\pi hmt) \left(\int_{t-\frac{1}{m}}^{0} [\sin|\cos](2\pi hmt) \cdot x(\tau) d\tau \right) dt \qquad (1)$$

where *t* (or *τ*) is time, the subscript [*s*|*c*] indicates either *s* (sin) or *c* (cosine), *h* is the harmonic number, a positive integer, and *m* is the characteristic frequency of the Prism. Equation (1) describes a family of discrete Fourier Transform (DFT)–style double integrals. At each stage the input signal is multiplied by a modulating sine or cosine function and is then integrated over the period of the characteristic frequency *m*. The harmonic number *h* (a positive integer) determines how many whole cycles of the modulation function occur over the period of integration 1/*m*. In any Prism, either one or two pairs of such integrals are updated every sample, using an efficient sliding window calculation.

For a sinusoidal input signal (or one frequency component in a multi-component signal)

$$x(\tau) = A\sin(2\pi f\tau + \phi) \tag{2}$$

where A is amplitude, f is frequency, and ϕ is the phase at time *t* = 0, the Prism output is one or both of G_s^h and G_c^h, defined as follows:

$$G_s^h = I_{ss}^h + I_{cc}^h = A\operatorname{sinc}^2(r)\frac{r^2}{r^2-h^2}\sin(\phi - 2\pi r) \tag{3}$$

$$G_c^h = I_{cs}^h - I_{sc}^h = A\operatorname{sinc}^2(r)\frac{hr}{r^2-h^2}\cos(\phi - 2\pi r) \tag{4}$$

Here *r* = *f*/*m* is the dimensionless frequency of the input sinusoid *x*(*τ*).

Where two outputs are generated, G_s^h and G_c^h have sine and cosine terms with the same phase offset, and so form an orthogonal pair. From this pair of values, estimates of the current frequency, amplitude and/or phase of the input sinusoid *x*(*τ*) can be calculated using a few simple steps [3]. Dual output Prisms can be used to create “trackers” monitoring one or more of these sinusoidal parameters. Here, we focus on Prisms generating the G_s^h output only, as these form the basis of the Gtap filter example presented in Figure 1.

The key mathematical property of the Prism, as detailed in [4], is that each integration stage can be performed on a sliding window (i.e. recursive) basis, because equations (3) and (4) are valid irrespective of the initial phase of the modulation functions. This means that when the integrals are updated, only the newest value need be added and the oldest value removed. This simplification results in a low and fixed computational cost for Prism update, however long the filter window. The analytical results developed in continuous time can be applied effectively unchanged in discrete, sampled time by using an efficient time series implementation of Romberg Integration [4], which generates high precision outputs. The DFT-style calculation also means that Prism design is trivial: given desired values of *m* and *h*, the equivalent of the tap coefficients are linearly spaced sine and cosine values, which are readily calculated. The combined properties of efficient, recursive calculation, trivial design, high precision, and linear phase, all support the use of the Prism as a convenient building block for signal processing.

The key to understanding the Prism's computational efficiency is to appreciate how it is possible to side-step the requirement for a convolution calculation. The impulse response for any non-trivial FIR filter must vary with position inside the filter window: this is clearly the case for the Prism filter shown in Figure 1. So, the weight of each sample, as a linear contribution to the overall filter output, must also vary as it passes through the data window. Until now, the default means for achieving this has been to determine, during filter design, the weight of each position in the filter window (i.e. by calculating or indeed designing/selecting the impulse response), and then, during filter operation, to compute the filter output explicitly as Σ (sample value × positional weight), via the computationally expensive means of convolution. In the Prism, a fixed (sinusoidal) weight is assigned to each sample in the first level integral. This weight does not change as it passes through the first level integral window, so a full convolution calculation is not required. Nevertheless, in the second level integrals and the filter output, the contribution of each input sample does vary, most obviously as a function of its residency time in the first level integrals [3]. So, the double integral structure of the Prism provides a means of generating the position-varying weighting required for any non-trivial filter, while only applying a single weight to each sample as it passes through the first level integrals, resulting in a computationally efficient evaluation. Only a limited number of impulse responses can be generated using this indirect technique. However, given the low computational cost, it becomes feasible to create a wider range of impulse responses by combining Prisms into networks.

Bandpass Gtap Filter Design

The filter in Figure 1 is generated from a design template consisting of a chain of six Prisms in series, each generating the G_s^h output (Figure 2). The bandpass filter design procedure is detailed in [5]; only a summary and worked example are provided here. The design principle is simple. The G_s^h output has two peaks in its gain ([2], [5]) with corresponding frequencies (for $h > 20$) of approximately $m \times (h - 0.371)$ and $m \times (h - 0.371)$ Hz. To create a bandpass filter with a single peak at a desired central frequency f_c, pairs of Prisms are applied in series, where first the lower peak and then the upper peak coincides with f_c. For any h, there are thus two corresponding values of m, one for each of the gain peaks. The selection of a suitable h for the Prism pair is based on the desired filter bandwidth: the peak width is proportional to m, so that a higher h and hence lower m results in a narrower peak (measured in Hz).

For the six Prism generic filter design, three Prism pairs are used, where the respective values of h are selected in a fixed ratio 6:4:3; this ratio has been found to provide good attenuation of the sidelobes [5]. A useful but simplified analogy for the bandpass filter design is the creation of a set of gear wheels (Figure 3). Each wheel represents a Prism: h is the number of (sinusoidal) gear teeth, while the wheel circumference is the Prism's time period i.e. $2/m$, and the sum of all six circumferences gives the time period of the entire filter. Note that this gear wheel analogy is a useful representation of the Prism m and h parameters to be selected in the design process, and their role in the generation of the modulation function in each Prism integral (Equation 1), but does not offer a meaningful analogue of the filtering process itself.

In Figure 3, the wheels are arranged in pairs (P_1 and P_2, etc), where each pair has the same number of teeth, but the size of the teeth (and the wheels) are slightly different. The blue wheels have teeth of circumferential length corresponding to the period of the lower gain peak, while the green wheels match the upper gain peak. The design template is generic: the gear ratio 6:4:3 delivers the desired performance. To construct any particular gear system, the requirement is simply to determine the number of teeth and the circumference of each wheel. With these parameters assigned, the construction of each individual gear wheels is straightforward, and the performance of the whole system is guaranteed. Similarly, the Prism bandpass filter design problem is specified by the required bandwidth and central frequency. These values are used to determine *h* and *m* for each Prism, from which the design is straightforward.

For example, to create the filter in Figure 1, the design parameters are: central frequency $f_c = 1$ MHz, bandwidth b = 0.025 Hz, and the sample rate f_s = 20 MHz. The generic design rules are as follows. The *h* values for each Prism pair are given by:

$$h = round\,(0.0371 \times f_c/b \times [6, 4, 3]) \qquad (5)$$

where the *round* function rounds to the nearest integer value (by analogy, each gear wheel must have an integral number of teeth). The linear scaling factor 0.0371 delivers the required bandwidth (using gain > -3 dB). For each *h* value, the corresponding values of *m* for the Prism pair (to ensure f_c coincides with the *h*-peak and *h+1*-peak respectively) are given by:

$$m_1 = f_c/(h + 0.371) \text{ and } m_2 = f_c/(h - 0.371) \qquad (6)$$

That's all that is required; once the *m* and *h* values are calculated, the corresponding Prisms are readily constructed for the desired sampling rate f_s. Table 1 shows the resulting values of *m* and *h* for the example of Figure 1, and assuming a sample rate f_s = 20MHz. Note that in this case the high f_c/b ratio results in large values of *h,* or, returning to the gear analogy, millions of teeth on each wheel.

Table 1. Bandpass filter parameters for f_c = 1 MHz, f_s = 20 MHz and b = 0.025 Hz

Prism Index, i	h^i	m^i (Hz)	Integral length (samples)
1	8,904,000	0.1123090796	178,079,992
2	8,904,000	0.1123090695	178,080,008
3	5,935,994	0.1684637935	118,719,872
4	5,935,994	0.1684637708	118,719,888
5	4,452,000	0.2246181693	89,039,992
6	4,452,000	0.2246181290	89,040,008
		Total Filter Length	**1,543,359,520**

Also shown in Table 1 are the lengths of each Prism integral in samples, given by f_s/m. Each Prism contains two integral stages, so the total sample length of each Prism is double the integral length. With the Prisms arranged in series, the total filter length is just over 1.5 billion samples. As discussed in [4], the evaluation cost of each single output Prism is just 13 multiplications and 33 additions, so the total computational burden of updating this filter is around 300 flops. By contrast, the conventional filter evaluation via convolution would require around 1.5 billion each of multiplications and additions, so the Prism calculation is approximately 10 million times more efficient. As will be seen below, such filters are readily run in real time on a single FPGA.

Given the simplicity of the design procedure there is considerable freedom in the selection of f_s, f_c, and b. Depending on certain implementation details, larger precision errors are observed for $f_c > 0.4\ f_s$, as the Nyquist limit is approached. In any practical implementation, available memory is likely to constrain how narrow a filter can be constructed; the sample length can be estimated using $1.93 \times f_s / b$, which gives 1.544×10^9 samples in this case. Depending on the number format used and other implementation details, the required memory size can be estimated accordingly [4]. The converse problem may also arise – if the filter passband is too wide, the values of h will drop below 20 and the filter performance will have reduced efficiency. However, between these two extremes, there is no further restriction on the ratio between f_c and f_s. In the example given here, an integer ratio of 1:20

was selected so that Fig 1a would have a discernible structure; for the vast majority of f_c / f_s ratios the complete impulse response would appear fully shaded with no visible features. An example of a non-integer ratio is given in the demonstration system described in the next section.

Demonstrator system

An FPGA-based demonstrator system has been created which implements the six Prism bandpass filter design of the type illustrated in Figures 1 and 2. It further includes a tracking Prism which calculates the frequency and amplitude of the filtered signal, so that the resulting measurement performance can be assessed. This system has been implemented using the Innovative Integration Inc PEX-7 COP-410 card, hosted in a desktop computer. The card provides ADC sampling at up to 200MHz, which in this application is decimated by a factor of 10 to give 20MHz sampling. Fast PCIe links to the host computer's 64 Gbyte memory provides ample storage for long Prisms. An Agilent 33522B Waveform generator is used to create test signals in real time. The two systems share a reference clock to provide a common frequency calibration, supporting high precision measurement. An application running on the host computer is used to configure, initiate and monitor Prism operation. An early demonstrator with a significantly more limited capability is described in [6].

Figure 4 shows the setup screen from the application. Its primary function is to enable the user to select the central frequency and bandwidth of the desired filter. On clicking the "Design" button, equations (5) and (6) are used to calculate the corresponding *m* and *h* values of the six Prisms. In the figure, the central frequency is arbitrarily selected to be 1,234,567.89 Hz, while the bandwidth chosen is 0.01 Hz. The corresponding values of *h*, *m* and the integral length of each Prism is displayed. For this filter design, using the same sample rate but narrower bandwidth (0.01 Hz vs 0.025 Hz) than for the example in Table 1, the values of *h* and the Prism lengths are correspondingly higher, with a total filter length of around 3.8 Gtaps.

Separate screens control data streaming and provide live monitoring. When streaming is initialised, the Prisms in the FPGA are configured, based on the current parameters values, and real-time processing of the ADC data is commenced. The data stream is passed through the bandpass filter and the tracking Prism to generate estimates of the frequency and amplitude of the original signal from the waveform generator. The current estimate of these parameters are displayed in real time; a graphic display and data logging capabilities are included in the application.

Figure 5 shows the results obtained using the bandpass filter and tracker to monitor step changes in the frequency (Figure 5a) and amplitude (Figure 5b). With the filter bandwidth b = 0.01 Hz, sample rate 20 MHz, and filter length 3.8 Gtaps, the step response period is approximately 190 s. For this example, a waveform generator is used to provide a low amplitude signal (in the range 0.5 mV – 1 mV) which has a frequency within the passband of the filter, and adding high amplitude white noise (standard deviation 180 mV); overall the signal-to-noise ratio is no greater than -54 dB.

Figure 5a shows the filter/tracking system responding to a step change (at t = 200s) in the input frequency, from 1,234,567.890 Hz to 1,234,567.891 Hz i.e. a step change of 1mHz, which is 10% of the bandwidth of the filter; as a fraction of the sample rate this is $5 \times 10^{-11} f_s$. This demonstrates the performance that can be provided using billion tap filtering: a one part per billion step change in signal frequency is clearly tracked in real time where the SNR is -54 dB. Similarly, Figure 5b shows how, in a separate experiment, the demonstration system is able to track a step change in the signal amplitude from 0.5 mV to 1 mV. The step change occurs at approximately t = 200s, and is completed within 190s.

Finally, we return to the convolution comparison to gauge the efficiency of the real-time demonstrator's computational performance. A 3.9 Gtap filter, running at 20 MHz and implemented using convolution, would require 156,000 Terraflops/s, which would earn a place in the top 3 of the June 2020 supercomputer rankings [1]. However, for the Prism-based filter, the computational load on the FPGA remains only 300 flops per sample or 6 GFlops/s in total, representing a reduction over convolution of approximately 25 million.

Next Steps

This paper has explained and demonstrated how, using Prism signal processing, the design and operation of billion tap filters is straightforward. The demonstrator system described here uses an FPGA, but other hardware architectures are equally suited to Prism signal processing, including microcontrollers, conventional processors and dedicated hardware. The implementation of Prism networks seems particularly suited to the architecture of GPUs, which is an area of current development. Another extension is applying Prism bandpass filtering to spectral analysis [7].

While perhaps few applications genuinely need the extremes of billion tap filtering, the same principles can be used to match more modest needs over a wider range of applications – in instrumentation, condition monitoring, or communications - requiring perhaps only a few hundred, or a few thousand, or a few million taps, where the design simplicity and computational efficiency of Prism signal processing may be useful.

One remaining limitation is the large memory requirement: a billion tap filter requires several gigabytes of memory. This issue can be addressed by introducing alternative Prism network structures and some additional techniques. Figure 6 shows the frequency response of a 6 layer Prism 'low-pass' filter Prism network, as described in [8]. Here, the gain for all frequencies over *m* is less than -138 dB. This low pass characteristic can be applied as a bandpass filter with an arbitrary central frequency when used in combination with heterodyning, which shifts the desired central frequency of the original signal into the passband of the low pass filter. This is demonstrated in [8], where tracking of the frequency, phase and amplitude of the original (pre-heterodyned) signal is achieved. A further step, available for the low pass but not the bandpass filter design, is to apply down-sampling (i.e. lowering the effective sample rate), to reduce the effective length of the filter, and therefore the memory requirement (as well as further reducing the computational load). This technique will be fully explained and demonstrated in a future article.

References

[1] R. G. Lyons, “Understanding Digital Signal Processing”, 3rd Edition, Prentice Hall, 2011.

[2] Top 50 Supercomputers https://www.top500.org/lists/top500/list/2020/06/ June 2020. Retrieved July 06, 2020.

[3] MP Henry, F Leach, M Davy, O Bushuev, MS Tombs, FB Zhou, and S Karout, “The Prism: Efficient Signal Processing for the Internet of Things”, IEEE Industrial Electronics Magazine, pp 2–10, December 2017.

[4] MP Henry. “The Prism: recursive FIR signal processing for instrumentation applications”. IEEE Transactions on Instrumentation and Measurement, April 2020.

[5] MP Henry, “Ultra narrowband filtering with Prism signal processing: design and simulation,” IECON, pp. 2748–2753, 2018.

[6] J. Owen, MP Henry, “384 TMAC/s FIR filtering on an Artix-7 FPGA using Prism signal processing,” IECON 2018, vol. 1, pp. 2659–2664, 2018.

[7] MP Henry, “Spectral analysis techniques using Prism signal processing”, Measurement, 169 (2021), 108491.

[8] MP Henry, “Low Cost, Low Pass Prism Filtering”, IEEE International Workshop on Metrology for Industry 4.0 & IoT, 2020.

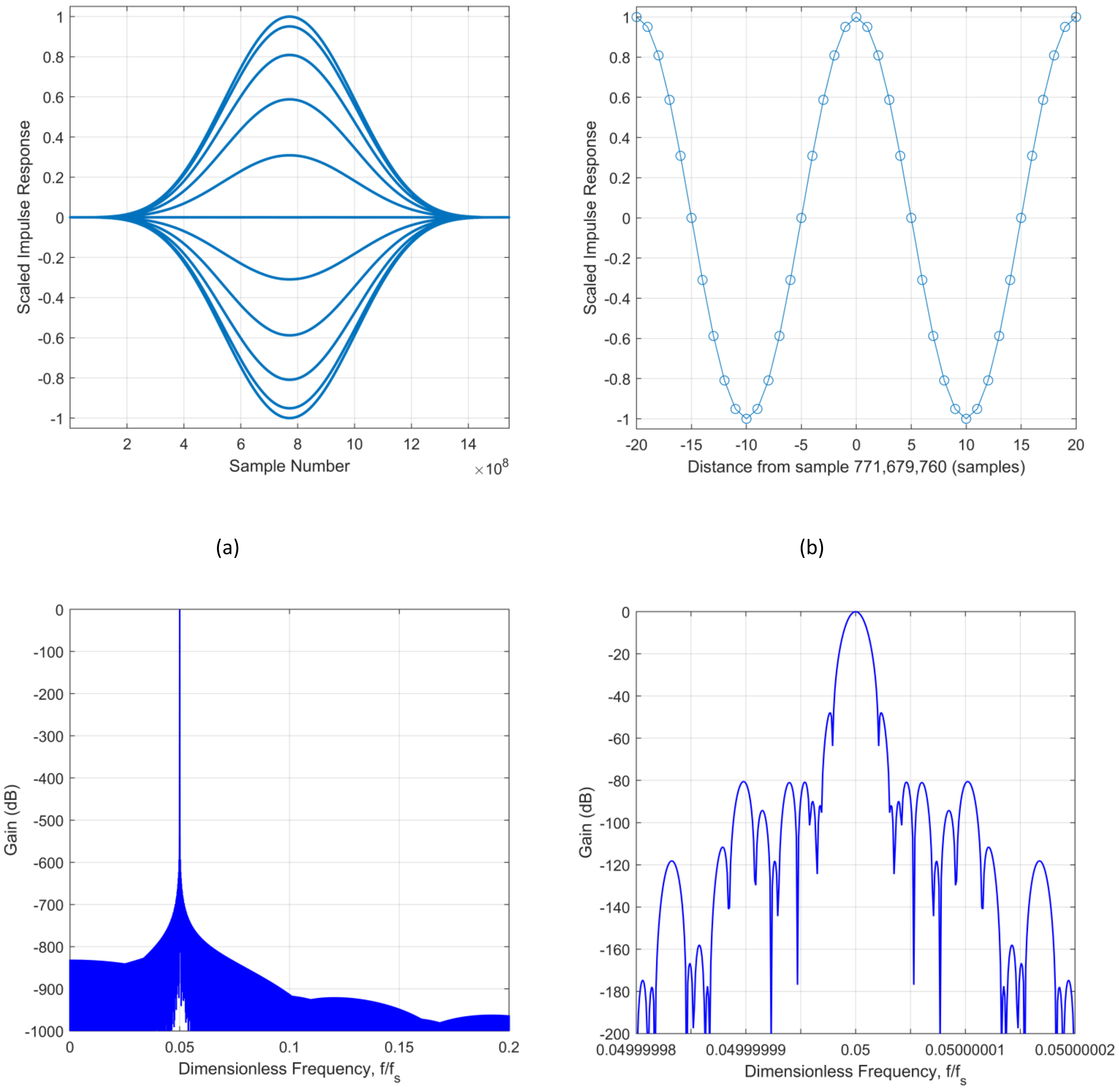


Figure 1. Design of a 1.5 Gtap FIR bandpass filter. (a) Complete impulse response; (b) Central region of impulse response; (c) Frequency response up to 0.2 f_s, where f_s is the sampling frequency; (d) Detail of frequency response around the central frequency of 0.05 f_s .

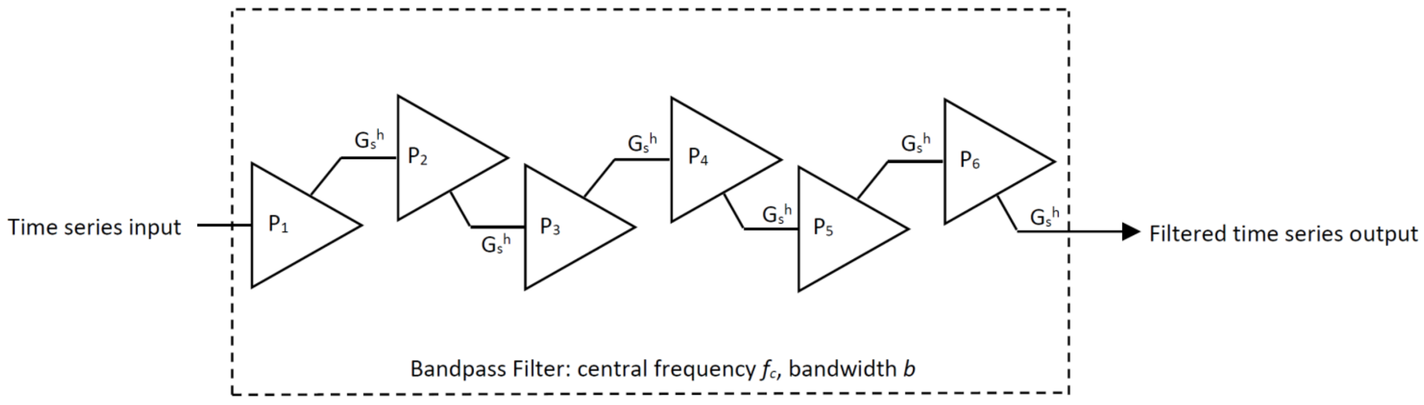

Time series input
P$_1$
P$_2$
P$_3$
P$_4$
P$_5$
P$_6$
G_s^h
Filtered time series output
Bandpass Filter: central frequency f_c, bandwidth b

Figure 2. Structure of bandpass filter, consisting of six single output Prisms in series

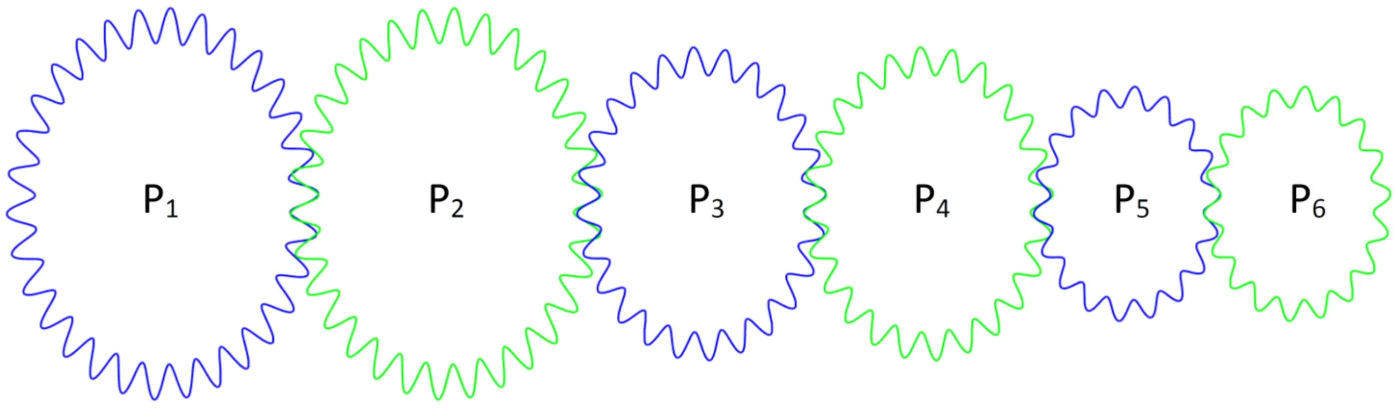
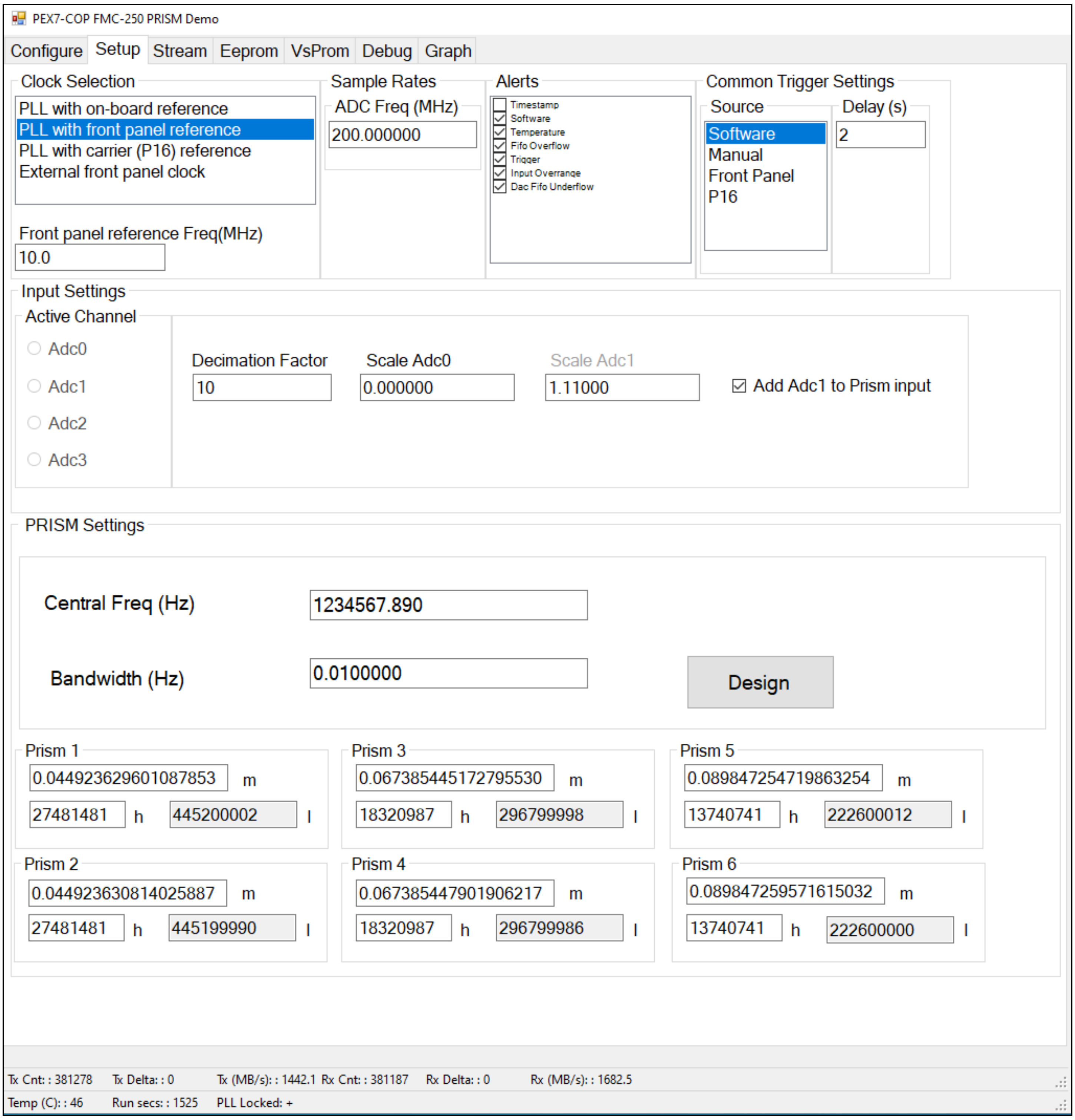


Figure 3. Gear wheel analogy for Prism bandpass filter.


PEX7-COP FMC-250 PRISM Demo

Configure | Setup | Stream | Eeprom | VsProm | Debug | Graph

Clock Selection
- PLL with on-board reference
- [✓ PLL with front panel reference]
- PLL with carrier (P16) reference
- External front panel clock

Front panel reference Freq(MHz): 10.0

Sample Rates
ADC Freq (MHz): 200.000000

Alerts
- ☐ Timestamp
- ☑ Software
- ☑ Temperature
- ☑ Fifo Overflow
- ☑ Trigger
- ☑ Input Overrange
- ☑ Dac Fifo Underflow

Common Trigger Settings
Source: [✓ Software] Manual Front Panel P16
Delay (s): 2

Input Settings
Active Channel: ○ Adc0 ○ Adc1 ○ Adc2 ○ Adc3

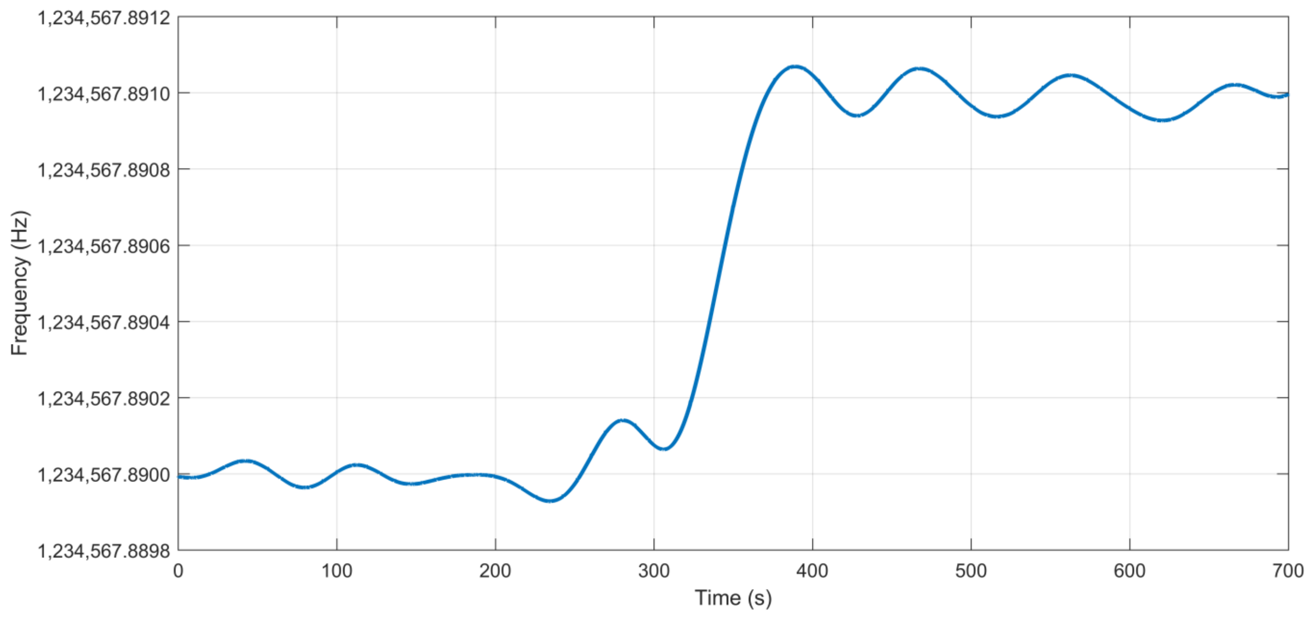
Decimation Factor: 10
Scale Adc0: 0.000000
Scale Adc1: 1.11000
☑ Add Adc1 to Prism input

PRISM Settings
Central Freq (Hz): 1234567.890
Bandwidth (Hz): 0.0100000
Design

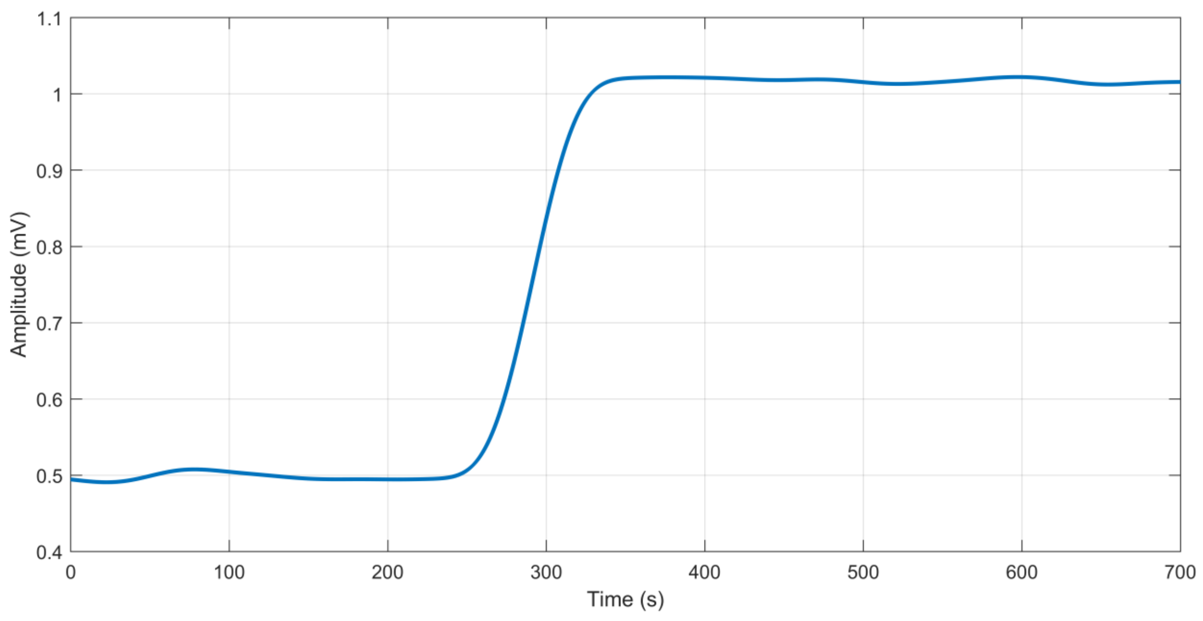
	m	h	l
Prism 1	0.044923629601087853	27481481	445200002
Prism 2	0.044923630814025887	27481481	445199990
Prism 3	0.067385445172795530	18320987	296799998
Prism 4	0.067385447901906217	18320987	296799986
Prism 5	0.089847254719863254	13740741	222600012
Prism 6	0.089847259571615032	13740741	222600000

Tx Cnt: : 381278 Tx Delta: : 0 Tx (MB/s): : 1442.1 Rx Cnt: : 381187 Rx Delta: : 0 Rx (MB/s): : 1682.5
Temp (C): : 46 Run secs: : 1525 PLL Locked: +

Figure 4. Prism setup screen from real-time FPGA demonstrator system. The user enters the desired central frequency and bandwidth, and the corresponding Prism parameters (*m*, *h*, and *l*, the length of each Prism integral) are calculated and displayed. Here, with 20 MHz sampling (200 MHz ADC decimated by a factor of 10) and a bandwidth of 0.01 Hz, the total filter length is approximately 3.8 Gtaps.



(a)



(b)

Figure 5. Results from the FPGA-based real-time bandpass filtering demonstrator, using f_c = 1,234,567.89 Hz and b = 0.01 Hz, tracking a frequency component in 180 mV std noise. (a) tracking results for 1mHz step change in component frequency; (b) tracking results for step change in component amplitude from 0.5mV to 1 mV.

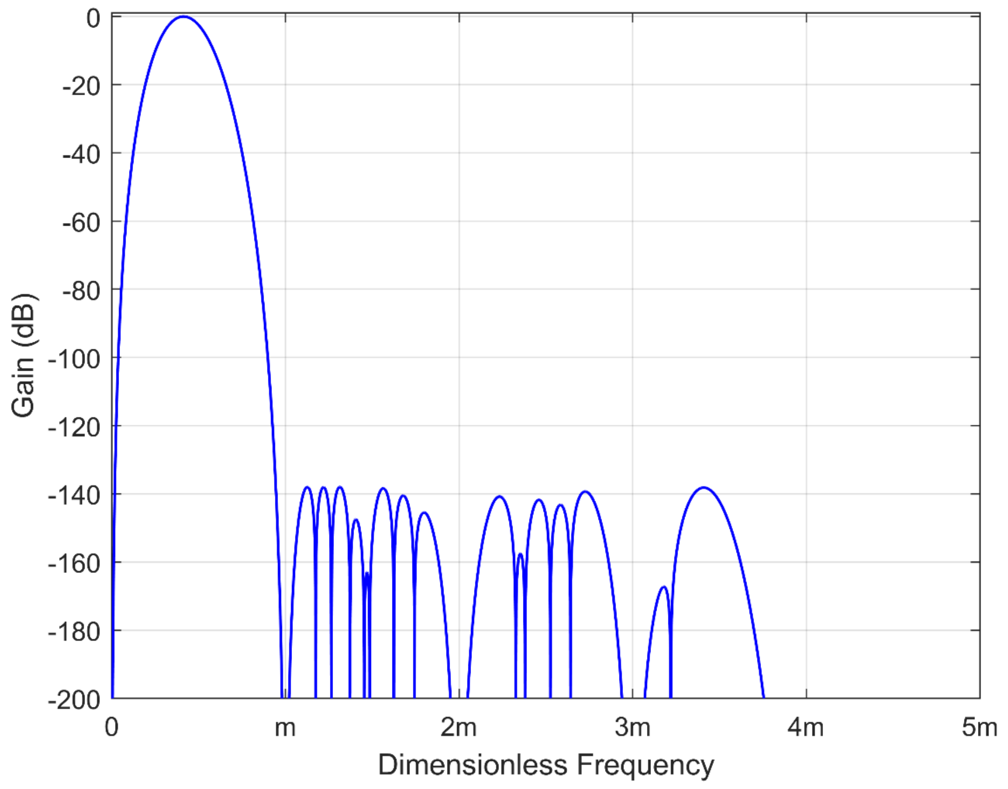


Figure 6: Frequency response of 'low pass' 6 Prism layer filter (from [8]). The design is optimised so that all frequencies above *m* Hz have an attenuation of at least -138 dB.