LASER INTERFEROMETER GRAVITATIONAL WAVE OBSERVATORY
- LIGO -
CALIFORNIA INSTITUTE OF TECHNOLOGY
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Technical Note **LIGO-T050060-00 - D** 4/7/05

IMPLEMENTING AN IIR FILTER IN HARDWARE

Daniel Sigg

Distribution of this draft:

all

This is an internal working note
of the LIGO Project.

LIGO Hanford Observatory
P.O. Box 159
Richland, WA 99352
Phone (509) 372-8106
FAX (509) 372-8137
E-mail: info@ligo.caltech.edu

LIGO Livingston Observatory
19100 LIGO Lane
Livingston, LA 70754
Phone (504) 686-3100
FAX (504) 686-7189
E-mail: info@ligo.caltech.edu

California Institute of Technology
LIGO Project - MS 51-33
Pasadena CA 91125
Phone (626) 395-2129
Fax (626) 304-9834
E-mail: info@ligo.caltech.edu

Massachusetts Institute of Technology
LIGO Project - MS NW17-161
Cambridge, MA 01239
Phone (617) 253-4824
Fax (617) 253-7014
E-mail: info@ligo.mit.edu

Table of Contents

1 INTRODUCTION AND OVERVIEW 3
2 CASCADING SECOND ORDER SECTIONS 3
3 SIMULATION RESULTS 6
3.1 SETUP 6
3.2 STIMULUS 6
3.3 FILTERS 6
3.4 RESULTS 7
3.4.1 Different Stimuli 7
3.4.2 Different Filters 7
3.4.3 Change the Number of Bits 7
3.4.4 Range Check 8
3.4.5 Direct Comparison 8
3.4.6 2kHz Sampling Rate 8
3.4.7 Conclusion 8
4 IMPLEMENTATION DETAILS 9
4.1 SCHEMATICS 9
4.2 STATE CODING 9
APPENDIX A SCHEMATICS AND CODE LISTINGS 11
APPENDIX B SPECTRA OF STIMULI 12
APPENDIX C FILTER TRANSFER FUNCTIONS 13
APPENDIX D SPECTRA OF DIFFERENT STIMULI 14
APPENDIX E SPECTRA OF DIFFERENT FILTERS 15
APPENDIX F EFFECT OF BIT RESOLUTION 16
APPENDIX G 2KHZ SAMPLING RATE 17
APPENDIX H DIRECT COMPARISON 18

1 Introduction and Overview

This document studies the implementation of an IIR filter in hardware. It works out a fixed-point number representation with sufficient accuracy, it demonstrates the filter performance with a set of low-pass elliptic filters and it targets an implementation in an FPGA device.

2 Cascading Second Order Sections

An IIR filter can be written as a product of second order sections[1],

$$H(z) = g \prod_{k=1}^{N_s} \frac{c_{0k}(1 + b_{1k}z^{-1} + b_{1k}z^{-2})}{1 - a_{1k}z^{-1} - a_{2k}z^{-2}}. \tag{1}$$

This representation is less prone to rounding errors than the direct form. In the above equation we separated out an overall gain factor, g, and individual gain factors, c_{0k}. The idea is to implement the c_{0k} multiplication as an efficient shift operation, therefore allowing each second order term to have near unity gain. This is important to maintain accuracy when both filter values are represented as fixed point numbers. A typical decimation filter will use a sum of scaled down input values added to a sum of previous output values to achieve its low pass filtering.

Now restricting ourselves to a single second order section and denoting input values by x_i and output values with y_i the filter section can be written as:

$$y_i = c_0(x_i + b_1 x_{i-1} + b_2 x_{i-2}) + a_1 y_{i-1} + a_2 y_{i-2}. \tag{2}$$

We have chosen the direct form I structure rather than the more traditional direct form II[2]. This allows us to use input and output as history values and, since we set up the filter to be near unity gain, a suitable fixed point representation is straight forward. Since the output value of the first second order section is the input value of the next one, the whole filter can be implemented with a single MAC (multiplier-accumulator) where the very first input value has to be scaled by the gain g (see Figure 1). History output values are shared with the history input values of the next section, therefore only requiring two additional history values compared to the direct form II structure.

A fixed point number representation can be viewed as a 2’s complement signed integer with n bits and an implicit decimal point. We can choose the number of digits for the input value, the output value, the history values, the filter coefficients and the accumulator. A typical hardware multiplier will have equal numbers of bits for the two operands and twice the number of bits in the result. The accumulator should be wide enough to contain intermediate results without overflow but can probably drop a few digits from the end of the multiplier output. Figure 2 shows the representation of the various values. We set the decimal point of the input value after the last bit, thus representing the input value as an integer. Typically, we the sign extend and left shift the

---

1. Here the filter coefficients a_{ik} have the opposite sign than in the convention used by foton.
2. A.V. Oppenheim & R.W. Schafer, “Discrete-Time Signal Processing”.

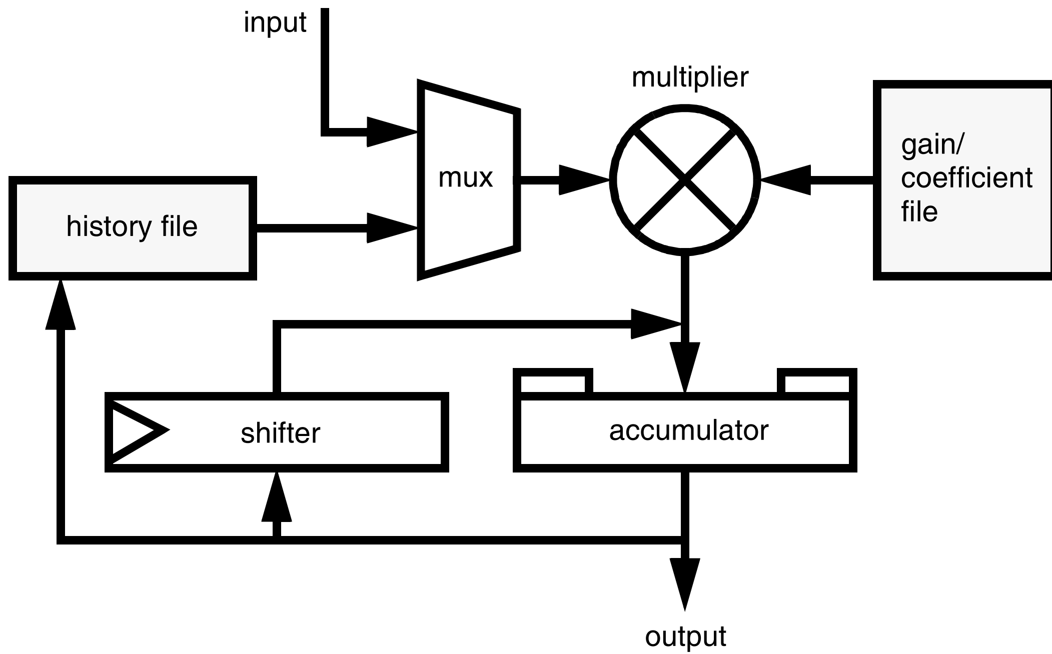


Figure 1: Diagram of a second order filter section that can be cycled multiple times to form higher order filters.

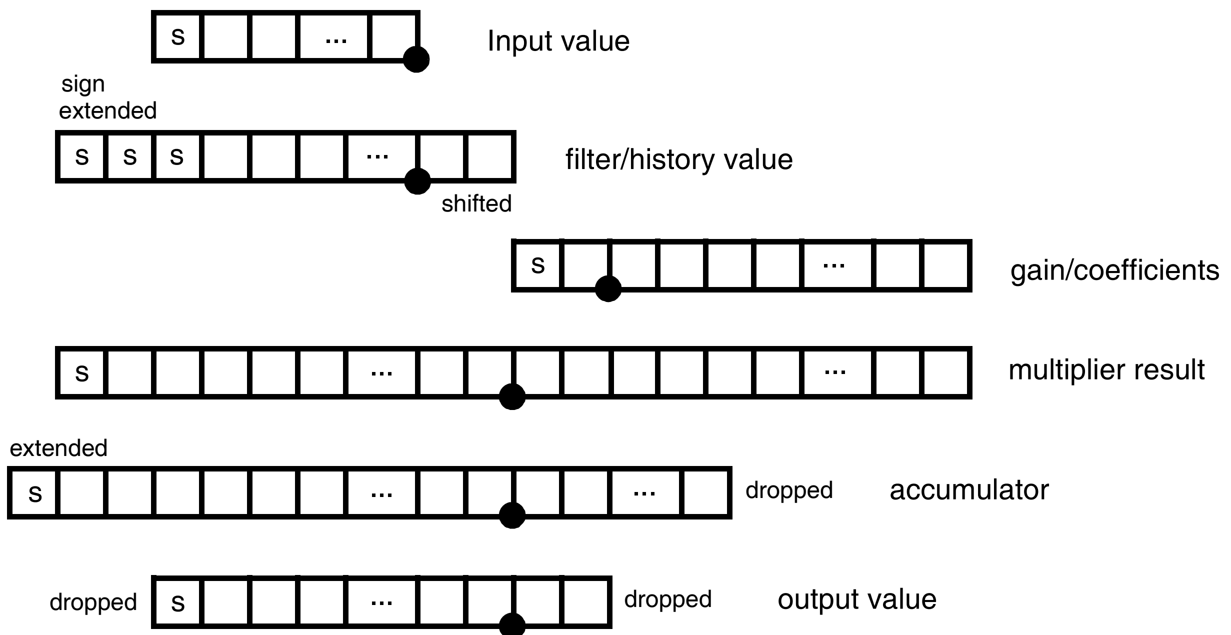


Figure 2: Fixed point number representation.

input value into the filter or history value. The left shift will minimize rounding errors through the filter. The filter coefficients a_2 and b_2 have a range from –1 to +1, whereas the coefficients a_1 and b_1 have a range from –2 to +2. This allows us to set the decimal point of the filter coefficients after the second bit. Since the coefficient c_0 is implemented as a shift operation, we are unable to keep the gain of each second order section at exactly unity. However, we are able to keep the intermediate filter gain always greater or equal than 1 and smaller than 2. This allows us to choose the fixed point representation of the gain value g the same as for the filter coefficients. Since a low pass filter will set $0 < c_0 \leq 1$ and the intermediate filter gain is always between 1 and 2, the accumulator has to add at least 2 additional bits at the front in order not to overflow—assuming the full precision of the filter and history values is used. Strictly speaking, the accumulator can overflow but only after one of the history values has as well. The multiplier produces a lot of digits that won't be significant in the final result. The accumulator hence can drop some at the end. To minimize rounding effects the accumulator should keep at least 3 bits beyond the precision of the filter output. The accumulator output is used to update the history values. Since the accumulator typically has more bits before the decimal point than the filter output value, special logic should be implemented to monitor overflow. The table below shows some typical values based on a device that implements 18 bit multipliers in hardware. Since we need more than 18 bit precision we form 35 bit words that can be handled by the multiplier in 4 passes. All values are signed.

Table 1: Fixed point representation of an example implementation.

Parameter	bits	decimal point	extended at front	added at end
ADC value	18	0	—	—
input value	32	9	5	9
history/filter value	35	12	0	3
filter/gain coefficient	35	33	—	—
multiplier result	70	45	0	0
accumulator	48	20	3	–25
filter/history value	35	12	–5	–8
final output	32	9	0	–3

In the above example the effective number of ADC bits through the filter calculation is 30. This leaves 5 bits (factor of 32) headroom. However, the number of bits reported in the output value is only 27 with a range from -2^{26} to $2^{26}-1$.

3 Simulation results

3.1 Setup

Three types of simulations were performed:

i) An "ideal" simulation using second order sections with double precision floating point numbers. This is the reference implementation.
ii) A behavioral simulation using second order sections with fixed point numbers. This was used to investigate the effect of finite precision and the best allocation of digits.
iii) A register-transfer-level simulation which implements 7 second order section computations in series. This is represents the implementation that targets an FPGA.

The listings and schematics are given in Appendix A.

3.2 Stimulus

The following stimuli were prepared:

i) Sine waves at frequencies of 10 Hz, 100 Hz, 1 kHz, 10 kHz and 100 kHz. To make sure the least significant bit is random high-pass filtered uniform noise was added. The amplitude was set at 1 assuming the ADC has a ±2 range.
ii) A two-tone excitation with sine waves at 10 Hz and 1 kHz. Again, high-pass filtered uniform noise was added. The amplitudes were set to 0.5 each.
iii) Uniform flat noise with notches around 100 Hz and 4 kHz. The rms amplitude was set to be around 0.5.
iv) Uniform noise with notches that was filtered to emphasize low frequencies and therefore better mimic our true signals. The rms amplitude was set to be around 0.5.
v) Two square wave sweeps starting at 160 Hz and 1 kHz and ending at 1.6 kHz and 10 kHz, respectively. The square wave amplitude was set at 1.99.

Spectral plots are given in Appendix F.

3.3 Filters

The following filters were investigated:

i) BUTTER4: A 4th order low-pass/high-pass butterworth filter with a corners at 7400 Hz and 74 kHz, respectively.
ii) ELP4: A 4th order elliptic filter with a design string ellip("LowPass",4,0.1,40,7400)gain(1.01158)
iii) ELP6: A 6th order elliptic filter with a design string ellip("LowPass",6,0.1,60,7400)gain(1.01158)
iv) ELP8: A 8th order elliptic filter with a design string ellip("LowPass",8,0.1,80,7400)gain(1.01158)
v) ELP10: A 10th order elliptic filter with a design string ellip("LowPass",10,0.1,100,7400)gain(1.01158)

vi) ELP12: A 12th order elliptic filter with a design string
ellip("LowPass",12,0.1,120,7400)gain(1.01158)

vii) ELP14: A 14th order elliptic filter with a design string
ellip("LowPass",14,0.1,140,7400)gain(1.01158)

viii) ELP4_6: A combination of a 4th and a 6th order filter with design string:
ellip("LowPass",4,0.1,40,7400)gain(1.01158)ellip("LowPass",6,0.1,60,7400)gain(1.01158)

ix) ELP6_6: A combination of two 6th order filters with design string:
ellip("LowPass",6,0.1,60,7400)gain(1.01158)ellip("LowPass",6,0.1,60,7400)gain(1.01158)

x) ELP6_8: A combination of a 6th and a 8th order filter with design string:
ellip("LowPass",6,0.1,60,7400)gain(1.01158)ellip("LowPass",8,0.1,80,7400)gain(1.01158)

The elliptic filters all had a corner of 7400 Hz. Appendix C shows plots of the filter transfer functions.

A second set of filters were prepared to investigate the lower sampling rate of 2048 Hz. The filters were identical but had a corner frequency of 925 Hz.

3.4 Results

The simulations were done slightly different than indicated in Table 1. There was no shifting the 32 bit value into the 35 bits of the multiplier. Therefore, when Table 1 indicates 27 bits of precision the simulation actually had to use 30 bits. The later number is the one indicated in the plots.

3.4.1 Different Stimuli

The resulting spectra of applying the ELP4 filters on the different stimuli is shown in Appendix B. A resolution of 28 bits was used. The quantization noise is consistently just below $10^{-8}/\sqrt{\mathrm{Hz}}$. Since we use a range of ±2 we can translate this into a voltage noise for a ±10 V ADC by multiplying with 5 V. Hence, the above quantization noise would correspond to about $50\ \mathrm{nV}/\sqrt{\mathrm{Hz}}$.

3.4.2 Different Filters

For this investigation we chose the 1 kHz sine wave excitation and applied all the different filters in turn. Again, a resolution of 28 bits was used. Due to their high Q second order sections we see that the ELP10, ELP12 and ELP12 filters enhance the quantization noise near their corner frequency significantly. They should be replaced by the corresponding ELP4_6, ELP6_6 and ELP6_8 filters, respectively. The resulting spectra are shown in Appendix E.

3.4.3 Change the Number of Bits

We investigated the effect of the number of significant bits carried in the filter values. Appendix F shows the resulting spectra for 22 bits, 24 bits, 26 bits, 28 bits, 30 bits and 32 bits. The plots show that at 30 bits resolution for the filter values and their intermediate history values the quantization noise is dominated by the resolution of the filter coefficients rather than the filter value itself.

The above simulations were done with a range of up to 32 for the overall filter gain value. We repeated the simulation with a reduced range of up to 2 and saw no difference. We also simulated

the effect of dropping the last 25 bits from the output of the multiplier up front, so that we can use a 48 bit accumulator, but saw no effect.

3.4.4 Range Check

We checked to see if the representation of Table 1 has enough headroom by using the square wave sweeps as an excitation and writing down the largest number that occurred either as a history value or as a filter output value. The following table shows that only a small fraction of the actual range is used and that internal saturation is unlikely.

Table 2: Percent of range used.

Excitation	Exc. Amplitude	Filter	Used Range (%)
square wave seep 1 kHz to 10 kHz	1.99	ELP8	7.0
square wave seep 1 kHz to 10 kHz	1.99	ELP10	5.8
square wave seep 1 kHz to 10 kHz	1.99	ELP12	6.3
square wave seep 1 kHz to 10 kHz	1.99	ELP14	7.5
square wave seep 1 kHz to 10 kHz	1.99	ELP4_6	6.3
square wave seep 160 Hz to 1.6 kHz	1.99	ELP8	6.9
1 kHz sine wave	1.00	ELP8	3.0
10 kHz sine wave	1.00	ELP8	2.6

3.4.5 Direct Comparison

Appendix H shows time series of all three types of simulation as well as their difference. From the difference histogram we see that the RTL and behavioral agree within 2×10^{-7} (of the total range) with the ideal filter output, whereas the difference between RTL and behavioral is only $\sim3\times10^{-8}$ of the total range. The above numbers are for a 1 kHz square wave of 1.99 amplitude. With a 1 kHz sine wave of amplitude 1.0 the numbers are about 2–3 times higher.

3.4.6 2 kHz Sampling Rate

The results of different filters with a corner frequency of 925 Hz and using a 1 kHz sine wave excitation are shown in Appendix G. The quantization noise is more than a factor of 10 higher than previously—indicating that the precision of the filter coefficients is no longer adequate. A decimation to a 2 kHz sampling rate will have to be done in two steps. For the first step we can use the output of from the filters with the 7400 Hz corner frequency.

3.4.7 Conclusion

We conclude that an IIR decimation filter implemented with 35 bit fixed point numbers can be implemented in an FPGA device and that it gives sufficient low quantization noise not to limit the performance of a real world ADC. Furthermore, our choice of parameters in Table 1 is sound to implement a decimation filter to go from a 524288 Hz sampling rate to a 16384 Hz sampling rate.

It is however not sufficient to go directly to a 2048 Hz sampling rate. In this case the decimation and filtering has to be done in two steps.

4 IMPLEMENTATION DETAILS

4.1 SCHEMATICS

See Appendix Table A.

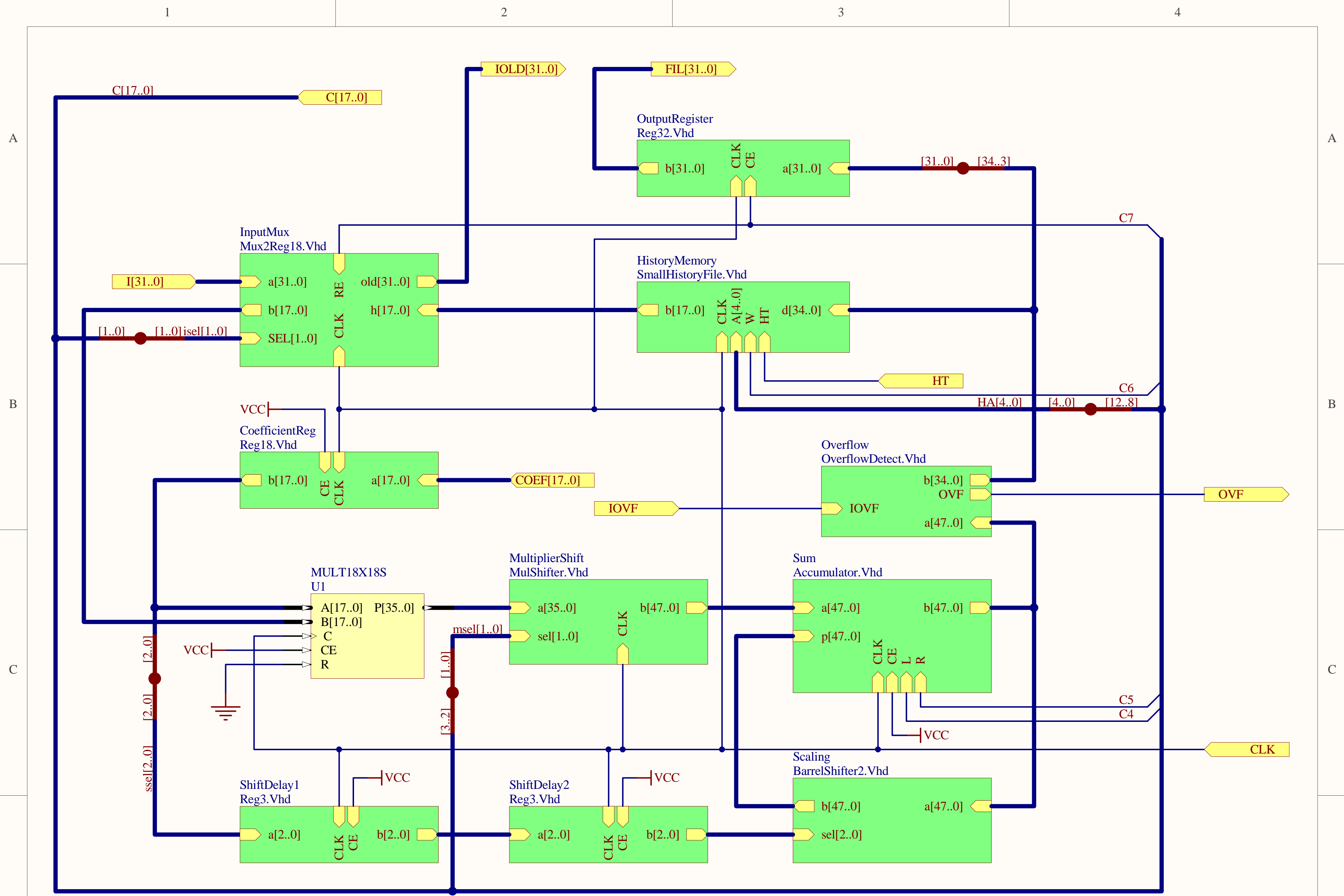
4.2 STATE CODING

The following table shows the different stages of the filter pipeline. Part of the control logic is encoded as microcode together with the filter coefficients. A C++ program to generate the memory initialization strings is shown in Appendix A as well.

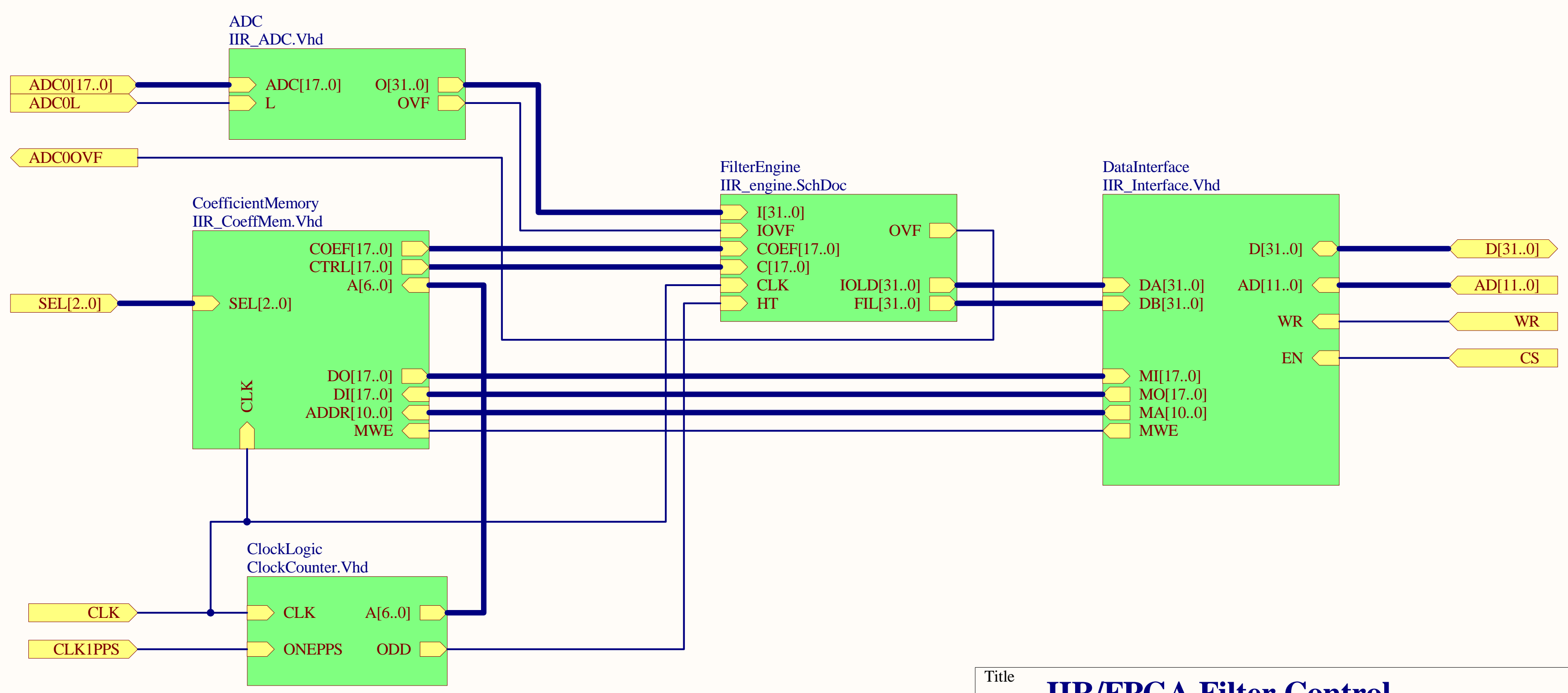
Table 3: Filter Pipeline: l=LSB, m=MSB, $y_{ik}=x_{ik+1}$.

	HistMem	InpMux	Coeff	CoeffReg	Mul	MulShift	Accu
0	–	–	g,l	–	–	–	–
1	–	x_{00},l	g,m	g,l	–	–	–
2	–	x_{00},l	g,l	g,m	gx_{00},ll	–	–
3	–	x_{00},m	g,m	g,l	gx_{00},lm	gx_{00},ll	0
4	x_{-20},l	x_{00},m	b_{20},l	g,m	gx_{00},ml	gx_{00},lm	$+gx_{00}$,ll
5	x_{-20},l	x_{-20},l	b_{20},m	b_{20},l	gx_{00},mm	gx_{00},ml	$+gx_{00}$,lm
6	x_{-20},m	x_{-20},l	b_{20},l	b_{20},m	$b_{20}x_{-20}$,ll	gx_{00},mm	$+gx_{00}$,ml
7	x_{-20},m	x_{-20},m	b_{20},m	b_{20},l	$b_{20}x_{-20}$,lm	$b_{20}x_{-20}$,ll	gx_{00}
8	x_{-10},l	x_{-20},m	b_{10},l	b_{20},m	$b_{20}x_{-20}$,ml	$b_{20}x_{-20}$,lm	$+b_{20}x_{00}$,ll
9	x_{-10},l	x_{-10},l	b_{10},m	b_{10},l	$b_{20}x_{-20}$,mm	$b_{20}x_{-20}$,ml	$+b_{20}x_{00}$,lm
10	x_{-10},m	x_{-10},l	b_{10},l	b_{10},m	$b_{10}x_{-10}$,ll	$b_{20}x_{-20}$,mm	$+b_{20}x_{00}$,ml
11	x_{-10},m	x_{-10},m	b_{10},m	b_{10},l	$b_{10}x_{-10}$,lm	$b_{10}x_{-10}$,ll	$gx_{00}+b_{20}x_{-20}$
12	–	x_{-10},m	-lb(c_{00})	b_{10},m	$b_{10}x_{-10}$,ml	$b_{10}x_{-10}$,lm	$+b_{10}x_{-10}$,ll
13	y_{-20},l	–	a_{20},l	-lb(c_{00})	$b_{10}x_{-10}$,mm	$b_{10}x_{-10}$,ml	$+b_{10}x_{-10}$,lm
14	y_{-20},l	y_{-20},l	a_{20},m	a_{20},l	–	$b_{10}x_{-10}$,mm	$+b_{10}x_{-10}$,ml
15	y_{-20},m	y_{-20},l	a_{20},l	a_{20},m	$a_{20}y_{-20}$,ll	–	$gx_{00}+b_{20}x_{-20}+b_{10}x_{-10}$
16	y_{-20},m	y_{-20},m	a_{20},m	a_{20},l	$a_{20}y_{-20}$,lm	$a_{20}y_{-20}$,ll	$c_{00}\Sigma x$
17	y_{-10},l	y_{-20},m	a_{10},l	a_{20},m	$a_{20}y_{-20}$,ml	$a_{20}y_{-20}$,lm	$+a_{20}y_{-20}$,ll
18	y_{-10},l	y_{-10},l	a_{10},m	a_{10},l	$a_{20}y_{-20}$,mm	$a_{20}y_{-20}$,ml	$+a_{20}y_{-20}$,lm
19	y_{-10},m	y_{-10},l	a_{10},l	a_{10},m	$a_{10}y_{-10}$,ll	$a_{20}y_{-20}$,mm	$+a_{20}y_{-20}$,ml
20	y_{-10},m	y_{-10},m	a_{10},m	a_{10},l	$a_{10}y_{-10}$,lm	$a_{10}y_{-10}$,ll	$c_{00}\Sigma x+a_{20}y_{-20}$
21	↓	y_{-10},m	↓	a_{10},m	$a_{10}y_{-10}$,ml	$a_{10}y_{-10}$,lm	$+a_{10}y_{-10}$,ll
22		↓		↓	$a_{10}y_{-10}$,mm	$a_{10}y_{-10}$,ml	$+a_{10}y_{-10}$,lm
23					↓	$a_{10}y_{-10}$,mm	$+a_{10}y_{-10}$,ml
24						↓	$c_{00}\Sigma x+\Sigma y = y_{00} = x_{01}$
							↓
118	x_{-26},m		b_{26},m				
119	x_{-16},l	x_{-26},m	b_{16},l	b_{26},m			
120	x_{-16},m	x_{-16},l	b_{16},l	b_{16},l	$b_{26}x_{-26}$,mm		
121	x_{-16},l	x_{-16},m	b_{16},m	b_{16},l	$b_{16}x_{-16}$,ll	$b_{26}x_{-26}$,mm	
122	x_{-16},m	x_{-16},l	b_{16},m	b_{16},m	$b_{16}x_{-16}$,lm	$b_{16}x_{-16}$,ll	$c_{06}\Sigma x+a_{26}y_{-26}$
123	–	x_{-16},m	–	b_{16},m	$b_{16}x_{-16}$,ml	$b_{16}x_{-16}$,lm	$+a_{16}y_{-16}$,ll
124	–	–	–	–	$b_{16}x_{-16}$,mm	$b_{16}x_{-16}$,ml	$+a_{16}y_{-16}$,lm
125	–	–	–	–	–	$b_{16}x_{-16}$,mm	$+a_{16}y_{-16}$,ml
126	–	–	–	–	–	–	$c_{06}\Sigma x+\Sigma y = y_{06}$
127	x_{-16},m	–	–	–	–	–	$c_{06}\Sigma x+\Sigma y = y_{06}$

APPENDIX A SCHEMATICS AND CODE LISTINGS



Title	IIR/FPGA Filter Engine	
Size: A	Number: D050000-00	Revision: 0
Date: 4/17/2005		Sheet 1 of 2
File: C:\User\..\IIR_engine.SchDoc		Drawn By: Daniel Sigg



Title	**IIR/FPGA Filter Control**		
Size A	Number D050000-00		Revision 0
Date:	4/17/2005	Sheet 1 of 2	
File:	C:\User\..\IIR_top.SchDoc	Drawn By:	Daniel Sigg

```vhdl
--------------------------------------------------------------------------------
-- SubModule IIR_ADC
-- Created   4/12/2005 5:51:13 PM
--------------------------------------------------------------------------------
Library IEEE;
Use IEEE.Std_Logic_1164.all;

entity IIR_ADC is port
   (
     ADC        : in    std_logic_vector(17 downto 0);  -- AD7679
     L          : in    std_logic;  -- busy signal of ADC
     O          : out   std_logic_vector(31 downto 0);  -- sign extended ADC value
     OVF        : out   std_logic    -- overflow
   );
end IIR_ADC;
--------------------------------------------------------------------------------

--------------------------------------------------------------------------------
architecture Structure of IIR_ADC is

   signal ext : std_logic_vector(31 downto 0);
   signal reg : std_logic_vector(31 downto 0) := (others => '0');

begin
   -- sign extend and shift ADC value into 32 bit word
   ext(8 downto 0) <= (others => '0');
   ext(26 downto 9) <= ADC;
   ext(31 downto 27) <= (others => ADC(17));
   -- load new ADC value into register
   reg <= ext when falling_edge(L);
   O <= reg;

   -- check overflow on registered input
   overflow: process (reg) is
   begin
      if (reg(26 downto 15) = B"011111111111") or
         (reg(26 downto 15) = B"100000000000") then
         OVF <= '1';
      else
         OVF <= '0';
      end if;
   end process overflow;

end Structure;
--------------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------------
-- SubModule ClockCounter
-- Created   4/12/2005 7:06:29 PM
--------------------------------------------------------------------------------
library IEEE;
use IEEE.Std_Logic_1164.all;
use IEEE.Numeric_Std.all;

entity ClockCounter is port
   (
     CLK      : in    std_logic;
     ONEPPS   : in    std_logic;
     A        : out   std_logic_vector(6 downto 0);
     ODD      : out   std_logic
   );
end ClockCounter;
--------------------------------------------------------------------------------

--------------------------------------------------------------------------------
architecture RTL of ClockCounter is

-- Signal Declarations
   signal counter : std_logic_vector(7 downto 0) := B"11110110";
   signal short1pps : std_logic := '0';
   signal load : std_logic := '0';

begin
   -- make sure 1pps is shorter than a CLK period
   short_onepps: process (ONEPPS, load) is
   begin
      if load = '1' then
         short1pps <= '0';
      elsif rising_edge(onepps) then
         short1pps <= '1';
      end if;
   end process short_onepps;

   -- load counter with 1 on next CLK after 1PPS
   load <= short1pps when falling_edge(CLK);

   -- binary counter with load
   count: process (CLK) is
   begin
      if rising_edge(CLK) then
         if load = '1' then
            counter <= B"00000001";
         else
            counter <= std_logic_vector(unsigned(counter) + 1);
         end if;
      end if;
   end process count;

   -- set output values
   A <= counter(6 downto 0);
   ODD <= counter(7);

end architecture RTL;
--------------------------------------------------------------------------------
```

```vhdl
--------------------------------------------------------------------------------
-- SubModule IIR_Interface
-- Created   4/12/2005 6:49:14 PM
--------------------------------------------------------------------------------
library IEEE;
use IEEE.Std_Logic_1164.all;

entity IIR_Interface is port
   (
     AD         : in    std_logic_vector(11 downto 0);
     D          : inout std_logic_vector(31 downto 0);
     WR         : in    std_logic;
     EN         : in    std_logic;
     MI         : in    std_logic_vector(17 downto 0);
     MO         : out   std_logic_vector(17 downto 0);
     MA         : out   std_logic_vector(10 downto 0);
     MWE        : out   std_logic;
     DA         : in    std_logic_vector(31 downto 0);
     DB         : in    std_logic_vector(31 downto 0)
   );
end IIR_Interface;
--------------------------------------------------------------------------------

--------------------------------------------------------------------------------
architecture RTL of IIR_Interface is

-- Signal Declarations
   signal muxsel : std_logic_vector(0 downto 0);
   signal muxout : std_logic_vector(31 downto 0);
   signal memout : std_logic_vector(31 downto 0);
   signal dout : std_logic_vector(31 downto 0);

begin
   -- mux address
   muxsel(0) <= AD(0);
   -- memory address out
   MA <= AD(10 downto 0);
   -- ADC mux
   datamux: process (DA, DB, MI, muxsel) is
   begin
      case muxsel is
         when B"0" =>
            muxout <= DA;
         when B"1" =>
            muxout <= DB;
         when others =>
            muxout <= DA;
      end case;
   end process datamux;
   -- extend memory output to 32 bits
   memout(17 downto 0) <= MI;
   memout(31 downto 18) <= (others => '0');
   -- ADC/memory mux
   dout <= muxout when AD(11) = '0'
           else memout;
   -- data out
   D <= dout when (EN = '1') and (WR = '0')
        else (others => 'Z');
   -- data in
   MO <= To_X01 (D(17 downto 0));
   -- write to memory
   MWE <= '1' when (EN = '1') and (WR = '1') and (AD(11) = '1')
          else '0';
end architecture RTL;
--------------------------------------------------------------------------------
```

```vhdl
------------------------------------------------------------
-- VHDL IIR_CoeffMem
-- 2005 4 14 17 19 17
-- Created By "DXP VHDL Generator"
-- "Copyright (c) 2002-2004 Altium Limited"
------------------------------------------------------------

------------------------------------------------------------
-- VHDL IIR_CoeffMem
------------------------------------------------------------

Library IEEE;
Use     IEEE.std_logic_1164.all;
--synopsys translate_off
library UNISIM;
use unisim.vcomponents.all;
--synopsys translate_on

entity IIR_CoeffMem is
  port
  (
    A    : In    STD_LOGIC_VECTOR(6 downto 0);
    ADDR : In    STD_LOGIC_VECTOR(10 downto 0);
    CLK  : In    STD_LOGIC;
    COEF : Out   STD_LOGIC_VECTOR(17 downto 0);
    CTRL : Out   STD_LOGIC_VECTOR(17 downto 0);
    DI   : In    STD_LOGIC_VECTOR(17 downto 0);
    DO   : Out   STD_LOGIC_VECTOR(17 downto 0);
    MWE  : In    STD_LOGIC;
    SEL  : In    STD_LOGIC_VECTOR(2 downto 0)
  );

end IIR_CoeffMem;
------------------------------------------------------------

------------------------------------------------------------
architecture RTL of IIR_CoeffMem is

   component RAMB16_S18_S36
      -- pragma translate_off
      generic
      (
        -- "Read during Write" attribute for functional simulation
        WRITE_MODE_A : string := "WRITE_FIRST" ; -- WRITE_FIRST(default)/READ_FIRST/ NO_CHANGE
        -- "Read during Write" attribute for functional simulation
        WRITE_MODE_B : string := "WRITE_FIRST" ; -- WRITE_FIRST(default)/READ_FIRST/ NO_CHANGE
        -- Output value after configuration
        INIT_A : bit_vector(17 downto 0) := (others => '0');
        -- Output value after configuration
        INIT_B : bit_vector(35 downto 0) := (others => '0');
        -- Output value if SSR active
        SRVAL_A : bit_vector(17 downto 0) := (others => '0');
        -- Output value if SSR active
        SRVAL_B : bit_vector(35 downto 0) := (others => '0');
        -- Plus bits initial content
        INIT_00 : bit_vector(255 downto 0) :=
           X"001963d10119000044080000442900004018000040150641008593ab00010641";
        INIT_01 : bit_vector(255 downto 0) :=
           X"081b15064c492ab74c0b150648292ab748180002001a0512040963d1042a0512";
        INIT_02 : bit_vector(255 downto 0) :=
           X"091900004c0800004c29000048180000481 9ea4b0c093d2e0c29ea4b08193d2e";
        INIT_03 : bit_vector(255 downto 0) :=
           X"5449537b540b0c6b5029537b50180003001a08740c09fb2c0c2a08740819fb2c";
        INIT_04 : bit_vector(255 downto 0) :=
           X"5408000054290000501800005019f2081409e2081429f2081019e208101b0c6b";
        INIT_05 : bit_vector(255 downto 0) :=
           X"5c0b03ba5829105258180005001a35b11409b045142a35b11019b04511190000";
        INIT_06 : bit_vector(255 downto 0) :=
           X"5c28000058180000 5819fa121c099ce91c29fa1218199ce9181b03ba5c491052";
        INIT_07 : bit_vector(255 downto 0) :=
           X"602800006018000000180000 1c0800001c280000181800001918 00005c080000";
        INIT_08 : bit_vector(255 downto 0) :=
           X"6018000060180000240800002428000020180000201800006448000064080000";
        INIT_09 : bit_vector(255 downto 0) :=
```

```
            X"6818000000180000240800002428000020180000211800006408000064280000";
        INIT_0a : bit_vector(255 downto 0) :=
            X"681800002c0800002c280000281800002818000006c4800006c0800006828000"0;
```

```
            X"3c280000381800003818000007c4800007c0800007828000078180000000180000";
         INIT_2f : bit_vector(255 downto 0) :=
            X"00013f21380000003b0000007c0800007c28000078180000781800003c080000";
         INIT_30 : bit_vector(255 downto 0) :=
            X"00185fd40119000044080000442900004018000040152368008579ce00012368";
         INIT_31 : bit_vector(255 downto 0) :=
            X"081b0bd74c49372d4c0b0bd74829372d48180004001a035904085fd4042a0359";
         INIT_32 : bit_vector(255 downto 0) :=
            X"091900004c0800004c29000048180000481 9f3f50c0962a70c29f3f5081962a7";
         INIT_33 : bit_vector(255 downto 0) :=
            X"54494624540b09fa502946245018000 2001a03a90c08f51a0c2a03a90818f51a";
         INIT_34 : bit_vector(255 downto 0) :=
            X"540800005429000050180000501 9f57614081f991429f57610181f99101b09fa";
         INIT_35 : bit_vector(255 downto 0) :=
            X"5c0b073b5828e16a58180003001a04851409f340142a04851019f34011190000";
         INIT_36 : bit_vector(255 downto 0) :=
            X"5c29000058180000581 9f7af1c0916811c29f7af18191681181b073b5c48e16a";
         INIT_37 : bit_vector(255 downto 0) :=
            X"6029b70c60180002001a06d81c0874641c2a06d818187464191900005c080000";
         INIT_38 : bit_vector(255 downto 0) :=
            X"601800006019f9da240943462429f9da20194346201b04966449b70c640b0496";
         INIT_39 : bit_vector(255 downto 0) :=
            X"68180003001a0f79240955e3242a0f79201955e32119000064080000642 90000";
         INIT_3a : bit_vector(255 downto 0) :=
            X"6819fba82c0966f62c29fba8281966f6281b02756c4835f46c0b0275682835f4";
         INIT_3b : bit_vector(255 downto 0) :=
            X"001a70d82c080a292c2a70d828180a29291900006c0800006c29000068180000";
         INIT_3c : bit_vector(255 downto 0) :=
            X"3408500b3429fd303018500b301b00c4744816a6740b00c4702816a670180006";
         INIT_3d : bit_vector(255 downto 0) :=
            X"3408000034280000301800003118000074080000742800007018000007019fd30";
         INIT_3e : bit_vector(255 downto 0) :=
            X"3c280000381800003818000007c4800007c0800007828000078180000000180000";
         INIT_3f : bit_vector(255 downto 0) :=
            X"000179ce380000003b0000007c0800007c28000078180000781800003c080000"
      );
      -- pragma translate_on
      port
      (
        ADDRA : in  STD_LOGIC_VECTOR(9 downto 0);
        ADDRB : in  STD_LOGIC_VECTOR(8 downto 0);
        CLKA  : in  STD_LOGIC;
        CLKB  : in  STD_LOGIC;
        DIA   : in  STD_LOGIC_VECTOR(15 downto 0);
        DIB   : in  STD_LOGIC_VECTOR(31 downto 0);
        DIPA  : in  STD_LOGIC_VECTOR(1 downto 0);
        DIPB  : in  STD_LOGIC_VECTOR(3 downto 0);
        DOA   : out STD_LOGIC_VECTOR(15 downto 0);
        DOB   : out STD_LOGIC_VECTOR(31 downto 0);
        DOPA  : out STD_LOGIC_VECTOR(1 downto 0);
        DOPB  : out STD_LOGIC_VECTOR(3 downto 0);
        ENA   : in  STD_LOGIC;
        ENB   : in  STD_LOGIC;
        SSRA  : in  STD_LOGIC;
        SSRB  : in  STD_LOGIC;
        WEA   : in  STD_LOGIC;
        WEB   : in  STD_LOGIC
      );
   end component RAMB16_S18_S36;

   attribute INIT_00 : string;
   attribute INIT_00 of U_RAM : label is
      "001963d10119000044080000442900004018000040150641008593ab00010641";
   attribute INIT_01 : string;
   attribute INIT_01 of U_RAM : label is
      "081b15064c492ab74c0b150648292ab748180002001a0512040963d1042a0512";
   attribute INIT_02 : string;
   attribute INIT_02 of U_RAM : label is
      "091900004c0800004c290000481800004819ea4b0c093d2e0c29ea4b08193d2e";
   attribute INIT_03 : string;
   attribute INIT_03 of U_RAM : label is
      "5449537b540b0c6b5029537b50180003001a08740c09fb2c0c2a08740819fb2c";
   attribute INIT_04 : string;
   attribute INIT_04 of U_RAM : label is
```

```
        "540800005429000050180000501 9f2081409e2081429f2081019e208101b0c6b";
    attribute INIT_05 : string;
    attribute INIT_05 of U_RAM : label is
        "5c0b03ba58291052581800050 01a35b11409b045142a35b11019b04511190000";
    attribute INIT_06 : string;
    attribute INIT_06 of U_RAM : label is
        "5c2800005818000058 19fa121c099ce91c29fa1218199ce9181b03ba5c491052";
    attribute INIT_07 : string;
    attribute INIT_07 of U_RAM : label is
        "602800006018000000180000 1c0800001c2800001818000019180000 5c080000";
    attribute INIT_08 : string;
    attribute INIT_08 of U_RAM : label is
        "60180000601800002408000024280000201800002018000064480000 64080000";
    attribute INIT_09 : string;
    attribute INIT_09 of U_RAM : label is
        "68180000001800002408000024280000201800002118000064080000 64280000";
    attribute INIT_0a : string;
    attribute INIT_0a of U_RAM : label is
        "681800002c0800002c28000028180000281800006c4800006c080000 68280000";
    attribute INIT_0b : string;
    attribute INIT_0b of U_RAM : label is
        "001800002c0800002c28000028180000291800006c0800006c280000 68180000";
    attribute INIT_0c : string;
    attribute INIT_0c of U_RAM : label is
        "340800003428000030180000301800007448000074080000702800007 0180000";
    attribute INIT_0d : string;
    attribute INIT_0d of U_RAM : label is
        "340800003428000030180000311800007408000074280000701800007 0180000";
    attribute INIT_0e : string;
    attribute INIT_0e of U_RAM : label is
        "3c280000381800003818000 07c4800007c08000078280000781800000 0180000";
    attribute INIT_0f : string;
    attribute INIT_0f of U_RAM : label is
        "000193ab380000003b0000007c0800007c28000078180000781800003c080000";
    attribute INIT_10 : string;
    attribute INIT_10 of U_RAM : label is
        "0019c569011900004408000044290000401800004015ad510084f9af0001ad51";
    attribute INIT_11 : string;
    attribute INIT_11 of U_RAM : label is
        "081b10a74c49d67f4c0b10a74829d67f48180004001a040f0409c569042a040f";
    attribute INIT_12 : string;
    attribute INIT_12 of U_RAM : label is
        "091900004c0800004c290000481800004819eeef0c0997c80c29eeef081997c8";
    attribute INIT_13 : string;
    attribute INIT_13 of U_RAM : label is
        "5449a6df540b0bf45029a6df50180002001a052d0c09be250c2a052d0819be25";
    attribute INIT_14 : string;
    attribute INIT_14 of U_RAM : label is
        "5408000054290000501800005019f30114085202 1429f30110185202101b0bf4";
    attribute INIT_15 : string;
    attribute INIT_15 of U_RAM : label is
        "5c0b06625828af8858180002001a0a24140947b3142a0a24101947b311190000";
    attribute INIT_16 : string;
    attribute INIT_16 of U_RAM : label is
        "5c2900005818000058 19f7e41c0917791c29f7e418191779181b06625c48af88";
    attribute INIT_17 : string;
    attribute INIT_17 of U_RAM : label is
        "60293d0760180006001a46351c09b8991c2a46351819b89919190000 5c080000";
    attribute INIT_18 : string;
    attribute INIT_18 of U_RAM : label is
        "6018000060 19fbf82408415f2429fbf82018415f201b01ed64493d07640b01ed";
    attribute INIT_19 : string;
    attribute INIT_19 of U_RAM : label is
        "68180000001800002408000024280000201800002118000064080000 64280000";
    attribute INIT_1a : string;
    attribute INIT_1a of U_RAM : label is
        "681800002c0800002c28000028180000281800006c4800006c080000 68280000";
    attribute INIT_1b : string;
    attribute INIT_1b of U_RAM : label is
        "001800002c0800002c28000028180000291800006c0800006c280000 68180000";
    attribute INIT_1c : string;
    attribute INIT_1c of U_RAM : label is
        "340800003428000030180000301800007448000074080000702800007 0180000";
    attribute INIT_1d : string;
```

```
    attribute INIT_1d of U_RAM : label is
       "3408000034280000301800003118000074080000742800007018000070180000";
    attribute INIT_1e : string;
    attribute INIT_1e of U_RAM : label is
       "3c28000038180000381800007c4800007c080000782800007818000000180000";
    attribute INIT_1f : string;
    attribute INIT_1f of U_RAM : label is
       "0000f9af380000003b0000007c0800007c280000781800007818000003c080000";
    attribute INIT_20 : string;
    attribute INIT_20 of U_RAM : label is
       "001965fb011900004408000044290000401800004015 60ea00853f21000160ea";
    attribute INIT_21 : string;
    attribute INIT_21 of U_RAM : label is
       "081b0dd44c493ed04c0b0dd448293ed048180004001a039a040965fb042a039a";
    attribute INIT_22 : string;
    attribute INIT_22 of U_RAM : label is
       "091900004c0800004c29000048180000 4819f1e50c0860340c29f1e508186034";
    attribute INIT_23 : string;
    attribute INIT_23 of U_RAM : label is
       "54498ad2540b0af950298ad250180002001a04240c09694d0c2a04240819694d";
    attribute INIT_24 : string;
    attribute INIT_24 of U_RAM : label is
       "5408000054290000501800005019f4471408db071429f4471018db07101b0af9";
    attribute INIT_25 : string;
    attribute INIT_25 of U_RAM : label is
       "5c0b072158282bf458180002001a05de14092945142a05de1019294511190000";
    attribute INIT_26 : string;
    attribute INIT_26 of U_RAM : label is
       "5c290000581800005819f7831c09c7a61c29f7831819c7a6181b07215c482bf4";
    attribute INIT_27 : string;
    attribute INIT_27 of U_RAM : label is
       "6029d7266018000 3001a0c8c1c088d0e1c2a0c8c18188d0e191900005c080000";
    attribute INIT_28 : string;
    attribute INIT_28 of U_RAM : label is
       "601800006019fa662409c0f62429fa662019c0f6201b03c56449d726640b03c5";
    attribute INIT_29 : string;
    attribute INIT_29 of U_RAM : label is
       "6818000 6001a5a392409be57242a5a392019be572119000064080000 64290000";
    attribute INIT_2a : string;
    attribute INIT_2a of U_RAM : label is
       "6819fcc62c09cac22c29fcc62819cac2281b01286c491e766c0b012868291e76";
    attribute INIT_2b : string;
    attribute INIT_2b of U_RAM : label is
       "001800002c0800002c28000028180000291800006c0800006c28000068180000";
    attribute INIT_2c : string;
    attribute INIT_2c of U_RAM : label is
       "3408000034280000301800003018000074480000740800007028000070180000";
    attribute INIT_2d : string;
    attribute INIT_2d of U_RAM : label is
       "3408000034280000301800003118000074080000742800007018000070180000";
    attribute INIT_2e : string;
    attribute INIT_2e of U_RAM : label is
       "3c28000038180000381800007c4800007c080000782800007818000000180000";
    attribute INIT_2f : string;
    attribute INIT_2f of U_RAM : label is
       "00013f21380000003b0000007c0800007c2800007818000078180000 3c080000";
    attribute INIT_30 : string;
    attribute INIT_30 of U_RAM : label is
       "00185fd401190000440800004429000040180000401523680 08579ce00012368";
    attribute INIT_31 : string;
    attribute INIT_31 of U_RAM : label is
       "081b0bd74c49372d4c0b0bd74829372d48180004001a035904085fd4042a0359";
    attribute INIT_32 : string;
    attribute INIT_32 of U_RAM : label is
       "091900004c0800004c290000481800004819f3f50c0962a70c29f3f5081962a7";
    attribute INIT_33 : string;
    attribute INIT_33 of U_RAM : label is
       "54494624540b09fa5029462450180002001a03a90c08f51a0c2a03a90818f51a";
    attribute INIT_34 : string;
    attribute INIT_34 of U_RAM : label is
       "5408000054290000501800005019f57614081f991429f57610181f99101b09fa";
    attribute INIT_35 : string;
    attribute INIT_35 of U_RAM : label is
       "5c0b073b5828e16a58180003001a04851409f340142a04851019f34011190000";
```

```
    attribute INIT_36 : string;
    attribute INIT_36 of U_RAM : label is
       "5c29000058180000 5819f7af1c0916811c29f7af18191681181b073b5c48e16a";
    attribute INIT_37 : string;
    attribute INIT_37 of U_RAM : label is
       "6029b70c60180002001a06d81c0874641c2a06d818187464191900005c080000";
    attribute INIT_38 : string;
    attribute INIT_38 of U_RAM : label is
       "601800006019f9da240943462429f9da20194346201b04966449b70c640b0496";
    attribute INIT_39 : string;
    attribute INIT_39 of U_RAM : label is
       "68180003001a0f79240955e3242a0f79201955e3211900006408000064290000";
    attribute INIT_3a : string;
    attribute INIT_3a of U_RAM : label is
       "6819fba82c0966f62c29fba8281966f6281b02756c4835f46c0b0275682835f4";
    attribute INIT_3b : string;
    attribute INIT_3b of U_RAM : label is
       "001a70d82c080a292c2a70d828180a29291900006c0800006c29000068180000";
    attribute INIT_3c : string;
    attribute INIT_3c of U_RAM : label is
       "3408500b3429fd303018500b301b00c4744816a6740b00c4702816a670180006";
    attribute INIT_3d : string;
    attribute INIT_3d of U_RAM : label is
       "3408000034280000301800003118000074080000742800007018000070 19fd30";
    attribute INIT_3e : string;
    attribute INIT_3e of U_RAM : label is
       "3c28000038180000381800007c4800007c080000782800007818000000180000";
    attribute INIT_3f : string;
    attribute INIT_3f of U_RAM : label is
       "000179ce380000003b0000007c0800007c280000781800007818 00003c080000";

    signal dob : std_logic_vector (31 downto 0);
    signal addrb : std_logic_vector (8 downto 0);

 begin
    addrb(6 downto 0) <= A;
    addrb(8 downto 7) <= sel(1 downto 0);
    -- port A for configuration, port B for IIR filter engine
    U_RAM : RAMB16_S18_S36
      port map
      (
        ADDRA => ADDR(9 downto 0),
        ADDRB => addrb,
        CLKA  => CLK,
        CLKB  => CLK,
        DIA   => DI(15 downto 0),
        DIB   => (others => '0'),
        DIPA  => DI(17 downto 16),
        DIPB  => (others => '0'),
        DOA   => DO(15 downto 0),
        DOB   => dob,
        DOPA  => DO(17 downto 16),
        DOPB  => CTRL(17 downto 14),
        ENA   => '1',
        ENB   => '1',
        SSRA  => '0',
        SSRB  => '0',
        WEA   => MWE,
        WEB   => '0'
      );
    CTRL(13 downto 0) <= dob (31 downto 18);
    COEF(17 downto 0) <= dob (17 downto 0);

 end architecture RTL;
 ------------------------------------------------------------
```

```
-----------------------------------------------------------------------------
-- SubModule SmallHistoryFile
-- Created   4/10/2005 7:28:13 PM
-----------------------------------------------------------------------------
library IEEE;
use IEEE.Std_Logic_1164.all;
use IEEE.Numeric_Std.all;
--synopsys translate_off
library UNISIM;
use unisim.vcomponents.all;
--synopsys translate_on

entity SmallHistoryFile is port
   (
     d        : in    std_logic_vector(34 downto 0); -- data in
     b        : out   std_logic_vector(17 downto 0); -- data out
     CLK      : in    std_logic;                     -- clock
     A        : in    std_logic_vector(4 downto 0);  -- address
     HT       : in    std_logic;                     -- toggle order of history in memory
     W        : in    std_logic                      -- write enable
   );
end SmallHistoryFile;
-----------------------------------------------------------------------------

-----------------------------------------------------------------------------
architecture RTL of SmallHistoryFile is

   -- Component Declarations
   component RAM16X1D
      --synopsys translate_off
      generic
      (
        INIT : bit_vector := X"0000"
      );
      --synopsys translate_on
      port
      (
        A0    : in  STD_LOGIC;
        A1    : in  STD_LOGIC;
        A2    : in  STD_LOGIC;
        A3    : in  STD_LOGIC;
        D     : in  STD_LOGIC;
        DPO   : out STD_LOGIC;
        DPRA0 : in  STD_LOGIC;
        DPRA1 : in  STD_LOGIC;
        DPRA2 : in  STD_LOGIC;
        DPRA3 : in  STD_LOGIC;
        SPO   : out STD_LOGIC;
        WCLK  : in  STD_LOGIC;
        WE    : in  STD_LOGIC
      );
   end component RAM16X1D;

   attribute INIT : string;
   attribute INIT of all: label is "0000";
--   attribute INIT of bit_msb: label is "0000";

   -- Signal Declarations
   signal A4T, A4TN, MSB : std_logic;
   signal memin : std_logic_vector(35 downto 0);

begin
   MSB <= A(0);  -- selects the MSB/LSB (only used for read out)
   -- A(3..1) are for selecting the second order section, A(4) selects old/new
   A4T <= A(4) xor HT;  -- makes sure history values are interchanged everytime filter runs
   A4TN <= not A4T;      -- this one is for write, the above is for reading
    -- split 35 bit input into two 18 bit vectors; don't write garbage during simulation init
   memin(16 downto 0) <= To_StdLogicVector (To_bitvector (d(16 downto 0)));
   memin(17) <= '0';
   memin(35 downto 18) <= To_StdLogicVector (To_bitvector (d(34 downto 17)));

   -- memory is 18 times 2bit with output select for MSB/LSB and an output register
   memory: for i in 17 downto 0 generate
      signal mout : std_logic_vector(1 downto 0);
```

```
      signal mo : std_logic;
   begin
      -- select memory: dual port, 2 x 1 bit for LSB and MSB
      bit_lsb: component RAM16X1D
         port map
         (
            A0 => A(1),
            A1 => A(2),
            A2 => A(3),
            A3 => A4TN,
            D => memin(i),
            SPO => open,
            DPRA0 => A(1),
            DPRA1 => A(2),
            DPRA2 => A(3),
            DPRA3 => A4T,
            DPO => mout(0),
            WCLK => CLK,
            WE => W
         );
      bit_msb: component RAM16X1D
         port map
         (
            A0 => A(1),
            A1 => A(2),
            A2 => A(3),
            A3 => A4TN,
            D => memin(i+18),
            SPO => open,
            DPRA0 => A(1),
            DPRA1 => A(2),
            DPRA2 => A(3),
            DPRA3 => A4T,
            DPO => mout(1),
            WCLK => CLK,
            WE => W
         );
      -- select LSB/MSB of output
      lsbsel : process (MSB, mout) is
      begin
         case MSB is
            when '0' =>
               mo <= mout(0);
            when '1' =>
               mo <= mout(1);
            when others =>
               mo <= '0';
         end case;
      end process lsbsel;
      -- add register for output
      reg: process (CLK, mo) is
      begin
         if rising_edge (CLK) then
            b(i) <= mo;
         end if;
      end process reg;
   end generate memory;

end architecture RTL;
--------------------------------------------------------------------------------
```

```vhdl
--------------------------------------------------------------------------------
-- SubModule Mux2Reg18
-- Created   4/10/2005 5:55:36 PM
--------------------------------------------------------------------------------
Library IEEE;
Use IEEE.Std_Logic_1164.all;

entity Mux2Reg18 is port
   (
     a         : in    std_logic_vector(31 downto 0); -- ADC input
     old       : out   std_logic_vector(31 downto 0); -- old ADC input
     h         : in    std_logic_vector(17 downto 0); -- history input
     b         : out   std_logic_vector(17 downto 0); -- output
     CLK       : in    std_logic;                     -- clock
     RE        : in    std_logic;                     -- register enable
     SEL       : in    std_logic_vector(1 downto 0)   -- input select
   );
end Mux2Reg18;
--------------------------------------------------------------------------------

--------------------------------------------------------------------------------
architecture RTL of Mux2Reg18 is

   signal reg : std_logic_vector(31 downto 0);     -- register to delay input
   signal muxout : std_logic_vector(17 downto 0); -- output of ADC/history mux


begin
   -- ADC input register
   register1: process (CLK, RE) is
   begin
      if rising_edge (CLK) then
         if RE = '1' then
            reg <= a;
         end if;
      end if;
   end process register1;

   -- ADC delay register for straight through path
   register2: process (CLK, RE) is
   begin
      if rising_edge (CLK) then
         if RE = '1' then
            old <= reg;
         end if;
      end if;
   end process register2;

   -- input select mux
   mux: process (reg, h, sel) is
   begin
      case sel is
         when B"00" =>
            muxout(17) <= '0';
            muxout(16 downto 3) <= reg(13 downto 0);
            muxout(2 downto 0) <= (others => '0');
         when B"01" =>
            muxout <= reg(31 downto 14);
         when B"10" | B"11" =>
            muxout <= h;
         when others =>
            muxout <= h;
      end case;
   end process mux;
   -- write output
   b <= muxout when rising_edge (CLK);

end architecture RTL;
--------------------------------------------------------------------------------
```

```vhdl
------------------------------------------------------------------------------
-- SubModule MulShifter
-- Created   4/8/2005 8:30:57 PM
------------------------------------------------------------------------------
library IEEE;
use IEEE.Std_Logic_1164.all;
use IEEE.Numeric_Std.all;

entity MulShifter is
   port (
     a        : in    std_logic_vector(35 downto 0);
     b        : out   std_logic_vector(47 downto 0);
     sel      : in    std_logic_vector (1 downto 0);
     CLK      : in    std_logic
   );
end MulShifter;
------------------------------------------------------------------------------

------------------------------------------------------------------------------
architecture RTL of MulShifter is

   --constant mulwidth : integer := 18;
   --constant width : integer := 48;
   --constant drop : integer := 25;

   signal s : std_logic_vector (72 downto 0);

begin
   shifter: process (a, sel) is
   begin
      case sel is
         -- LSB * LSB: no shift, no sign extend
         when B"00" =>
            s(33 downto 0) <= a(33 downto 0);
            s(72 downto 34) <= (others => '0');
         -- MSB * LSB or LSB * MSB: shift 17 bits, sign extend
         when B"01" =>
            s(16 downto 0) <= (others => '0');
            s(52 downto 17) <= a;
            s(72 downto 53) <= (others => a(a'left));
         -- MSB * MSB: shift 34 bits, sign extend
         when B"10" | B"11" =>
            s(33 downto 0) <= (others => '0');
            s(69 downto 34) <= a;
            s(72 downto 70) <= (others => a(a'left));
         when others =>
            s(72 downto 0) <= (others => '0');
      end case;
   end process shifter;
   -- output register
   b <= s(72 downto 25) when rising_edge (CLK);
end architecture RTL;
------------------------------------------------------------------------------
```

```vhdl
--------------------------------------------------------------------------------
-- SubModule Accumulator
-- Created   4/8/2005 8:30:57 PM
--------------------------------------------------------------------------------
library IEEE;
use IEEE.Std_Logic_1164.all;
use IEEE.Numeric_Std.all;

entity Accumulator is
--   generic (
--     width    : natural := 48
--   );
   port (
     a        : in    std_logic_vector(47 downto 0);
     b        : out   std_logic_vector(47 downto 0);
     p        : in    std_logic_vector(47 downto 0);
     R        : in    std_logic;
     L        : in    std_logic;
     CLK      : in    std_logic;
     CE       : in    std_logic
   );
end Accumulator;
--------------------------------------------------------------------------------

--------------------------------------------------------------------------------
architecture RTL of Accumulator is

   signal sum : signed(47 downto 0);

begin
   accu: process (CLK) is
   begin
      if rising_edge (CLK) then
         if CE = '1' then
            if R = '1' then
               sum <= (others => '0');
            elsif L = '1' then
               sum <= signed(p);
            else
               sum <= signed(a) + signed(sum);
            end if;
         end if;
      end if;
   end process accu;
   b <= std_logic_vector(sum);
end architecture RTL;
--------------------------------------------------------------------------------
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

--
-- performs the operation b <= shift_right (a, sel)
--
entity BarrelShifter2 is
   port (a : in std_logic_vector (47 downto 0);
         b : out std_logic_vector (47 downto 0);
         sel : in std_logic_vector (2 downto 0));
end entity BarrelShifter2;


architecture RTL of BarrelShifter2 is

   subtype data_type is std_logic_vector (47 downto 0);

   function shift_right (a : in data_type; n : in natural) return data_type is
   begin
      return std_logic_vector (shift_right (signed(a), n));
   end;

begin
   Shifter: process (a, sel)
   begin
      case sel is
         when B"000" =>
            b <= a;
         when B"001" =>
            b <= shift_right (a, 1);
         when B"010" =>
            b <= shift_right (a, 2);
         when B"011" =>
            b <= shift_right (a, 3);
         when B"100" =>
            b <= shift_right (a, 4);
         when B"101" =>
            b <= shift_right (a, 5);
         when B"110" =>
            b <= shift_right (a, 6);
         when B"111" =>
            b <= shift_right (a, 7);
         when others =>
            b <= a;
      end case;
   end process Shifter;
end architecture RTL;
```

```vhdl
--------------------------------------------------------------------------------
-- SubModule OverflowDetect
-- Created   4/14/2005 3:56:08 PM
--------------------------------------------------------------------------------
Library IEEE;
Use IEEE.Std_Logic_1164.all;

entity OverflowDetect is port
   (
     a         : in    std_logic_vector(47 downto 0);
     b         : out   std_logic_vector(34 downto 0);
     IOVF      : in    std_logic;
     OVF       : out   std_logic
   );
end OverflowDetect;
--------------------------------------------------------------------------------

--------------------------------------------------------------------------------
architecture RTL of OverflowDetect is

   signal overflow : std_logic;

begin
   -- detect overflow
   overflow <= '1' when (a(46) /= a(47)) or (a(45) /= a(47)) or
                        (a(44) /= a(47)) or (a(43) /= a(47)) or
                        (a(42) /= a(47)) or (IOVF = '1')
           else '0';
   OVF <= overflow;

   -- mark overflow in output
   mark: process (overflow, a) is
   begin
      if overflow = '1' and a(36) = '0' then
         b <= ('0', '0', '0', '0', '0', '0', others => '1');
      elsif overflow = '1' and a(36) = '1' then
         b <= ('1', '1', '1', '1', '1', '1', others => '0');
      else
         b <= a (42 downto 8);
      end if;
   end process mark;

end architecture RTL;
--------------------------------------------------------------------------------
```

```
------------------------------------------------------------------------------
-- SubModule Reg32
-- Created   4/10/2005 7:00:40 PM
------------------------------------------------------------------------------
library IEEE;
use IEEE.Std_Logic_1164.all;

entity Reg32 is port
   (
     a        : in    std_logic_vector(31 downto 0);
     b        : out   std_logic_vector(31 downto 0);
     CLK      : in    std_logic;
     CE       : in    std_logic
   );
end Reg32;
------------------------------------------------------------------------------

------------------------------------------------------------------------------
architecture RTL of Reg32 is
begin
   reg: process (CLK, CE) is
   begin
      if rising_edge (CLK) then
         if CE = '1' then
            b <= a;
         end if;
      end if;
   end process reg;
end architecture RTL;
------------------------------------------------------------------------------
```

```
-------------------------------------------------------------------------------
-- SubModule Reg18
-- Created   4/10/2005 6:07:36 PM
-------------------------------------------------------------------------------
library IEEE;
use IEEE.Std_Logic_1164.all;

entity Reg18 is port
   (
     a        : in    std_logic_vector(17 downto 0);
     b        : out   std_logic_vector(17 downto 0);
     CLK      : in    std_logic;
     CE       : in    std_logic
   );
end Reg18;
-------------------------------------------------------------------------------

-------------------------------------------------------------------------------
architecture RTL of Reg18 is

begin
   reg: process (CLK, CE) is
   begin
      if rising_edge (CLK) then
         if CE = '1' then
            b <= a;
         end if;
      end if;
   end process reg;
end architecture RTL;
-------------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------------
-- SubModule Reg3
-- Created   4/11/2005 12:16:29 PM
--------------------------------------------------------------------------------
library IEEE;
use IEEE.Std_Logic_1164.all;

entity Reg3 is port
   (
     a        : in    std_logic_vector(2 downto 0);
     b        : out   std_logic_vector(2 downto 0);
     CLK      : in    std_logic;
     CE       : in    std_logic
   );
end Reg3;
--------------------------------------------------------------------------------

--------------------------------------------------------------------------------
architecture RTL of Reg3 is

begin
   reg: process (CLK, CE) is
   begin
      if rising_edge (CLK) then
         if CE = '1' then
            b <= a;
         end if;
      end if;
   end process reg;
end architecture RTL;
--------------------------------------------------------------------------------
```

```
--
-- Test Bench for IIR filter
--
library ieee;
use std.textio.all;
use ieee.numeric_std.all;
use ieee.std_logic_1164.all;
use ieee.std_logic_textio.all;


entity iir_RTL_test_bench is
end entity iir_RTL_test_bench;


architecture test of iir_RTL_test_bench is
   -- Path
   constant path : string := "C:\User\Daniel\Protel\FPGA_Biquad\";
   -- Input file. Format: x
   constant sim_in : string (1 to 50) :=
         "simdata\stim_sqrsweep1_17s.txt###################";
   -- Output file. Format: t  y
   constant sim_out : string :=
         "simdata\resp_sqr1_30b_1g_48a_16s.txt#############";
   -- master clock
   constant clock_time : time := 0.5**26*1000 ms;   --14.9 ns;
   -- master clock high
   constant clock_hi   : time := 0.5 * clock_time;
   -- master clock low
   constant clock_low  : time := clock_time - clock_hi;
   -- ADC clock multiplier
   constant adc_clock_time : integer := 128;
   -- ADC clock multiplier high
   constant adc_clock_hi   : integer := 100;
   -- ADC clock multiplier low
   constant adc_clock_low  : integer := adc_clock_time - adc_clock_hi;
   -- 1pps clock multiplier
   constant onepps_clock_time : integer := 2**26;
   -- 1pps clock multiplier high
   constant onepps_clock_hi   : integer := 4;
   -- 1pps clock multiplier low
   constant onepps_clock_low  : integer := onepps_clock_time - onepps_clock_hi;
   -- simulation T0 time shift multiplier
   constant t0 : integer := -10;

   -- adc value type
   subtype adc_type is std_logic_vector (17 downto 0);
   -- filter output type
   subtype output_type is std_logic_vector (31 downto 0);

   -- clock signal
   signal clk : std_logic := '1';
   -- ADC signal
   signal adc_clk : std_logic := '0';
   -- 1 pps signal
   signal onepps : std_logic := '0';

   -- ADC input value
   signal x1 : adc_type := (others => '0');
   signal x1r : real := 0.0;
   -- ADC/filter overflow output value
   signal x1_ovf : std_logic := '0';
   -- latched overflow
   signal overflow : std_logic := '0';
   -- filter output value
   signal y1 : output_type := (others => '0');
   signal y1r : real := 0.0;

   -- IIR_top component
   component IIR is port
   (
      AD       : In    STD_LOGIC_VECTOR(11 downto 0);
      ADC0     : In    STD_LOGIC_VECTOR(17 downto 0);
      ADC0L    : In    STD_LOGIC;
      ADC0OVF  : Out   STD_LOGIC;
```

```
      CLK     : In    STD_LOGIC;
      CLK1PPS : In    STD_LOGIC;
      CS      : In    STD_LOGIC;
      D       : InOut STD_LOGIC_VECTOR(31 downto 0);
      SEL     : In    STD_LOGIC_VECTOR(2 downto 0);
      WR      : In    STD_LOGIC
   );
   end component IIR;


begin

   ---------------------------------------------------------------------------
   -- Clock generator
   ---------------------------------------------------------------------------
   clock_gen: process is
      variable count : integer := t0;
   begin
      clk <= '0' after clock_hi, '1' after clock_low;
      if count mod adc_clock_time = 0 then
         adc_clk <= '1';
      elsif count mod adc_clock_time = adc_clock_hi then
         adc_clk <= '0';
      end if;
      if count mod onepps_clock_time = 0 then
         onepps <= '1';
      elsif count mod onepps_clock_time = onepps_clock_hi then
         onepps <= '0';
      end if;
      wait for clock_time;
      count := count + 1;
   end process clock_gen;


   ---------------------------------------------------------------------------
   -- Stimulus: Impulse response
   ---------------------------------------------------------------------------
--   stimulus_impulse: process (clk) is
--      -- filter values
--      variable r : real := 0.0;
--   begin
--      if rising_edge (adc_clk) then
--         r := 1.0;
--         x1r <= r;
--         x1 <= convert_adc_value (r, 2.0**(adc_type'length-2));
--      end if;
--   end process stimulus_impulse;


   ---------------------------------------------------------------------------
   -- Stimulus: read excitation from file
   ---------------------------------------------------------------------------
   stimulus_file: process is
      -- filter output file
      file input : text;
      -- line
      variable l : line;
      -- status
      variable status : file_open_status;
      -- filter values
      variable r : real := 0.0;
      -- string length
      variable len : integer := sim_in'length;

      --
      -- convert a real value into a ADC value
      --
      function convert_adc_value (a : in real;
            scaling : in real := 2.0**(adc_type'length-2)) return adc_type is

         subtype adc_signed is signed (adc_type'length-1 downto 0);
         constant q : real := 2.0**(adc_type'length-1);
         constant max : adc_signed := (adc_type'left => '0', others => '1');
         variable x : real;
```

```
          variable xx : real;
          variable y : adc_signed := (others => '0');

       begin
          x := a * scaling;
          xx := abs (x);
          if (xx >= q) then
             y := max;
          else
             for i in y'left-1 downto 0 loop
                if (xx >= q/2.0) then
                   y(i) := '1';
                   xx := xx - q/2.0;
                else
                   y(i) := '0';
                end if;
                xx := 2.0 * xx;
             end loop;
             if (xx > q/2.0) and (y /= max) then
                y := y + 1;
             end if;
          end if;
          if (x < 0.0) then
             y := -y;
          end if;
          return std_logic_vector(y);
       end function convert_adc_value;

    begin
       for i in 1 to sim_in'length loop
          if sim_in (i) = '#' then
             len := i - 1;
             exit;
          end if;
       end loop;
       -- read filter coefficients from file
       file_open (status, input, path & sim_in(1 to len), read_mode);
       assert status = open_ok
          report "File open failed for " & sim_in(1 to len) severity failure;
       report "Input file is " & sim_in(1 to len);
       -- read data from disk
       while not endfile (input) loop
          readline (input, l);
          read (l, r);
          wait until rising_edge (adc_clk);
          xlr <= r;
          x1 <= convert_adc_value (r, 2.0**(adc_type'length-2));
       end loop;
       file_close (input);
       -- done
       wait;
    end process stimulus_file;


    ---------------------------------------------------------------------------
    -- Filter instantiation
    ---------------------------------------------------------------------------
    fiter1: component IIR
       port map (
          ADC0 => x1,
          ADC0L => adc_clk,
          ADC0OVL => x1_ovf,
          SEL => B"000",
          CLK => clk,
          CLK1PPS => onepps,
          AD => B"000000000001",
          D => y1,
          WR => '0',
          CS => '1'
       );

    -- convert output to real
    real_output: process (y1) is
       -- ADC output to real
```

```
      function adc_to_real (x : in std_logic_vector;
                            decimal : in integer := -1) return real is
         variable y, q : real;
         variable n : integer;
      begin
         if (decimal < 0) or (decimal > x'length) then
            n := x'length;
         else
            n := decimal;
         end if;
         q := 2.0**(n-1);
         if x(x'left) = '1' then
            y := -q;
         else
            y := 0.0;
         end if;
         for i in x'range loop
            if (i /= x'left) and (x(i) = '1') then
               y := y + q;
            end if;
            q := q / 2.0;
         end loop;
         return y;
      end function adc_to_real;
   begin
   y1r <= adc_to_real (y1, output_type'length - (adc_type'length - 2 + 9));
   end process real_output;

   -- latch overflow
   overflow <= '1' when x1_ovf = '1';


   ------------------------------------------------------------------------------
   -- Write results to disk
   ------------------------------------------------------------------------------
   recording: process is
      -- filter output file
      file output : text;
      -- line
      variable l : line;
      -- status
      variable status : file_open_status;
      -- number of data points to skip
      constant skip : integer := 16384;
      -- number of data points to keep
      constant keep : integer := 16*16384;
      -- recording enabled?
      constant enabled : boolean := true;
      -- string length
      variable len : integer := sim_out'length;

   begin
      if enabled then
         -- skip
         for i in 1 to skip loop
            wait until rising_edge (adc_clk);
         end loop;
         --
         -- read filter coefficients from file
         for i in 1 to sim_out'length loop
            if sim_out (i) = '#' then
               len := i - 1;
               exit;
            end if;
         end loop;
         file_open (status, output, path & sim_out(1 to len), write_mode);
         assert status = open_ok
            report "File open failed for " & sim_out(1 to len) severity failure;
         report "Output file is " & sim_out(1 to len);
         -- write data to disk
         for i in 1 to keep loop
            wait until rising_edge (adc_clk);
            write (l, y1r, right, 20, 14);
            writeline (output, l);
```

```
          end loop;
          file_close (output);
          report "Overflow is " & std_logic'image (overflow);
      end if;
      -- done
      wait;
   end process recording;

end architecture test;
```

```
--
-- IIR filter types
--
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
-- use ieee.math_real.all;

package iir_types is
   --
   -- Biquad types and constant (real)
   --
   -- biquad coefficients
   type biquad_record is record
      a1: real;
      a2: real;
      b1: real;
      b2: real;
   end record biquad_record;
   -- maximum biquad sections
   constant biquad_max : natural := 16;
   -- array of biquads
   type biquad_array is array (natural range 1 to biquad_max) of biquad_record;
   -- iir coefficient record
   type iir_record is record
      gain: real;
      num_biquad: natural;
      biquads: biquad_array;
   end record iir_record;
   -- history array of reals
   type history_array is array (natural range 1 to biquad_max) of real;

   --
   -- Biquad types and constant (signed fixed point)
   --
   -- number of digits in a's and b's
   constant filter_coeff_digits : natural := 35;
   -- number of digits in muliplier
   constant multiplier_digits : natural := 35;
   -- number of digits in filter gain
   constant filter_gain_digits : natural := filter_coeff_digits;
   -- position of decimal point (count from left) in muliplier filter coefficient
   constant multiplier_coeff_decimal : natural := 2;
   -- position of decimal point (count from left) in muliplier filter gain
   constant multiplier_gain_decimal : natural := 2 + 0*4;
   -- number of digits in filter values
   constant filter_value_digits : natural := multiplier_digits;
   -- number of digits in final filter output
   constant filter_output_digits : natural := 32;
   -- number of digits in adc values
   constant adc_digits : natural := 18;
   -- number of digits in the accumulators
   constant accumulator_digits : natural := 73;
   -- number of dropped digits in the accumulator
   constant accumulator_dropped_digits : natural := 25;
   -- number of digits in shift value
   constant filter_shift_digits : natural := 4;
   -- right shift after converting adc to filter value
   constant adc_filter_shift : natural := 2 + (filter_value_digits - filter_output_digits);
   -- right shift after converting filter value to muliplier
   constant filter_mul_shift : natural := multiplier_digits - filter_value_digits;
   -- right shift after converting filter coefficients to muliplier
   constant coeff_mul_shift : natural := 0;
   -- right shift after converting gain to multiplier
   constant gain_mul_shift : natural := coeff_mul_shift;
   -- right shift before converting gain muliplier result back to filter value
   constant mul_filter_shift : natural := multiplier_digits-multiplier_gain_decimal;
   -- right shift after converting filter value to accumulator precision
   constant filter_accumulator_shift : natural :=
      (accumulator_digits-(multiplier_digits-multiplier_coeff_decimal)-
       (multiplier_digits-filter_mul_shift));
   -- right shift after converting muliplier result to accumulator precision
   constant mul_accumulator_shift : natural := accumulator_digits-2*multiplier_digits;
   -- right shift before converting accumulator values to filter values
```

```
    constant accumulator_filter_shift : natural :=
       accumulator_digits - filter_value_digits - filter_accumulator_shift;
    -- right shift before converting filter value to output value
    constant filter_out_shift : natural := 0;
    -- output decimation
    constant out_decimation : natural := 32;

    -- adc value type
    subtype adc_type is signed (adc_digits-1 downto 0);
    -- filter value type
    subtype filter_value_type is signed (filter_value_digits-1 downto 0);
    -- filter output type
    subtype filter_output_type is signed (filter_output_digits-1 downto 0);
    -- filter coefficient type
    subtype filter_coeff_type is signed (filter_coeff_digits-1 downto 0);
    -- multiplier input type
    subtype multiplier_type is signed (multiplier_digits-1 downto 0);
    -- multiplier output type
    subtype multiplier_result_type is signed (2*multiplier_digits-1 downto 0);
    -- accumulator type
    subtype accumulator_type is signed (accumulator_digits-accumulator_dropped_digits-1 downto 0);
    -- filter shift type
    subtype filter_shift_type is signed (filter_shift_digits-1 downto 0);
    -- biquad coefficients
    type biquad_coeffs is record
       a1: filter_coeff_type; -- sign changed to normal convention!
       a2: filter_coeff_type; -- sign changed to normal convention!
       b1: filter_coeff_type;
       b2: filter_coeff_type;
       c0: filter_shift_type;
    end record biquad_coeffs;
    -- array of biquad coeffcients
    type biquad_coeffs_array is array (natural range 1 to biquad_max) of biquad_coeffs;
   -- iir coefficient record
    type iir_coeffs is record
       gain: filter_coeff_type;
       num_biquad: natural;
       biquads: biquad_coeffs_array;
    end record iir_coeffs;
    -- biquad history
    type biquad_history is record
       x1 : filter_value_type; -- old x
       x2 : filter_value_type; -- old old x
       y1 : filter_value_type; -- old y
       y2 : filter_value_type; -- old old y
    end record biquad_history;
    -- array of biquad history values
    type iir_history is array (natural range 1 to biquad_max) of biquad_history;

    --
    --
    -- signed to hex string
    procedure signed_to_hex (str : out string; len : out integer;
       x : in signed; digits : in integer := -1);
    -- signed to string
    -- the decimal point is specified counting from left
    procedure signed_to_string (str : out string; len : out integer;
       x : in signed; decimal : in integer := -1);
    -- signed to real
    -- the decimal point is specified counting from left
    function signed_to_real (x : in signed; decimal : in integer := -1) return real;

    --
    -- Biquad subroutines (real)
    --
    -- computes a biquad
    procedure ideal_biquad (x : in real; y : out real;
          h1: inout real; h2: inout real; coeff: in biquad_record);

    -- computes an IIR filter on a single value
    procedure ideal_iir (x : in real; y : out real;
          h1: inout history_array; h2: inout history_array;
          iir : in iir_record);
```

```vhdl
    --
    -- Biquad subroutines (signed fixed point)
    --
    -- convert a real value into adc
    function convert_adc_value (a : in real;
       scaling : in real := 2.0**(adc_digits-2)) return adc_type;

    -- convert a real filter coefficient into a signed
    function convert_filter_coeff (a : in real) return filter_coeff_type;

    -- convert a filter gain into a multiplier shift
    function convert_filter_gain (g : in real) return filter_shift_type;


    -- convert one signed vector to another
    -- the output array must be at least as long as the input array
    -- empty bits at the end are filled with zeros
    -- values are then shifted and sign extended if necessary
    procedure convert_vector_shift (x : in signed;
          y : out signed; shift : in natural);

    -- convert one signed vector to another
    -- the output array must be at equal or shorter than the input array
    -- the value is first shifted to the right and sign extended if necessary
    -- excess bits at the front are dropped, the overflow flag is set appropriately
    procedure convert_shift_vector (x : in signed;
          y : out signed; shift : in natural; ovf : out boolean);

    -- computes a biquad
    procedure fixedpoint_biquad (x : in filter_value_type;
          y : out filter_value_type; h: inout biquad_history;
          coeff: in biquad_coeffs; overflow : out boolean);

    -- computes an IIR filter on a single value
    procedure fixedpoint_iir (x : in adc_type;
          y : out filter_output_type; h: inout iir_history;
          iir : in iir_coeffs; overflow : out boolean;
          maxrange : out real);

  end package iir_types;


  package body iir_types is

    --
    -- signed to hex
    --
    procedure signed_to_hex (str : out string; len : out integer;
       x : in signed; digits : in integer := -1) is
       variable d, n : integer;
       variable y,mask : signed (x'high downto x'low);
       variable s : string (1 to str'high-str'low+1);
       variable l : integer := 0;
    begin
       d := digits;
       y := x;
       mask := (others => '0');
       mask(x'low+0) := '1';
       mask(x'low+1) := '1';
       mask(x'low+2) := '1';
       mask(x'low+3) := '1';
       loop
          n := to_integer (y and mask);
          if (n < 10) then
             s(1 to l+1) := (character'val(character'pos('0')+n)) & s(1 to l);
          else
             s(1 to l+1) := (character'val(character'pos('A')+n-10)) & s(1 to l);
          end if;
          l := l + 1;
          y := y srl 4;
          d := d - 1;
          exit when (d = 0) or ((d < 0) and (y = 0));
       end loop;
```

```
        str := s;
        len := l;
    end procedure signed_to_hex;

    --
    -- signed to string
    --
    procedure signed_to_string (str : out string; len : out integer;
        x : in signed; decimal : in integer := -1) is
        variable y : signed (x'high downto x'low);
        variable y2 : signed (x'high+4 downto x'low) := (others => '0');
        variable s : string (1 to 100);
        variable l, l2 : integer := 0;
        variable n : integer;
    begin
        if (decimal < 0) or (decimal > x'length) then
            n := x'length;
        else
            n := decimal;
        end if;
        y := x srl (x'length - n);
        signed_to_hex (s, l, y);
        if (n /= x'length) then
            s(1 to l+1) := s(1 to l) & '.';
            l := l + 1;
            y := x;
            if n > 0 then
                y(x'high downto x'high-n+1) := (others => '0');
            end if;
            y2(x'high downto x'low) := y;
            l2 := (x'length - n) / 4;
            if (x'length - n) mod 4 /= 0 then
                y2 := y2 sll (4 - ((x'length - n) mod 4));
                l2 := l2 +1;
            end if;
            signed_to_hex (s (l+1 to 100), l2, y2, l2);
            l := l + l2;
        end if;
        str := s;
        len := l;
    end procedure signed_to_string;

    --
    -- signed to real
    --
    function signed_to_real (x : in signed; decimal : in integer := -1) return real is
        variable y, q : real;
        variable n : integer;
    begin
        if (decimal < 0) or (decimal > x'length) then
            n := x'length;
        else
            n := decimal;
        end if;
        q := 2.0**(n-1);
        if x(x'left) = '1' then
            y := -q;
        else
            y := 0.0;
        end if;
        for i in x'range loop
            if (i /= x'left) and (x(i) = '1') then
                y := y + q;
            end if;
            q := q / 2.0;
        end loop;
        return y;
    end function signed_to_real;

    --
    -- computes a biquad
    --
    procedure ideal_biquad (x : in real; y : out real;
        h1: inout real; h2: inout real; coeff: in biquad_record) is
```

```
      variable h0 : real;
   begin
      -- apply single second order section
      h0 := x - h1 * coeff.a1 - h2 * coeff.a2;
      y  := h0 + h1 * coeff.b1 + h2 * coeff.b2;
      -- update history buffers
      h2 := h1;
      h1 := h0;
   end procedure ideal_biquad;

   --
   -- computes an IIR filter on a single value
   --
   procedure ideal_iir (x : in real; y : out real;
         h1: inout history_array; h2: inout history_array;
         iir : in iir_record) is
      variable r : real;
      variable s : real;
   begin
      r := iir.gain * x;
      for i in 1 to iir.num_biquad loop
         ideal_biquad (r, s, h1(i), h2(i), iir.biquads(i));
         r := s;
      end loop;
      y := r;
   end procedure ideal_iir;


   --
   -- convert a real value into a ADC value
   --
   function convert_adc_value (a : in real;
         scaling : in real := 2.0**(adc_digits-2)) return adc_type is
      constant q : real := 2.0**(adc_digits-1);
      constant max : adc_type := (adc_type'left => '0', others => '1');
      variable x : real;
      variable xx : real;
      variable y : adc_type := (others => '0');
   begin
      x := a * scaling;
      xx := abs (x);
      if (xx >= q) then
         y := max;
      else
         for i in y'left-1 downto 0 loop
            if (xx >= q/2.0) then
               y(i) := '1';
               xx := xx - q/2.0;
            else
               y(i) := '0';
            end if;
            xx := 2.0 * xx;
         end loop;
         if (xx > q/2.0) and (y /= max) then
            y := y + 1;
         end if;
      end if;
      if (x < 0.0) then
         y := -y;
      end if;
      return y;
   end function convert_adc_value;

   --
   -- convert a real filter coefficient into a signed
   --
   function convert_filter_coeff (a : in real) return filter_coeff_type is
      constant q : real := 2.0**(multiplier_coeff_decimal-2);
      constant max : filter_coeff_type := (filter_coeff_type'left => '0', others => '1');
      variable c : filter_coeff_type := (others => '0');
      variable aa : real;
      variable s : string (1 to 100);
      variable l : integer;
   begin
```

```
      aa := abs (a);
      if (aa > 2.0*q) then
         c := max;
         report "filter coefficient is too large " & real'image (a) severity error;
      else
         for i in c'left-1 downto 0 loop
            if (aa >= q) then
               c(i) := '1';
               aa := aa - q;
            else
               c(i) := '0';
            end if;
            aa := 2.0 * aa;
         end loop;
         if (aa > q) and (c /= max) then
            c := c + 1;
         end if;
      end if;
      if (a < 0.0) then
         c := -c;
      end if;
      return c;
   end function convert_filter_coeff;

   --
   -- convert a real filter gain into a multiplier shift
   --
   function convert_filter_gain (g : in real) return filter_shift_type is
      variable gain : real;
      variable shift : filter_shift_type;
      variable n : integer;
   begin
      gain := abs (g);
      n := -2**(filter_shift_digits-1);
      for i in -2**(filter_shift_digits-1) to 2**(filter_shift_digits-1)-1 loop
         if (gain > 2.0**i) then
            n := i;
         else
            exit;
         end if;
      end loop;
      shift := to_signed (n, filter_shift_digits);
      return shift;
   end function convert_filter_gain;

   --
   -- convert one std vector to another
   --
   procedure convert_vector_shift (x : in signed;
         y : out signed; shift : in natural) is
      variable temp : signed (y'left downto y'right);
   begin
      if y'left - y'right >= x'left - x'right then
         temp(y'left downto (y'left-(x'left-x'right))) := x;
         temp((y'left-x'left-1) downto y'right) := (others => '0');
      else
         temp(y'left downto y'right) := x(x'left downto (x'left-(y'left-y'right)));
      end if;
      y := shift_right (temp, shift);
   end procedure convert_vector_shift;

   --
   -- convert one std vector to another
   --
   procedure convert_shift_vector (x : in signed;
         y : out signed; shift : in natural; ovf : out boolean) is
      variable temp : signed (x'left downto x'right);
      variable b : boolean := false;
   begin
      temp := shift_right (x, shift);
      y := temp(y'left+x'right downto x'right);
      for i in x'left downto y'left+x'right+1 loop
         b := b or (temp(i) /= temp(y'left));
      end loop;
```

```
        ovf := b;
    end procedure convert_shift_vector;

    --
    -- computes a biquad
    --
    procedure fixedpoint_biquad (x : in filter_value_type;
          y : out filter_value_type; h: inout biquad_history;
          coeff: in biquad_coeffs; overflow : out boolean) is
       variable y0 : filter_value_type;
       variable x1, x2, y1, y2 : multiplier_type;
       variable a1, a2, b1, b2 : multiplier_type;
       variable bx1, bx2, ay1, ay2 : multiplier_result_type;
       variable s, s0, s1, s2, s3, s4 : accumulator_type;
       variable ovf : boolean;
    begin
       -- convert filter values and filter coefficients to muliplier type
       convert_vector_shift (h.x1, x1, filter_mul_shift);
       convert_vector_shift (h.x2, x2, filter_mul_shift);
       convert_vector_shift (h.y1, y1, filter_mul_shift);
       convert_vector_shift (h.y2, y2, filter_mul_shift);
       a1 := coeff.a1;
       a2 := coeff.a2;
       b1 := coeff.b1;
       b2 := coeff.b2;
       -- do multiplication
       bx1 := b1 * x1;
       bx2 := b2 * x2;
       ay1 := a1 * y1;
       ay2 := a2 * y2;
       -- convert mulplier result to accumulator precision
       convert_vector_shift (x, s0, filter_accumulator_shift);
       convert_vector_shift (bx1, s1, mul_accumulator_shift);
       convert_vector_shift (bx2, s2, mul_accumulator_shift);
       convert_vector_shift (ay1, s3, mul_accumulator_shift);
       convert_vector_shift (ay2, s4, mul_accumulator_shift);
       -- sum over input values
       s := s0 + s1 + s2;
       -- scaling
       if (coeff.c0 > 0) then
          s := shift_left (s, to_integer (coeff.c0));
       else
          s := shift_right (s, -to_integer(coeff.c0));
       end if;
       -- add ouput values
       s := s + s3 + s4;
       -- convert sum back to filter value
       convert_shift_vector (s, y0, accumulator_filter_shift-accumulator_dropped_digits, ovf);
       y := y0;
       overflow := ovf;
       -- update history
       h.x2 := h.x1;
       h.x1 := x;
       h.y2 := h.y1;
       h.y1 := y0;
    end procedure fixedpoint_biquad;

    --
    -- computes an IIR filter on a single value
    --
    procedure fixedpoint_iir (x : in adc_type;
          y : out filter_output_type; h: inout iir_history;
          iir : in iir_coeffs; overflow : out boolean;
          maxrange : out real) is
       constant maximum : adc_type := ('0', others => '1');
       constant minimum : adc_type := ('1', others => '0');

       function calc_range (x : in signed) return real is
          variable xr : real;
       begin
          xr := abs (signed_to_real (x, 1));
          return xr;
       end function calc_range;
```

```
      variable x0, y0 : filter_value_type;
      variable xm, gain : multiplier_type;
      variable gx0 : multiplier_result_type;
      variable ovf, o : boolean := false;
      variable rang, mrang : real;
   begin
      -- overflow
      ovf := (x = maximum) or (x = minimum);
      mrang := 0.0;
      -- gain multiplication
      convert_vector_shift (x, x0, adc_filter_shift);
      rang := calc_range (x0);
      if rang > mrang then
         mrang := rang;
      end if;
      convert_vector_shift (x0, xm, filter_mul_shift);
      convert_vector_shift (iir.gain, gain, gain_mul_shift);
      gx0 := xm * gain;
      convert_shift_vector (gx0, x0, mul_filter_shift, o);
      ovf := ovf or o;
      rang := calc_range (x0);
      if rang > mrang then
         mrang := rang;
      end if;
      -- iterate over second order sections
      for i in 1 to iir.num_biquad loop
         fixedpoint_biquad (x0, y0, h(i), iir.biquads(i), o);
         x0 := y0;
         ovf := ovf or o;
         rang := calc_range (x0);
         if rang > mrang then
            mrang := rang;
         end if;
      end loop;
      convert_shift_vector (x0, y, filter_out_shift, o);
      overflow := ovf or o;
      maxrange := mrang;
   end procedure fixedpoint_iir;

end package body iir_types;
```

```
-- Test Bench for IIR filter
--
library ieee;
use std.textio.all;
use ieee.numeric_std.all;
use ieee.std_logic_1164.all;
use ieee.std_logic_textio.all;
use work.iir_types.all;

entity iir_test_bench is
end entity iir_test_bench;


architecture test of iir_test_bench is
   -- Path
   constant path : string := "C:\User\Daniel\Protel\FPGA_Biquad\";
   -- File with IIR filter coefficients
   constant filter_coeff : string := "filters\iir_coeff.txt";
   -- Input file. Format: x1
   constant sim_in : string (1 to 50) :=
         "simdata\stim_sqrsweep1_17s.txt####################";
   -- Output file. Format: t  y1  y2
   constant sim_out : string :=
         "simdata\resp_sqr1_30b_1g_48a_16s.txt##############";
   -- Filter module name
   constant filter_module : string := "ELP8 #####";
   -- ADC clock
   constant adc_clock : time := 2 ns;

   -- filter coefficients
   shared variable initdone : boolean := false;
   shared variable iir_coeff : iir_record;
   shared variable iir_param : iir_coeffs;

   -- clock signal
   signal clk : std_logic;

   -- filter input value (ideal biquad)
   signal x1 : real := 0.0;
   -- filter output value
   signal y1 : real := 0.0;
   -- history values
   signal h1 : history_array;
   signal h2 : history_array;

   -- filter input value (fixed point biquad)
   signal x2 : adc_type := (others => '0');
   -- filter output value
   signal y2 : filter_output_type := (others => '0');
   signal yy2 : real := 0.0;
   -- maximum range (relative)
   signal mrange : real;

   -- difference
   signal dy : real := 0.0;

begin

   ----------------------------------------------------------------------------
   -- Initialization
   ----------------------------------------------------------------------------
   init: process is
      -- filter coeff file
      file coeff : text;
      -- line
      variable l : line;
      -- status
      variable status : file_open_status;
      -- module
      variable module : string (1 to filter_module'length);
      -- string length
      variable modlen : integer := filter_module'length;
      -- good?
      variable good : boolean;
```

```
        -- gain values
        variable dcgain, gain, gainerr : real;
        -- printout
        variable s1, s2, s3, s4 : string (1 to 100);
        variable l1, l2, l3, l4 : integer;

        -- skip leading blank characters
        procedure skip_blank (l : inout line) is
           -- character
           variable ch : character;
        begin
           while l'length /= 0 and
                 (l(l'left) = ' ' or l(l'left) = character'val(160) or l(l'left) = HT) loop
              read (l, ch);
           end loop;
        end procedure skip_blank;

        -- skip a number field
        procedure skip_number (l : inout line) is
           -- character
           variable ch : character;
        begin
           while l'length /= 0 and
                 (l(l'left) >= '0' and l(l'left) <= '9') loop
              read (l, ch);
           end loop;
        end procedure skip_number;

        -- skip a name field
        procedure skip_name (l : inout line) is
           -- character
           variable ch : character;
        begin
           while l'length /= 0 and
                  not (l(l'left) = ' ' or l(l'left) = character'val(160) or l(l'left) = HT) loop
              read (l, ch);
           end loop;
        end procedure skip_name;

     begin
        --
        -- read filter coefficients from file
        file_open (status, coeff, path & filter_coeff);
        assert status = open_ok
           report "File open failed for " & filter_coeff severity failure;
        -- prepare filter module string
        for i in 1 to filter_module'length loop
           if filter_module (i) = '#' then
              modlen := i - 1;
              exit;
           end if;
        end loop;
        report "Filter is " & filter_module(1 to modlen) & " from file " & filter_coeff;
        iir_coeff.gain := 1.0;
        iir_coeff.num_biquad := 0;
        -- read line by line
        while not endfile (coeff) loop
           readline (coeff, l);
           if (l'length < modlen) then
              read (l, module(1 to l'length));
              module(l'length+1 to module'length) := (others => '#');
           else
              read (l, module(1 to modlen));
              module(modlen+1 to module'length) := (others => '#');
           end if;
           if module = filter_module then
              skip_blank (l); skip_number (l);
              skip_blank (l); skip_number (l);
              skip_blank (l);
              read (l, iir_coeff.num_biquad, good);
              assert good
                 report "Unable to read number of biquad sections" severity error;
              skip_blank (l); skip_number (l);
              skip_blank (l); skip_number (l);
```

```
            skip_blank (l); skip_name (l);
            skip_blank (l);
            read (l, iir_coeff.gain, good);
            assert good
               report "Unable to read gain" severity error;
            for i in 1 to iir_coeff.num_biquad loop
               skip_blank (l);
               read (l, iir_coeff.biquads(i).a1, good);
               assert good
                  report "Unable to read coefficient a1" severity error;
               skip_blank (l);
               read (l, iir_coeff.biquads(i).a2, good);
               assert good
                  report "Unable to read coefficient a2" severity error;
               skip_blank (l);
               read (l, iir_coeff.biquads(i).b1, good);
               assert good
                  report "Unable to read coefficient b1" severity error;
               skip_blank (l);
               read (l, iir_coeff.biquads(i).b2, good);
               assert good
                  report "Unable to read coefficient b2" severity error;
               if i /= iir_coeff.num_biquad then
                  readline (coeff, l);
               end if;
            end loop;
         end if;
      end loop;
      -- prepare fixed point version
      iir_param.num_biquad := iir_coeff.num_biquad;
      gain := 1.0;
      gainerr := 1.0;
      for i in iir_coeff.num_biquad downto 1 loop
         iir_param.biquads(i).a1 := convert_filter_coeff (-iir_coeff.biquads(i).a1);
         iir_param.biquads(i).a2 := convert_filter_coeff (-iir_coeff.biquads(i).a2);
         iir_param.biquads(i).b1 := convert_filter_coeff (iir_coeff.biquads(i).b1);
         iir_param.biquads(i).b2 := convert_filter_coeff (iir_coeff.biquads(i).b2);
         dcgain := (1.0+iir_coeff.biquads(i).b1+iir_coeff.biquads(i).b2)/
                   (1.0+iir_coeff.biquads(i).a1+iir_coeff.biquads(i).a2);
         report "DC Gain " & real'image (dcgain) & "  gain err " & real'image (gainerr);
         iir_param.biquads(i).c0 := convert_filter_gain (gainerr/dcgain);
         gain := gain * 2.0**(to_integer(iir_param.biquads(i).c0));
         gainerr := gainerr / dcgain / 2.0**(to_integer(iir_param.biquads(i).c0));
      end loop;
      gain := iir_coeff.gain / gain *
              2.0**(multiplier_coeff_decimal-multiplier_gain_decimal);
      assert abs(gain) < 2.0 report "Gain out of range " & real'image(gain) severity error;
      iir_param.gain := convert_filter_coeff (gain);
      -- print
      report "IIR COEFFICIENTS (REAL)";
      report "biquad num = " & integer'image (iir_coeff.num_biquad);
      report "gain = " & real'image (iir_coeff.gain);
      for i in 1 to iir_coeff.num_biquad loop
         report "a1 = " & real'image (iir_coeff.biquads(i).a1) & "   " &
                "a2 = " & real'image (iir_coeff.biquads(i).a2) & "   " &
                "b1 = " & real'image (iir_coeff.biquads(i).b1) & "   " &
                "b2 = " & real'image (iir_coeff.biquads(i).b2);
      end loop;
      report "IIR COEFFICIENTS (FIXED POINT)";
      report "biquad num = " & integer'image (iir_param.num_biquad);
      signed_to_string (s1, l1, iir_param.gain, multiplier_gain_decimal);
      report "gain = " & s1(1 to l1);
      for i in 1 to iir_param.num_biquad loop
         signed_to_string (s1, l1, iir_param.biquads(i).a1, multiplier_coeff_decimal);
         signed_to_string (s2, l2, iir_param.biquads(i).a2, multiplier_coeff_decimal);
         signed_to_string (s3, l3, iir_param.biquads(i).b1, multiplier_coeff_decimal);
         signed_to_string (s4, l4, iir_param.biquads(i).b2, multiplier_coeff_decimal);
         report "a1 = " & s1(1 to l1) & "   " &
                "a2 = " & s2(1 to l2) & "   " &
                "b1 = " & s3(1 to l3) & "   " &
                "b2 = " & s4(1 to l4) & "   " &
                "c0 = " & integer'image (to_integer(iir_param.biquads(i).c0)) & " (" &
                real'image (2.0**to_integer(iir_param.biquads(i).c0)) & ")";
      end loop;
```

```
      file_close (coeff);
      gain := 1.0e-200;
      assert (gain > 0.0) report "real is not double precision!!!";
      initdone := true;
      wait;
   end process init;


   ---------------------------------------------------------------------------
   -- Clock generator
   ---------------------------------------------------------------------------
   clock_gen: process is
      -- filter values
      variable r : real := 0.0;
   begin
      clk <= '1' after adc_clock / 2, '0' after adc_clock;
      wait for adc_clock;
   end process clock_gen;


   ---------------------------------------------------------------------------
   -- Stimulus: Impulse response
   ---------------------------------------------------------------------------
--   stimulus_impulse: process (clk) is
--      -- filter values
--      variable r : real := 0.0;
--   begin
--      if falling_edge (clk) then
--         r := 1.0;
--         x1 <= r;
--         x2 <= convert_adc_value (r, 2.0**(adc_digits-2));
--      end if;
--   end process stimulus_impulse;


   ---------------------------------------------------------------------------
   -- Stimulus: read excitation from file
   ---------------------------------------------------------------------------
   stimulus_file: process is
      -- filter output file
      file input : text;
      -- line
      variable l : line;
      -- status
      variable status : file_open_status;
      -- filter values
      variable r : real := 0.0;
      -- string length
      variable len : integer := sim_in'length;
   begin
      for i in 1 to sim_in'length loop
         if sim_in (i) = '#' then
            len := i - 1;
            exit;
         end if;
      end loop;
      -- read filter coefficients from file
      file_open (status, input, path & sim_in(1 to len), read_mode);
      assert status = open_ok
         report "File open failed for " & sim_in(1 to len) severity failure;
      report "Input file is " & sim_in(1 to len);
      -- read data from disk
      while not endfile (input) loop
         readline (input, l);
         read (l, r);
         wait until falling_edge (clk);
         x1 <= r;
         x2 <= convert_adc_value (r, 2.0**(adc_digits-2));
      end loop;
      file_close (input);
      -- done
      wait;
   end process stimulus_file;
```

```
---------------------------------------------------------------------------
-- Ideal filter
---------------------------------------------------------------------------
ideal: process (clk) is
   -- filter values
   variable s : real := 0.0;
   -- history values
   variable hist1 : history_array := (others => 0.0);
   variable hist2 : history_array := (others => 0.0);
begin
   --
   -- run filter
   if rising_edge (clk) then
      ideal_iir (x1, s, hist1, hist2, iir_coeff);
      y1 <= s;
      h1 <= hist1;
      h2 <= hist2;
   end if;
end process ideal;


---------------------------------------------------------------------------
-- Fixed point filter
---------------------------------------------------------------------------
fixedpoint: process (clk) is
   -- filter values
   variable s : filter_output_type := (others => '0');
   -- history values
   variable hist : iir_history := (others =>
      (x1 => (others => '0'),
       x2 => (others => '0'),
       y1 => (others => '0'),
       y2 => (others => '0')));
   -- overflow
   variable overflow : boolean;
   -- maximum range
   variable maxrange : real;
begin
   --
   -- run filter
   if rising_edge (clk) then
      fixedpoint_iir (x2, s, hist, iir_param,
                      overflow, maxrange);
      assert not overflow report "Overflow has occured";
      y2 <= s;
      if maxrange > mrange then
         mrange <= maxrange;
      end if;
   end if;
end process fixedpoint;


---------------------------------------------------------------------------
-- Compare results
---------------------------------------------------------------------------
compare: process (y1, y2) is
   --variable yy2 : real;
   variable delta : real;
begin
   yy2 <= signed_to_real (y2, adc_filter_shift-
      (filter_value_digits - filter_output_digits)+2);
   delta := yy2 - y1;
   dy <= delta;
end process compare;


---------------------------------------------------------------------------
-- Write results to disk
---------------------------------------------------------------------------
recording: process is
   -- filter output file
   file output : text;
   -- line
```

```
        variable l : line;
        -- status
        variable status : file_open_status;
        -- number of data points to skip
        constant skip : integer := 512*1024;
        -- number of data points to keep
        constant keep : integer := 16*512*1024;
        -- recording enabled?
        constant enabled : boolean := true;
        -- string length
        variable len : integer := sim_out'length;
    begin
        if enabled then
           -- skip
           for i in 1 to skip loop
              wait until falling_edge (clk);
           end loop;
           --
           -- read filter coefficients from file
           for i in 1 to sim_out'length loop
              if sim_out (i) = '#' then
                 len := i - 1;
                 exit;
              end if;
           end loop;
           file_open (status, output, path & sim_out(1 to len), write_mode);
           assert status = open_ok
              report "File open failed for " & sim_out(1 to len) severity failure;
           report "Output file is " & sim_out(1 to len);
           -- write data to disk
           for i in 0 to keep-1 loop
              wait until falling_edge (clk);
              if i mod out_decimation = 0 then
                 --write (l, real(i) * 2.0**(-19), right, 12, 9);
                 write (l, y1, right, 20, 14);
                 write (l, signed_to_real (y2, adc_filter_shift+filter_out_shift-1), right, 20, 14);
                 writeline (output, l);
              end if;
           end loop;
           file_close (output);
           report "Maximum used range is " & real'image (mrange);
        end if;
        -- done
        wait;
    end process recording;

end architecture test;
```

```
#include <time.h>
#include "iirutil.hh"
#include "FilterDesign.hh"
#include "filterwiz/FilterFile.hh"
#include <math.h>
#include <string>
#include <iostream>
#include <fstream>
#include <iomanip>

using namespace ligogui;
using namespace filterwiz;
using namespace std;


const double PI = 3.14159265358979323846;
const double fS = 524288.;
const double dT = 17.0;
const bool writefiles = true;
const int kMaxSOS = 7;

typedef unsigned long coeff_list[4][128];
typedef unsigned long parity_list[4][32];

typedef char code_type[41];
typedef code_type code_list[128];

const code_list code = {
  // List of coefficients and micro code
  // bit encoded, LSB last in list, spaces ignored
  //
  //        history file address
  //        *     load input/output
  //        *     *history write enable
  //        *     * accumulator reset
  //        *     *  accumulator load
  //        *     *    multiplier shift: 00-no shift; 01-17b shift; 1X-34b shift
  //        *     *    * input selection: 00-input,lsb; 01-input,msb; 1X-history
  //        *     *    *    coefficients
  //        *     *    *    *                       iteration: value
  "00000 00000 0000 0000 000000000000000000" , // code   0: g, msb
  "00000 00000 0010 0001 000000000000000000" , // code   1: g, lsb
  "00000 10000 0000 0101 000000000000000000" , // code   2: g, msb

  "00000 10000 0000 0110 000000000000000000" , // code   3: b20, lsb
  "00000 10001 0000 1010 000000000000000000" , // code   4: b20, msb
  "00000 10001 0000 0010 000000000000000000" , // code   5: b20, lsb
  "00000 00000 0100 0110 000000000000000000" , // code   6: b20, msb
  "00000 00000 0000 0110 000000000000000000" , // code   7: b10, lsb
  "00000 00001 0000 1010 000000000000000000" , // code   8: b10, msb
  "00000 00001 0000 0010 000000000000000000" , // code   9: b10, lsb
  "00000 00000 0000 0110 000000000000000000" , // code  10: b10, msb

  "00000 10010 0000 0110 000000000000000000" , // code  11: -1b(c00)
```

```
    "00000 10010 0000 1010 000000000000000000", // code  12: a20, lsb
    "00000 10011 0000 0010 000000000000000000", // code  13: a20, msb
    "00000 10011 0001 0010 000000000000000000", // code  14: a20, lsb
    "00000 00010 0000 0110 000000000000000000", // code  15: a20, msb
    "00000 00010 0000 0110 000000000000000000", // code  16: a10, lsb
    "00000 00011 0000 1010 000000000000000000", // code  17: a10, msb
    "00000 00011 0000 0010 000000000000000000", // code  18: a10, lsb
    "00000 10010 0000 0110 000000000000000000", // code  19: a10, msb

    "00000 10010 0000 0110 000000000000000000", // code  20: b21, lsb
    "00000 10011 0000 1010 000000000000000000", // code  21: b21, msb
    "00000 10011 0000 0010 000000000000000000", // code  22: b21, lsb
    "00000 00010 0100 0110 000000000000000000", // code  23: b21, msb
    "00000 00010 0000 0110 000000000000000000", // code  24: b11, lsb
    "00000 00011 0000 1010 000000000000000000", // code  25: b11, msb
    "00000 00011 0000 0010 000000000000000000", // code  26: b11, lsb
    "00000 00000 0000 0110 000000000000000000", // code  27: b11, msb

    "00000 10100 0000 0110 000000000000000000", // code  28: -lb(c01)
    "00000 10100 0000 1010 000000000000000000", // code  29: a21, lsb
    "00000 10101 0000 0010 000000000000000000", // code  30: a21, msb
    "00000 10101 0001 0010 000000000000000000", // code  31: a21, lsb
    "00000 00100 0000 0110 000000000000000000", // code  32: a21, msb
    "00000 00100 0000 0110 000000000000000000", // code  33: a11, lsb
    "00000 00101 0000 1010 000000000000000000", // code  34: a11, msb
    "00000 00101 0000 0010 000000000000000000", // code  35: a11, lsb
    "00000 10100 0000 0110 000000000000000000", // code  36: a11, msb

    "00000 10100 0000 0110 000000000000000000", // code  37: b22, lsb
    "00000 10101 0000 1010 000000000000000000", // code  38: b22, msb
    "00000 10101 0000 0010 000000000000000000", // code  39: b22, lsb
    "00000 00100 0100 0110 000000000000000000", // code  40: b22, msb
    "00000 00100 0000 0110 000000000000000000", // code  41: b12, lsb
    "00000 00101 0000 1010 000000000000000000", // code  42: b12, msb
    "00000 00101 0000 0010 000000000000000000", // code  43: b12, lsb
    "00000 00000 0000 0110 000000000000000000", // code  44: b12, msb

    "00000 10110 0000 0110 000000000000000000", // code  45: -lb(c02)
    "00000 10110 0000 1010 000000000000000000", // code  46: a22, lsb
    "00000 10111 0000 0010 000000000000000000", // code  47: a22, msb
    "00000 10111 0001 0010 000000000000000000", // code  48: a22, lsb
    "00000 00110 0000 0110 000000000000000000", // code  49: a22, msb
    "00000 00110 0000 0110 000000000000000000", // code  50: a12, lsb
    "00000 00111 0000 1010 000000000000000000", // code  51: a12, msb
    "00000 00111 0000 0010 000000000000000000", // code  52: a12, lsb
    "00000 10110 0000 0110 000000000000000000", // code  53: a12, msb

    "00000 10110 0000 0110 000000000000000000", // code  54: b23, lsb
    "00000 10111 0000 1010 000000000000000000", // code  55: b23, msb
    "00000 10111 0000 0010 000000000000000000", // code  56: b23, lsb
    "00000 00110 0100 0110 000000000000000000", // code  57: b23, msb
    "00000 00110 0000 0110 000000000000000000", // code  58: b13, lsb
    "00000 00111 0000 1010 000000000000000000", // code  59: b13, msb
    "00000 00111 0000 0010 000000000000000000", // code  60: b13, lsb
```

```
    "00000 00000 0000 0110 000000000000000000", // code  61: b13, msb

    "00000 11000 0000 0110 000000000000000000", // code  62: -lb(c03)
    "00000 11000 0000 1010 000000000000000000", // code  63: a23, lsb
    "00000 11001 0000 0010 000000000000000000", // code  64: a23, msb
    "00000 11001 0001 0010 000000000000000000", // code  65: a23, lsb
    "00000 01000 0000 0110 000000000000000000", // code  66: a23, msb
    "00000 01000 0000 0110 000000000000000000", // code  67: a13, lsb
    "00000 01001 0000 1010 000000000000000000", // code  68: a13, msb
    "00000 01001 0000 0010 000000000000000000", // code  69: a13, lsb
    "00000 11000 0000 0110 000000000000000000", // code  70: a13, msb

    "00000 11000 0000 0110 000000000000000000", // code  71: b24, lsb
    "00000 11001 0000 1010 000000000000000000", // code  72: b24, msb
    "00000 11001 0000 0010 000000000000000000", // code  73: b24, lsb
    "00000 01000 0100 0110 000000000000000000", // code  74: b24, msb
    "00000 01000 0000 0110 000000000000000000", // code  75: b14, lsb
    "00000 01001 0000 1010 000000000000000000", // code  76: b14, msb
    "00000 01001 0000 0010 000000000000000000", // code  77: b14, lsb
    "00000 00000 0000 0110 000000000000000000", // code  78: b14, msb

    "00000 11010 0000 0110 000000000000000000", // code  79: -lb(c04)
    "00000 11010 0000 1010 000000000000000000", // code  80: a24, lsb
    "00000 11011 0000 0010 000000000000000000", // code  81: a24, msb
    "00000 11011 0001 0010 000000000000000000", // code  82: a24, lsb
    "00000 01010 0000 0110 000000000000000000", // code  83: a24, msb
    "00000 01010 0000 0110 000000000000000000", // code  84: a14, lsb
    "00000 01011 0000 1010 000000000000000000", // code  85: a14, msb
    "00000 01011 0000 0010 000000000000000000", // code  86: a14, lsb
    "00000 11010 0000 0110 000000000000000000", // code  87: a14, msb

    "00000 11010 0000 0110 000000000000000000", // code  88: b25, lsb
    "00000 11011 0000 1010 000000000000000000", // code  89: b25, msb
    "00000 11011 0000 0010 000000000000000000", // code  90: b25, lsb
    "00000 01010 0100 0110 000000000000000000", // code  91: b25, msb
    "00000 01010 0000 0110 000000000000000000", // code  92: b15, lsb
    "00000 01011 0000 1010 000000000000000000", // code  93: b15, msb
    "00000 01011 0000 0010 000000000000000000", // code  94: b15, lsb
    "00000 00000 0000 0110 000000000000000000", // code  95: b15, msb

    "00000 11100 0000 0110 000000000000000000", // code  96: -lb(c05)
    "00000 11100 0000 1010 000000000000000000", // code  97: a25, lsb
    "00000 11101 0000 0010 000000000000000000", // code  98: a25, msb
    "00000 11101 0001 0010 000000000000000000", // code  99: a25, lsb
    "00000 01100 0000 0110 000000000000000000", // code 100: a25, msb
    "00000 01100 0000 0110 000000000000000000", // code 101: a15, lsb
    "00000 01101 0000 1010 000000000000000000", // code 102: a15, msb
    "00000 01101 0000 0010 000000000000000000", // code 103: a15, lsb
    "00000 11100 0000 0110 000000000000000000", // code 104: a15, msb

    "00000 11100 0000 0110 000000000000000000", // code 105: b26, lsb
    "00000 11101 0000 1010 000000000000000000", // code 106: b26, msb
    "00000 11101 0000 0010 000000000000000000", // code 107: b26, lsb
    "00000 01100 0100 0110 000000000000000000", // code 108: b26, msb
```

```
    "00000 01100 0000 0110 000000000000000000" , // code 109: b16, lsb
    "00000 01101 0000 1010 000000000000000000" , // code 110: b16, msb
    "00000 01101 0000 0010 000000000000000000" , // code 111: b16, lsb
    "00000 00000 0000 0110 000000000000000000" , // code 112: b16, msb

    "00000 11110 0000 0110 000000000000000000" , // code 113: -lb(c06)
    "00000 11110 0000 1010 000000000000000000" , // code 114: a26, lsb
    "00000 11111 0000 0010 000000000000000000" , // code 115: a26, msb
    "00000 11111 0001 0010 000000000000000000" , // code 116: a26, lsb
    "00000 01110 0000 0110 000000000000000000" , // code 117: a26, msb
    "00000 01110 0000 0110 000000000000000000" , // code 118: a16, lsb
    "00000 01111 0000 1010 000000000000000000" , // code 119: a16, msb
    "00000 01111 0000 0010 000000000000000000" , // code 120: a16, lsb
    "00000 11110 0000 0110 000000000000000000" , // code 121: a16, msb

    "00000 11110 0000 0110 000000000000000000" , // code 122: --
    "00000 11111 0000 1010 000000000000000000" , // code 123: --
    "00000 11111 0000 0010 000000000000000000" , // code 124: --
    "00000 01110 1100 0000 000000000000000000" , // code 125: --
    "00000 01110 0000 0000 000000000000000000" , // code 126: --
    "00000 00000 0000 0000 000000000000000000"   // code 127: g,lsb
    };


unsigned long convert_coeff (double a, const char* p = "lsb")
    {
       const int q = 1.0;
       unsigned long long c = 0;
       bool lsb = strcmp (p, "lsb") == 0;

       double aa = fabs (a);
    // too large?
       if (aa > 2.0*q) {
          c = 0x3FFFFFFFFULL;
       }
       else {
       // convert bit by bit starting at the MSB
          aa += exp (-log(2.0)*34); // for correct rounding
       //int n = lsb ? 34 : 17;
          int n = 34;
          for (int i = n-1; i >= 0; --i) {
             if (aa >= q) {
                c |= 1ULL << i;
                aa -= q;
             }
             aa *= 2.0;
          }
       }
    // form negative if necessary
       if (a < 0.0) {
          c = -c;
       }
```

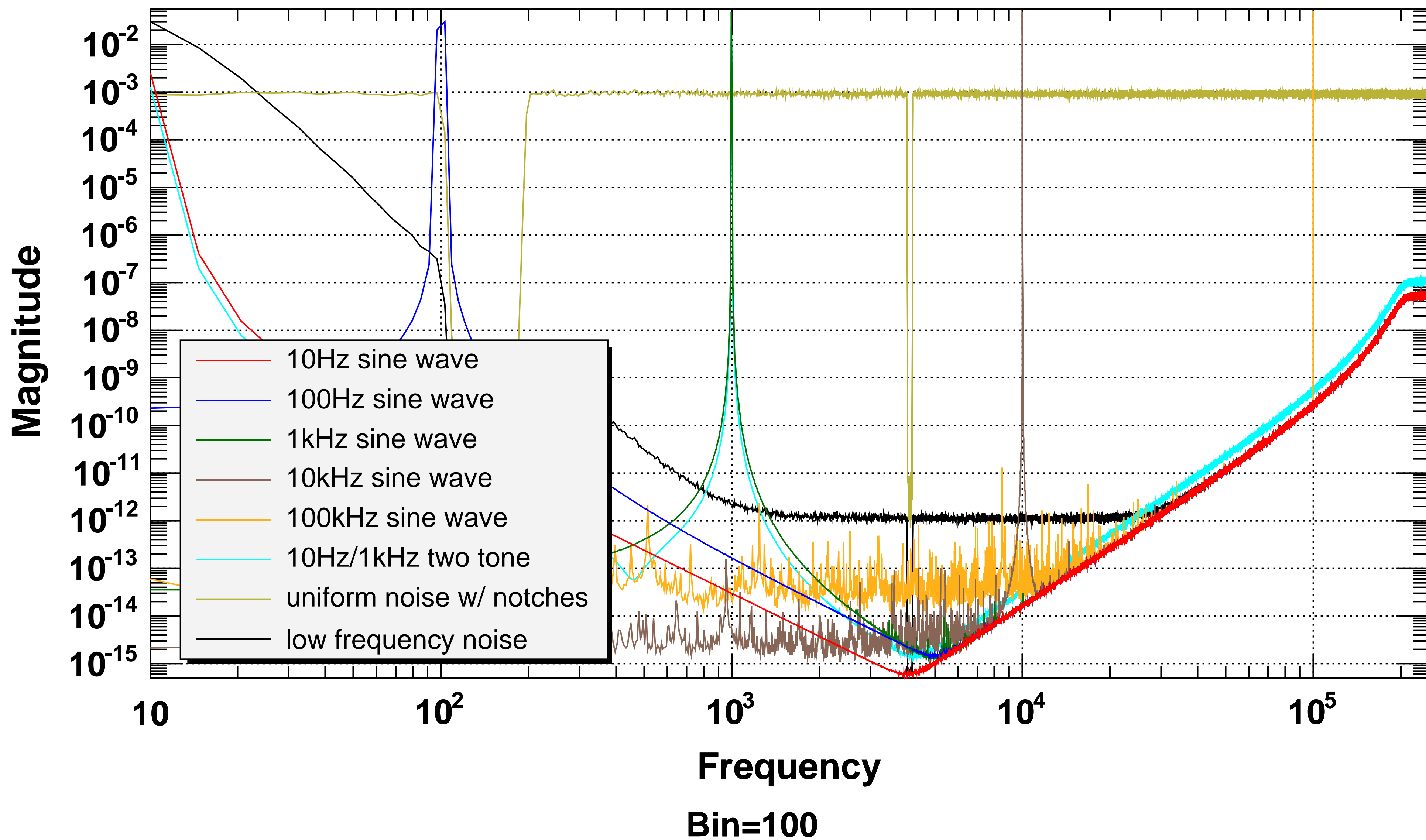
```
    // mask higher order bits
    if (lsb) {
        c &= 0x1FFFFULL; // get last 17 bits
    }
    else {
        unsigned long long mask = 0x3FFFFULL;
        c = (c & (mask << 17)) >> 17; // get bits 18 thru 35
    }
    return c;
}

unsigned long convert_shiftadjust (double g)
{
    double gain = fabs (g);
    int n = (1 << 3) - 1;
    for (int i = -(1 << 3) + 1; i <= 0; ++i) {
        if (gain > exp (log(2.0) * i)) {
            n = -i;
        }
        else {
            break;
        }
    }
    return n & 0xF;
}

unsigned long convert_code (const char* code)
{
    unsigned long c = 0;
    for (const char* p = code; *p; ++p) {
        if (!isspace (*p)) {
            c <<= 1;
            c |= (*p == '1') ? 1 : 0;
        }
    }
    return c & 0xFFFC0000;
}


int main(int argc, char **argv)
{

    if (argc <= 1) {
        cout << "Usage: coeffgen 'filterfile' {'filter'}" << endl;
        return 1;
    }
    FilterFile ff;
    const FilterModule* mod = 0;
```

```
if (!ff.read (argv[1])) {
   cerr << "Unable to open filter file " << argv[1] << endl;
   return 1;
}

coeff_list coeffs;
memset (coeffs, 0, sizeof (coeff_list));
parity_list parity;
memset (parity, 0, sizeof (parity_list));

for (int fil = 0; fil < 4; ++fil) {
// get coefficients
   if ((argc > fil + 2) && (mod = ff.find (argv[fil+2]))) {
      double fS = mod->getFSample();
      if (!(*mod)[0].valid()) {
         cerr << "Filter not valid at section 0 of " <<
            argv[fil+2] << endl;
      }
      else {
         cout << "Use filter " << (*mod)[0].getName() << " of " <<
            argv[fil+2] << endl;
         bool err = false;
         const FilterDesign& ds = (*mod)[0].filter();

         int order = iirorder (ds.get());
         if (order < 0) {
            cerr << "Not an IIR filter" << endl;
            err = true;
            order = 0;
         }
         int nba = 1;
         double* coeff = new double [1 + 4*order];
         if (!err) {
            if (!iir2z (ds.get(), nba, coeff, "o")) {
               cerr << "Unable to obtain online filter coefficients" <<
                  endl;
               err = true;
            }
         }
         int sosnum = (nba - 1) / 4;
         if (!err) {
            if ((sosnum < 1) || (sosnum > kMaxSOS)) {
               cerr << "Invalid number of SOSs " << sosnum << endl;
               err = true;
            }
         }
         if (!err) {
            cout << "Filter " << fil << ": Creating " <<
               (*mod)[0].getName() << " of " << argv[fil+2] << endl;
            for (int j = 0; j < sosnum; ++j) {
               coeffs[fil][3+j*17] = coeffs[fil][5+j*17] =
                  convert_coeff ( coeff[4+j*4], "lsb");
               coeffs[fil][4+j*17] = coeffs[fil][6+j*17] =
```

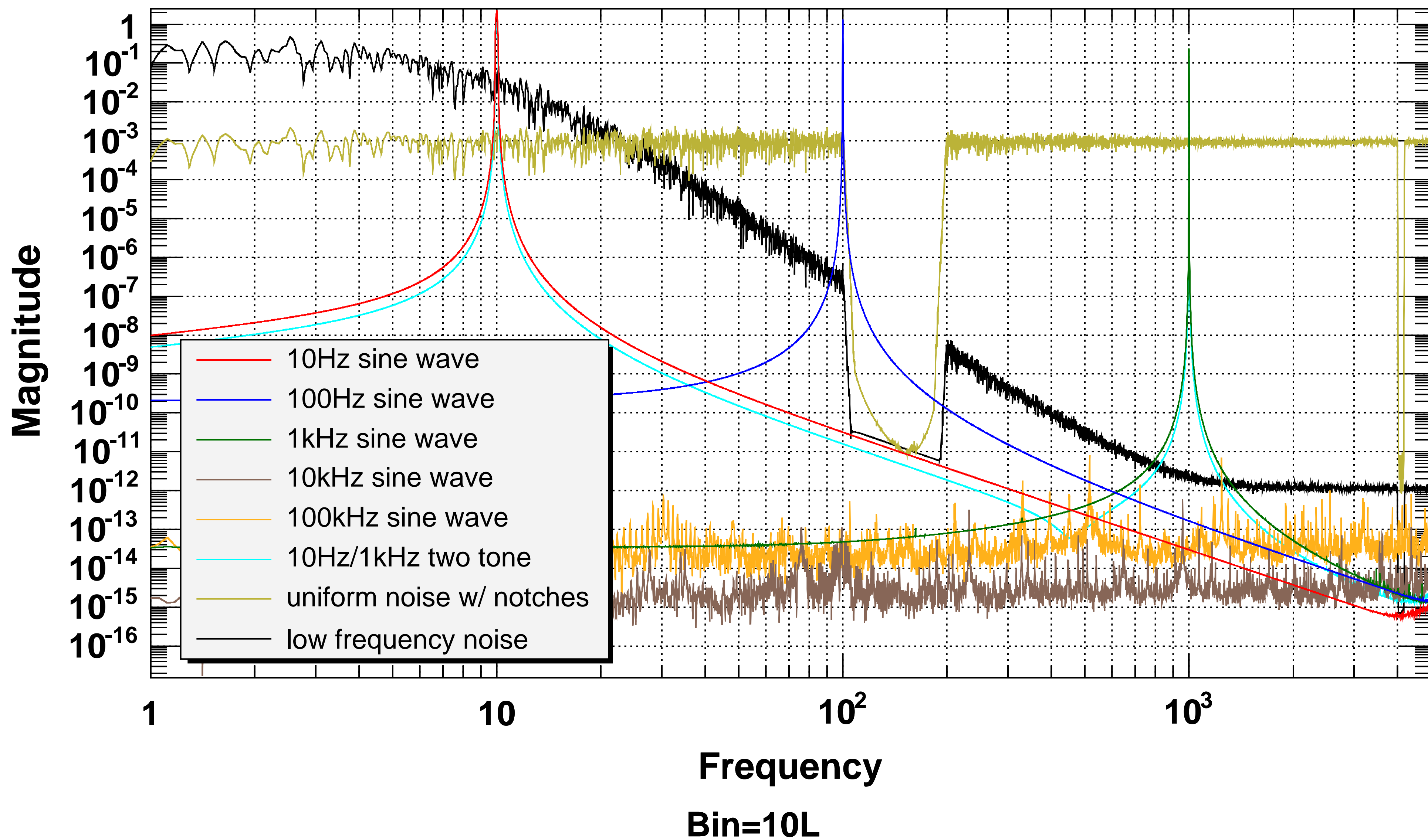
```
                convert_coeff ( coeff[4+j*4], "msb");
            coeffs[fil][7+j*17] = coeffs[fil][9+j*17] =
                convert_coeff ( coeff[3+j*4], "lsb");
            coeffs[fil][8+j*17] = coeffs[fil][10+j*17]=
                convert_coeff ( coeff[3+j*4], "msb");
            coeffs[fil][12+j*17]= coeffs[fil][14+j*17]=
                convert_coeff (-coeff[2+j*4], "lsb");
            coeffs[fil][13+j*17]= coeffs[fil][15+j*17]=
                convert_coeff (-coeff[2+j*4], "msb");
            coeffs[fil][16+j*17]= coeffs[fil][18+j*17]=
                convert_coeff (-coeff[1+j*4], "lsb");
            coeffs[fil][17+j*17]= coeffs[fil][19+j*17]=
                convert_coeff (-coeff[1+j*4], "msb");
        }
        double gain = 1.0;
        double gainerr = 1.0;
        for (int j = sosnum - 1; j >= 0; --j) {
            double dcgain = (1.0+coeff[3+j*4]+coeff[4+j*4])/
                (1.0+coeff[1+j*4]+coeff[2+j*4]);
            coeffs[fil][11+j*17]= convert_shiftadjust (gainerr/dcgain);
            double shiftgain = exp (log(2.0) * coeffs[fil][11+j*17]);
            gain = gain / shiftgain ;
            gainerr = gainerr / dcgain * shiftgain;
            cout <<
                "  Section " << j << ": dc gain is " << 1.0/dcgain <<
                "  shift adj. is <<" << coeffs[fil][11+j*17] <<
                "  cumm. gain err is " << gainerr << endl;
        }
        gain = coeff[0] / gain;
        cout << "  Remaining filter gain is " << setprecision(12) <<
            gain << setprecision(6) << endl;
        if (fabs (gain) >= 2.0) {
            cerr << "Gain out of range " << gain <<
                " expected <2.0   TRUNCATED!" << endl;
        }
        coeffs[fil][127] = coeffs[fil][1] = convert_coeff (gain, "lsb");
        coeffs[fil][0]   = coeffs[fil][2] = convert_coeff (gain, "msb");
      }
      delete [] coeff;
    }
  }
  // fill in micro code
  for (int j = 0; j < 128; ++j) {
      coeffs[fil][j] |= convert_code (code[j]);
  //printf ("0x%08lx  ", coeffs[fil][j]);
  //if (j % 4 == 3) printf ("\n");
  }
 }
 cout << endl << endl;

// write memory initialization file (generic)
 for (int i = 0; i < 4*16; ++i) {
   cout << "        INIT_" << setfill('0') << setw(2) << right << hex << i;
   cout << " : bit_vector(255 downto 0) :=" << endl;
```

```
        cout << "               X\"";
        for (int j = 7; j >= 0; --j) {
            cout << setfill('0') << setw(8) << hex <<
                coeffs[i / 16][8*(i % 16)+j];
        }
        cout << "\";" << endl;
    }
    cout << endl << endl;

// write generic map
//   for (int i = 0; i < 4*16; ++i) {
//      cout << "        INIT_" << setfill('0') << setw(2) << right << hex << i;
//      cout << " => X\"";
//      for (int j = 7; j >= 0; --j) {
//         cout << setfill('0') << setw(8) << hex <<
//            coeffs[i / 16][8*(i % 16)+j];
//      }
//      cout << "\"," << endl;
//   }
//   cout << endl << endl;

// write memory initialization file (attributes)
    for (int i = 0; i < 4*16; ++i) {
        cout << "   attribute INIT_" << setfill('0') << setw(2) << right <<
            hex << i;
        cout << " : string;" << endl;
        cout << "   attribute INIT_" << setfill('0') << setw(2) << right <<
            hex << i;
        cout << " of U_RAM : label is" << endl;
        cout << "      \"";
        for (int j = 7; j >= 0; --j) {
            cout << setfill('0') << setw(8) << hex <<
                coeffs[i / 16][8*(i % 16)+j];
        }
        cout << "\";" << endl;
    }

    return 0;
}
```

APPENDIX B SPECTRA OF STIMULI

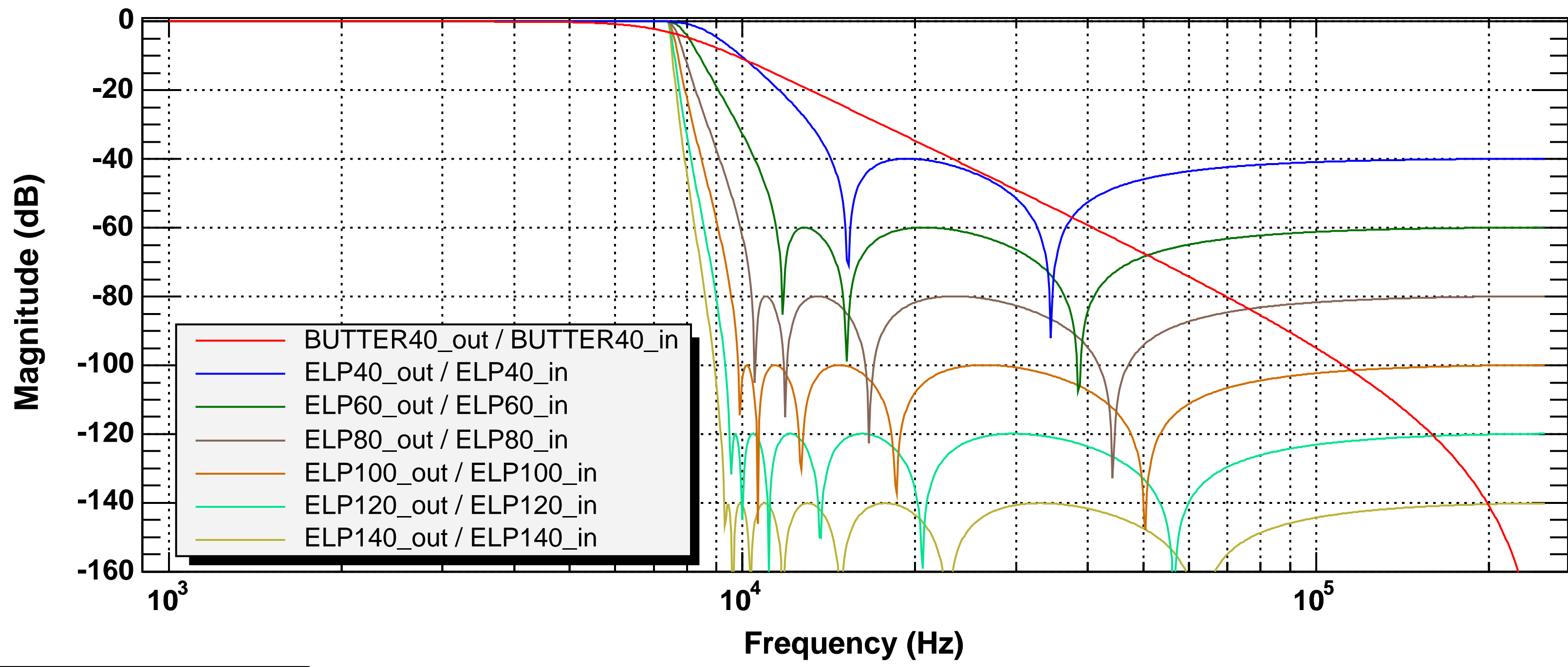


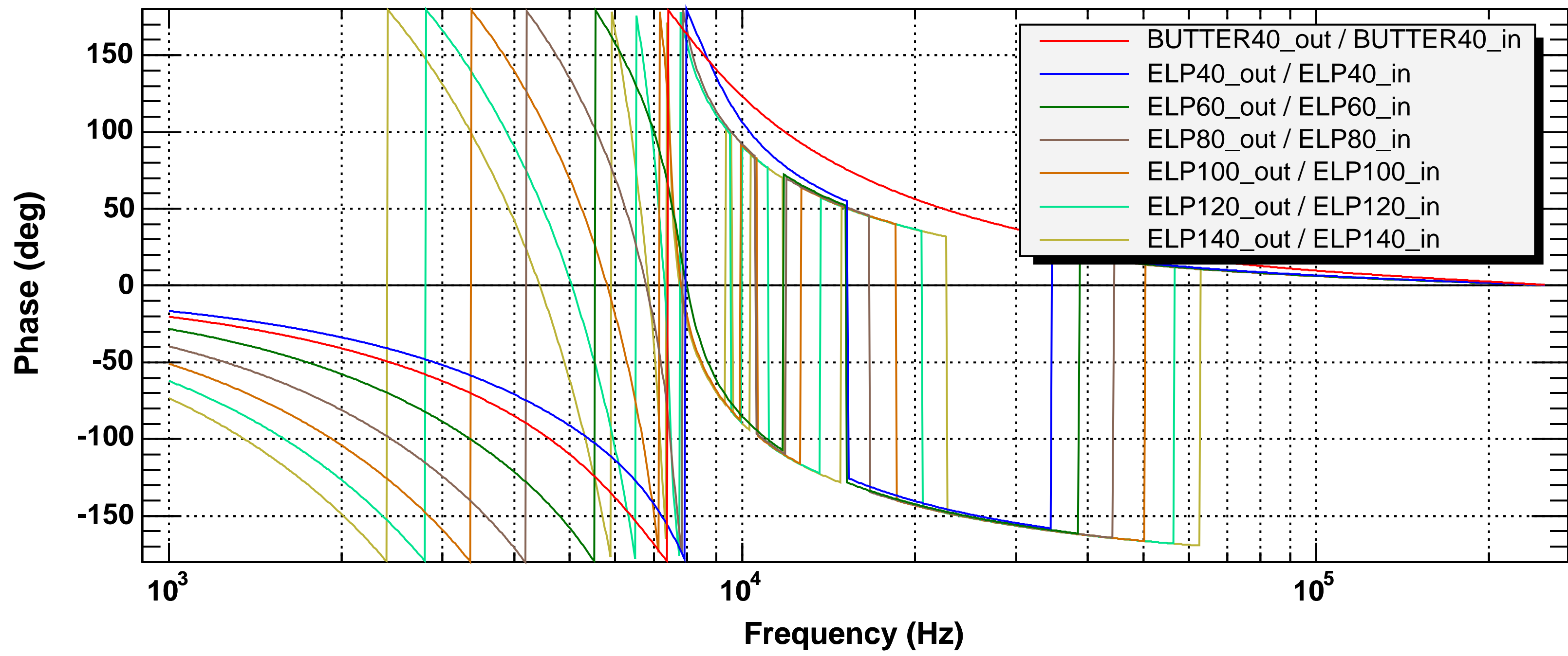
Stimulus Waveforms
Magnitude
Frequency
Bin=100
10Hz sine wave
100Hz sine wave
1kHz sine wave
10kHz sine wave
100kHz sine wave
10Hz/1kHz two tone
uniform noise w/ notches
low frequency noise



Stimulus Waveforms
Magnitude
Frequency
Bin=10L
10Hz sine wave
100Hz sine wave
1kHz sine wave
10kHz sine wave
100kHz sine wave
10Hz/1kHz two tone
uniform noise w/ notches
low frequency noise

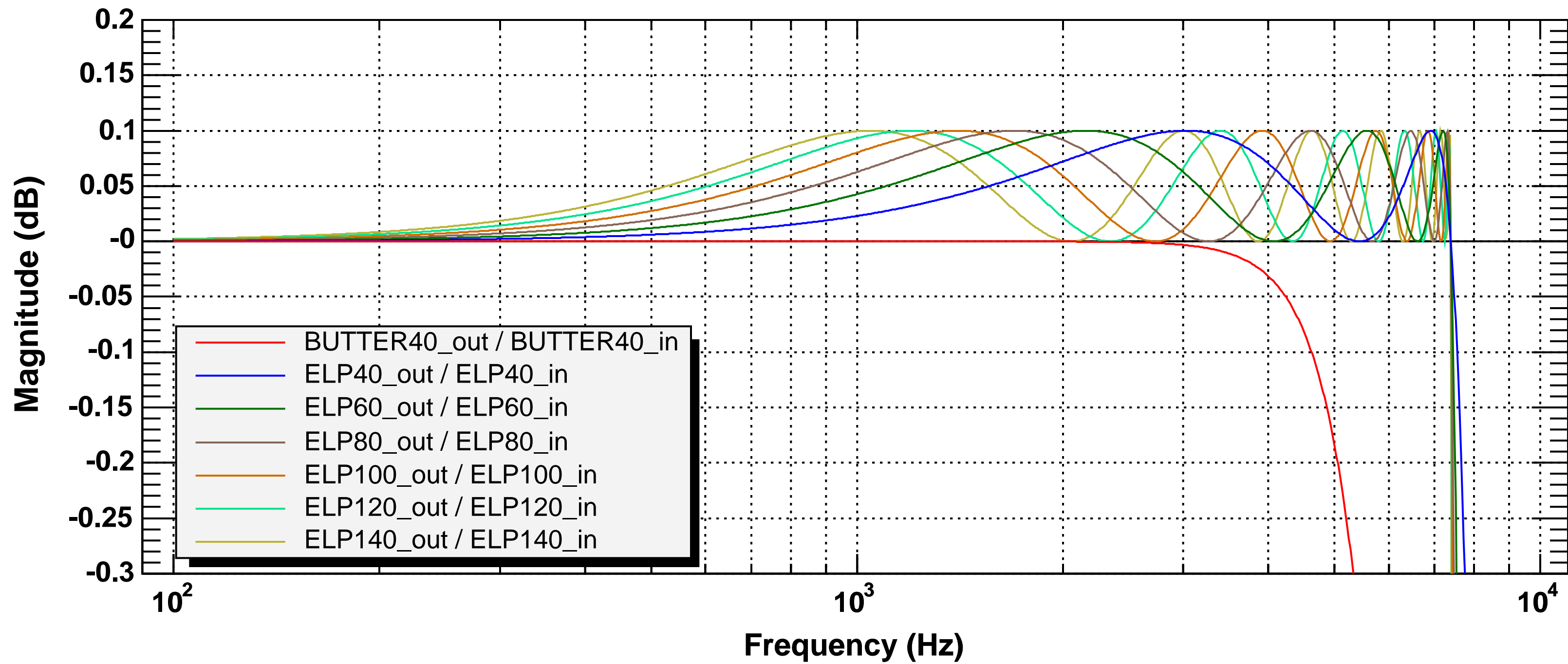
APPENDIX C FILTER TRANSFER FUNCTIONS

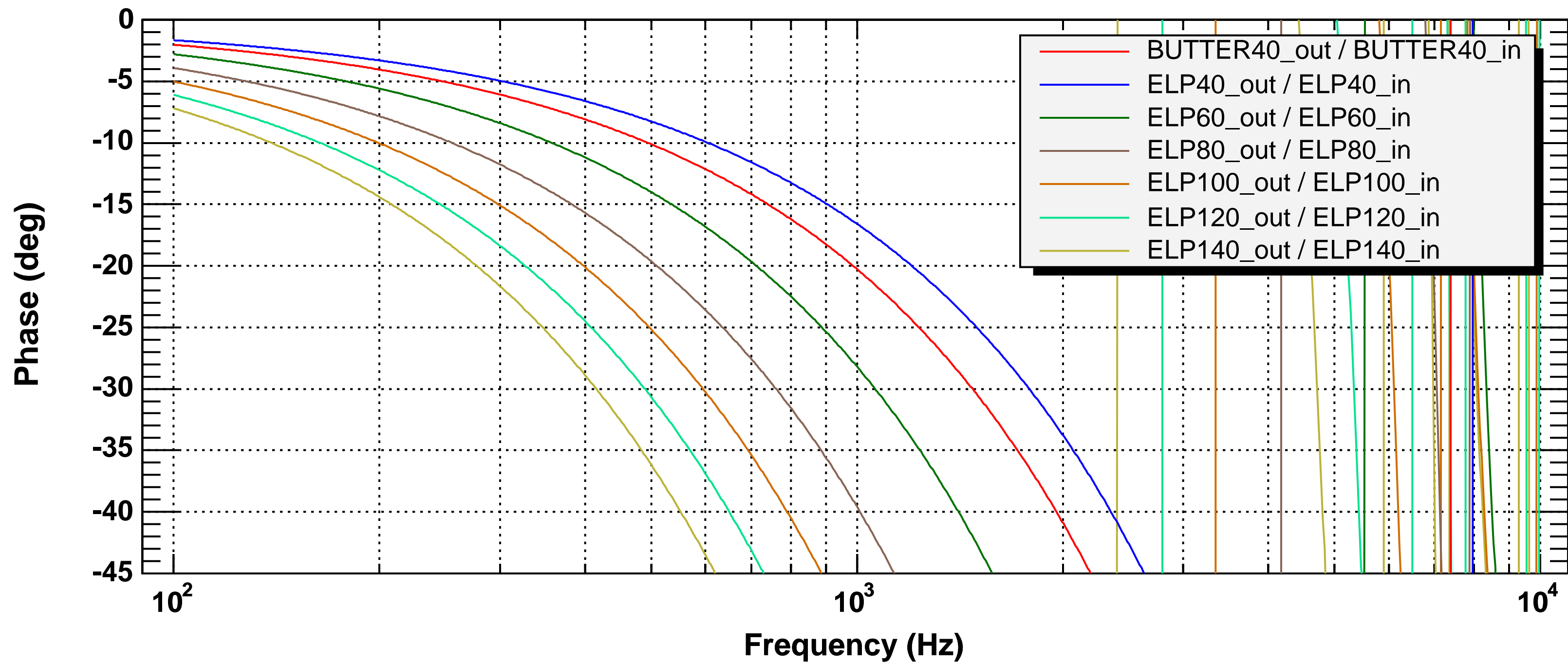




*T0=07/04/2005 19:03:56

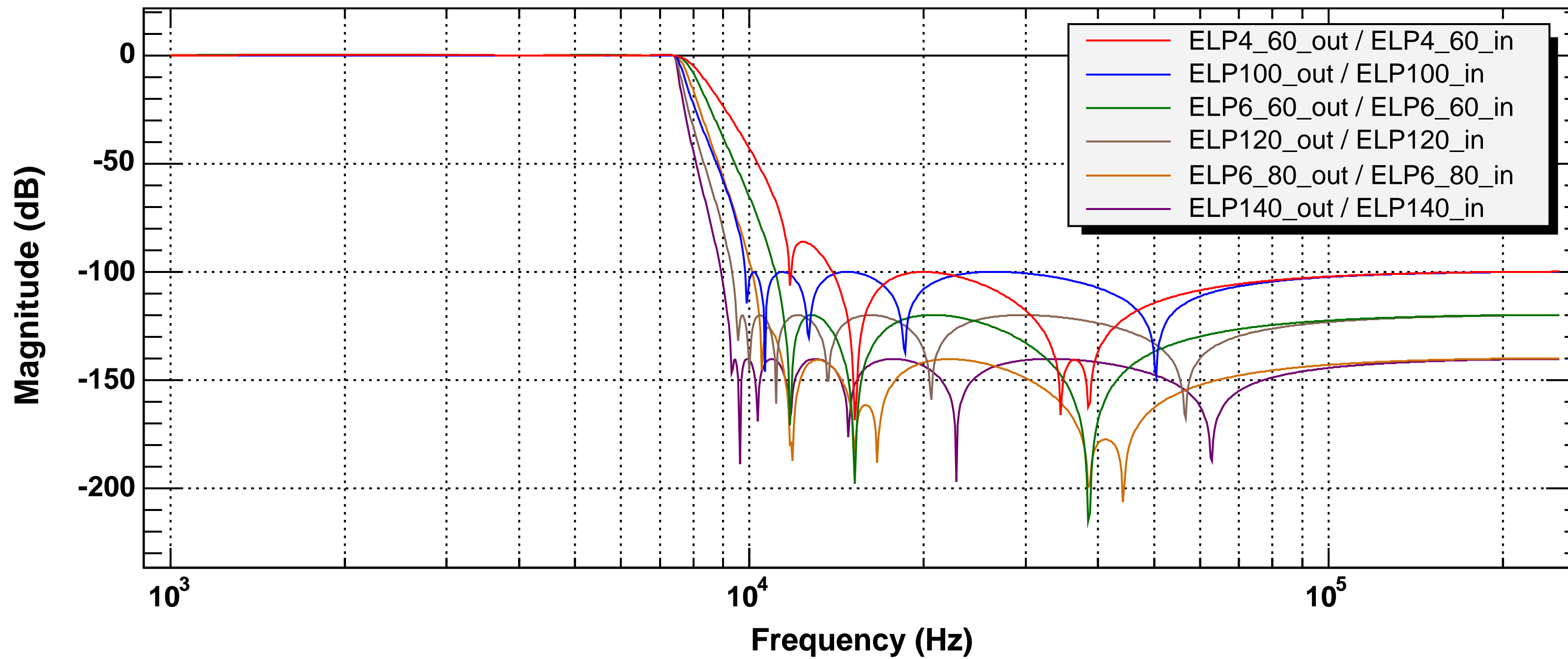
iir_coeff.txt

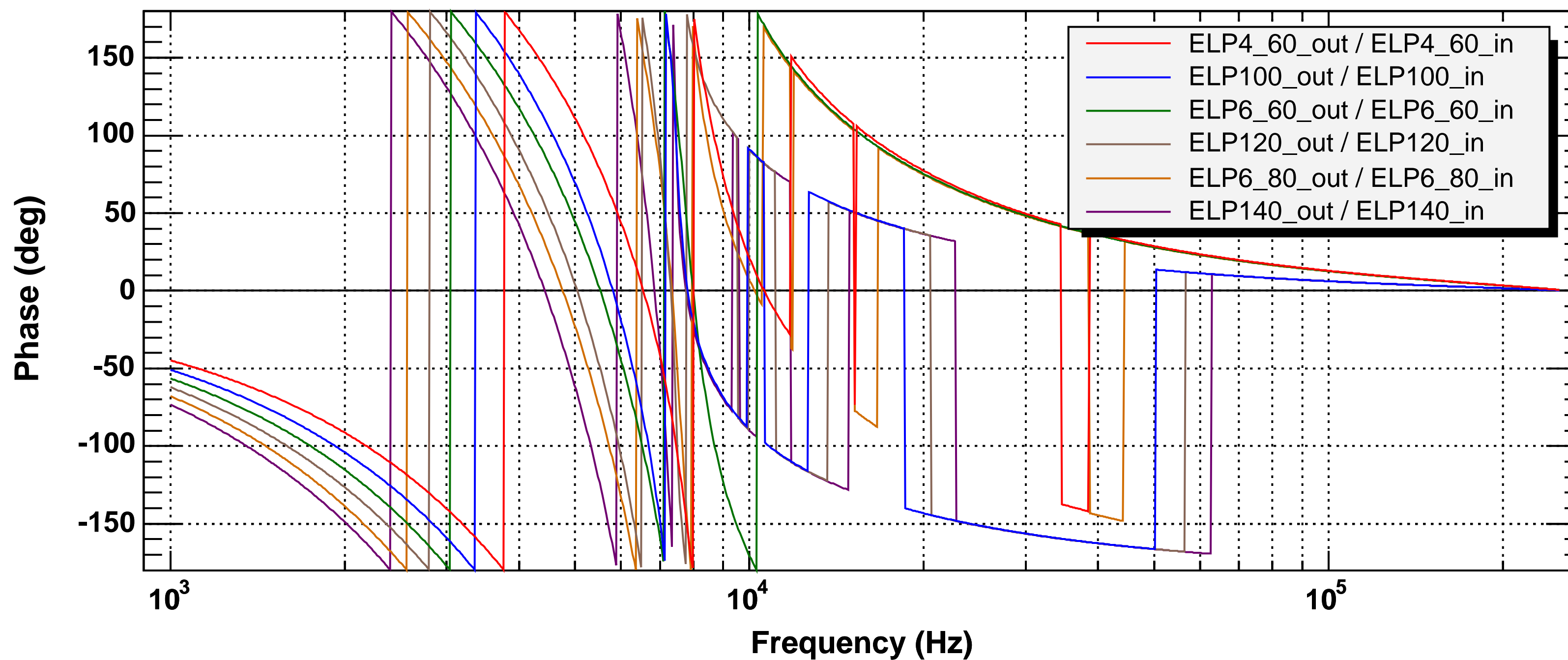




*T0=07/04/2005 19:09:47

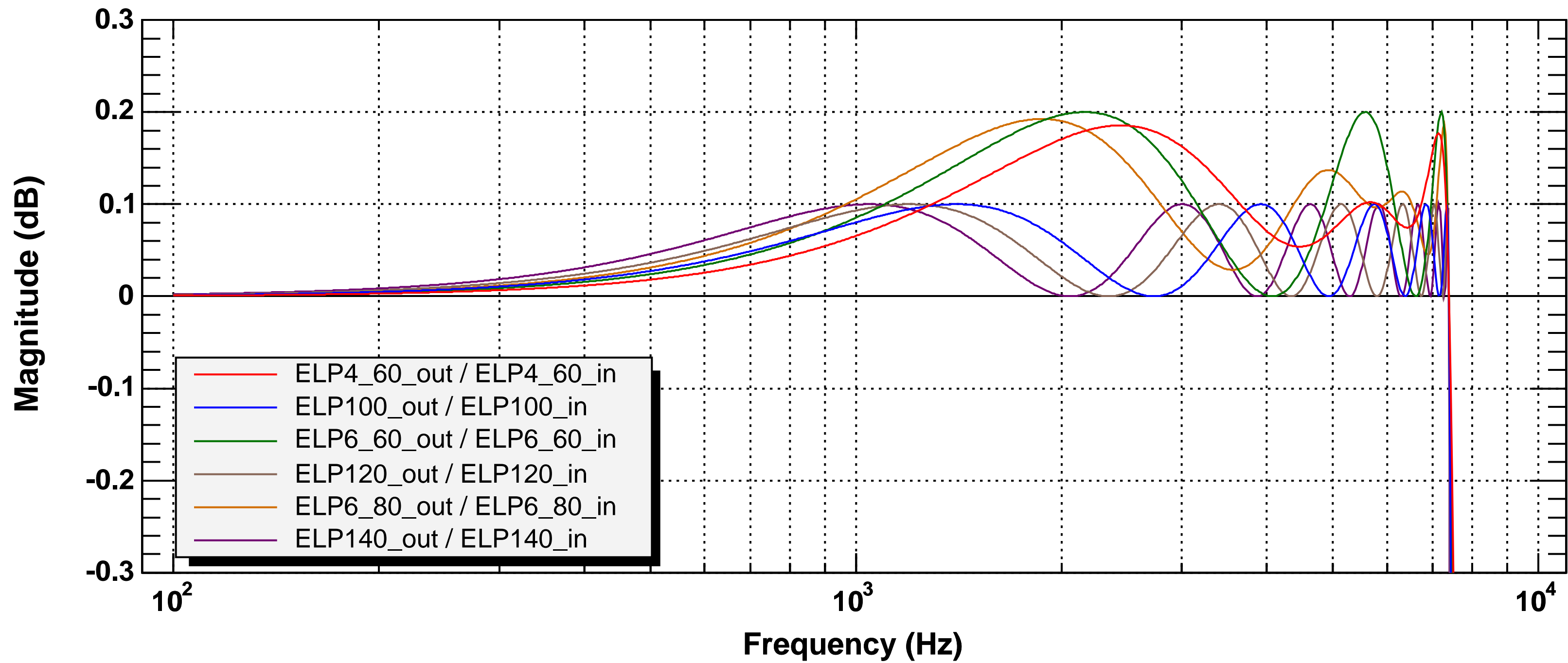
iir_coeff.txt

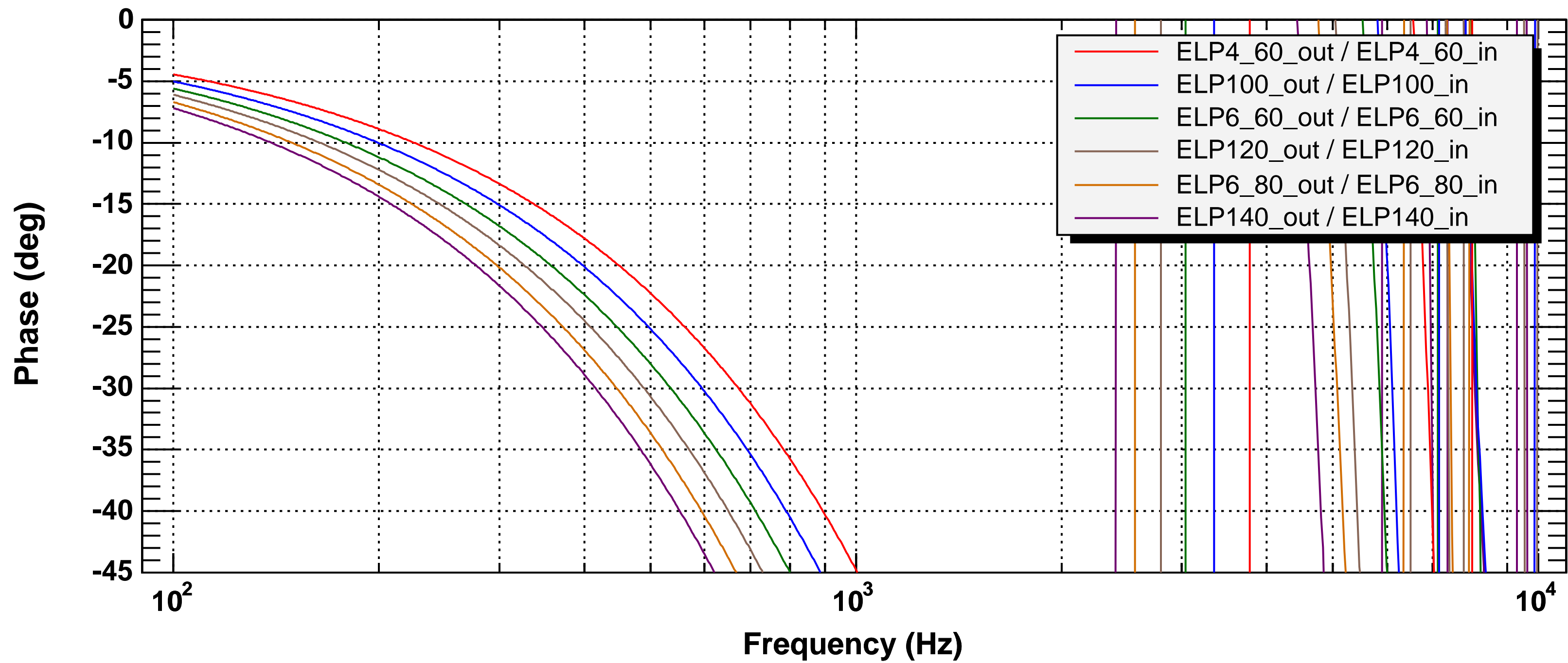




*T0=08/04/2005 16:29:19

iir_coeff.txt

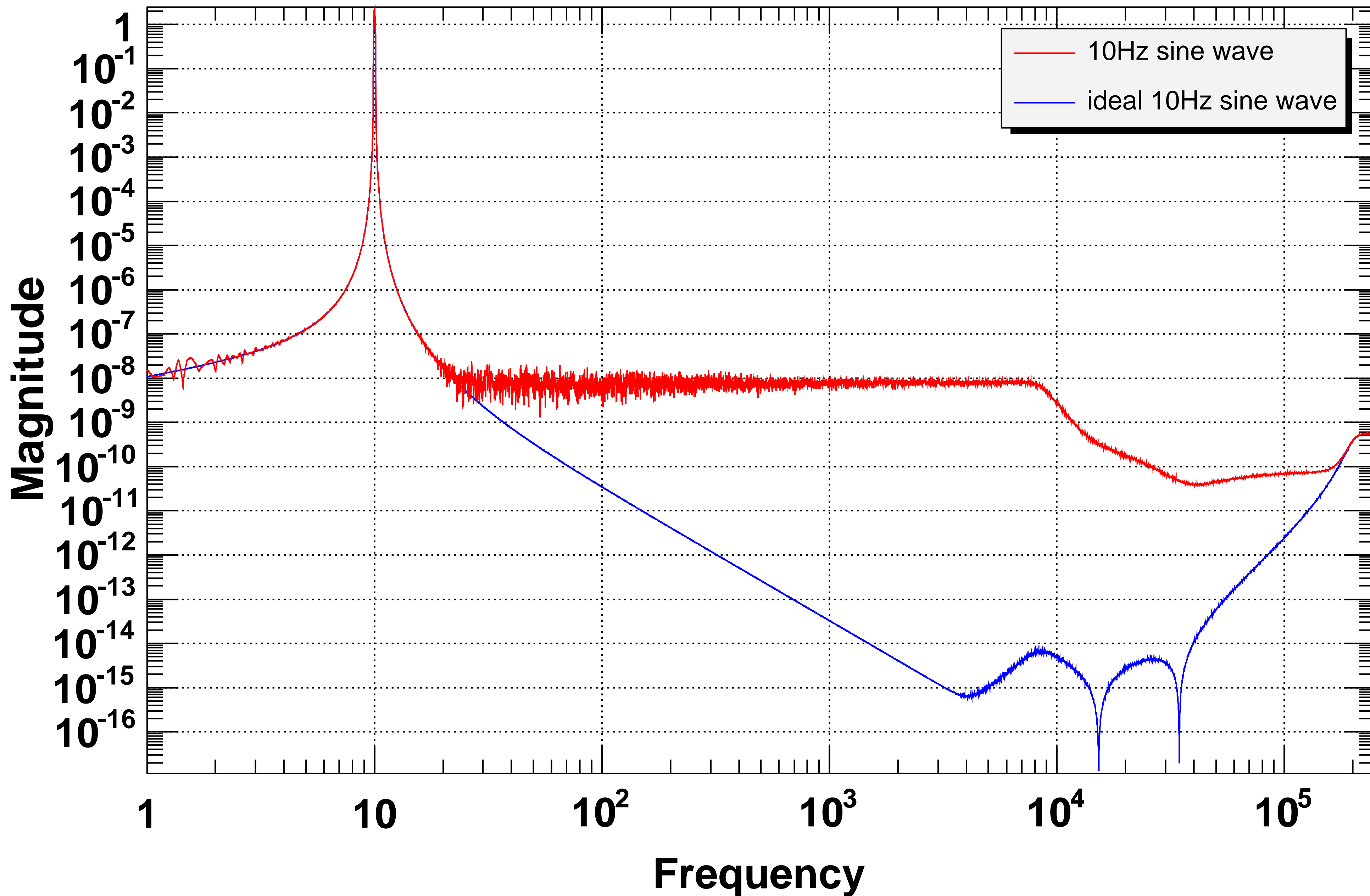




*T0=08/04/2005 16:37:36

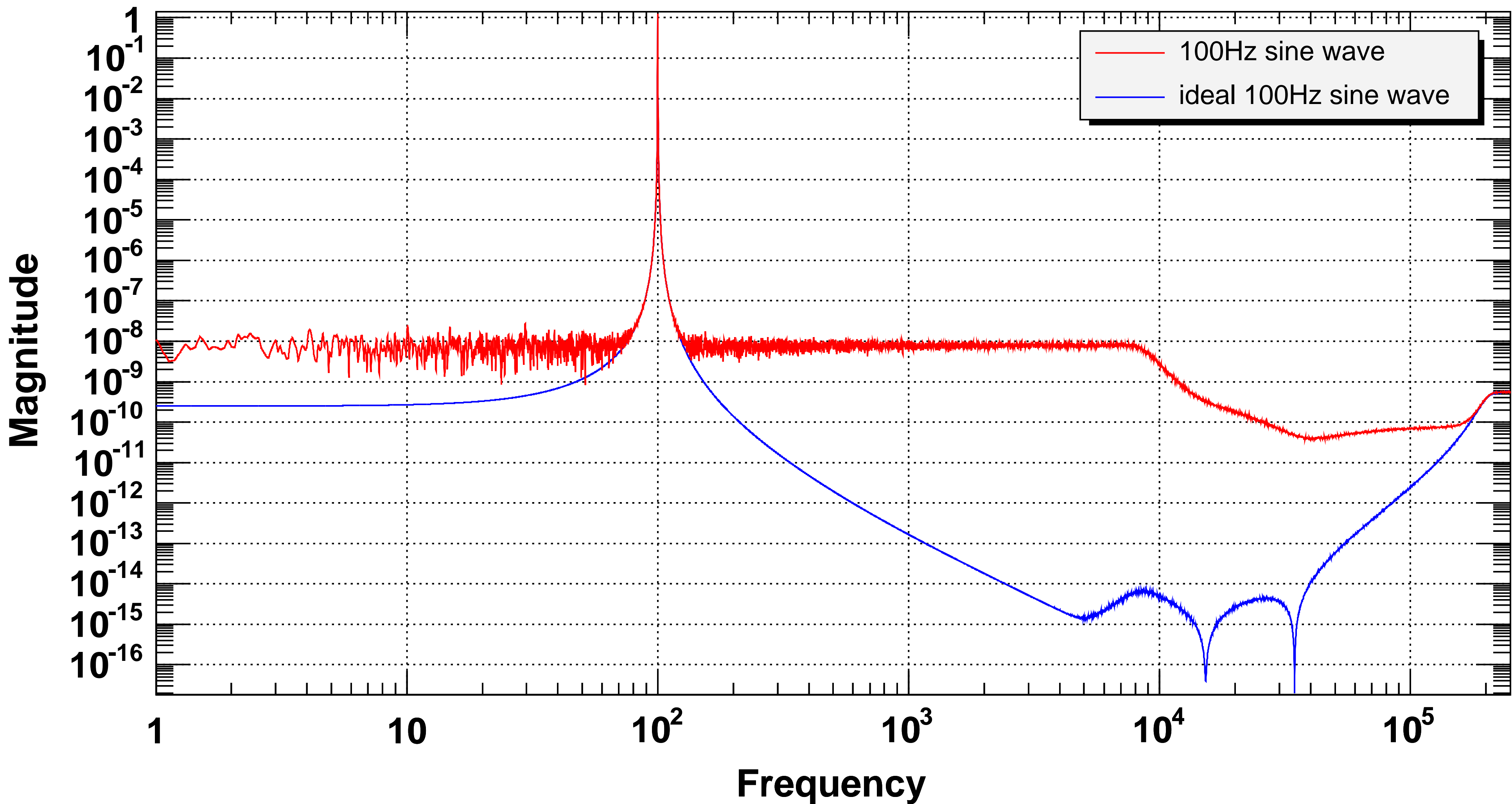
iir_coeff.txt

APPENDIX D SPECTRA OF DIFFERENT STIMULI

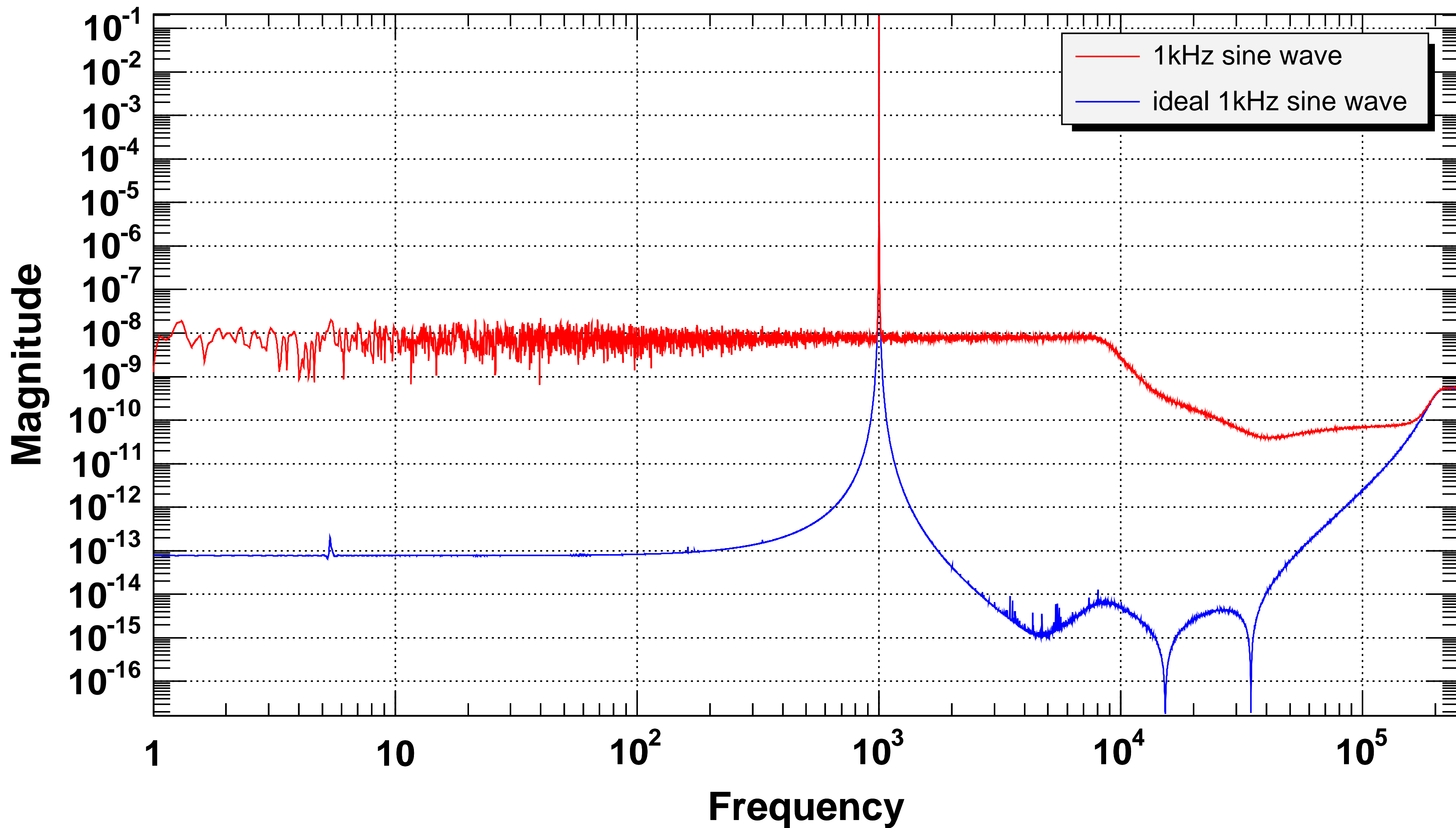


Power spectrum
10Hz sine wave
ideal 10Hz sine wave
Magnitude
Frequency
T0=06/01/1980 00:00:00
Bin=419L

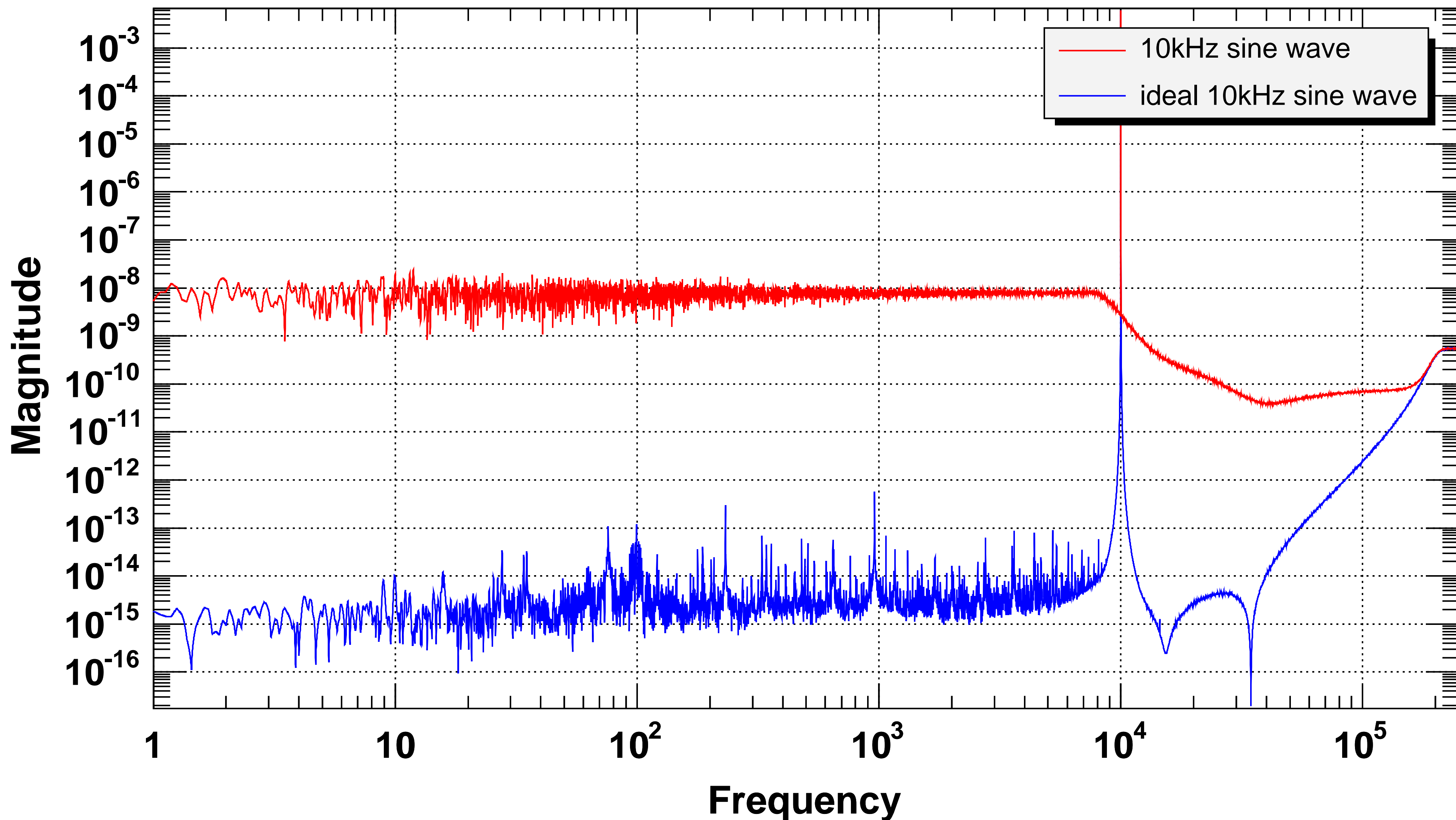
Power spectrum



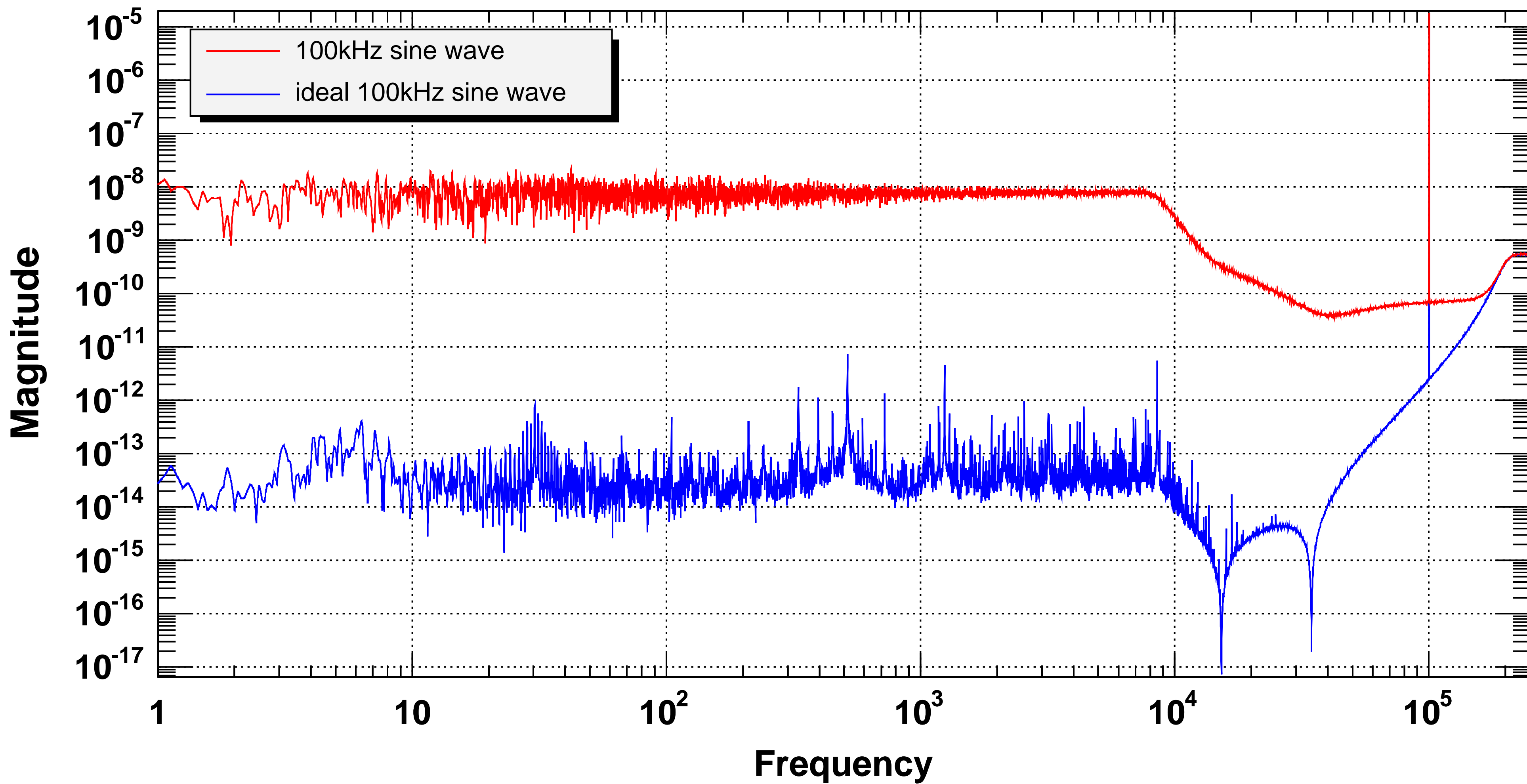
T0=06/01/1980 00:00:00 Bin=419L


Power spectrum
1kHz sine wave
ideal 1kHz sine wave
Magnitude
Frequency
T0=06/01/1980 00:00:00
Bin=419L

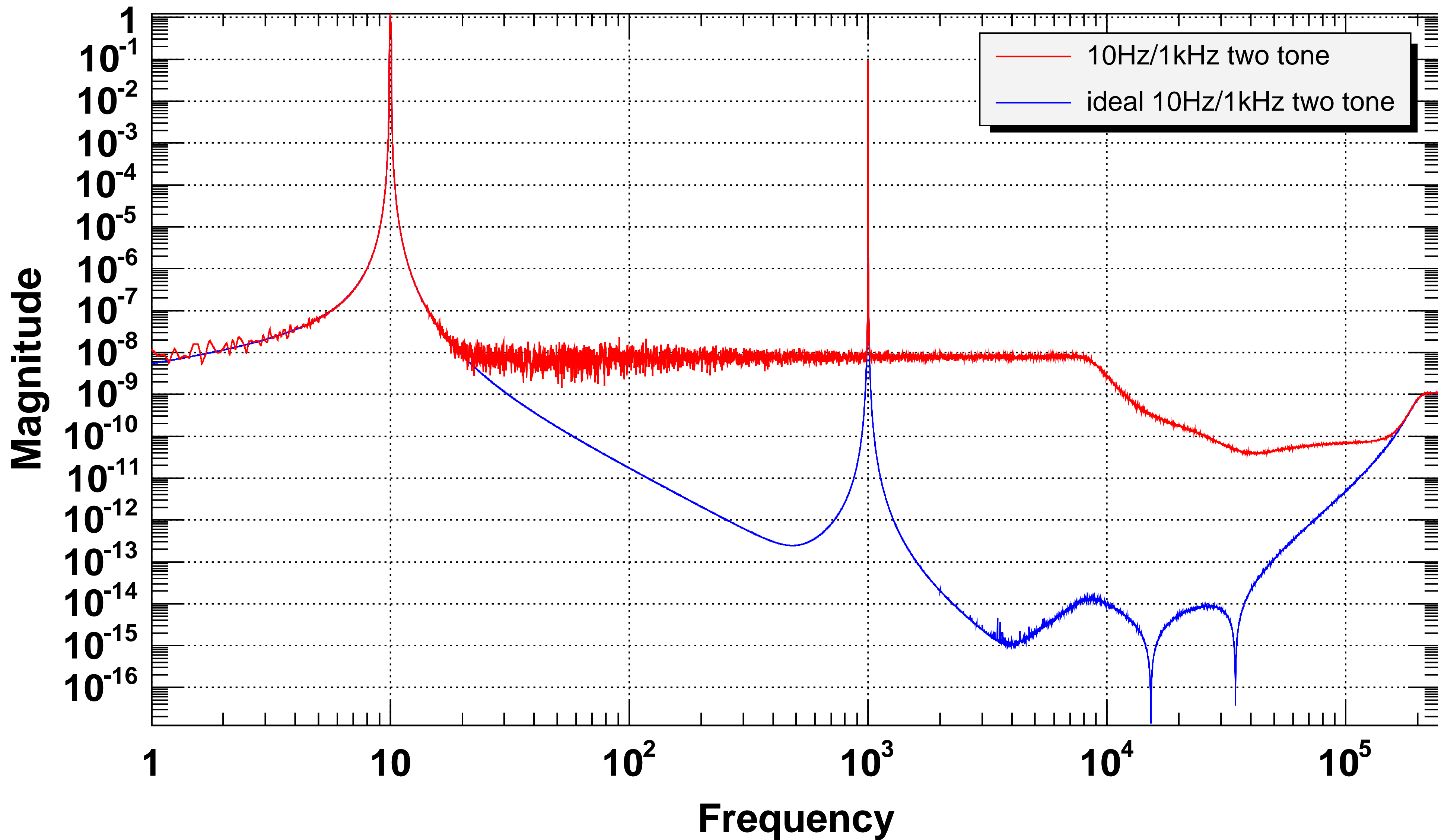
Power spectrum



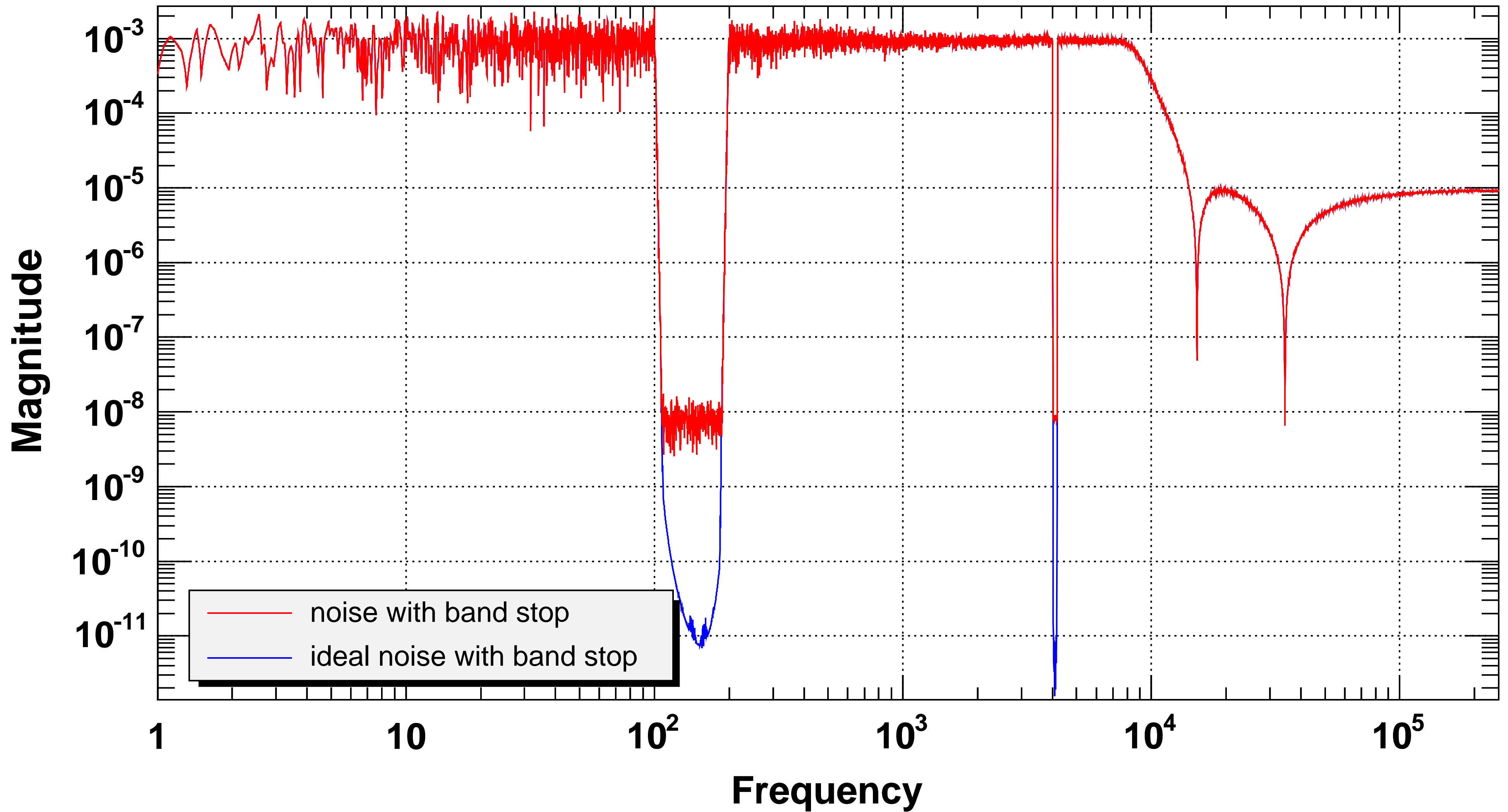
T0=06/01/1980 00:00:00 **Bin=419L**

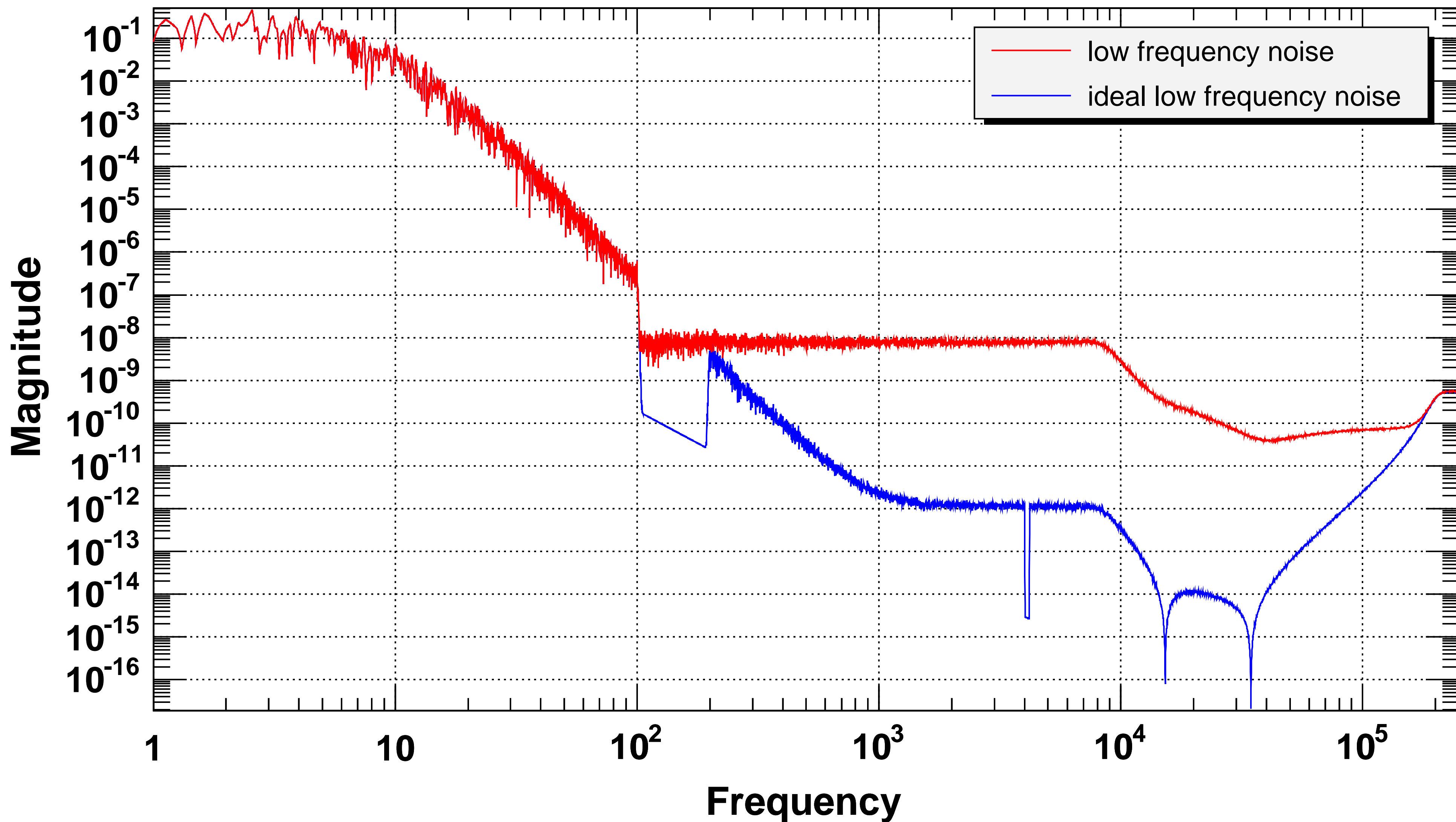


Power spectrum
100kHz sine wave
ideal 100kHz sine wave
Magnitude
Frequency
10^-5
10^-6
10^-7
10^-8
10^-9
10^-10
10^-11
10^-12
10^-13
10^-14
10^-15
10^-16
10^-17
1
10
10^2
10^3
10^4
10^5
T0=06/01/1980 00:00:00
Bin=419L



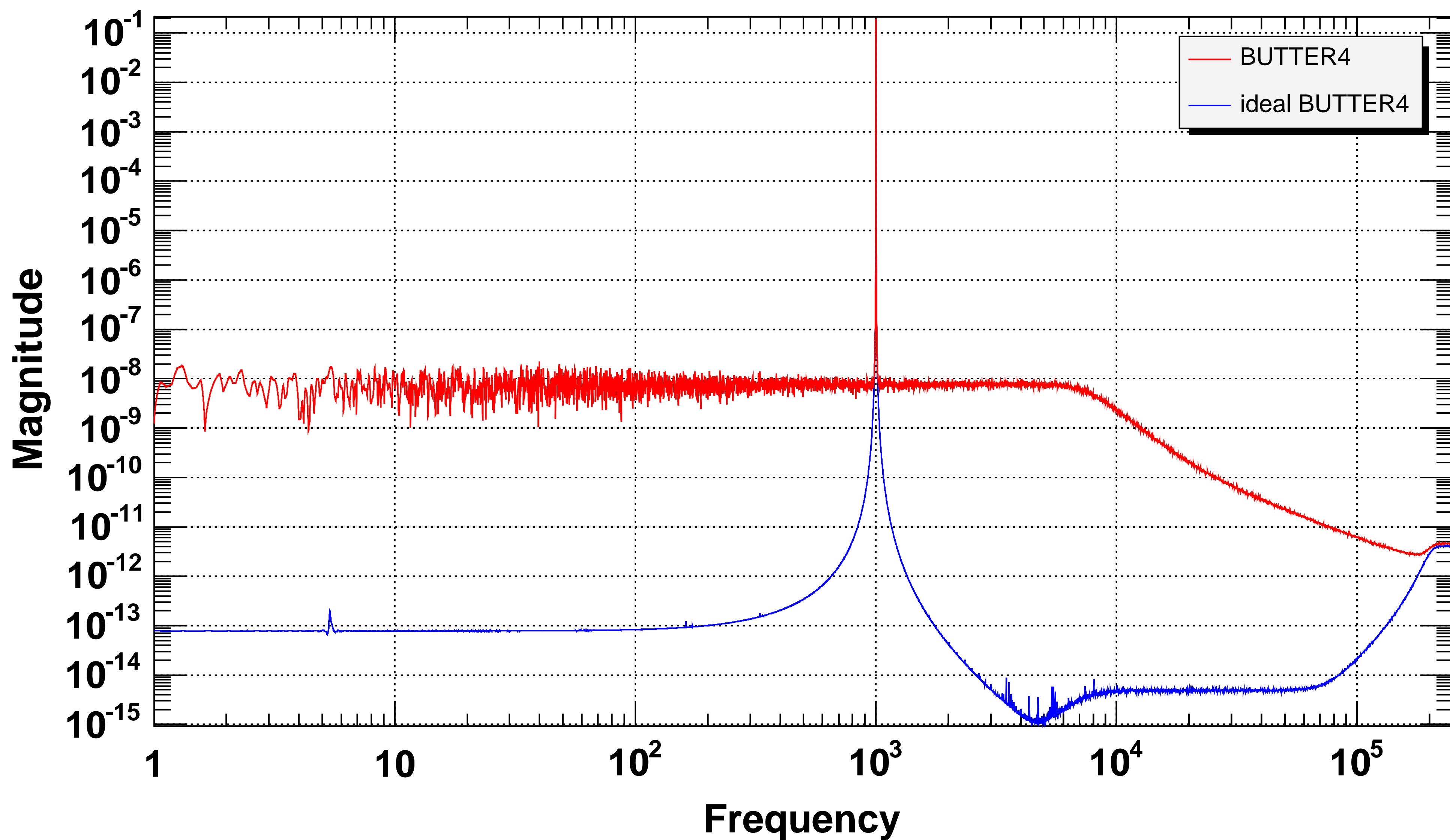
Power spectrum
Magnitude
Frequency
10Hz/1kHz two tone
ideal 10Hz/1kHz two tone
1
10
10^2
10^3
10^4
10^5
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
10^{-7}
10^{-8}
10^{-9}
10^{-10}
10^{-11}
10^{-12}
10^{-13}
10^{-14}
10^{-15}
10^{-16}
T0=06/01/1980 00:00:00
Bin=419L


Power spectrum
Magnitude
Frequency
noise with band stop
ideal noise with band stop
T0=06/01/1980 00:00:00
Bin=419L

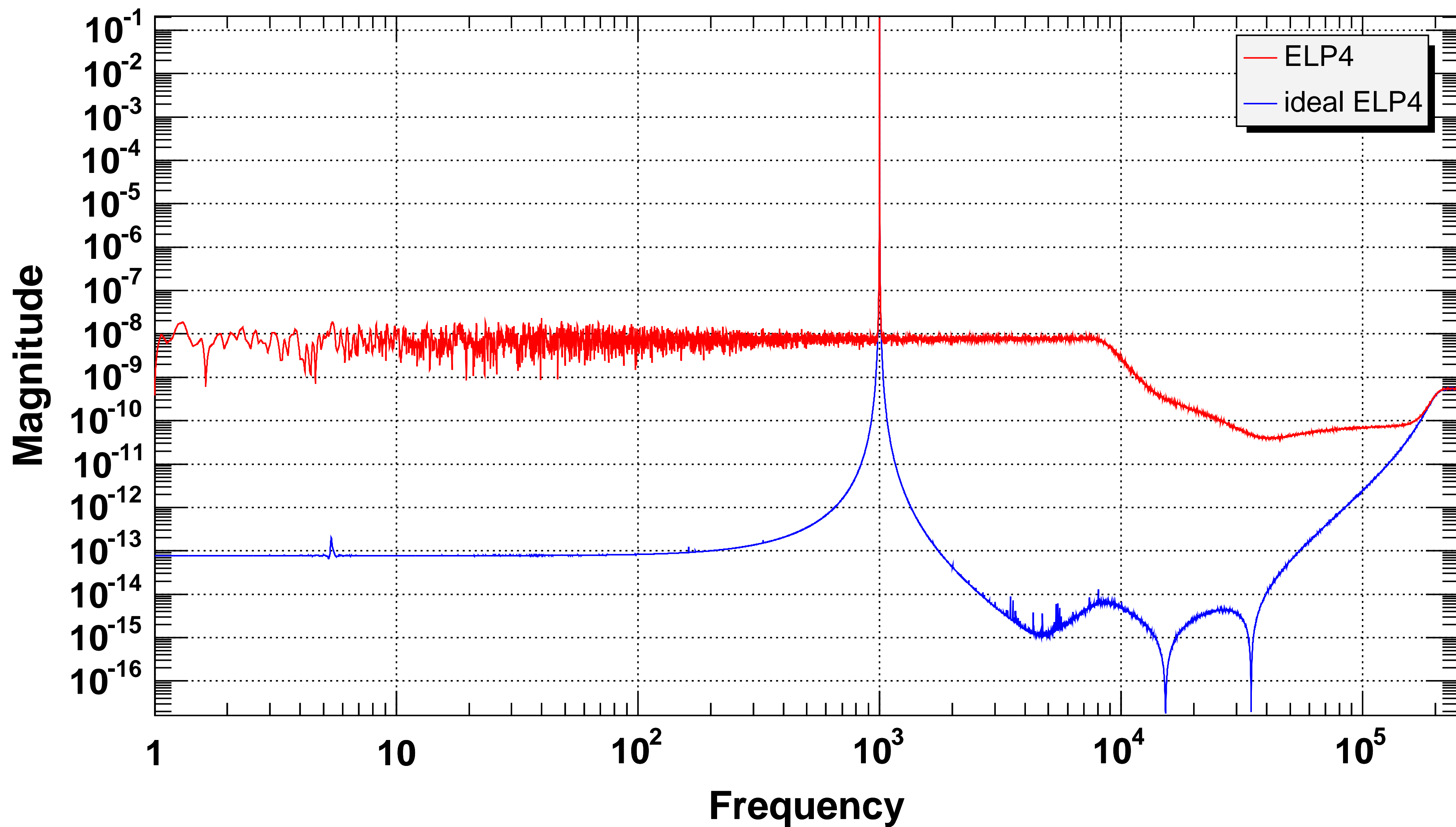


Power spectrum
low frequency noise
ideal low frequency noise
Magnitude
Frequency
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
10^{-7}
10^{-8}
10^{-9}
10^{-10}
10^{-11}
10^{-12}
10^{-13}
10^{-14}
10^{-15}
10^{-16}
1
10
10^{2}
10^{3}
10^{4}
10^{5}
T0=06/01/1980 00:00:00
Bin=419L

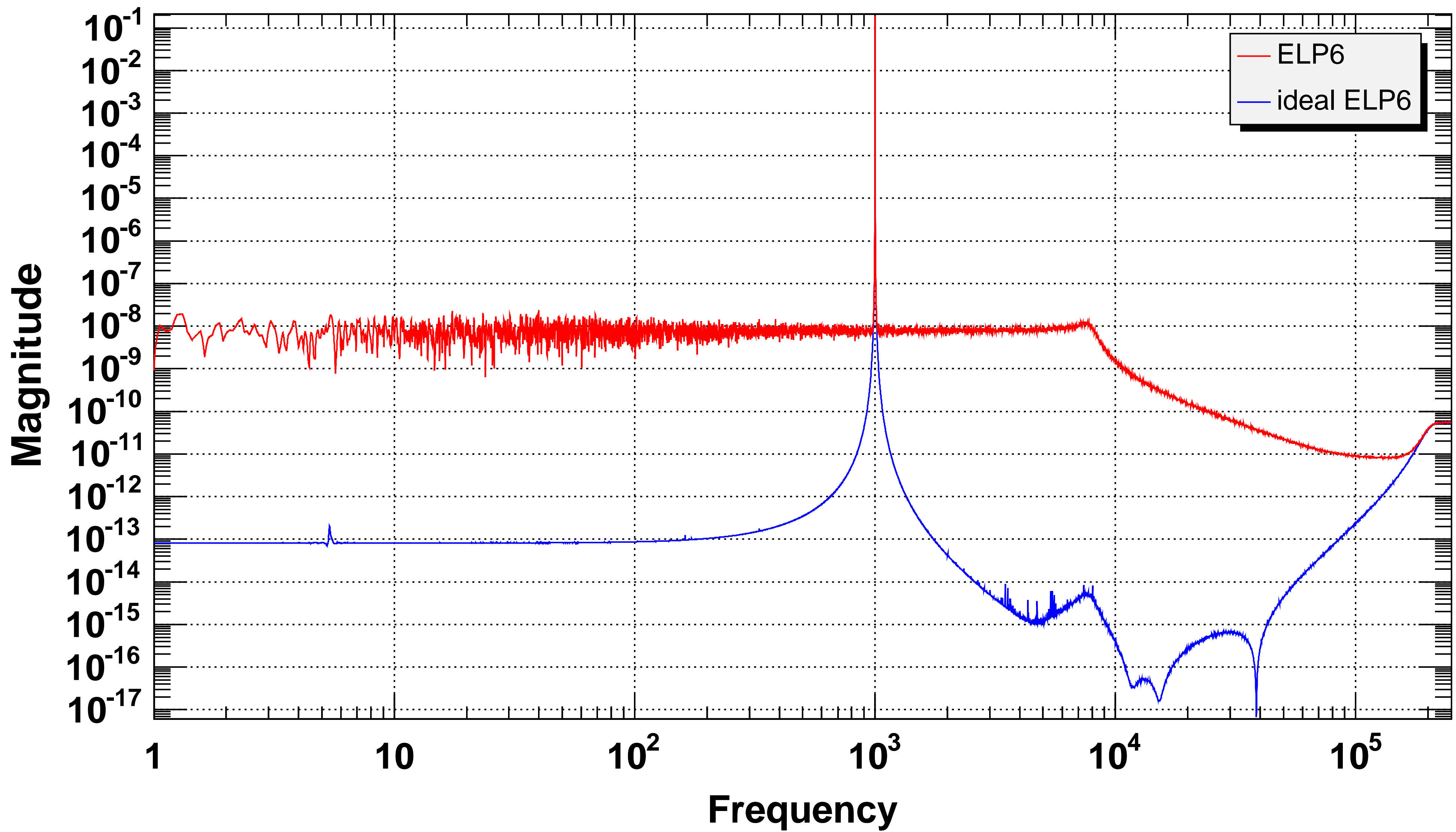
APPENDIX E SPECTRA OF DIFFERENT FILTERS



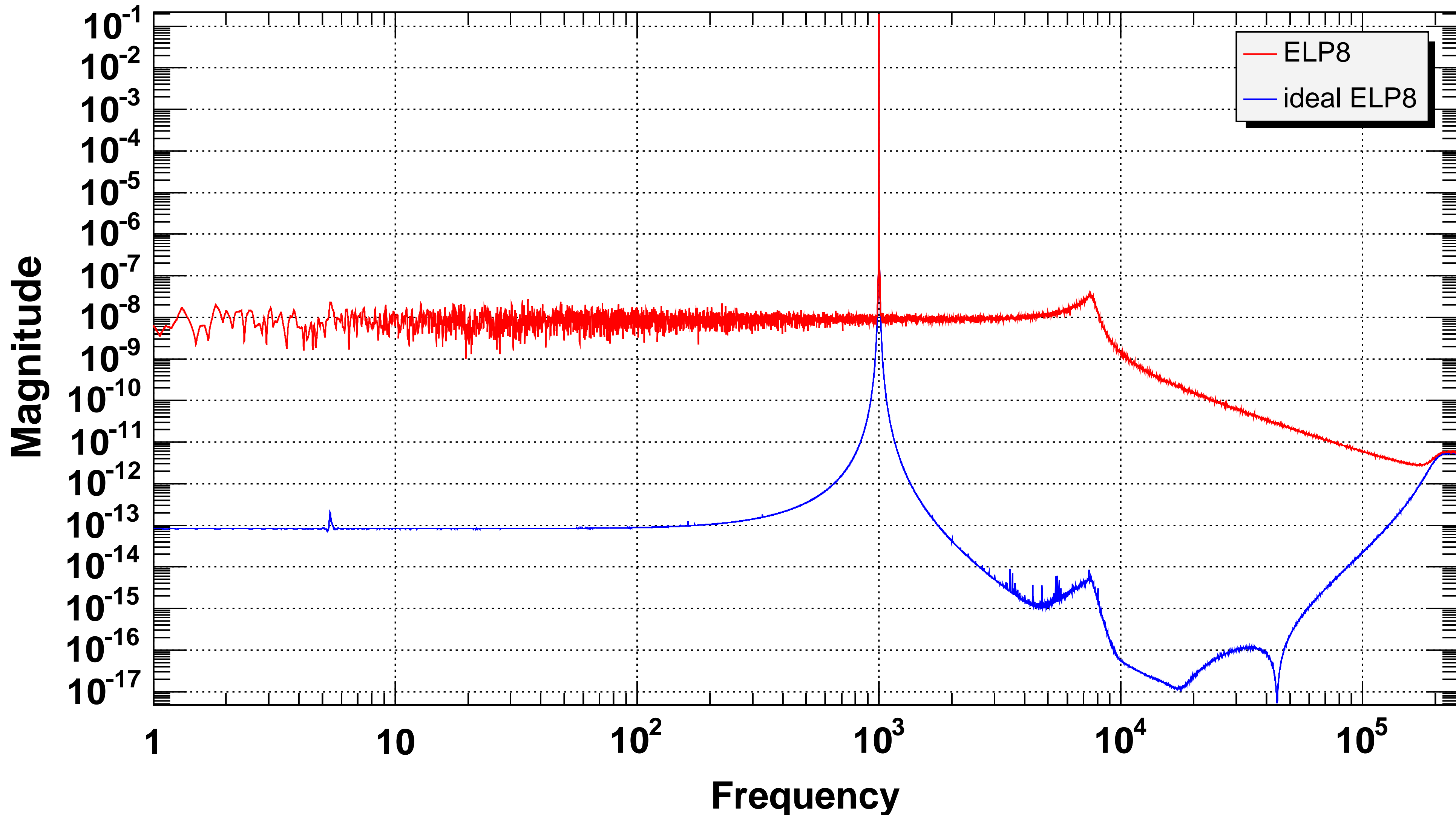
Power spectrum
BUTTER4
ideal BUTTER4
Magnitude
Frequency
T0=06/01/1980 00:00:00
Bin=419L


Power spectrum
ELP4
ideal ELP4
Magnitude
Frequency
T0=06/01/1980 00:00:00
Bin=419L

Power spectrum

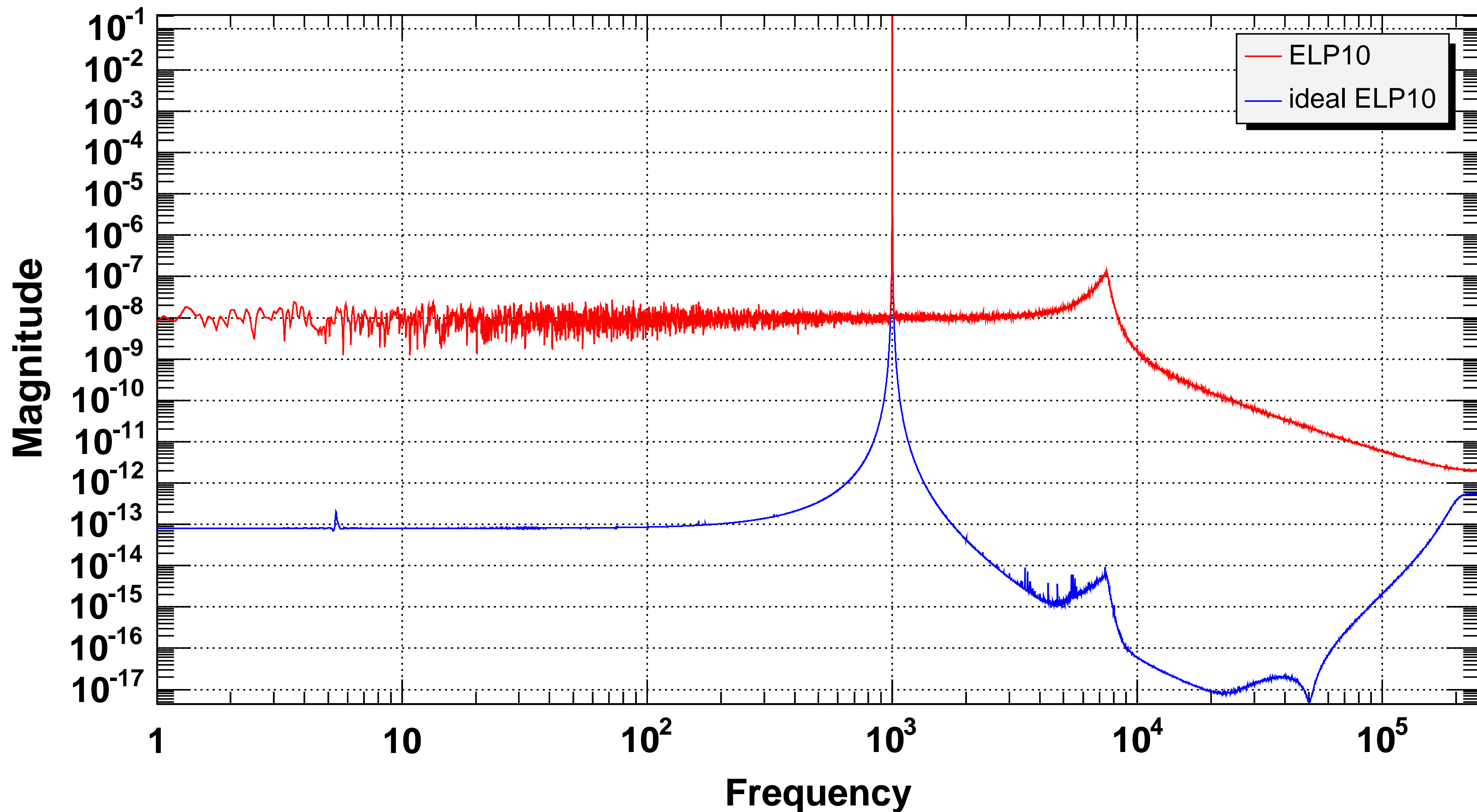


T0=06/01/1980 00:00:00 Bin=419L

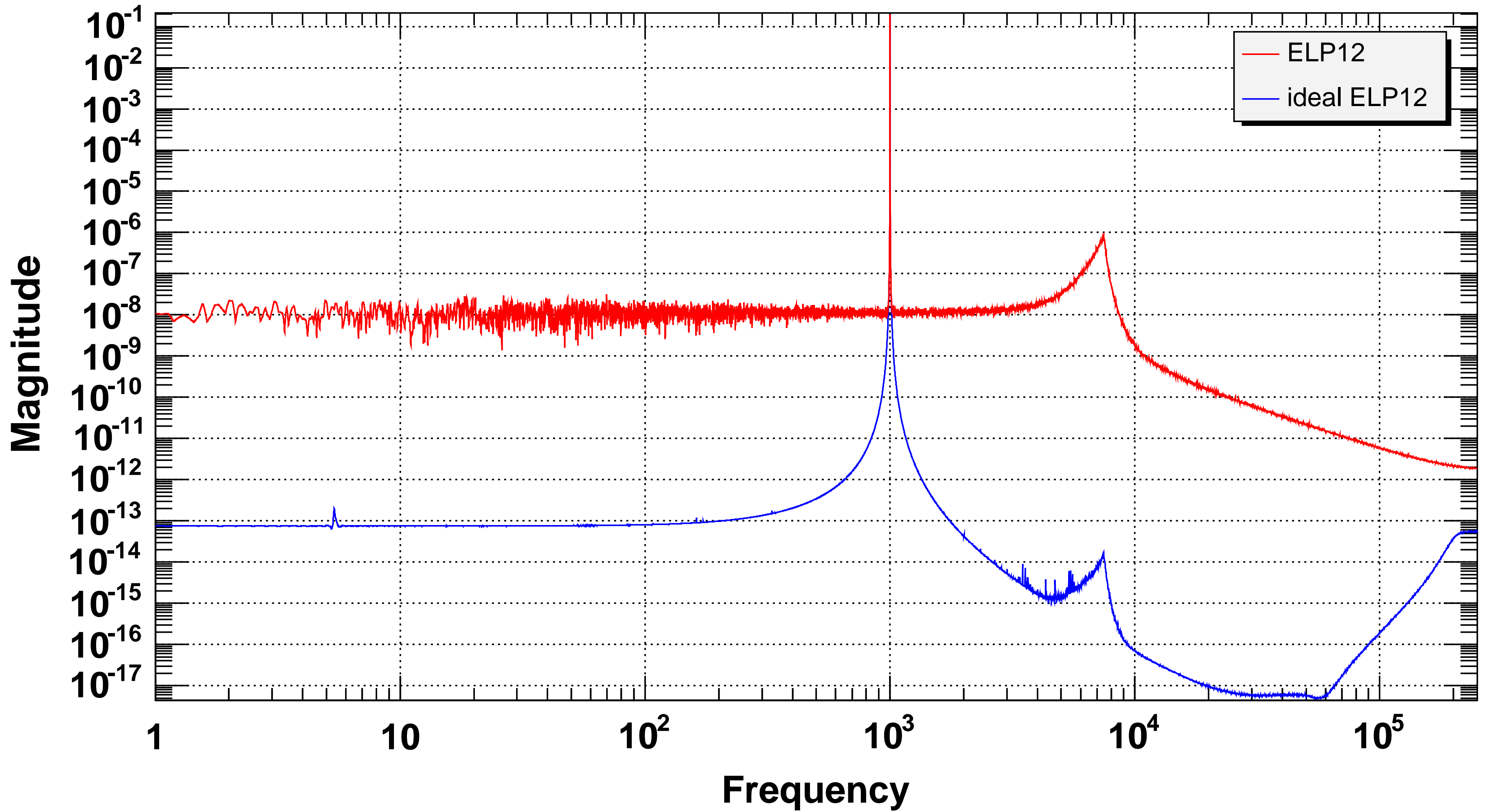


Power spectrum
ELP8
ideal ELP8
Magnitude
Frequency
T0=06/01/1980 00:00:00
Bin=419L

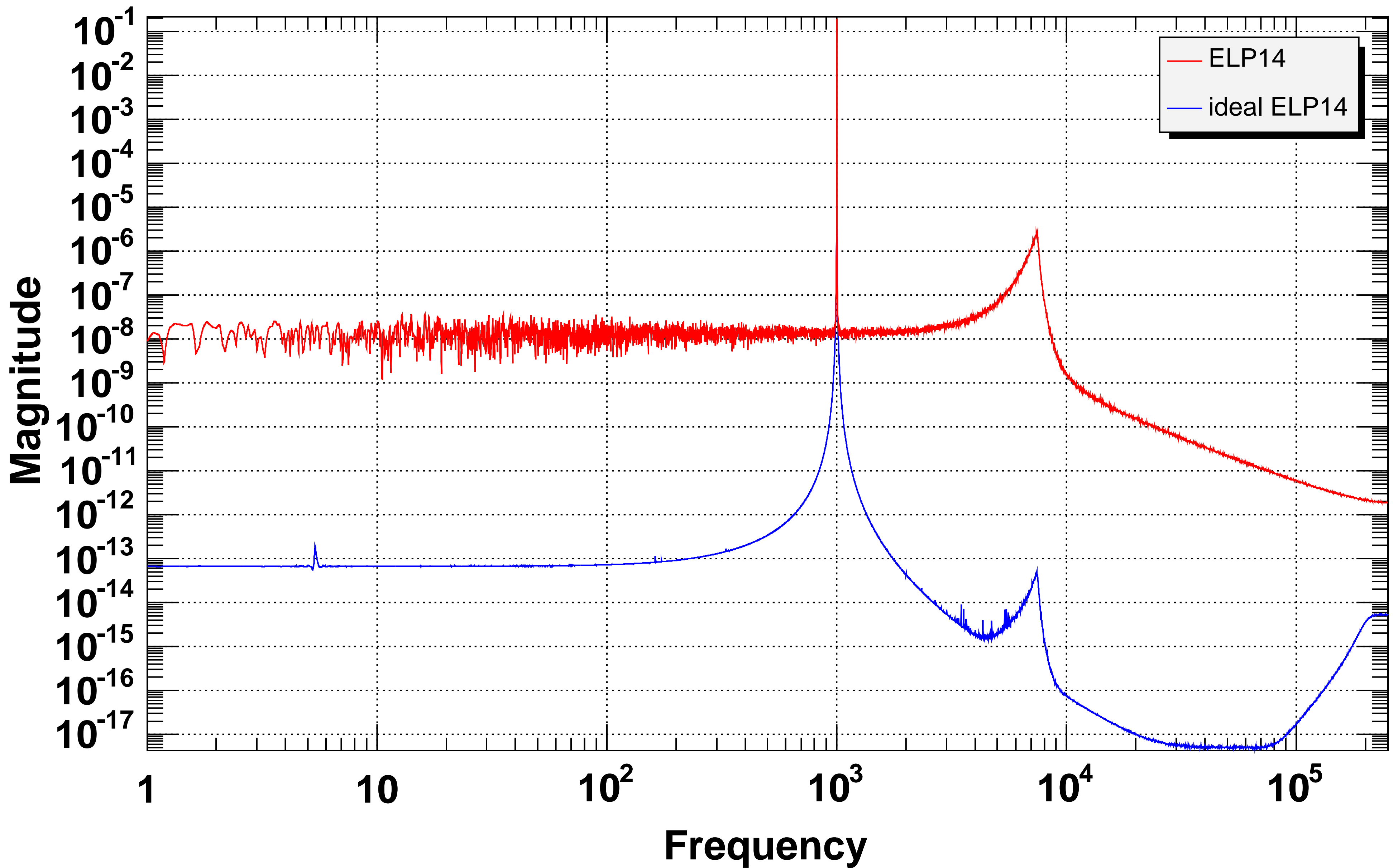
Power spectrum



T0=06/01/1980 00:00:00 Bin=419L

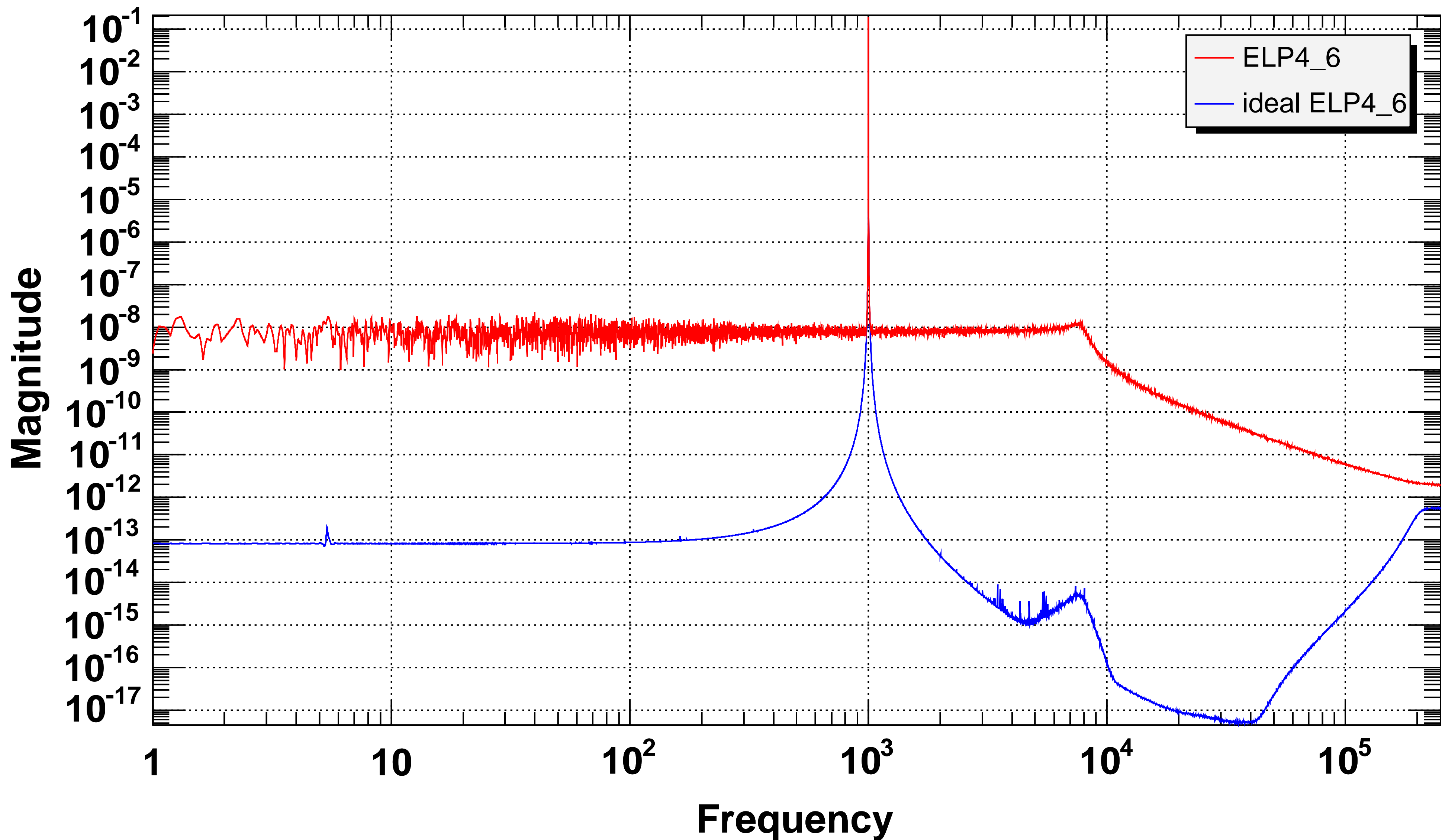

Power spectrum
ELP12
ideal ELP12
Magnitude
Frequency
T0=06/01/1980 00:00:00
Bin=419L

Power spectrum

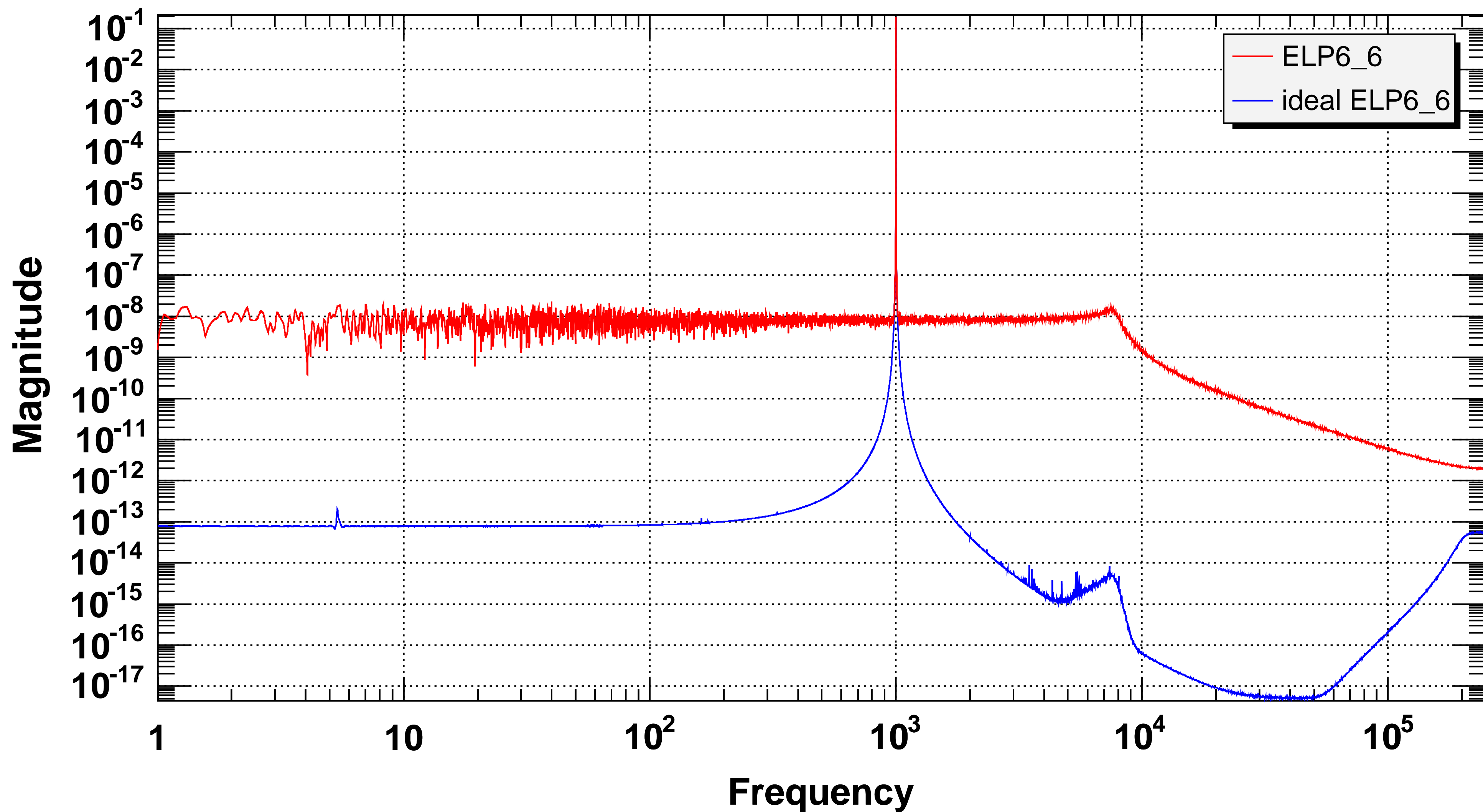


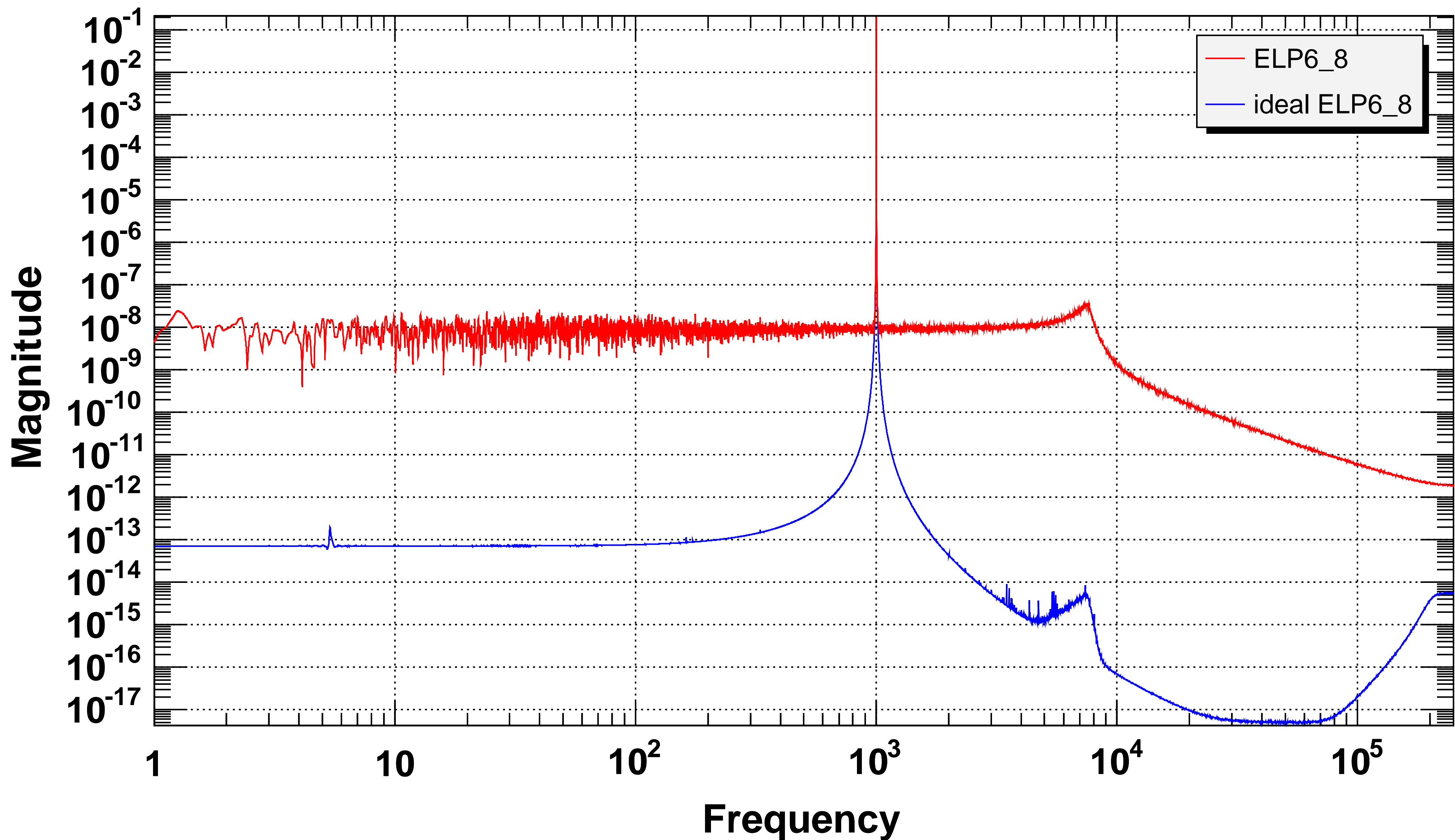
T0=06/01/1980 00:00:00 Bin=419L

Power spectrum



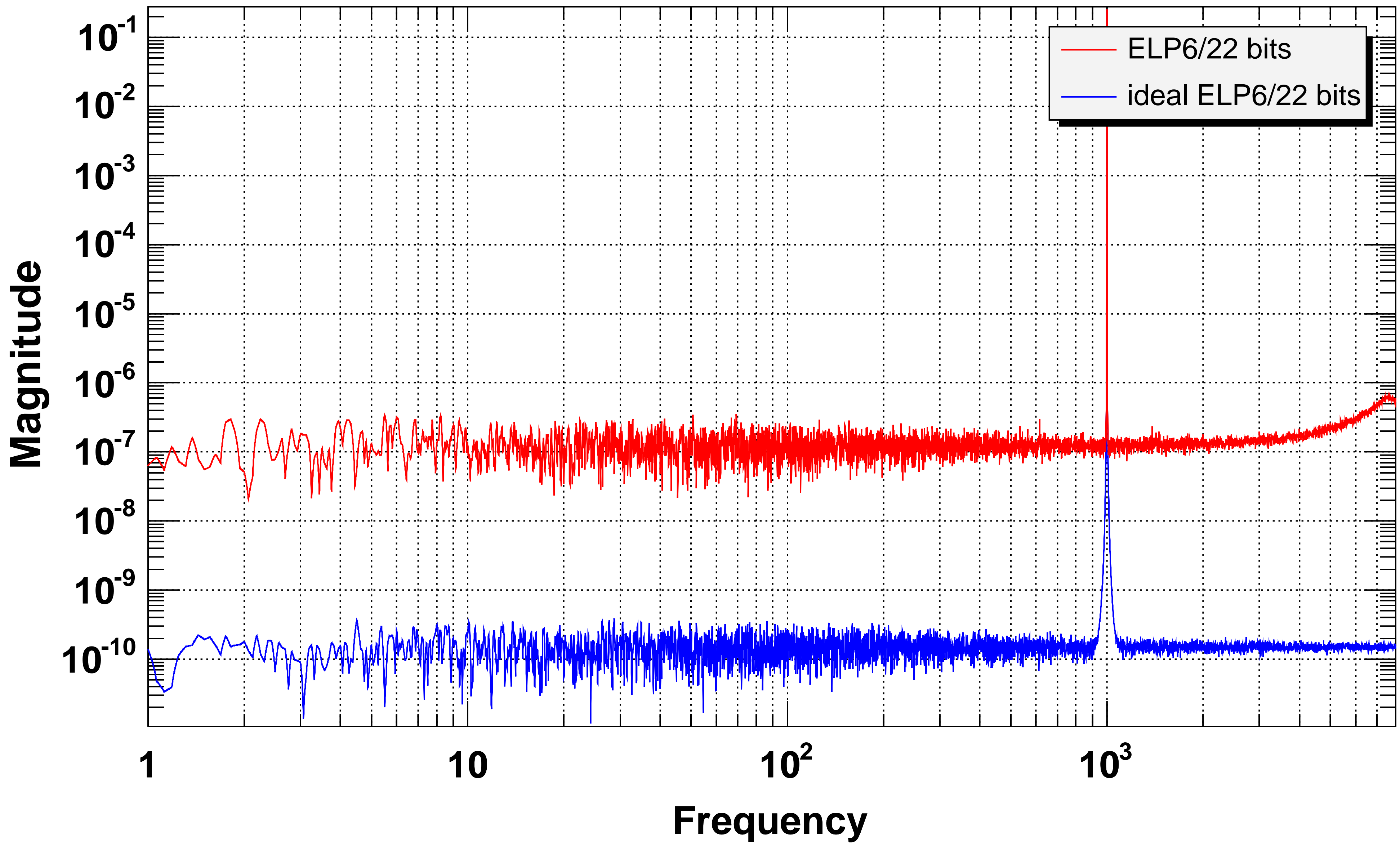
T0=06/01/1980 00:00:00 Bin=419L

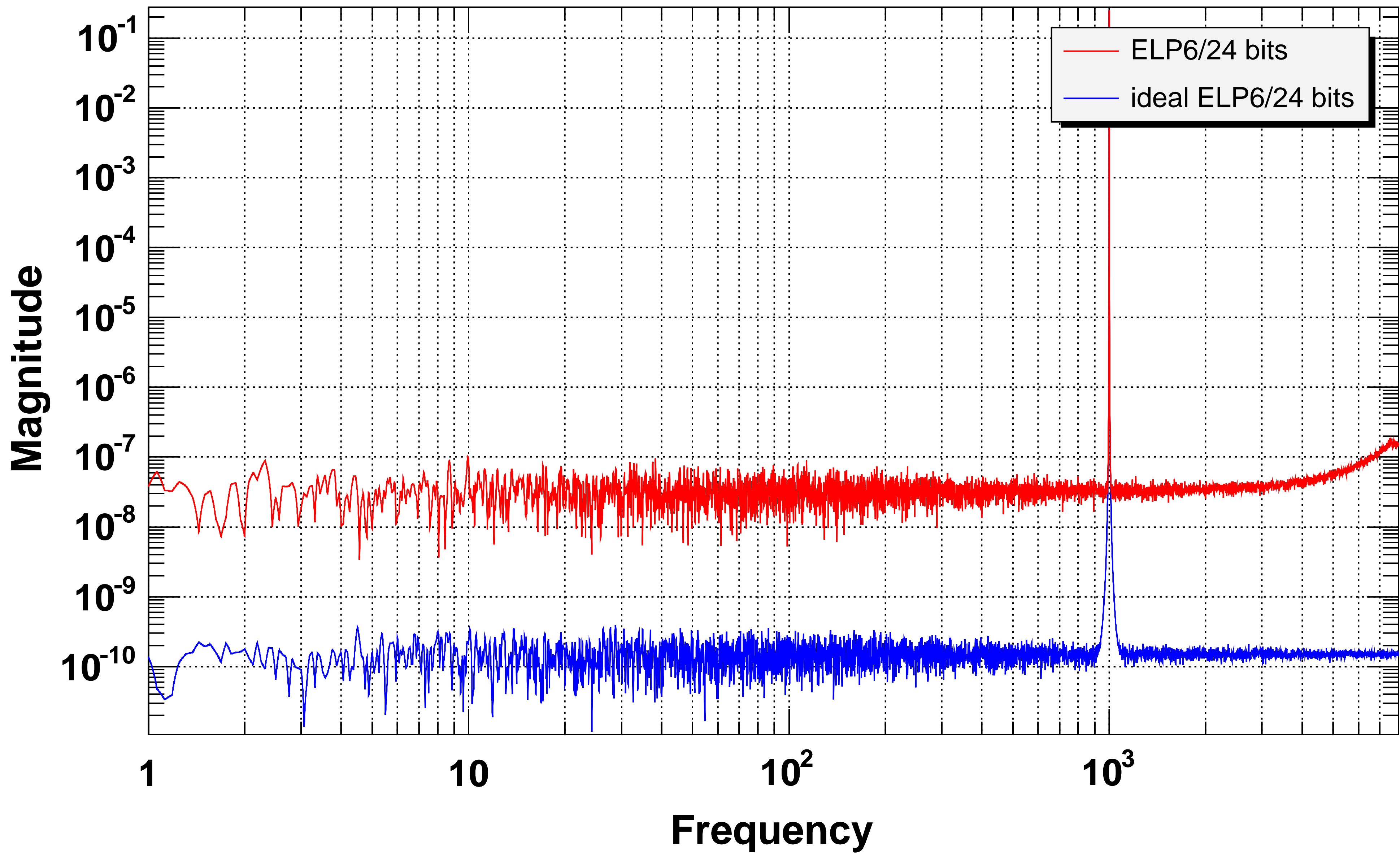

Power spectrum
ELP6_6
ideal ELP6_6
Magnitude
Frequency
T0=06/01/1980 00:00:00
Bin=419L



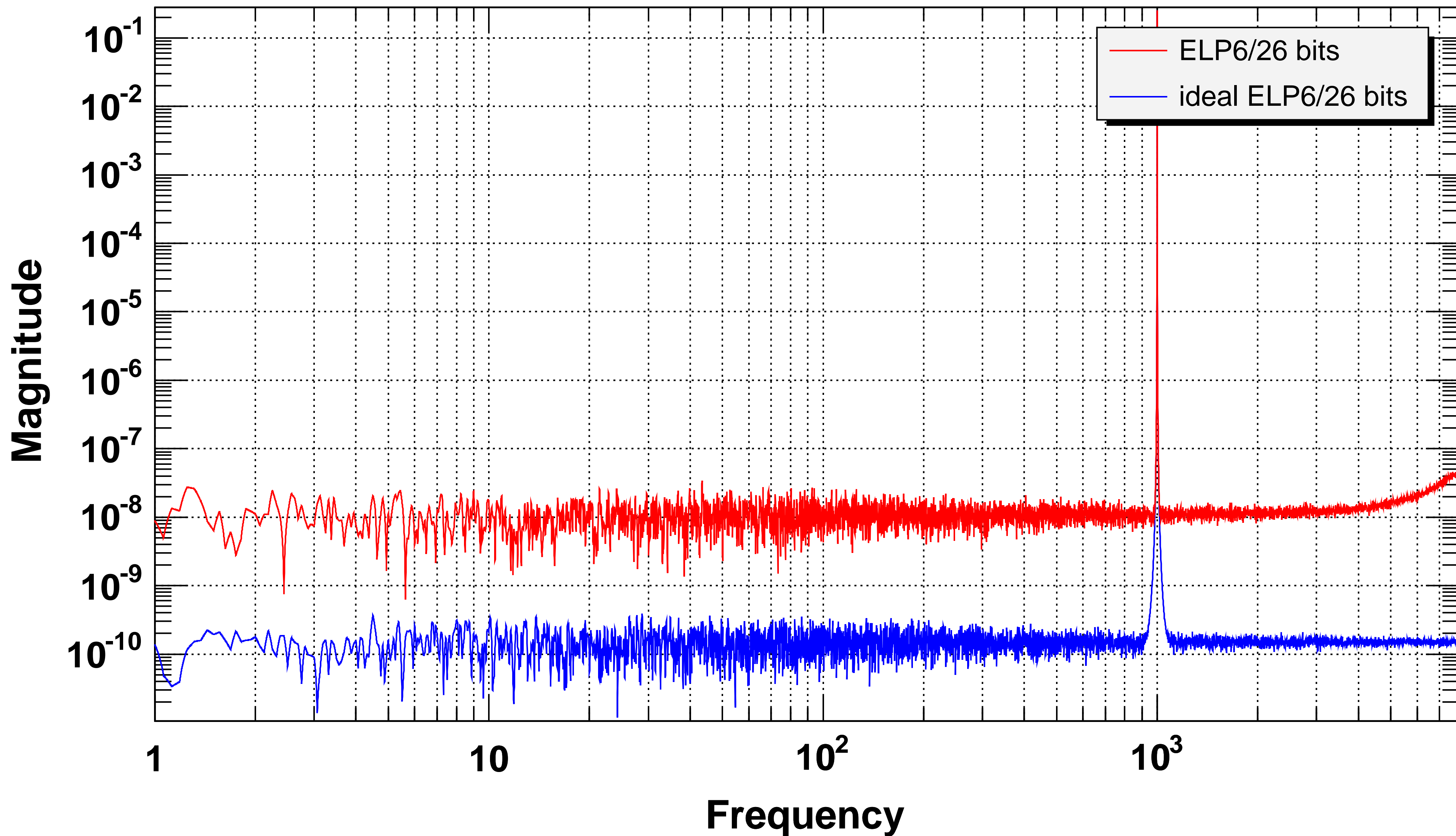
Power spectrum
ELP6_8
ideal ELP6_8
Magnitude
Frequency
T0=06/01/1980 00:00:00
Bin=419L

APPENDIX F EFFECT OF BIT RESOLUTION

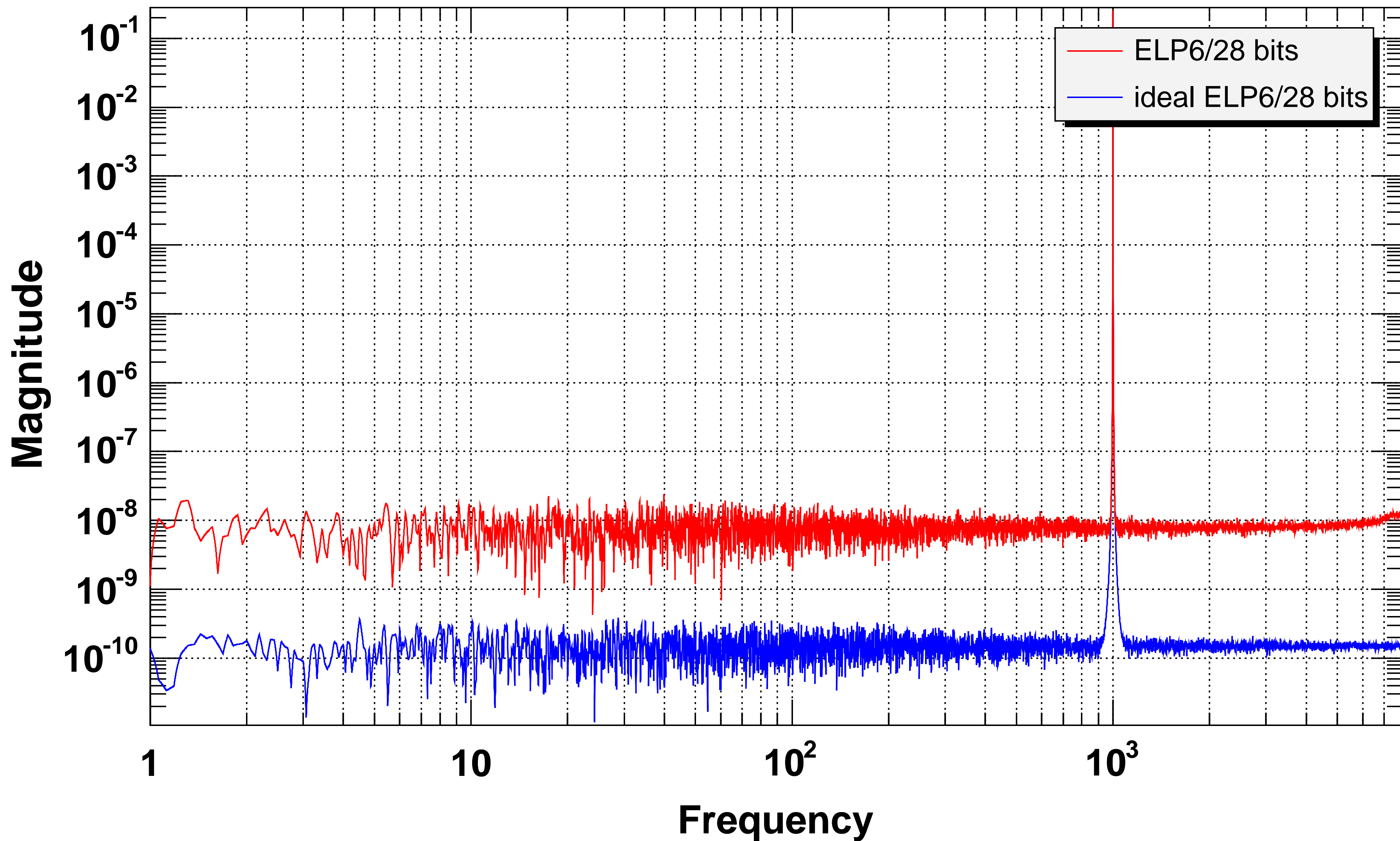

Power spectrum
ELP6/22 bits
ideal ELP6/22 bits
Magnitude
Frequency
T0=06/01/1980 00:00:00
Bin=13L

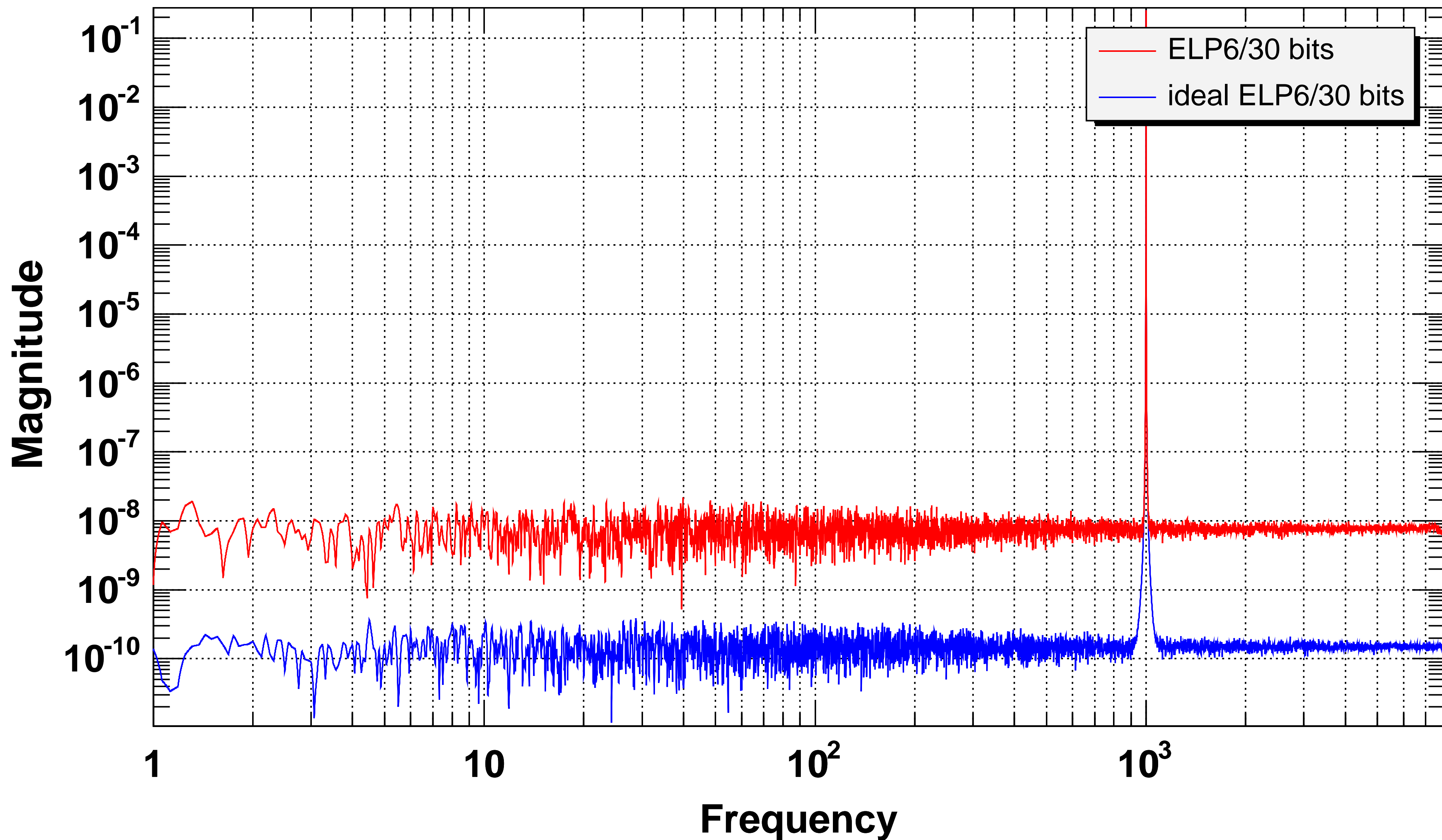


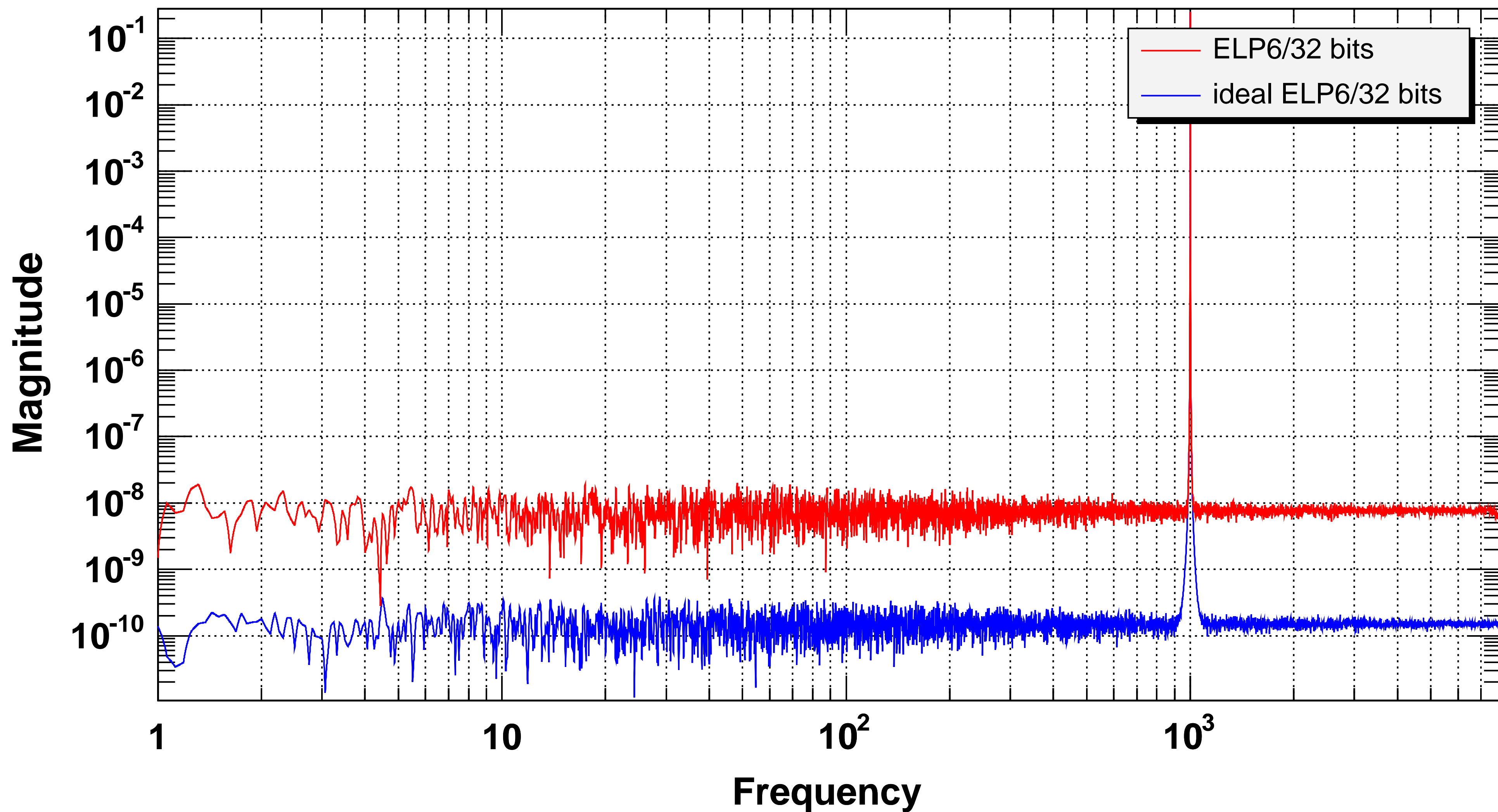
Power spectrum
ELP6/24 bits
ideal ELP6/24 bits
Magnitude
Frequency
T0=06/01/1980 00:00:00
Bin=13L



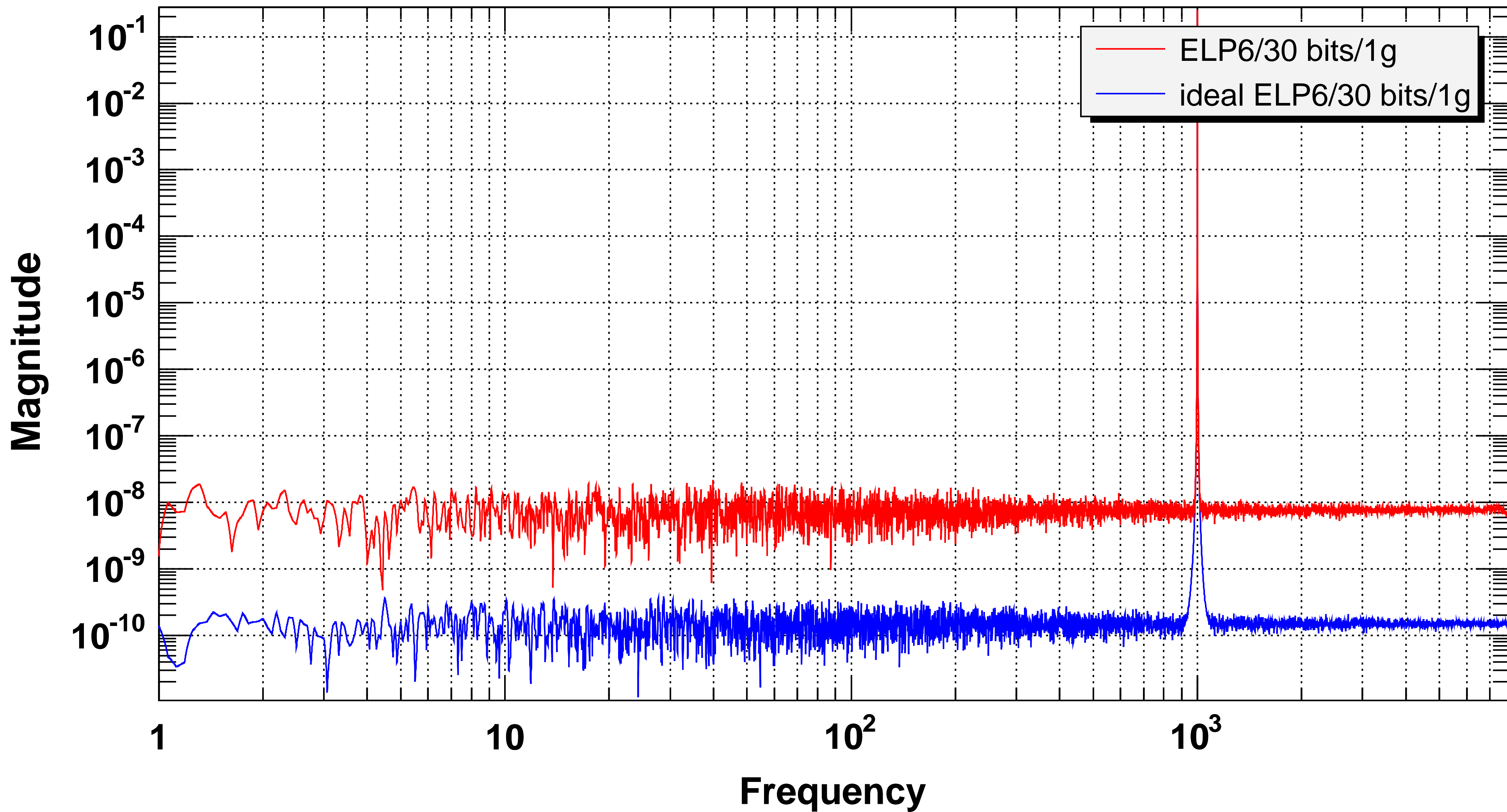
Power spectrum
ELP6/26 bits
ideal ELP6/26 bits
Magnitude
Frequency
T0=06/01/1980 00:00:00
Bin=13L

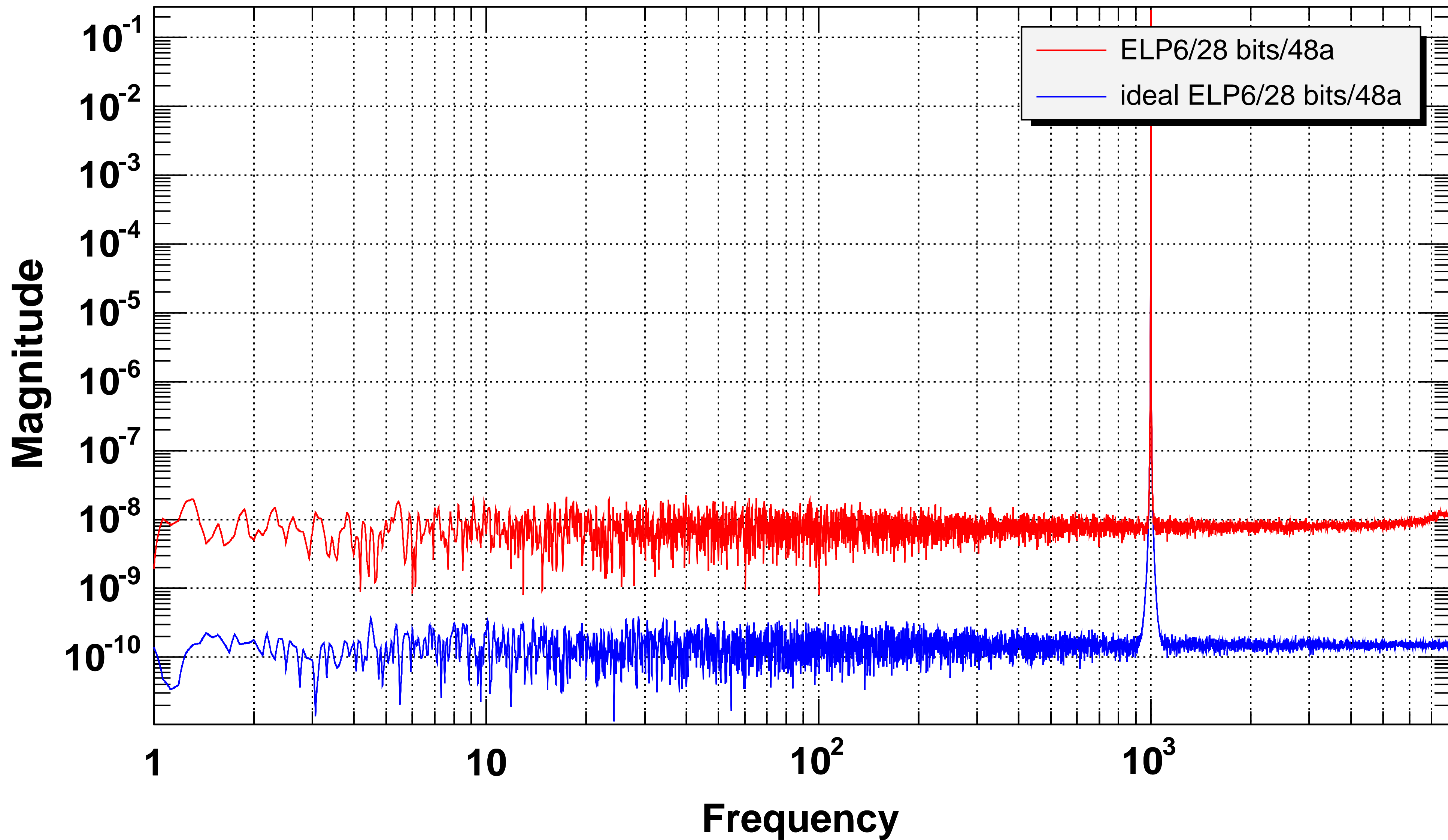

Power spectrum
ELP6/28 bits
ideal ELP6/28 bits
Magnitude
Frequency
T0=06/01/1980 00:00:00
Bin=13L


Power spectrum
ELP6/30 bits
ideal ELP6/30 bits
Magnitude
Frequency
T0=06/01/1980 00:00:00
Bin=13L

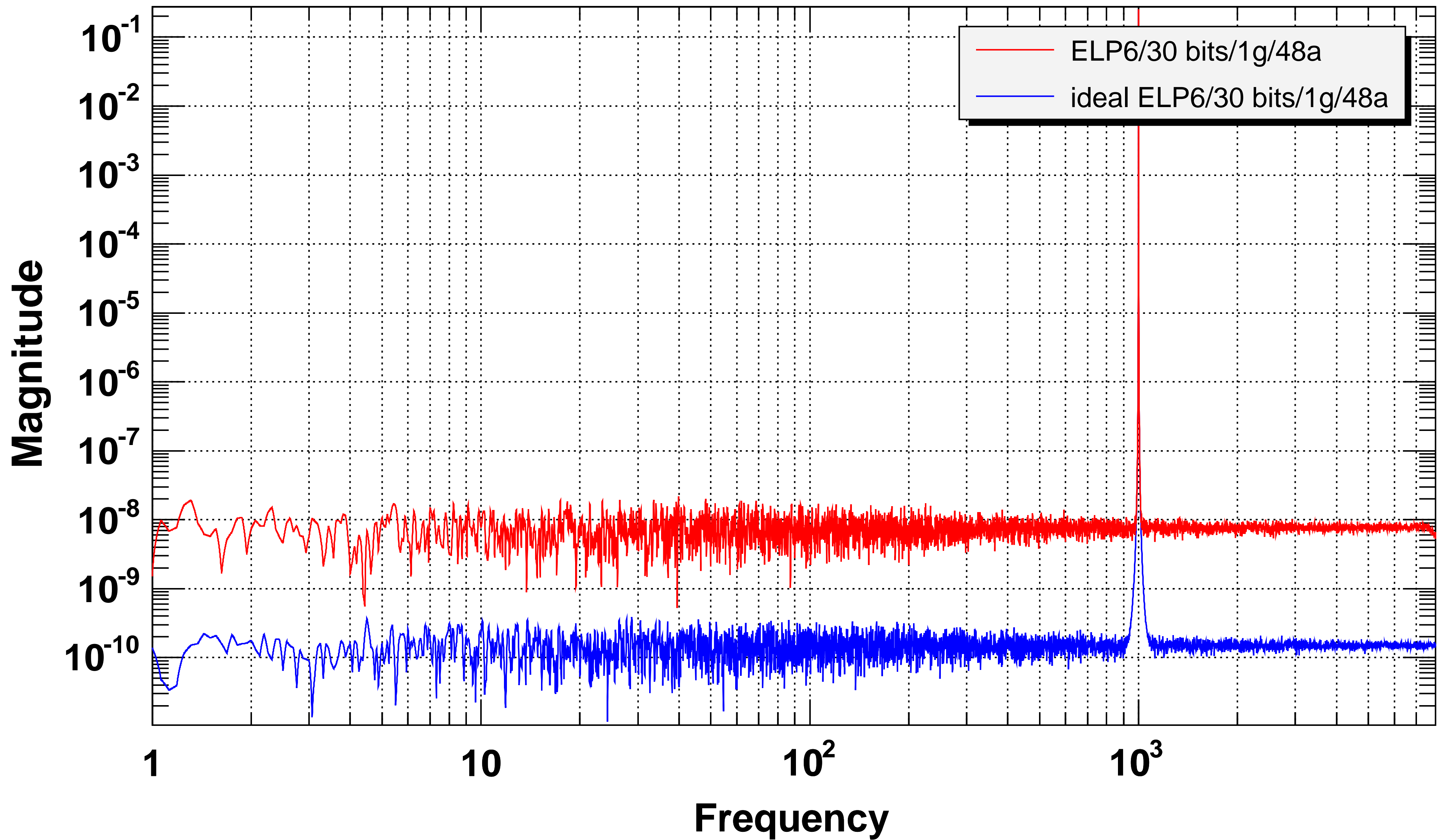


Power spectrum
ELP6/32 bits
ideal ELP6/32 bits
Magnitude
Frequency
T0=06/01/1980 00:00:00
Bin=13L


Power spectrum
ELP6/30 bits/1g
ideal ELP6/30 bits/1g
Magnitude
Frequency
T0=06/01/1980 00:00:00
Bin=13L

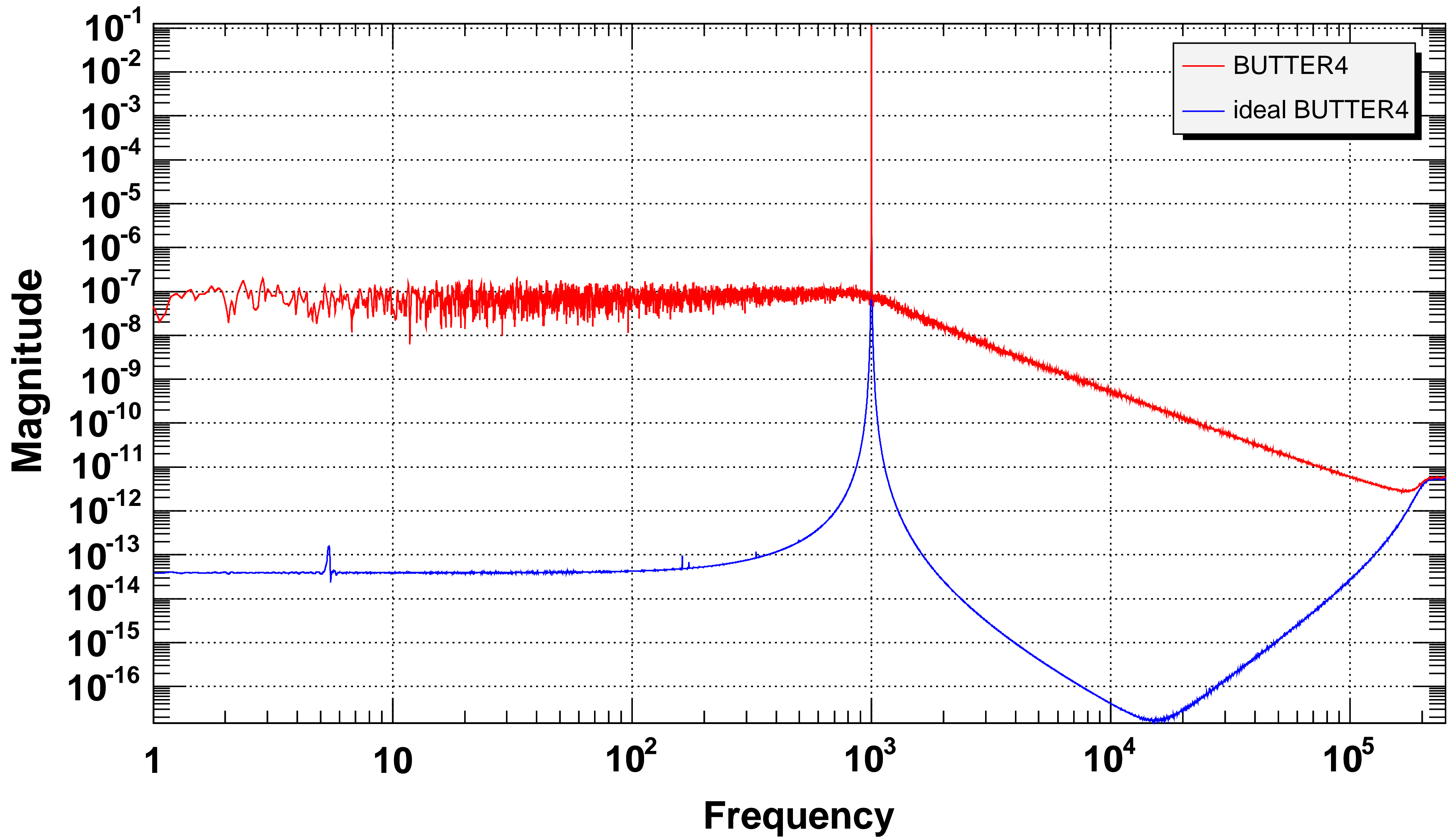


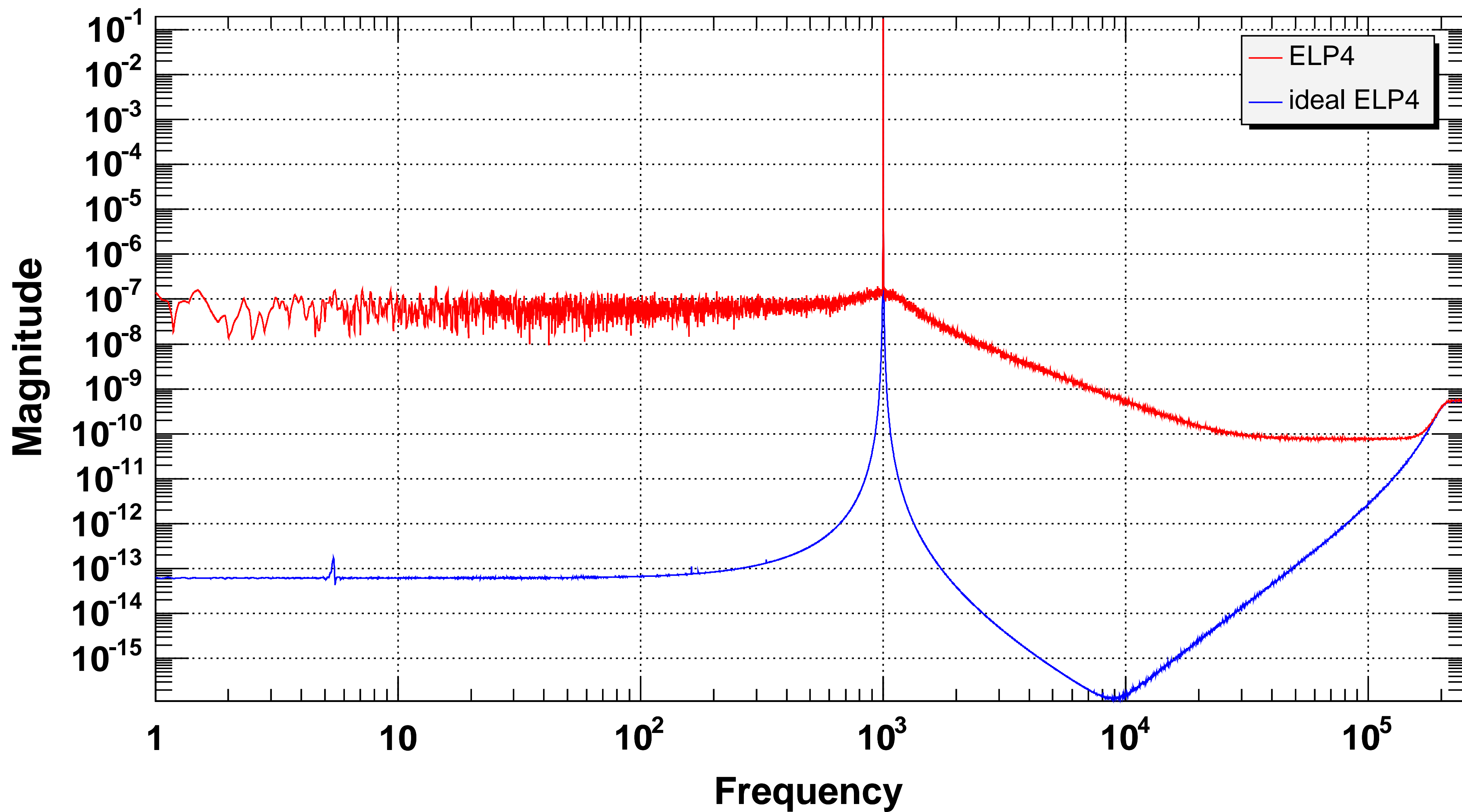
Power spectrum
Magnitude
Frequency
ELP6/28 bits/48a
ideal ELP6/28 bits/48a
T0=06/01/1980 00:00:00
Bin=13L

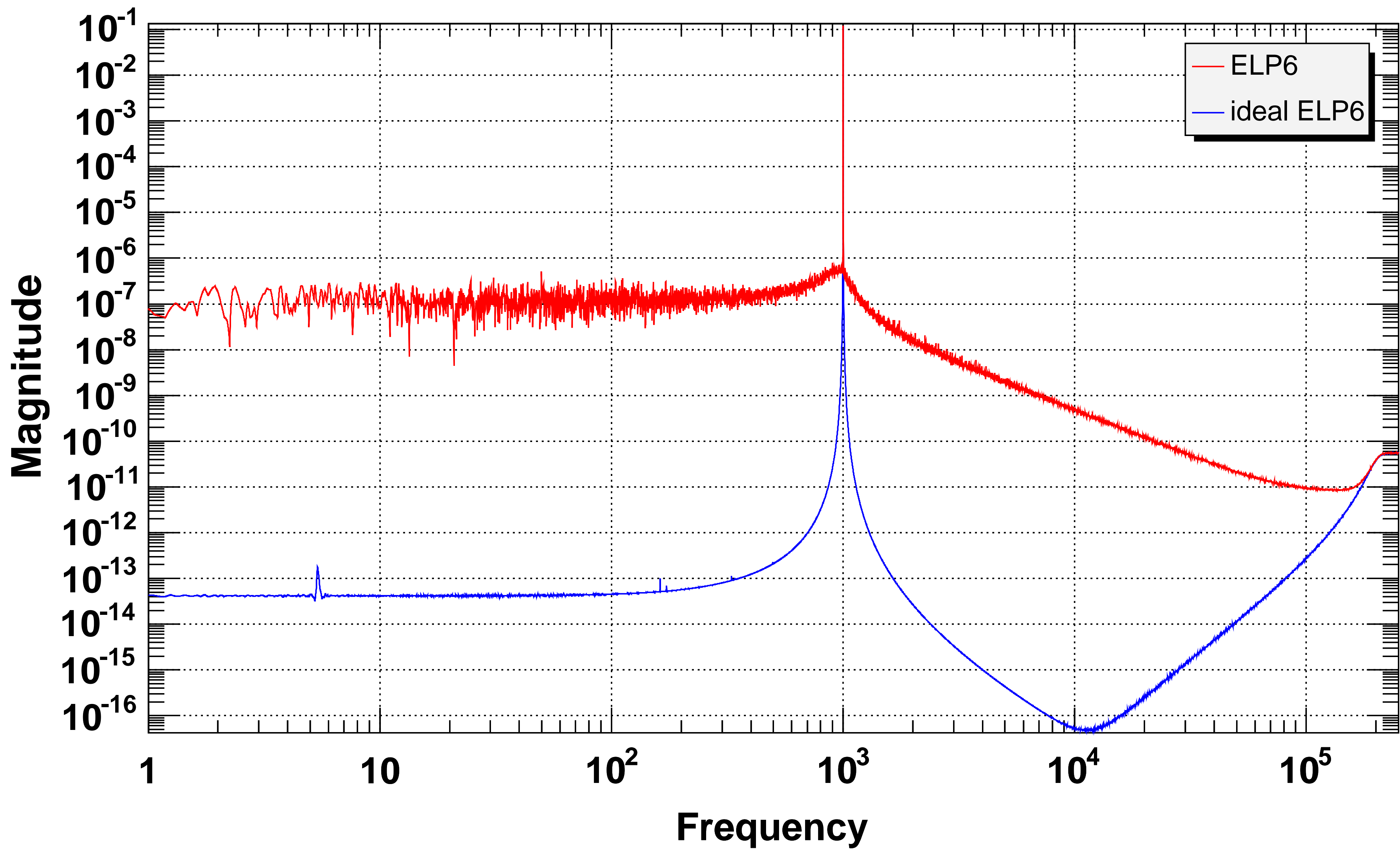


Power spectrum
ELP6/30 bits/1g/48a
ideal ELP6/30 bits/1g/48a
Magnitude
Frequency
T0=06/01/1980 00:00:00
Bin=13L

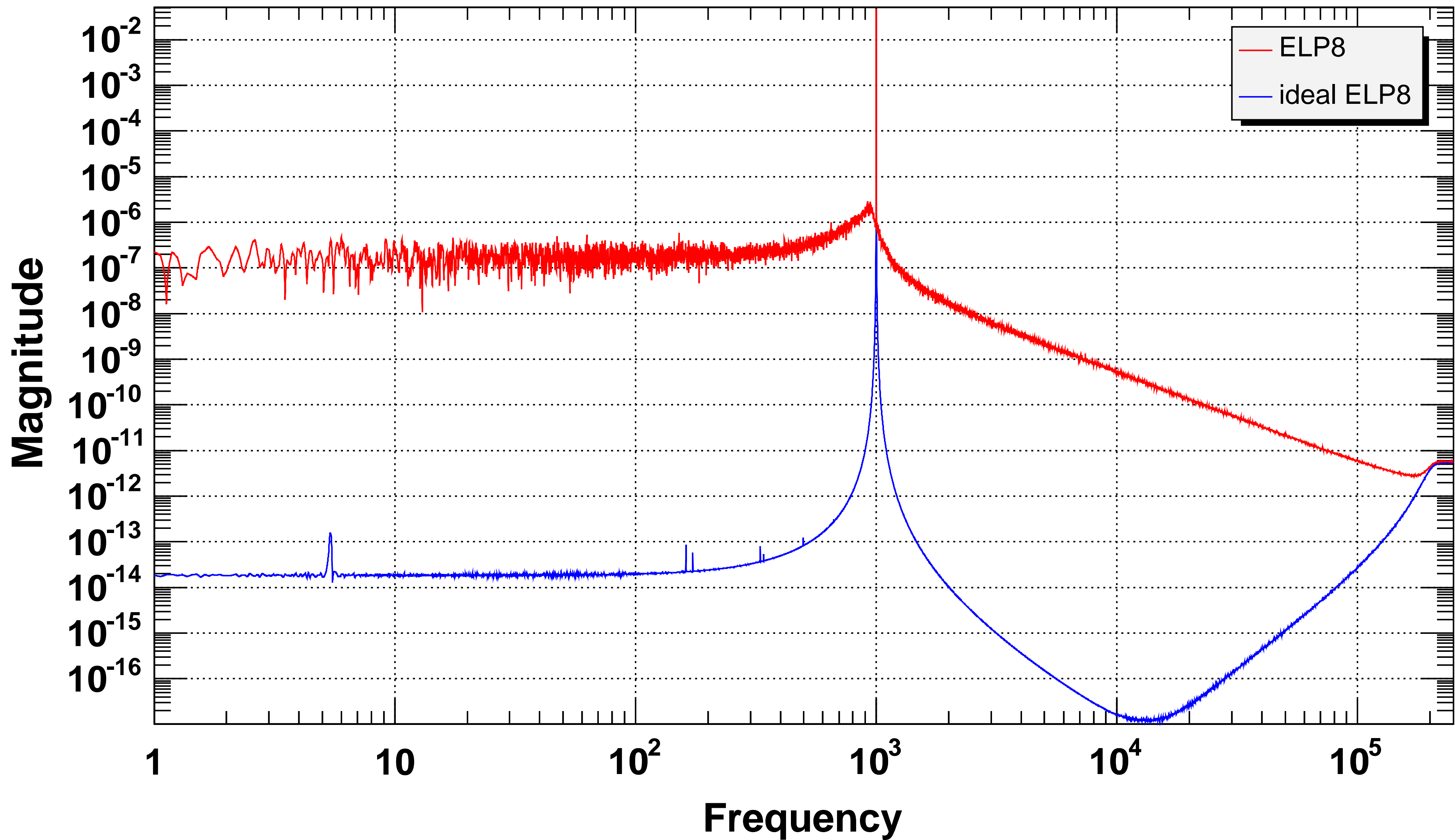
APPENDIX G 2 KHZ SAMPLING RATE


Power spectrum
BUTTER4
ideal BUTTER4
Magnitude
Frequency
T0=06/01/1980 00:00:00
Bin=419L

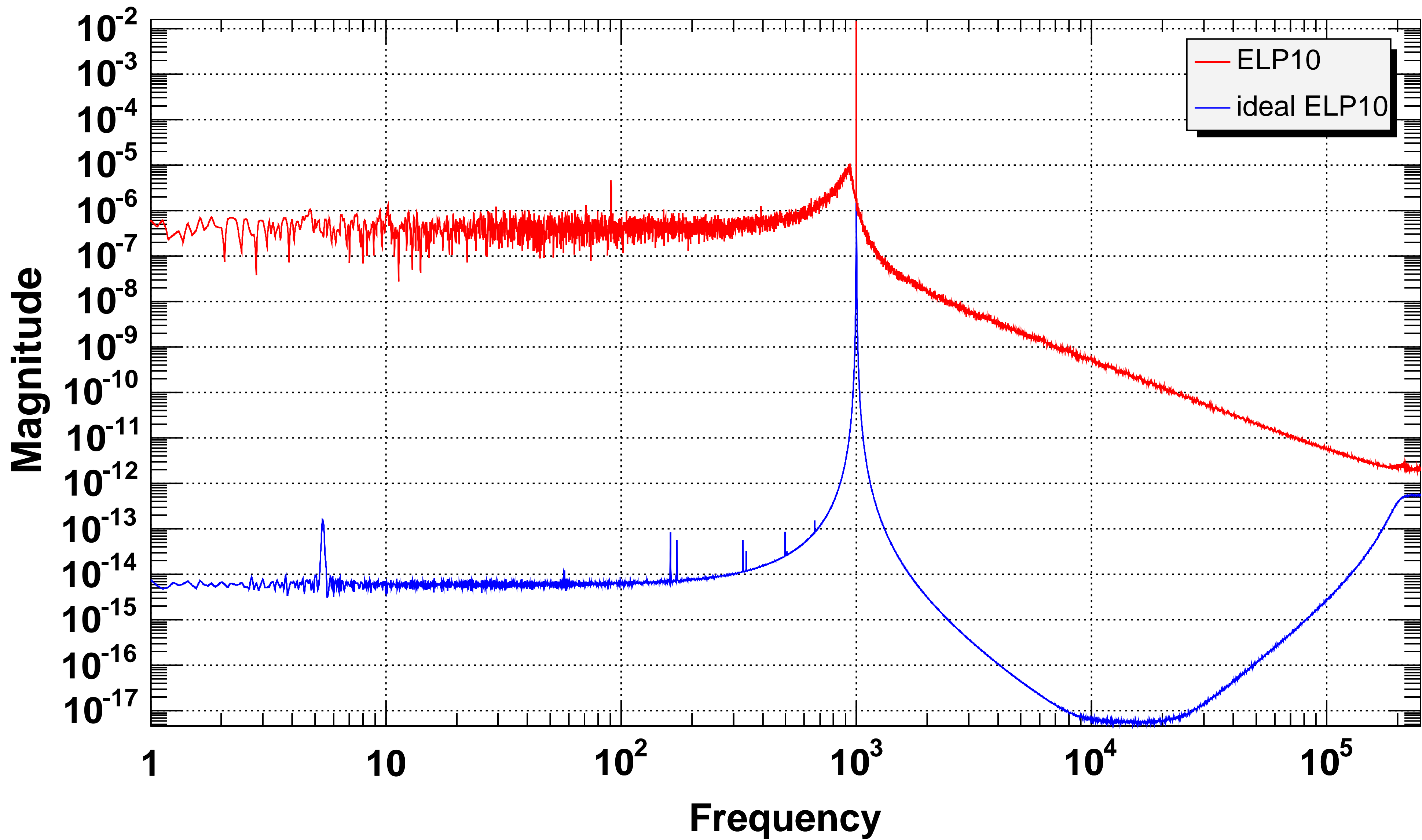

Power spectrum
Magnitude
Frequency
ELP4
ideal ELP4
T0=06/01/1980 00:00:00
Bin=419L



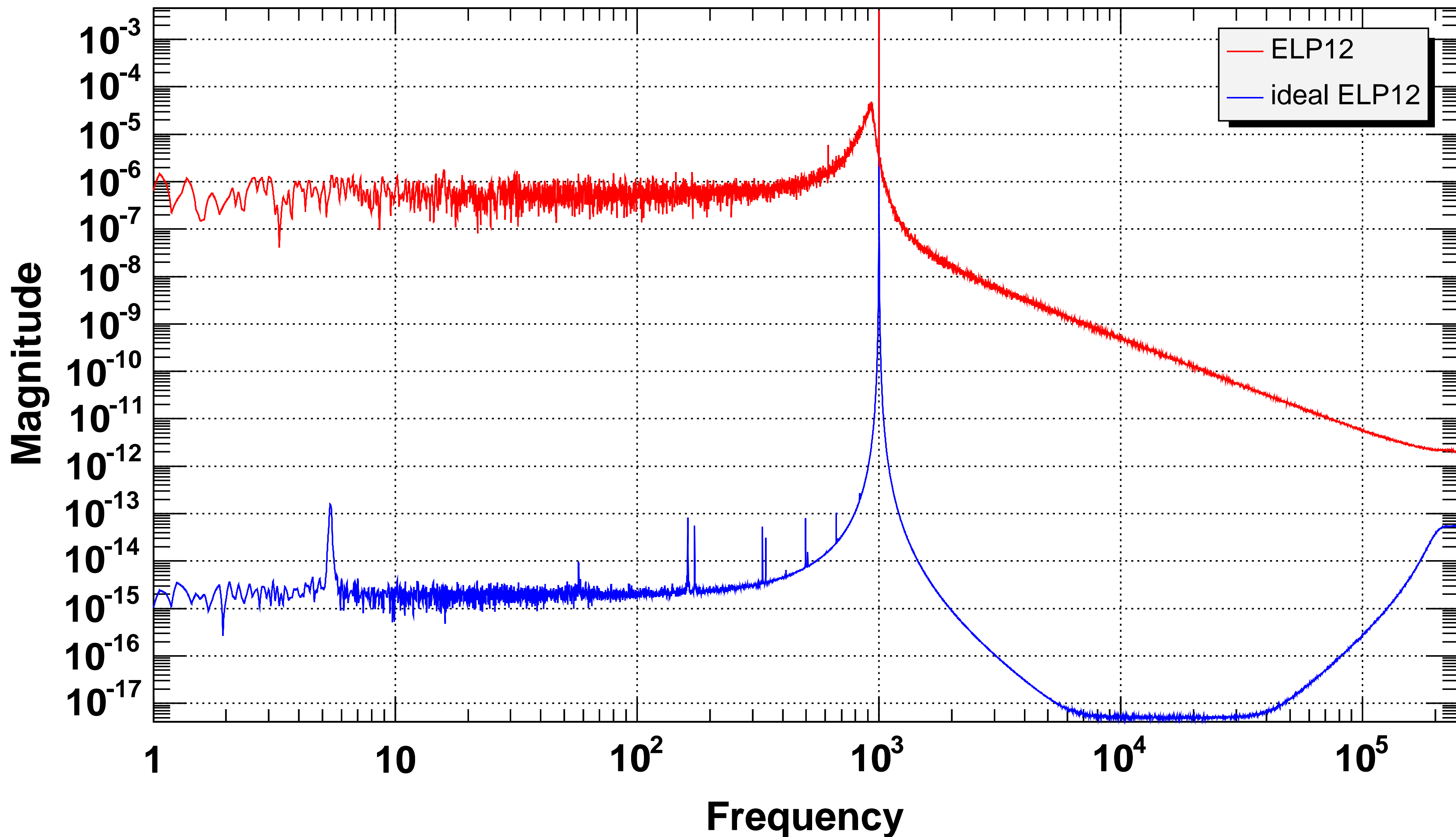
Power spectrum
ELP6
ideal ELP6
Magnitude
Frequency
T0=06/01/1980 00:00:00
Bin=419L

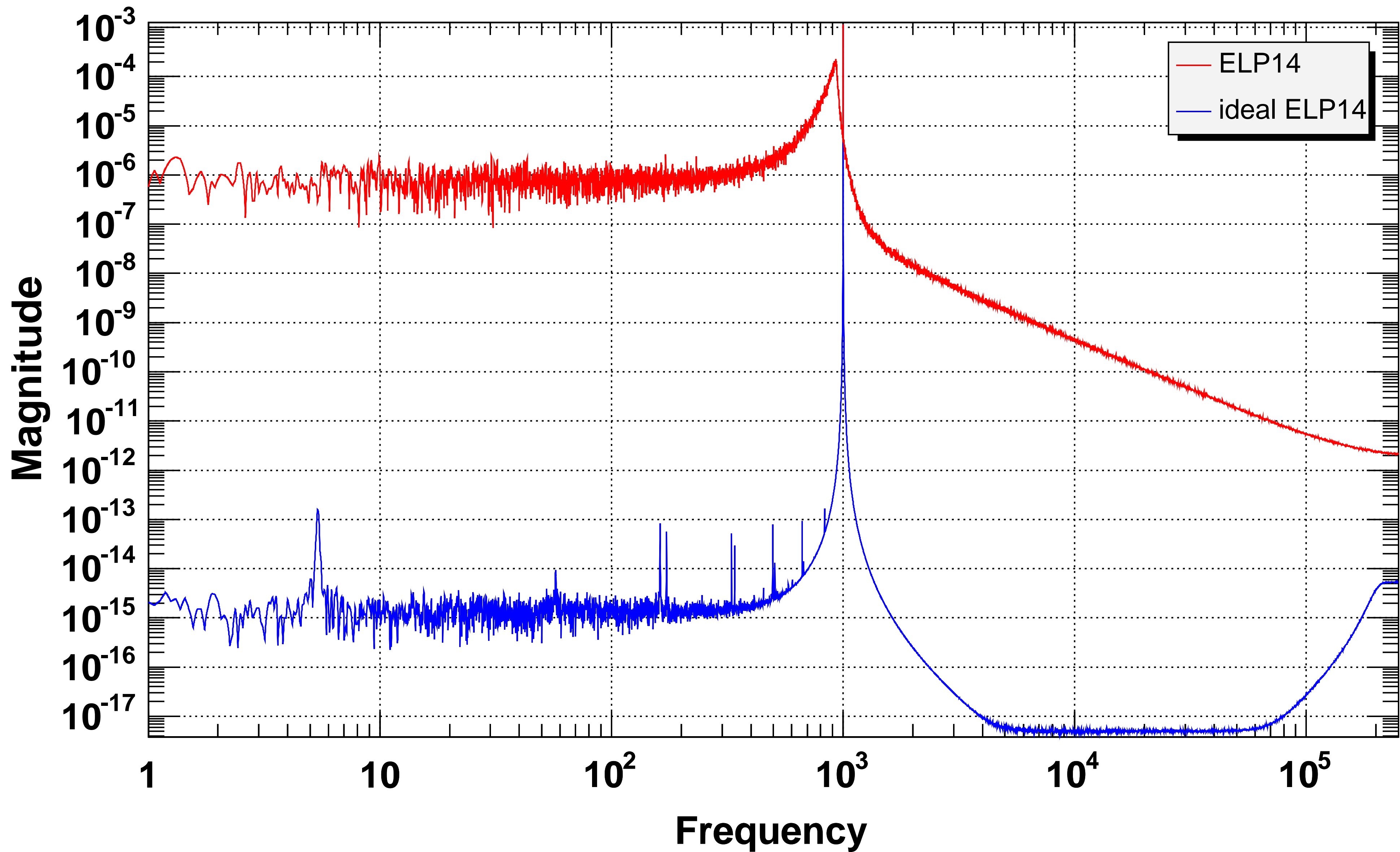


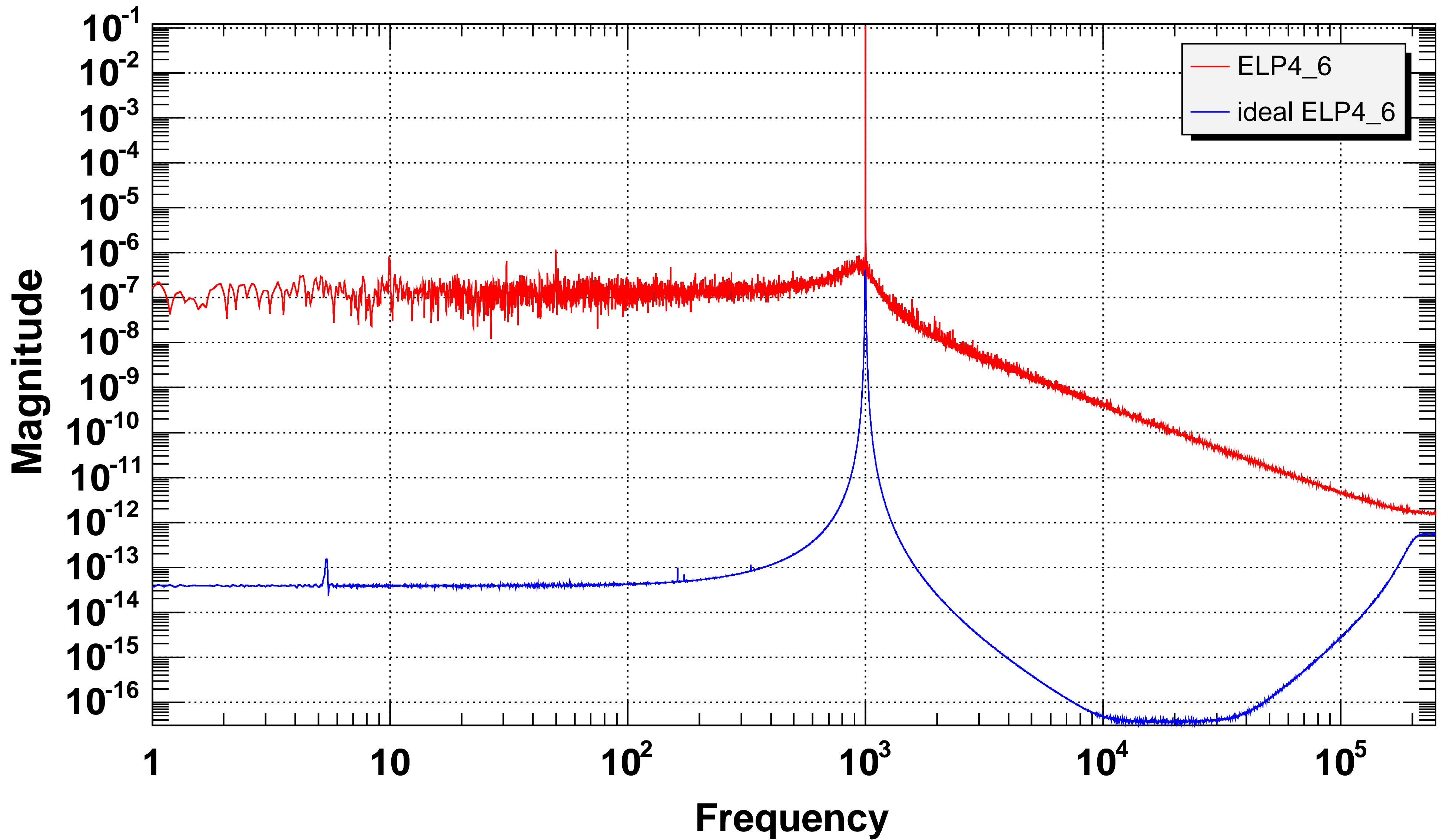
Power spectrum
ELP8
ideal ELP8
Magnitude
Frequency
T0=06/01/1980 00:00:00
Bin=419L

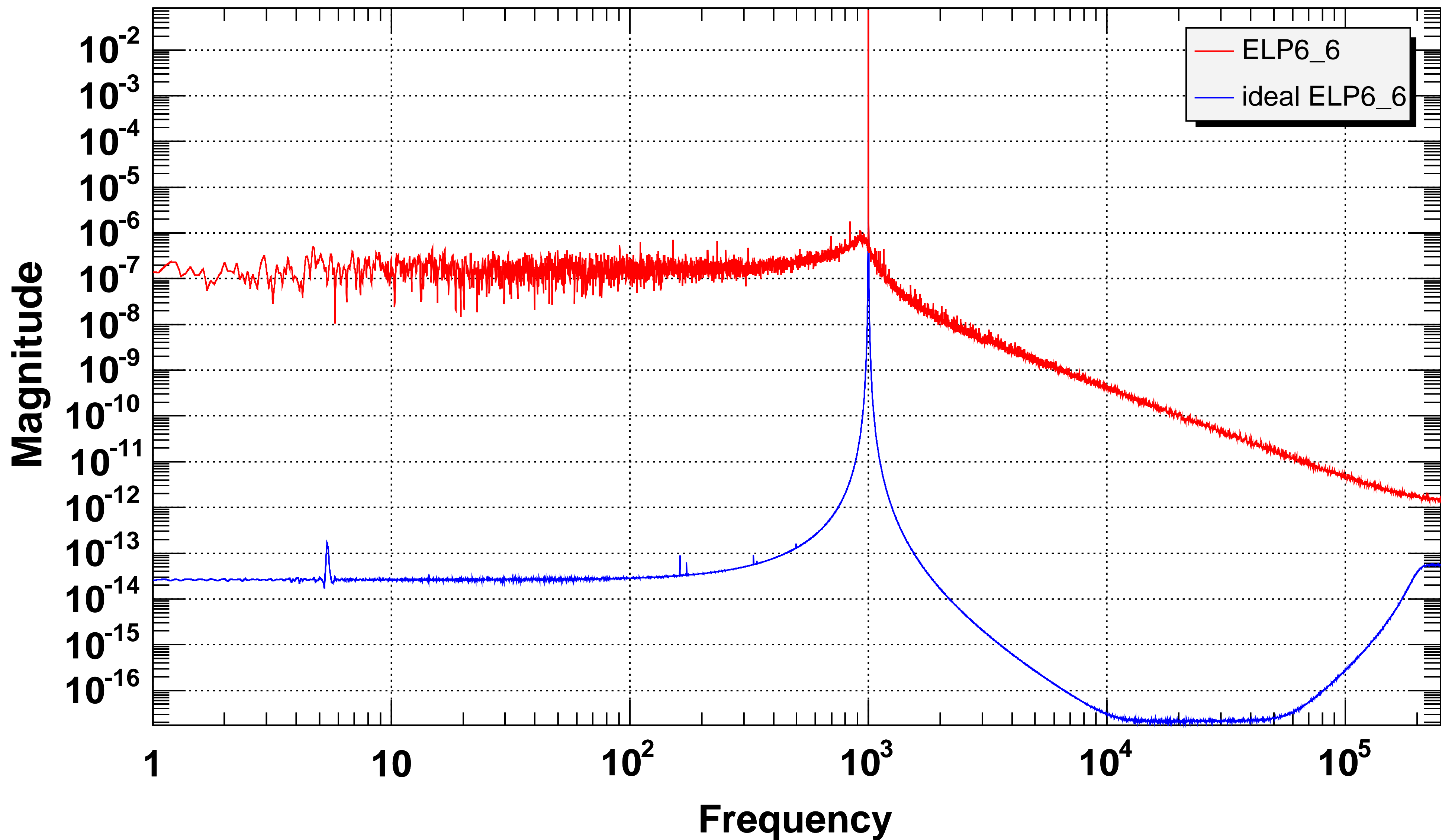


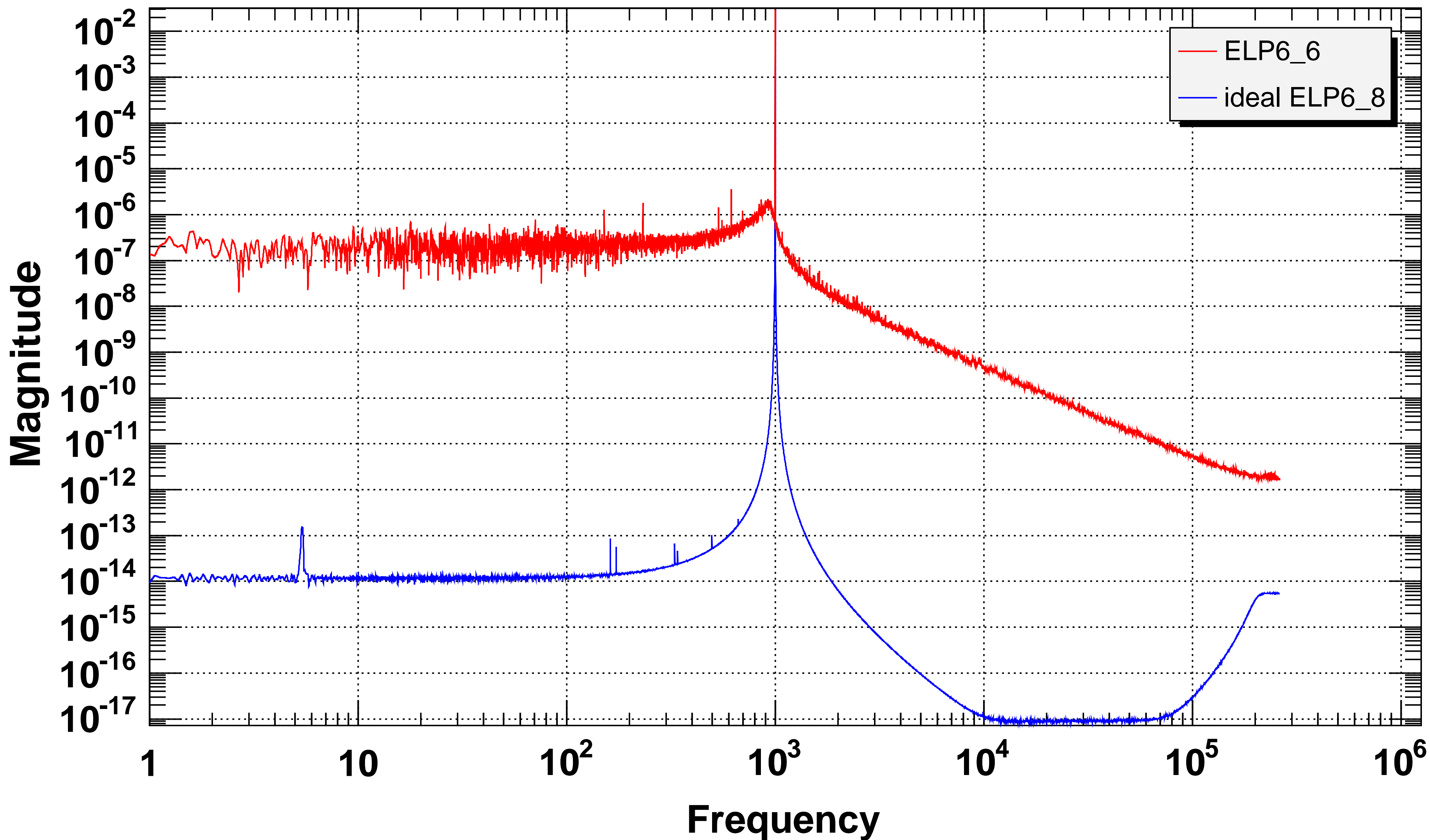
Power spectrum
ELP10
ideal ELP10
Magnitude
Frequency
T0=06/01/1980 00:00:00
Bin=419L


Power spectrum
ELP12
ideal ELP12
Magnitude
Frequency
T0=06/01/1980 00:00:00
Bin=419L


Power spectrum
ELP14
ideal ELP14
Magnitude
Frequency
T0=06/01/1980 00:00:00
Bin=419L


Power spectrum
ELP4_6
ideal ELP4_6
Magnitude
Frequency
T0=06/01/1980 00:00:00
Bin=419L


Power spectrum
ELP6_6
ideal ELP6_6
Magnitude
Frequency
1
10
10^2
10^3
10^4
10^5
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
10^{-7}
10^{-8}
10^{-9}
10^{-10}
10^{-11}
10^{-12}
10^{-13}
10^{-14}
10^{-15}
10^{-16}
T0=06/01/1980 00:00:00
Bin=419L



Power spectrum
ELP6_6
ideal ELP6_8
Magnitude
Frequency
T0=06/01/1980 00:00:00
Bin=419L

APPENDIX H DIRECT COMPARISON

Time series

Signal

3
2
1
0
-1
-2
-3

0 0.001 0.002 0.003 0.004 0.005 0.006 0.007 0.008 0.009 0.01

Time

fixed point (RTL)
fixed point (behaviour)
floating point (ideal)

Time series

Magnitude

$\times 10^{-6}$

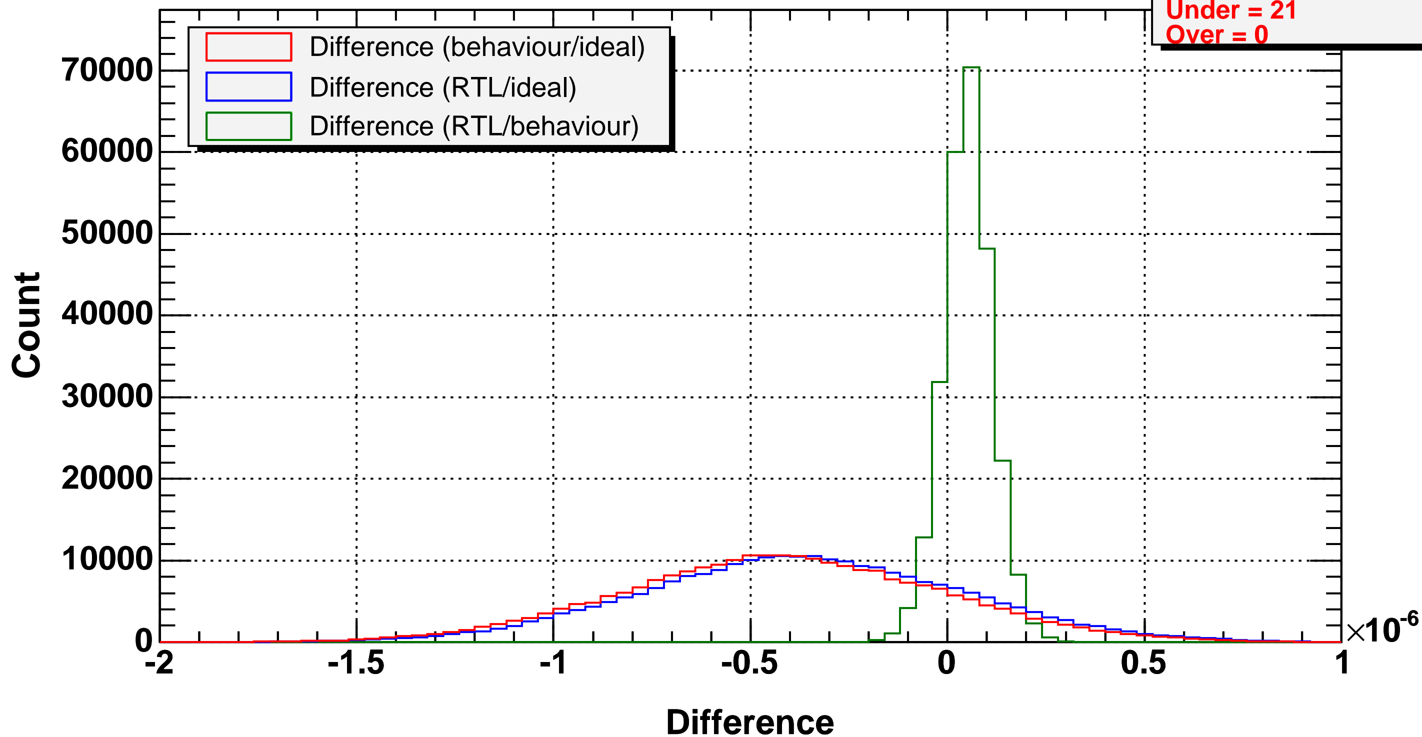
1
0.5
0
-0.5
-1
-1.5
-2

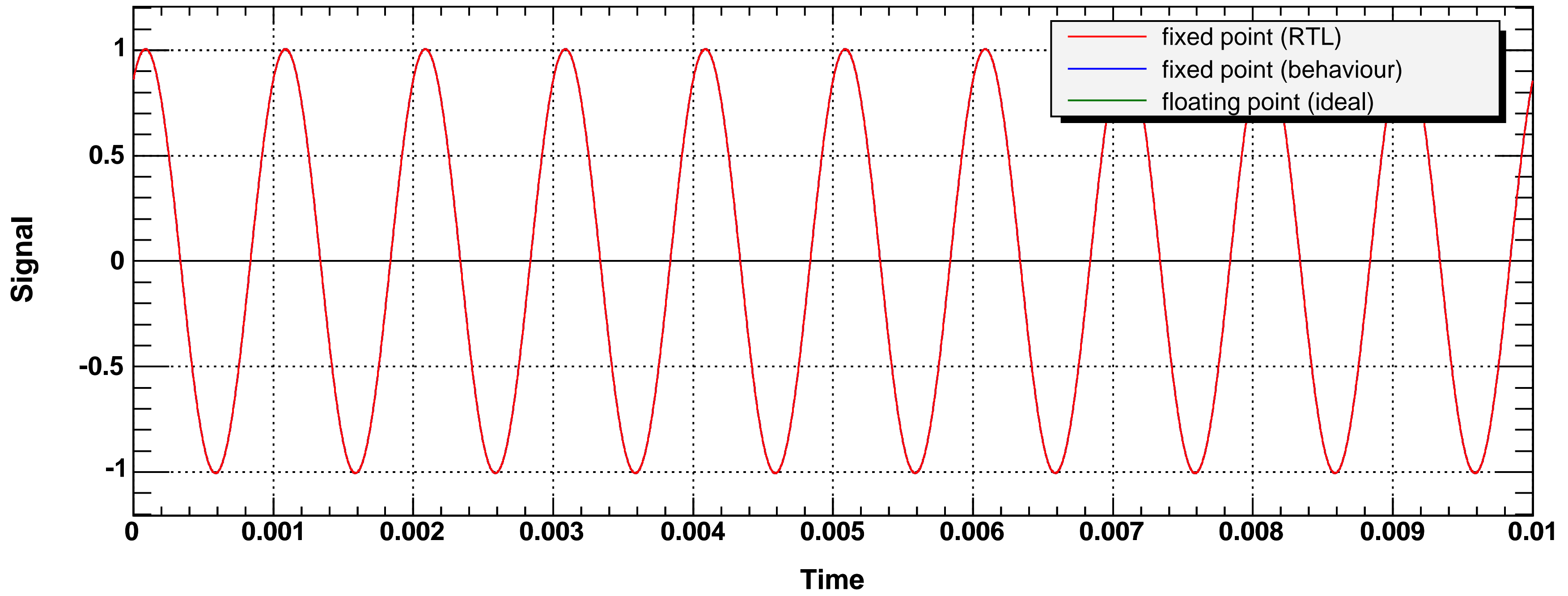
0 0.001 0.002 0.003 0.004 0.005 0.006 0.007 0.008 0.009 0.01

Time

Difference (RTL/ideal)
Difference (behaviour/ideal)
Difference (RTL/behaviour)

T0=06/01/1980 00:00:00

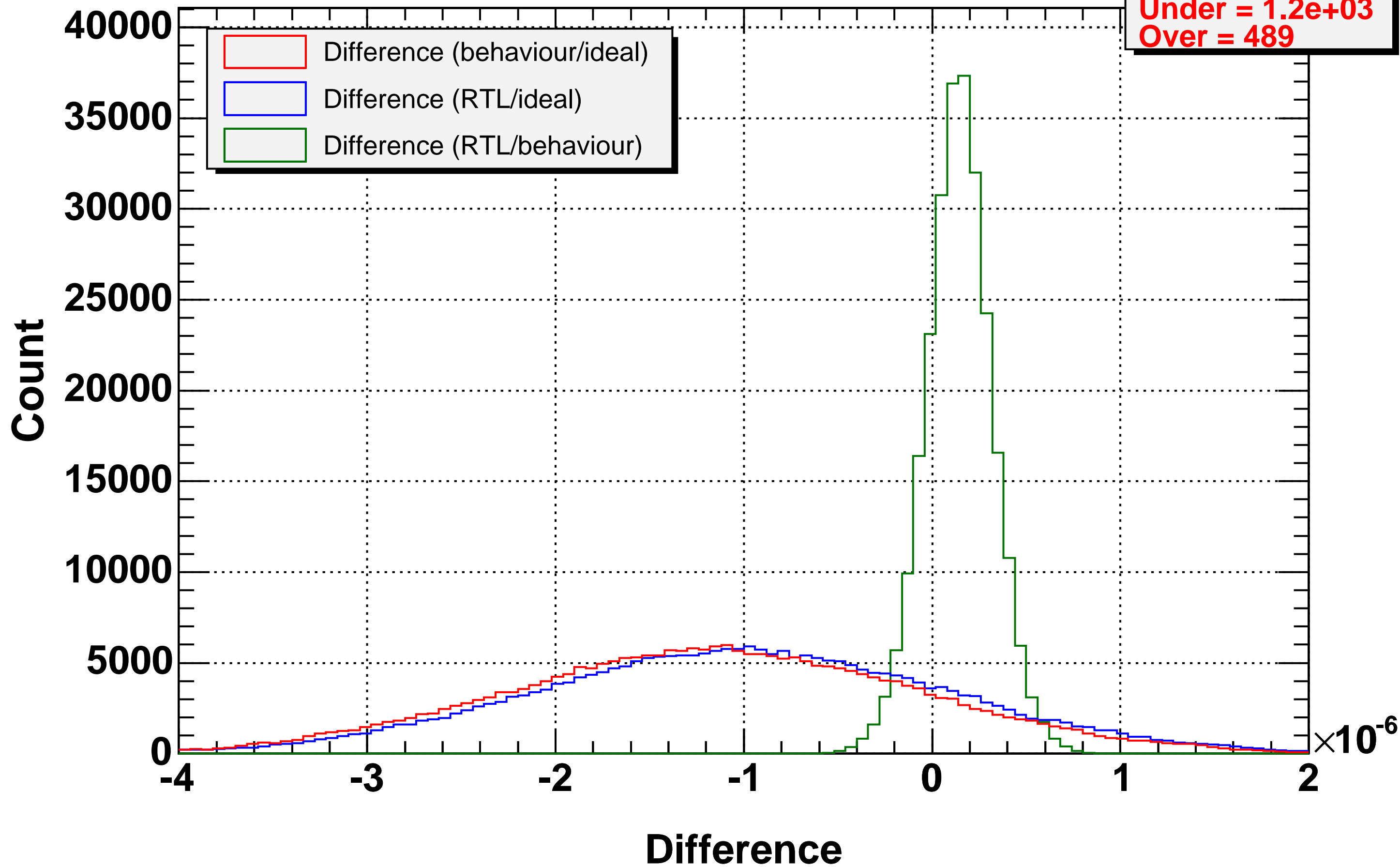

1-D Histogram
NEnt = 262144
Mean = -4.22e-07
SDev = 4.04e-07
Under = 21
Over = 0
Difference (behaviour/ideal)
Difference (RTL/ideal)
Difference (RTL/behaviour)
Count
70000
60000
50000
40000
30000
20000
10000
0
-2
-1.5
-1
-0.5
0
0.5
1
×10^-6
Difference
T0=06/01/1980 00:00:00



Time series
×10^-6
Difference (RTL/ideal)
Difference (behaviour/ideal)
Difference (RTL/behaviour)
Magnitude
3
2
1
0
-1
-2
-3
-4
0
0.001
0.002
0.003
0.004
0.005
0.006
0.007
0.008
0.009
0.01
Time

T0=06/01/1980 00:00:00

1-D Histogram



T0=06/01/1980 00:00:00