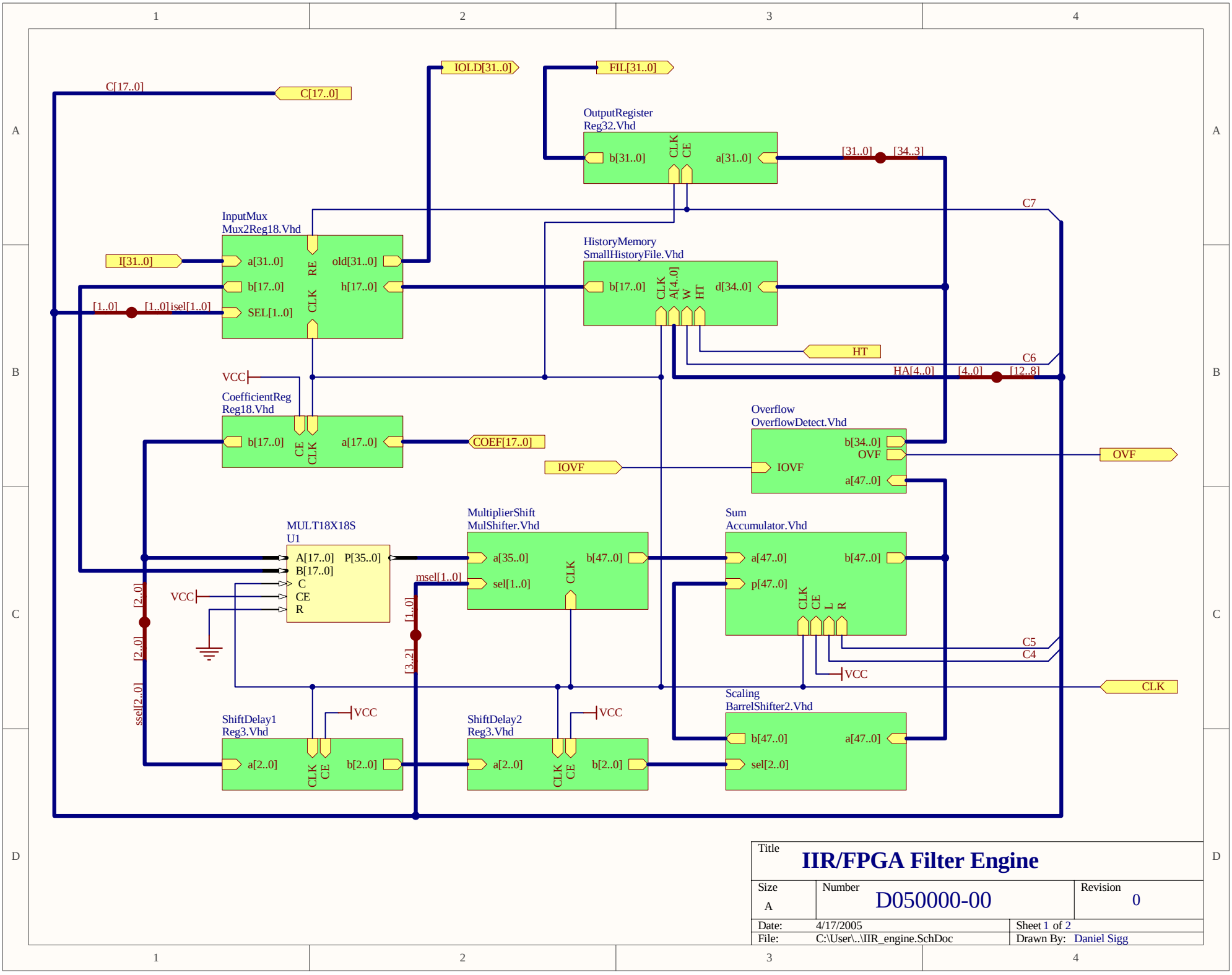
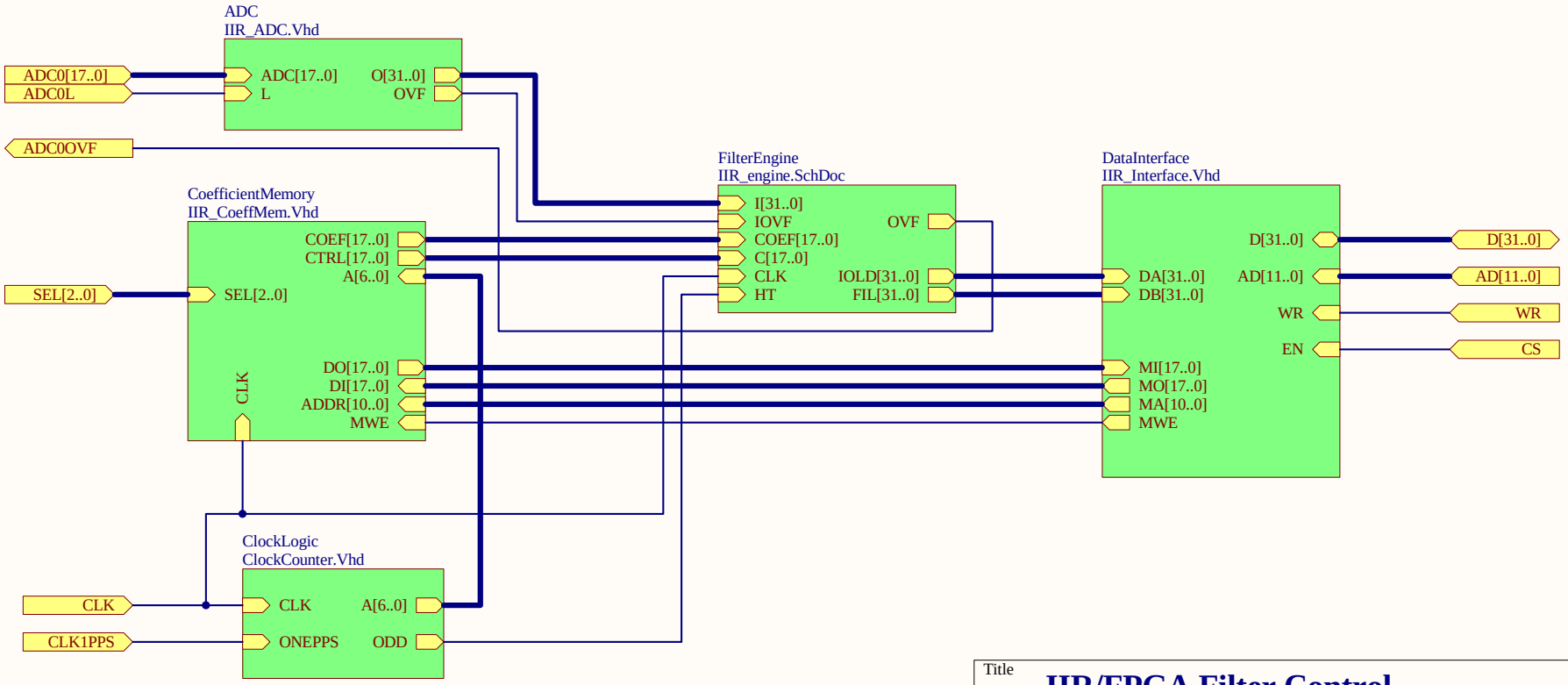
APPENDIX A SCHEMATICS AND CODE LISTINGS



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

C:\User\Daniel\Protel\FPGA_Biquad\IIR_ADC.Vhd

```
--------------------------------------------------------------------------------
-- 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;
--------------------------------------------------------------------------------
```

C:\User\Daniel\Protel\FPGA_Biquad\ClockCounter.Vhd

```
--------------------------------------------------------------------------------
-- 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;
--------------------------------------------------------------------------------
```

C:\User\Daniel\Protel\FPGA_Biquad\IIR_Interface.Vhd

```
--------------------------------------------------------------------------------
-- 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 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"091900004c0800004c2900004818000048l9ea4b0c093d2e0c29ea4b08193d2e";
          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"5c280000581800005819fa121c099ce91c29fa1218199ce9181b03ba5c491052";
          INIT_07 : bit_vector(255 downto 0) :=
             X"602800006018000001800001c0800001c280000181800001918000005c080000";
          INIT_08 : bit_vector(255 downto 0) :=
             X"601800006018000024080000242800002018000020180000644800006408000";
          INIT_09 : bit_vector(255 downto 0) :=
```

```
      X"6818000000180000240800002428000020180000211800006408000064280000";
   INIT_0a : bit_vector(255 downto 0) :=
      X"681800002c0800002c28000028180000281800006c4800006c08000068280000";
   INIT_0b : bit_vector(255 downto 0) :=
      X"001800002c0800002c28000028180000291800006c0800006c28000068180000";
   INIT_0c : bit_vector(255 downto 0) :=
      X"3408000034280000301800003018000074480000740800007028000070180000";
   INIT_0d : bit_vector(255 downto 0) :=
      X"3408000034280000301800003118000074080000742800007018000070180000";
   INIT_0e : bit_vector(255 downto 0) :=
      X"3c28000038180000381800007c4800007c080000782800007818000000180000";
   INIT_0f : bit_vector(255 downto 0) :=
      X"000193ab380000003b0000007c0800007c28000078180000781800003c080000";
   INIT_10 : bit_vector(255 downto 0) :=
      X"0019c569011900004408000044290000401800004015ad510084f9af0001ad51";
   INIT_11 : bit_vector(255 downto 0) :=
      X"081b10a74c49d67f4c0b10a74829d67f48180004001a040f0409c569042a040f";
   INIT_12 : bit_vector(255 downto 0) :=
      X"091900004c0800004c2900004818000048l9eeef0c0997c80c29eeef081997c8";
   INIT_13 : bit_vector(255 downto 0) :=
      X"5449a6df540b0bf45029a6df50180002001a052d0c09be250c2a052d0819be25";
   INIT_14 : bit_vector(255 downto 0) :=
      X"5408000054290000501800005019f301140852021429f30110185202101b0bf4";
   INIT_15 : bit_vector(255 downto 0) :=
      X"5c0b06625828af8858180002001a0a24140947b3142a0a24101947b311190000";
   INIT_16 : bit_vector(255 downto 0) :=
      X"5c290000581800005819f7e41c0917791c29f7e418191779181b06625c48af88";
   INIT_17 : bit_vector(255 downto 0) :=
      X"60293d0760180006001a46351c09b8991c2a46351819b899191900005c080000";
   INIT_18 : bit_vector(255 downto 0) :=
      X"601800006019fbf82408415f2429fbf82018415f201b01ed64493d07640b01ed";
   INIT_19 : bit_vector(255 downto 0) :=
      X"6818000000180000240800002428000020180000211800006408000064280000";
   INIT_1a : bit_vector(255 downto 0) :=
      X"681800002c0800002c28000028180000281800006c4800006c08000068280000";
   INIT_1b : bit_vector(255 downto 0) :=
      X"001800002c0800002c28000028180000291800006c0800006c28000068180000";
   INIT_1c : bit_vector(255 downto 0) :=
      X"3408000034280000301800003018000074480000740800007028000070180000";
   INIT_1d : bit_vector(255 downto 0) :=
      X"3408000034280000301800003118000074080000742800007018000070180000";
   INIT_1e : bit_vector(255 downto 0) :=
      X"3c28000038180000381800007c4800007c080000782800007818000000180000";
   INIT_1f : bit_vector(255 downto 0) :=
      X"0000f9af380000003b0000007c0800007c28000078180000781800003c080000";
   INIT_20 : bit_vector(255 downto 0) :=
      X"001965fb011900004408000044290000401800004015l0ea00853f21000160ea";
   INIT_21 : bit_vector(255 downto 0) :=
      X"081b0dd44c493ed04c0b0dd448293ed048180004001a039a040965fb042a039a";
   INIT_22 : bit_vector(255 downto 0) :=
      X"091900004c0800004c2900004818000048l9f1e50c0860340c29f1e508186034";
   INIT_23 : bit_vector(255 downto 0) :=
      X"54498ad2540b0af950298ad250180002001a04240c09694d0c2a04240819694d";
   INIT_24 : bit_vector(255 downto 0) :=
      X"5408000054290000501800005019f4471408db071429f4471018db07101b0af9";
   INIT_25 : bit_vector(255 downto 0) :=
      X"5c0b072158282bf458180002001a05de14092945142a05de1019294511190000";
   INIT_26 : bit_vector(255 downto 0) :=
      X"5c290000581800005819f7831c09c7a61c29f7831819c7a6181b07215c482bf4";
   INIT_27 : bit_vector(255 downto 0) :=
      X"6029d72660180003001a0c8c1c088d0e1c2a0c8c18188d0e191900005c080000";
   INIT_28 : bit_vector(255 downto 0) :=
      X"601800006019fa662409c0f62429fa662019c0f6201b03c56449d726640b03c5";
   INIT_29 : bit_vector(255 downto 0) :=
      X"68180006001a5a392409be57242a5a392019be57211900006408000064290000";
   INIT_2a : bit_vector(255 downto 0) :=
      X"6819fcc62c09cac22c29fcc62819cac2281b01286c491e766c0b012868291e76";
   INIT_2b : bit_vector(255 downto 0) :=
      X"001800002c0800002c28000028180000291800006c0800006c28000068180000";
   INIT_2c : bit_vector(255 downto 0) :=
      X"3408000034280000301800003018000074480000740800007028000070180000";
   INIT_2d : bit_vector(255 downto 0) :=
      X"3408000034280000301800003118000074080000742800007018000070180000";
   INIT_2e : bit_vector(255 downto 0) :=
```

```
            X"3c2800003818000038180000 7c4800007c0800007828000078180000 00180000";
         INIT_2f : bit_vector(255 downto 0) :=
            X"00013f21380000003b0000007c0800007c2800007818000078180000 3c080000";
         INIT_30 : bit_vector(255 downto 0) :=
            X"00185fd401190000440800004429000040180000401523680085 79ce00012368";
         INIT_31 : bit_vector(255 downto 0) :=
            X"081b0bd74c49372d4c0b0bd74829372d48180004001a035904085fd4042a0359";
         INIT_32 : bit_vector(255 downto 0) :=
            X"091900004c0800004c2900004818000048 19f3f50c0962a70c29f3f5081962a7";
         INIT_33 : bit_vector(255 downto 0) :=
            X"54494624540b09fa5029462450180002001a03a90c08f51a0c2a03a90818f51a";
         INIT_34 : bit_vector(255 downto 0) :=
            X"5408000054290000501800005019f57614081f991429f57610181f99101b09fa";
         INIT_35 : bit_vector(255 downto 0) :=
            X"5c0b073b5828e16a58180003001a04851409f340142a04851019f34011190000";
         INIT_36 : bit_vector(255 downto 0) :=
            X"5c2900005818000058 19f7af1c0916811c29f7af18191681181b073b5c48e16a";
         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"68180003001a0f79240955e3242a0f79201955e3211900006408000064290000";
         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"3408000034280000301800003118000074080000742800007018000070 19fd30";
         INIT_3e : bit_vector(255 downto 0) :=
            X"3c2800003818000038180000 7c4800007c0800007828000078180000 00180000";
         INIT_3f : bit_vector(255 downto 0) :=
            X"000179ce380000003b0000007c0800007c2800007818000078180000 3c080000"
      );
      -- 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
       "091900004c0800004c2900004818000048 19ea4b0c093d2e0c29ea4b08193d2e";
    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
```

```
      "540800005429000050180000 5019f2081409e2081429f2081019e208101b0c6b";
   attribute INIT_05 : string;
   attribute INIT_05 of U_RAM : label is
      "5c0b03ba5829105258180005001a35b11409b045142a35b11019b04511190000";
   attribute INIT_06 : string;
   attribute INIT_06 of U_RAM : label is
      "5c280000581800005819fa121c099ce91c29fa1218199ce9181b03ba5c491052";
   attribute INIT_07 : string;
   attribute INIT_07 of U_RAM : label is
      "602800006018000001800001c0800001c2800001818000019180000 5c080000";
   attribute INIT_08 : string;
   attribute INIT_08 of U_RAM : label is
      "601800006018000024080000242800002018000020180000644800006 4080000";
   attribute INIT_09 : string;
   attribute INIT_09 of U_RAM : label is
      "681800000018000024080000242800002018000021180000640800006 4280000";
   attribute INIT_0a : string;
   attribute INIT_0a of U_RAM : label is
      "681800002c0800002c28000028180000281800006c4800006c08000068280000";
   attribute INIT_0b : string;
   attribute INIT_0b of U_RAM : label is
      "001800002c0800002c28000028180000291800006c0800006c28000068180000";
   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
      "3c28000038180000381800007c4800007c080000782800007818000000180000";
   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
      "091900004c0800004c2900004818000048 19eeef0c0997c80c29eeef081997c8";
   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
      "540800005429000050180000 5019f301140852021429f30110185202101b0bf4";
   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
      "5c290000581800005819f7e41c0917791c29f7e418191779181b06625c48af88";
   attribute INIT_17 : string;
   attribute INIT_17 of U_RAM : label is
      "60293d0760180006001a46351c09b8991c2a46351819b899191900005c080000";
   attribute INIT_18 : string;
   attribute INIT_18 of U_RAM : label is
      "601800006019fbf82408415f2429fbf82018415f201b01ed64493d07640b01ed";
   attribute INIT_19 : string;
   attribute INIT_19 of U_RAM : label is
      "681800000018000024080000242800002018000021180000640800006 4280000";
   attribute INIT_1a : string;
   attribute INIT_1a of U_RAM : label is
      "681800002c0800002c28000028180000281800006c4800006c08000068280000";
   attribute INIT_1b : string;
   attribute INIT_1b of U_RAM : label is
      "001800002c0800002c28000028180000291800006c0800006c28000068180000";
   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
       "340800003428000030180000311800007408000074280000701800007018000";
    attribute INIT_1e : string;
    attribute INIT_1e of U_RAM : label is
       "3c28000038180000381800007c4800007c0800007828000078180000000180000";
    attribute INIT_1f : string;
    attribute INIT_1f of U_RAM : label is
       "0000f9af380000003b0000007c0800007c2800007818000078180000 3c080000";
    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
       "091900004c0800004c2900004818000048 19f1e50c0860340c29f1e508186034";
    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
       "5c2900005818000058 19f7831c09c7a61c29f7831819c7a6181b07215c482bf4";
    attribute INIT_27 : string;
    attribute INIT_27 of U_RAM : label is
       "6029d72660180003001a0c8c1c088d0e1c2a0c8c18188d0e191900005c080000";
    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
       "68180006001a5a392409be57242a5a392019be5721190000640800006429 0000";
    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
       "001800002c0800002c280000281800002918000 06c0800006c28000068180000";
    attribute INIT_2c : string;
    attribute INIT_2c of U_RAM : label is
       "340800003428000030180000301800007448000074080000702800007018 0000";
    attribute INIT_2d : string;
    attribute INIT_2d of U_RAM : label is
       "340800003428000030180000311800007408000074280000701800007018 0000";
    attribute INIT_2e : string;
    attribute INIT_2e of U_RAM : label is
       "3c28000038180000381800007c4800007c0800007828000078180000000180000";
    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
       "00185fd401190000440800004429000040180000401523680085 79ce00012368";
    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
       "091900004c0800004c2900004818000048 19f3f50c0962a70c29f3f5081962a7";
    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
       "5c2900005818000058l9f7af1c0916811c29f7af18191681181b073b5c48e16a";
    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
       "340800003428000030180000311800007408000074280000701800007019fd30";
    attribute INIT_3e : string;
    attribute INIT_3e of U_RAM : label is
       "3c2800003818000381800007c4800007c0800007828000078180000000180000";
    attribute INIT_3f : string;
    attribute INIT_3f of U_RAM : label is
       "000179ce380000003b0000007c0800007c2800007818000078180000 3c080000";

    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;
-------------------------------------------------------------------------------
```

C:\User\Daniel\Protel\FPGA_Biquad\Mux2Reg18.Vhd

```
--------------------------------------------------------------------------------
-- 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;
--------------------------------------------------------------------------------
```

C:\User\Daniel\Protel\FPGA_Biquad\MulShifter.Vhd

```
--------------------------------------------------------------------------------
-- 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;
--------------------------------------------------------------------------------
```

C:\User\Daniel\Protel\FPGA_Biquad\Accumulator.Vhd

```
--------------------------------------------------------------------------------
-- 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;
```

C:\User\Daniel\Protel\FPGA_Biquad\OverflowDetect.Vhd

```
--------------------------------------------------------------------------------
-- 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;
--------------------------------------------------------------------------------
```

C:\User\Daniel\Protel\FPGA_Biquad\Reg32.Vhd

```
--------------------------------------------------------------------------------
-- 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;
--------------------------------------------------------------------------------
```

C:\User\Daniel\Protel\FPGA_Biquad\Reg18.Vhd

```
--------------------------------------------------------------------------------
-- 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;
--------------------------------------------------------------------------------
```

C:\User\Daniel\Protel\FPGA_Biquad\Reg3.Vhd

```
--------------------------------------------------------------------------------
-- 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);
      x1r <= r;
      x1 <= convert_adc_value (r, 2.0**(adc_type'length-2));
   end loop;
   file_close (input);
   -- done
   wait;
end process stimulus_file;


-------------------------------------------------------------------------------
-- Filter instantination
-------------------------------------------------------------------------------
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;
```

```
#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 00000000000000000", // code   0: g, msb
  "00000 00000 0010 0001 00000000000000000", // code   1: g, lsb
  "00000 10000 0000 0101 00000000000000000", // code   2: g, msb

  "00000 10000 0000 0110 00000000000000000", // code   3: b20, lsb
  "00000 10001 0000 1010 00000000000000000", // code   4: b20, msb
  "00000 10001 0000 0010 00000000000000000", // code   5: b20, lsb
  "00000 00000 0100 0110 00000000000000000", // code   6: b20, msb
  "00000 00000 0000 0110 00000000000000000", // code   7: b10, lsb
  "00000 00001 0000 1010 00000000000000000", // code   8: b10, msb
  "00000 00001 0000 0010 00000000000000000", // code   9: b10, lsb
  "00000 00000 0000 0110 00000000000000000", // code  10: b10, msb

  "00000 10010 0000 0110 00000000000000000", // code  11: -lb(c00)
```

```
  "00000 10010 0000 1010 00000000000000000", // code  12: a20, lsb
  "00000 10011 0000 0010 00000000000000000", // code  13: a20, msb
  "00000 10011 0001 0010 00000000000000000", // code  14: a20, lsb
  "00000 00010 0000 0110 00000000000000000", // code  15: a20, msb
  "00000 00010 0000 0110 00000000000000000", // code  16: a10, lsb
  "00000 00011 0000 1010 00000000000000000", // code  17: a10, msb
  "00000 00011 0000 0010 00000000000000000", // code  18: a10, lsb
  "00000 10010 0000 0110 00000000000000000", // code  19: a10, msb

  "00000 10010 0000 0110 00000000000000000", // code  20: b21, lsb
  "00000 10011 0000 1010 00000000000000000", // code  21: b21, msb
  "00000 10011 0000 0010 00000000000000000", // code  22: b21, lsb
  "00000 00010 0100 0110 00000000000000000", // code  23: b21, msb
  "00000 00010 0000 0110 00000000000000000", // code  24: b11, lsb
  "00000 00011 0000 1010 00000000000000000", // code  25: b11, msb
  "00000 00011 0000 0010 00000000000000000", // code  26: b11, lsb
  "00000 00000 0000 0110 00000000000000000", // code  27: b11, msb

  "00000 10100 0000 0110 00000000000000000", // code  28: -lb(c01)
  "00000 10100 0000 1010 00000000000000000", // code  29: a21, lsb
  "00000 10101 0000 0010 00000000000000000", // code  30: a21, msb
  "00000 10101 0001 0010 00000000000000000", // code  31: a21, lsb
  "00000 00100 0000 0110 00000000000000000", // code  32: a21, msb
  "00000 00100 0000 0110 00000000000000000", // code  33: a11, lsb
  "00000 00101 0000 1010 00000000000000000", // code  34: a11, msb
  "00000 00101 0000 0010 00000000000000000", // code  35: a11, lsb
  "00000 10100 0000 0110 00000000000000000", // code  36: a11, msb

  "00000 10100 0000 0110 00000000000000000", // code  37: b22, lsb
  "00000 10101 0000 1010 00000000000000000", // code  38: b22, msb
  "00000 10101 0000 0010 00000000000000000", // code  39: b22, lsb
  "00000 00100 0100 0110 00000000000000000", // code  40: b22, msb
  "00000 00100 0000 0110 00000000000000000", // code  41: b12, lsb
  "00000 00101 0000 1010 00000000000000000", // code  42: b12, msb
  "00000 00101 0000 0010 00000000000000000", // code  43: b12, lsb
  "00000 00000 0000 0110 00000000000000000", // code  44: b12, msb

  "00000 10110 0000 0110 00000000000000000", // code  45: -lb(c02)
  "00000 10110 0000 1010 00000000000000000", // code  46: a22, lsb
  "00000 10111 0000 0010 00000000000000000", // code  47: a22, msb
  "00000 10111 0001 0010 00000000000000000", // code  48: a22, lsb
  "00000 00110 0000 0110 00000000000000000", // code  49: a22, msb
  "00000 00110 0000 0110 00000000000000000", // code  50: a12, lsb
  "00000 00111 0000 1010 00000000000000000", // code  51: a12, msb
  "00000 00111 0000 0010 00000000000000000", // code  52: a12, lsb
  "00000 10110 0000 0110 00000000000000000", // code  53: a12, msb

  "00000 10110 0000 0110 00000000000000000", // code  54: b23, lsb
  "00000 10111 0000 1010 00000000000000000", // code  55: b23, msb
  "00000 10111 0000 0010 00000000000000000", // code  56: b23, lsb
  "00000 00110 0100 0110 00000000000000000", // code  57: b23, msb
  "00000 00110 0000 0110 00000000000000000", // code  58: b13, lsb
  "00000 00111 0000 1010 00000000000000000", // code  59: b13, msb
  "00000 00111 0000 0010 00000000000000000", // code  60: b13, lsb
```

```
   "00000 00000 0000 0110 00000000000000000", // code  61: b13, msb

   "00000 11000 0000 0110 00000000000000000", // code  62: -lb(c03)
   "00000 11000 0000 1010 00000000000000000", // code  63: a23, lsb
   "00000 11001 0000 0010 00000000000000000", // code  64: a23, msb
   "00000 11001 0001 0010 00000000000000000", // code  65: a23, lsb
   "00000 01000 0000 0110 00000000000000000", // code  66: a23, msb
   "00000 01000 0000 0110 00000000000000000", // code  67: a13, lsb
   "00000 01001 0000 1010 00000000000000000", // code  68: a13, msb
   "00000 01001 0000 0010 00000000000000000", // code  69: a13, lsb
   "00000 11000 0000 0110 00000000000000000", // code  70: a13, msb

   "00000 11000 0000 0110 00000000000000000", // code  71: b24, lsb
   "00000 11001 0000 1010 00000000000000000", // code  72: b24, msb
   "00000 11001 0000 0010 00000000000000000", // code  73: b24, lsb
   "00000 01000 0100 0110 00000000000000000", // code  74: b24, msb
   "00000 01000 0000 0110 00000000000000000", // code  75: b14, lsb
   "00000 01001 0000 1010 00000000000000000", // code  76: b14, msb
   "00000 01001 0000 0010 00000000000000000", // code  77: b14, lsb
   "00000 00000 0000 0110 00000000000000000", // code  78: b14, msb

   "00000 11010 0000 0110 00000000000000000", // code  79: -lb(c04)
   "00000 11010 0000 1010 00000000000000000", // code  80: a24, lsb
   "00000 11011 0000 0010 00000000000000000", // code  81: a24, msb
   "00000 11011 0001 0010 00000000000000000", // code  82: a24, lsb
   "00000 01010 0000 0110 00000000000000000", // code  83: a24, msb
   "00000 01010 0000 0110 00000000000000000", // code  84: a14, lsb
   "00000 01011 0000 1010 00000000000000000", // code  85: a14, msb
   "00000 01011 0000 0010 00000000000000000", // code  86: a14, lsb
   "00000 11010 0000 0110 00000000000000000", // code  87: a14, msb

   "00000 11010 0000 0110 00000000000000000", // code  88: b25, lsb
   "00000 11011 0000 1010 00000000000000000", // code  89: b25, msb
   "00000 11011 0000 0010 00000000000000000", // code  90: b25, lsb
   "00000 01010 0100 0110 00000000000000000", // code  91: b25, msb
   "00000 01010 0000 0110 00000000000000000", // code  92: b15, lsb
   "00000 01011 0000 1010 00000000000000000", // code  93: b15, msb
   "00000 01011 0000 0010 00000000000000000", // code  94: b15, lsb
   "00000 00000 0000 0110 00000000000000000", // code  95: b15, msb

   "00000 11100 0000 0110 00000000000000000", // code  96: -lb(c05)
   "00000 11100 0000 1010 00000000000000000", // code  97: a25, lsb
   "00000 11101 0000 0010 00000000000000000", // code  98: a25, msb
   "00000 11101 0001 0010 00000000000000000", // code  99: a25, lsb
   "00000 01100 0000 0110 00000000000000000", // code 100: a25, msb
   "00000 01100 0000 0110 00000000000000000", // code 101: a15, lsb
   "00000 01101 0000 1010 00000000000000000", // code 102: a15, msb
   "00000 01101 0000 0010 00000000000000000", // code 103: a15, lsb
   "00000 11100 0000 0110 00000000000000000", // code 104: a15, msb

   "00000 11100 0000 0110 00000000000000000", // code 105: b26, lsb
   "00000 11101 0000 1010 00000000000000000", // code 106: b26, msb
   "00000 11101 0000 0010 00000000000000000", // code 107: b26, lsb
   "00000 01100 0100 0110 00000000000000000", // code 108: b26, msb
```

```
  "00000 01100 0000 0110 0000000000000000", // code 109: b16, lsb
  "00000 01101 0000 1010 0000000000000000", // code 110: b16, msb
  "00000 01101 0000 0010 0000000000000000", // code 111: b16, lsb
  "00000 00000 0000 0110 0000000000000000", // code 112: b16, msb

  "00000 11110 0000 0110 0000000000000000", // code 113: -lb(c06)
  "00000 11110 0000 1010 0000000000000000", // code 114: a26, lsb
  "00000 11111 0000 0010 0000000000000000", // code 115: a26, msb
  "00000 11111 0001 0010 0000000000000000", // code 116: a26, lsb
  "00000 01110 0000 0110 0000000000000000", // code 117: a26, msb
  "00000 01110 0000 0110 0000000000000000", // code 118: a16, lsb
  "00000 01111 0000 1010 0000000000000000", // code 119: a16, msb
  "00000 01111 0000 0010 0000000000000000", // code 120: a16, lsb
  "00000 11110 0000 0110 0000000000000000", // code 121: a16, msb

  "00000 11110 0000 0110 0000000000000000", // code 122: --
  "00000 11111 0000 1010 0000000000000000", // code 123: --
  "00000 11111 0000 0010 0000000000000000", // code 124: --
  "00000 01110 1100 0000 0000000000000000", // code 125: --
  "00000 01110 0000 0000 0000000000000000", // code 126: --
  "00000 00000 0000 0000 0000000000000000"  // 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 = 0x3FFFFFFFULL;
  }
  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;
```

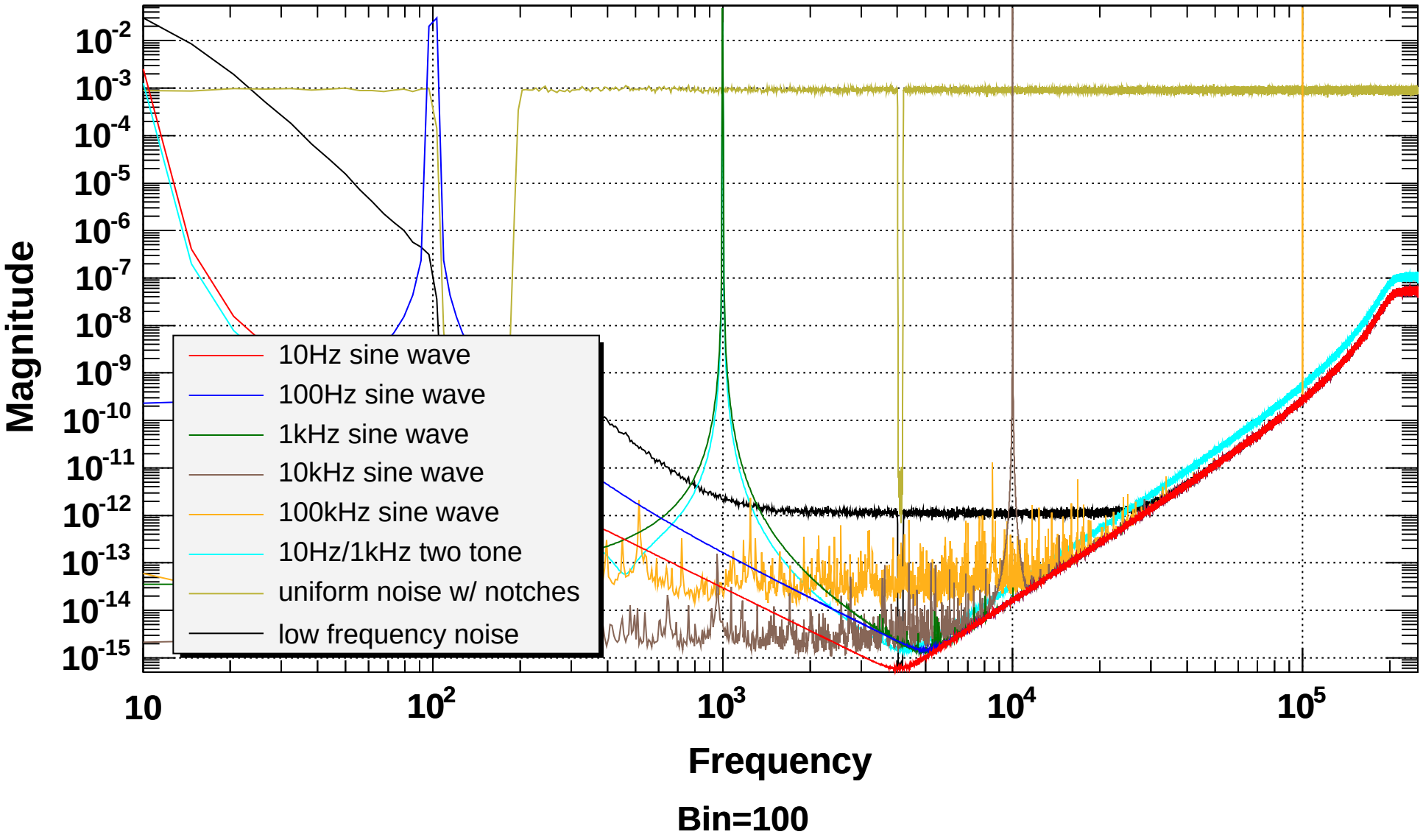
```
    if (!ff.read (argv[1])) {
      cerr << "Unable to open filter file " << argv[1] << endl;
      return 1;
    }

    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;
```

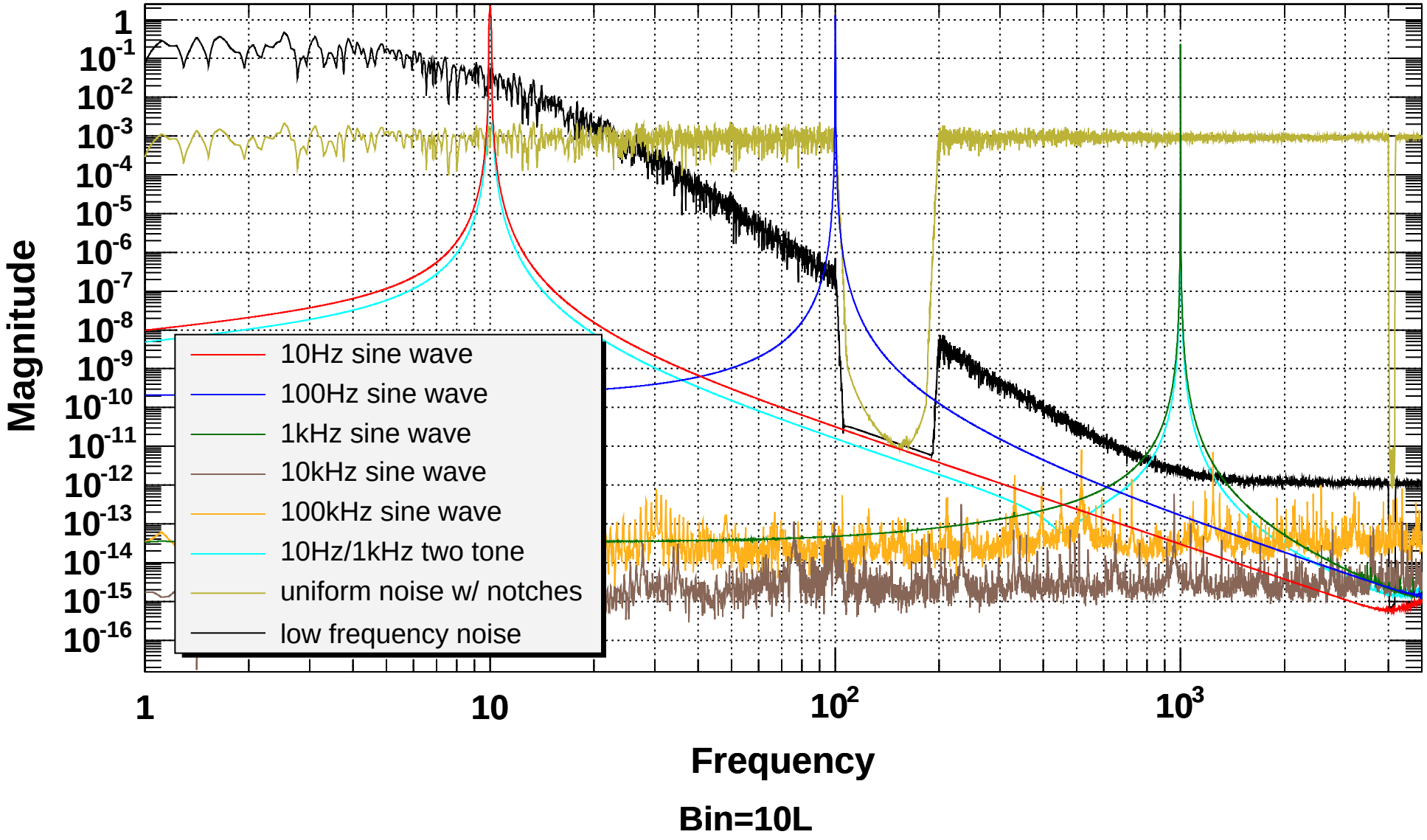
```
    cout << "           X\"";
    for (int j = 7; j >= 0; --j) {
      cout << setfill('0') << setw(8) << hex <<
          coeffs[i / 16][8*(i % 16)+j];
    }
    cout << "\";" << endl;
  }
  cout << endl << endl;

  // 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

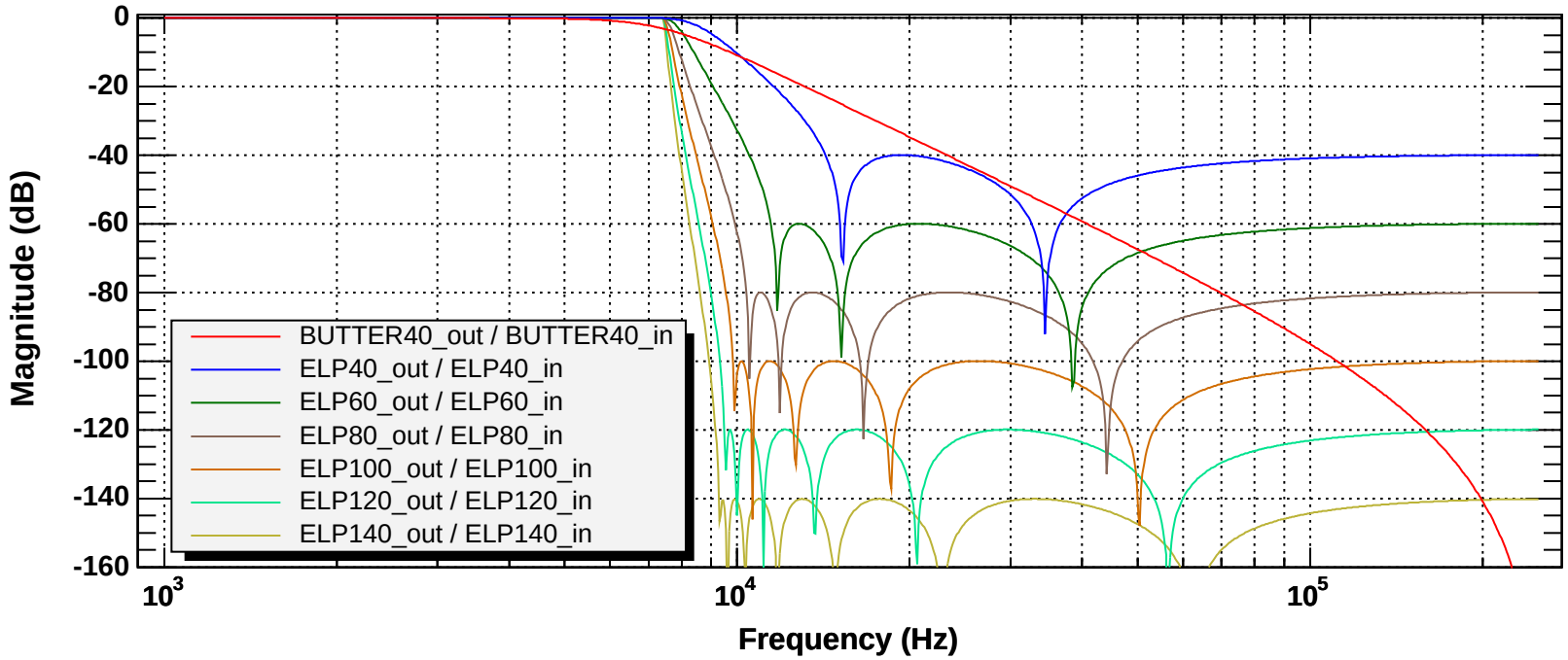


Stimulus Waveforms

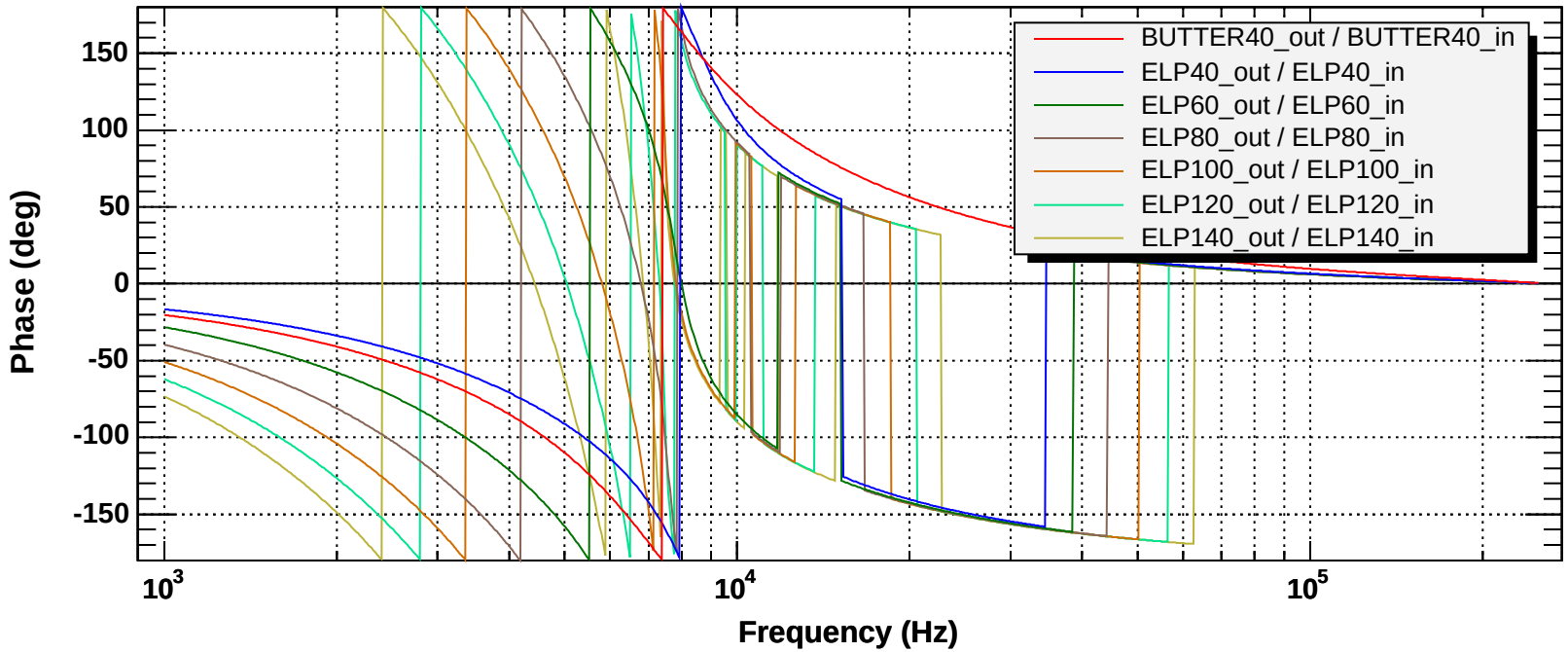


APPENDIX C FILTER TRANSFER FUNCTIONS

Transfer function



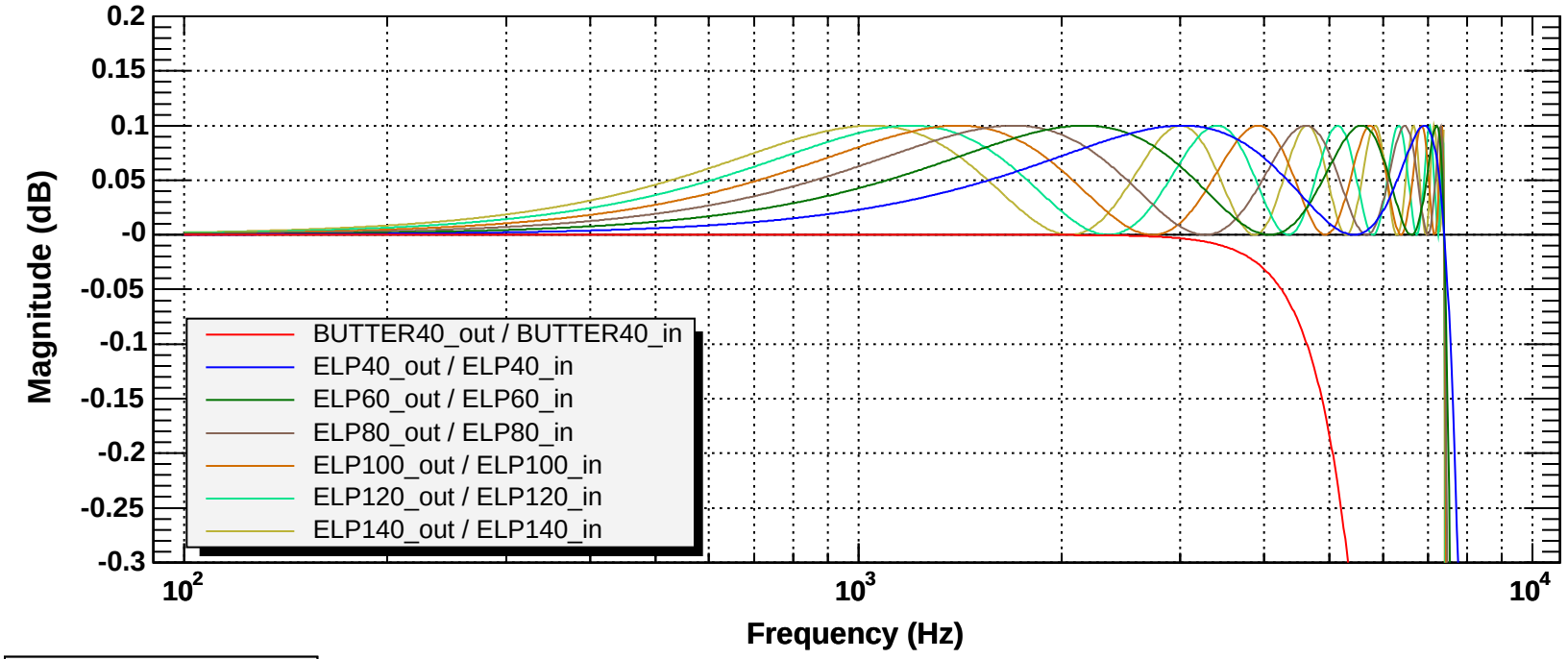
Transfer function



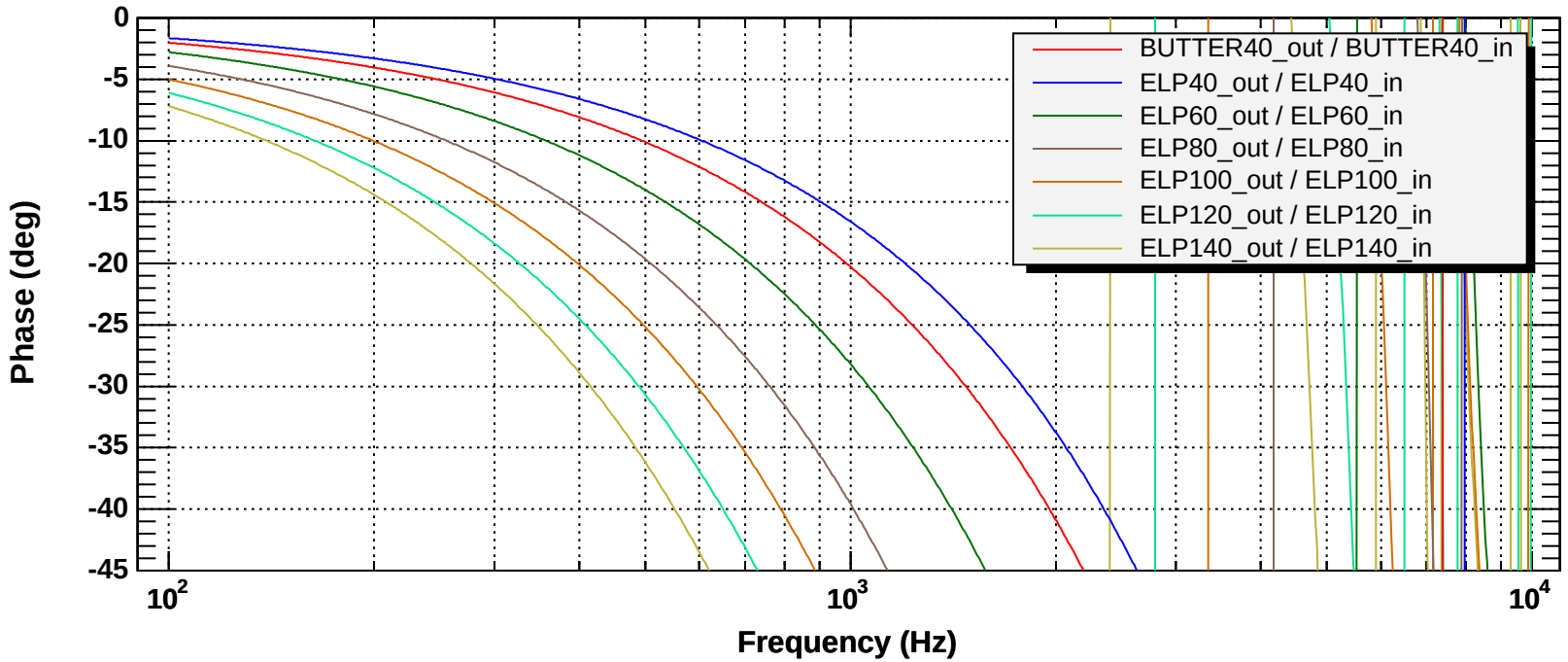
*T0=07/04/2005 19:03:56

iir_coeff.txt

Transfer function



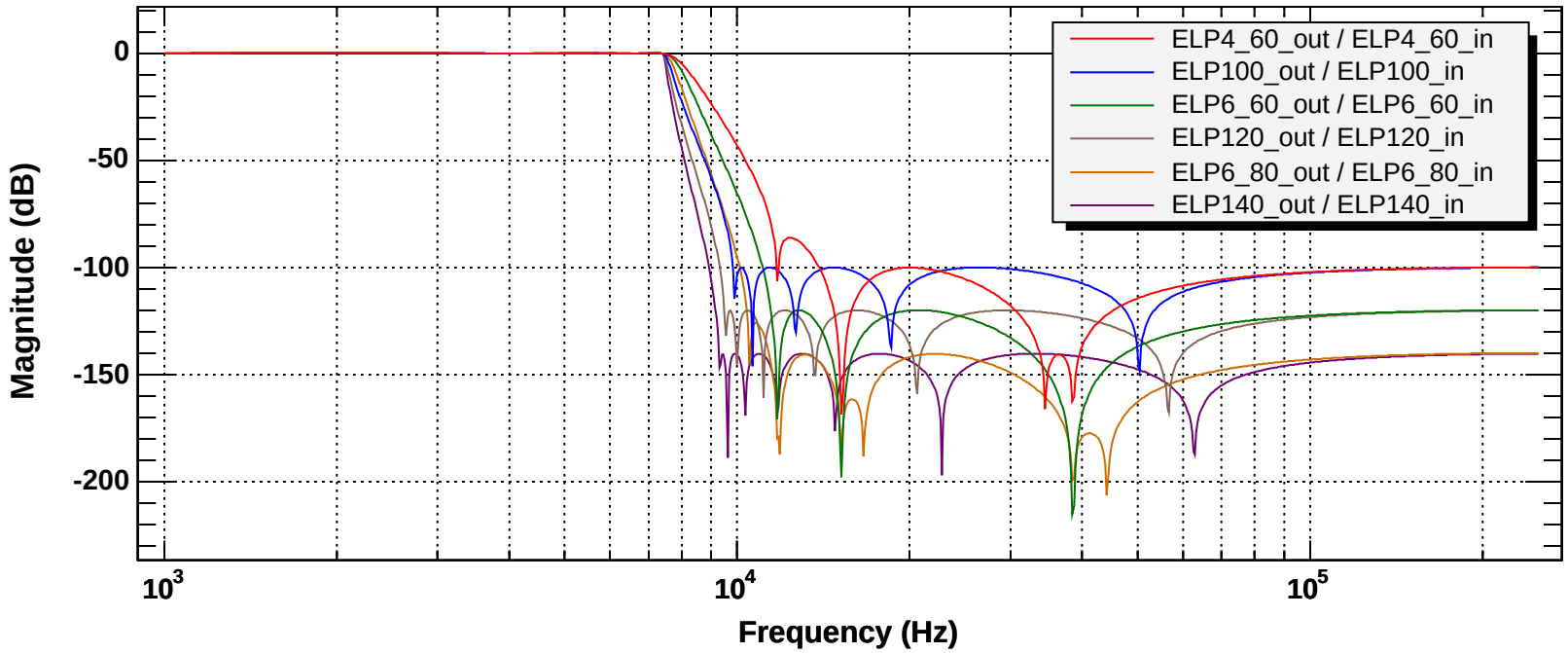
Transfer function



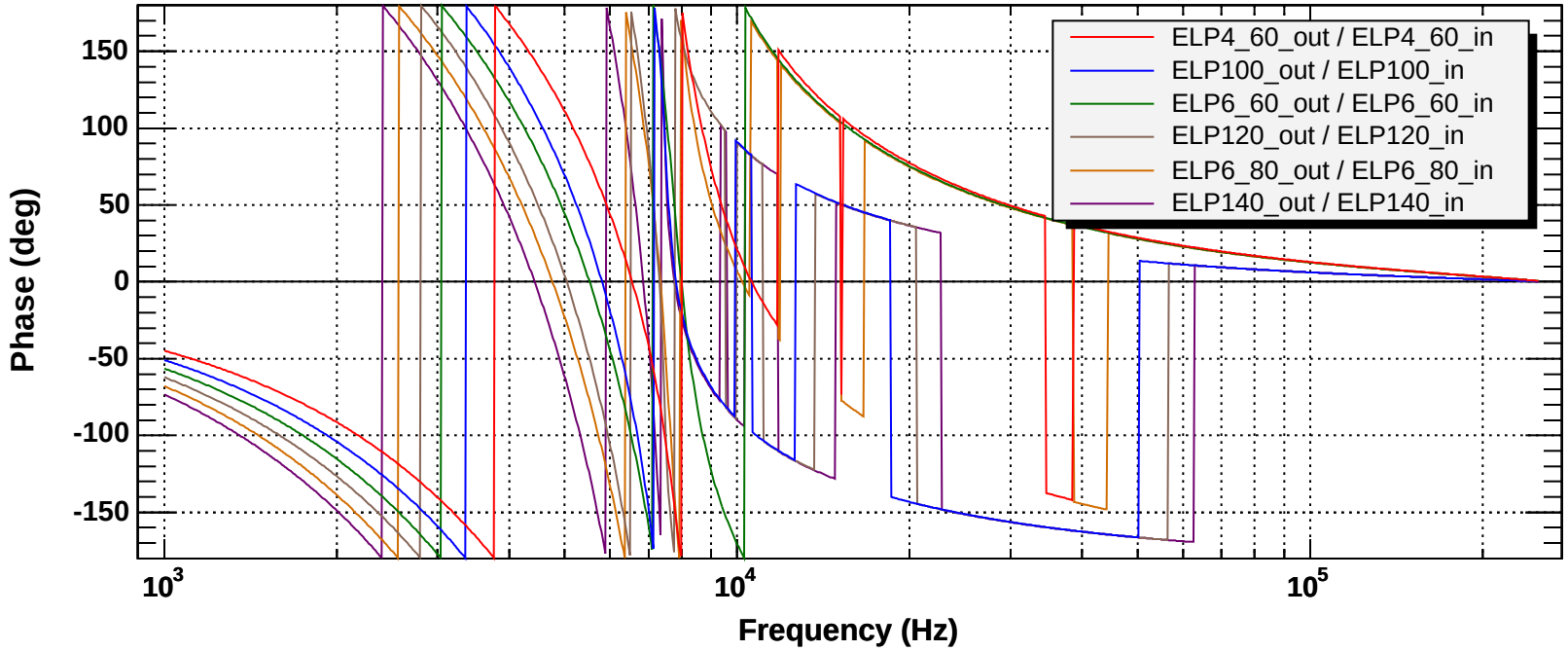
*T0=07/04/2005 19:09:47

iir_coeff.txt

Transfer function



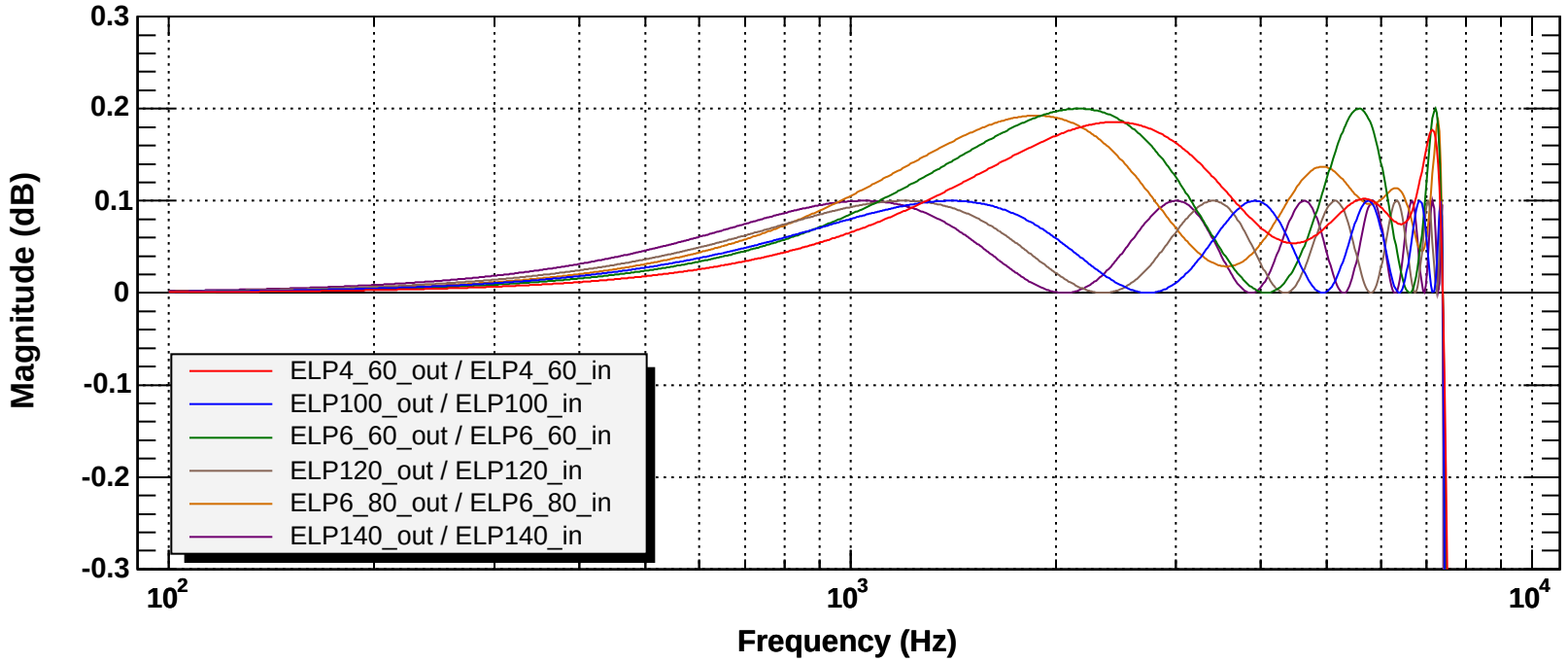
Transfer function



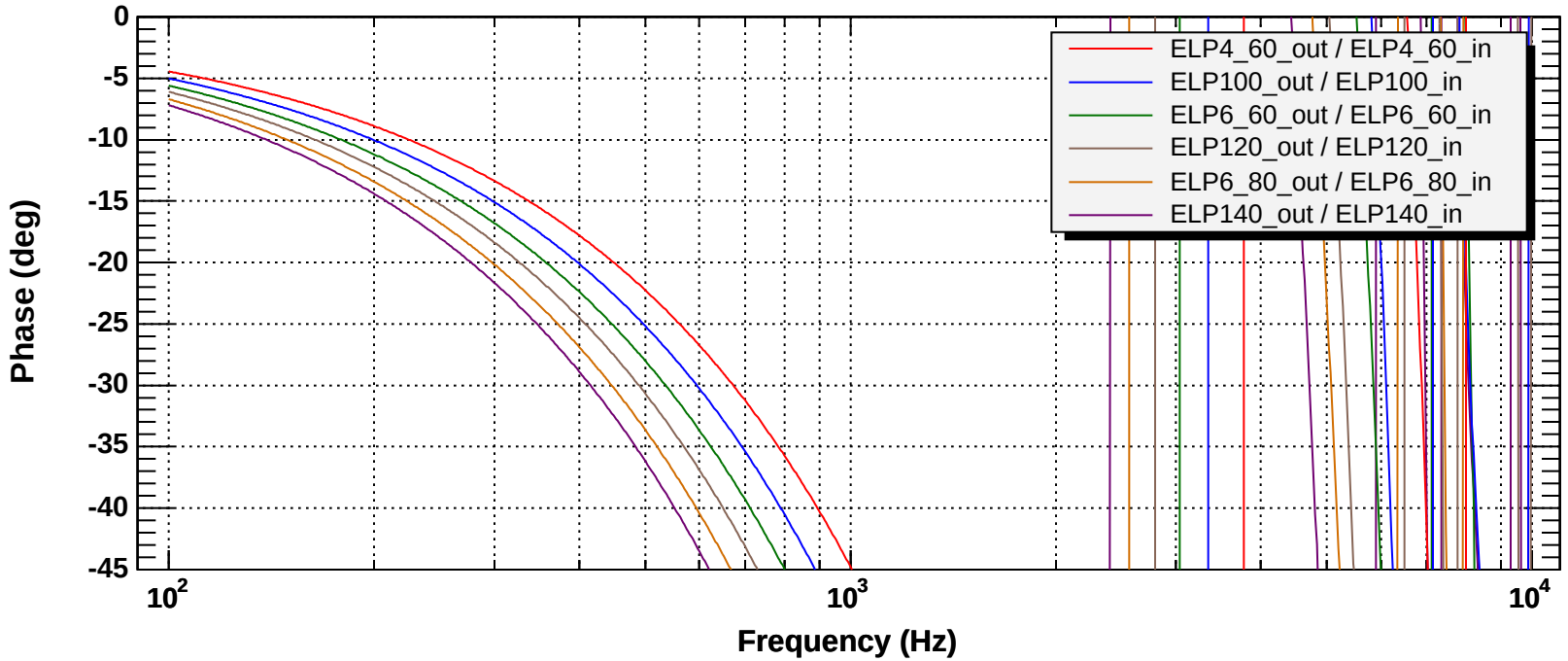
*T0=08/04/2005 16:29:19

iir_coeff.txt

Transfer function



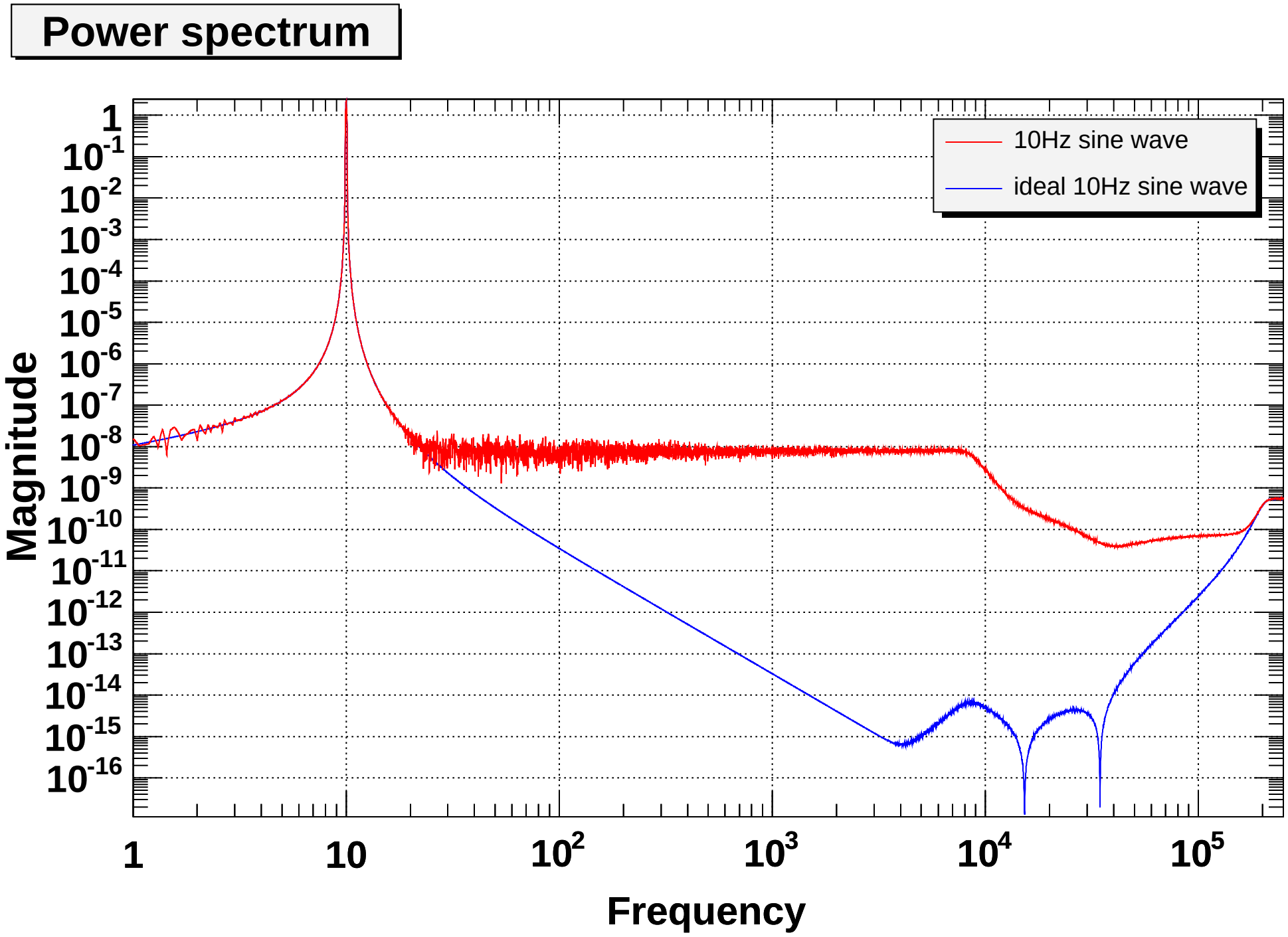
Transfer function

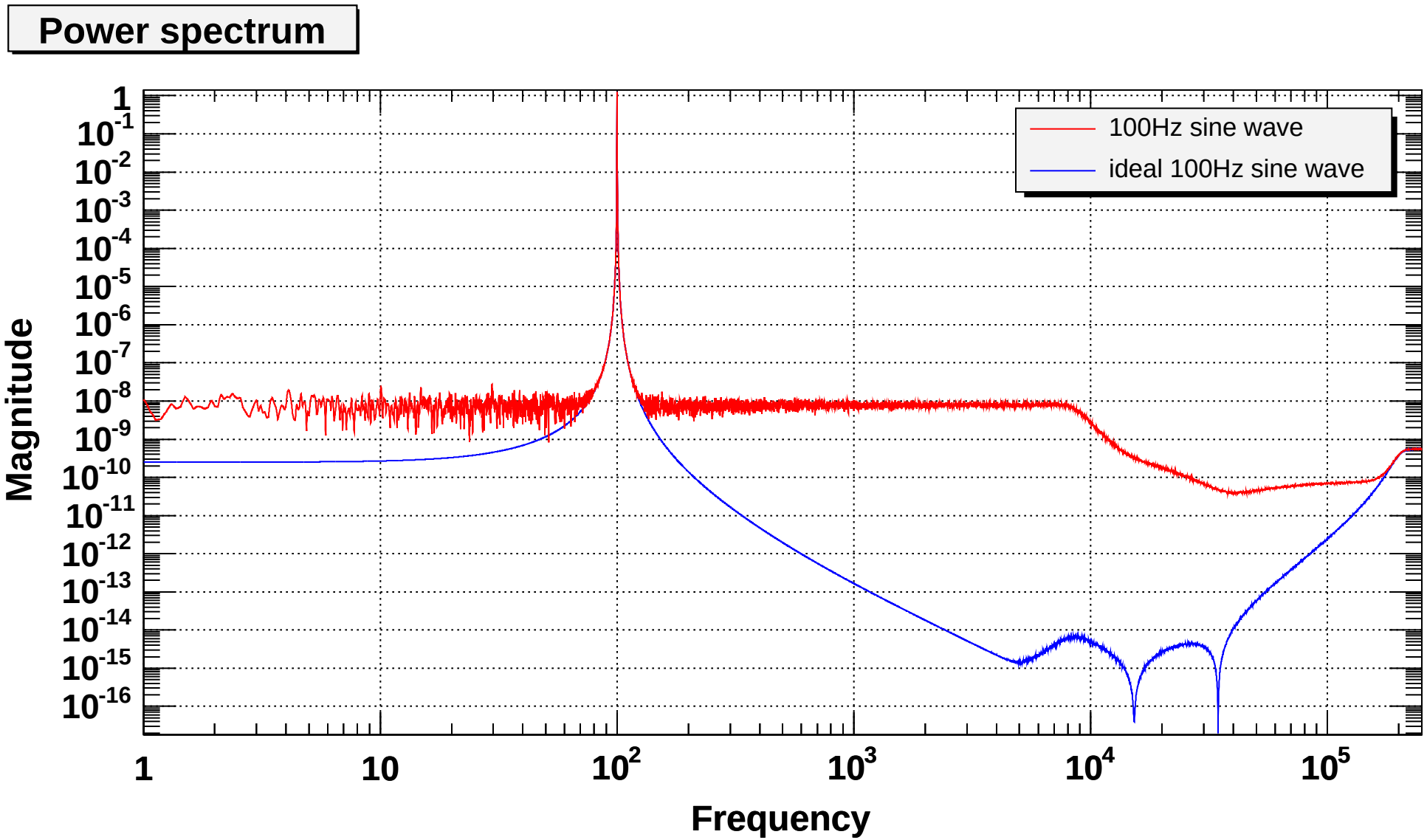


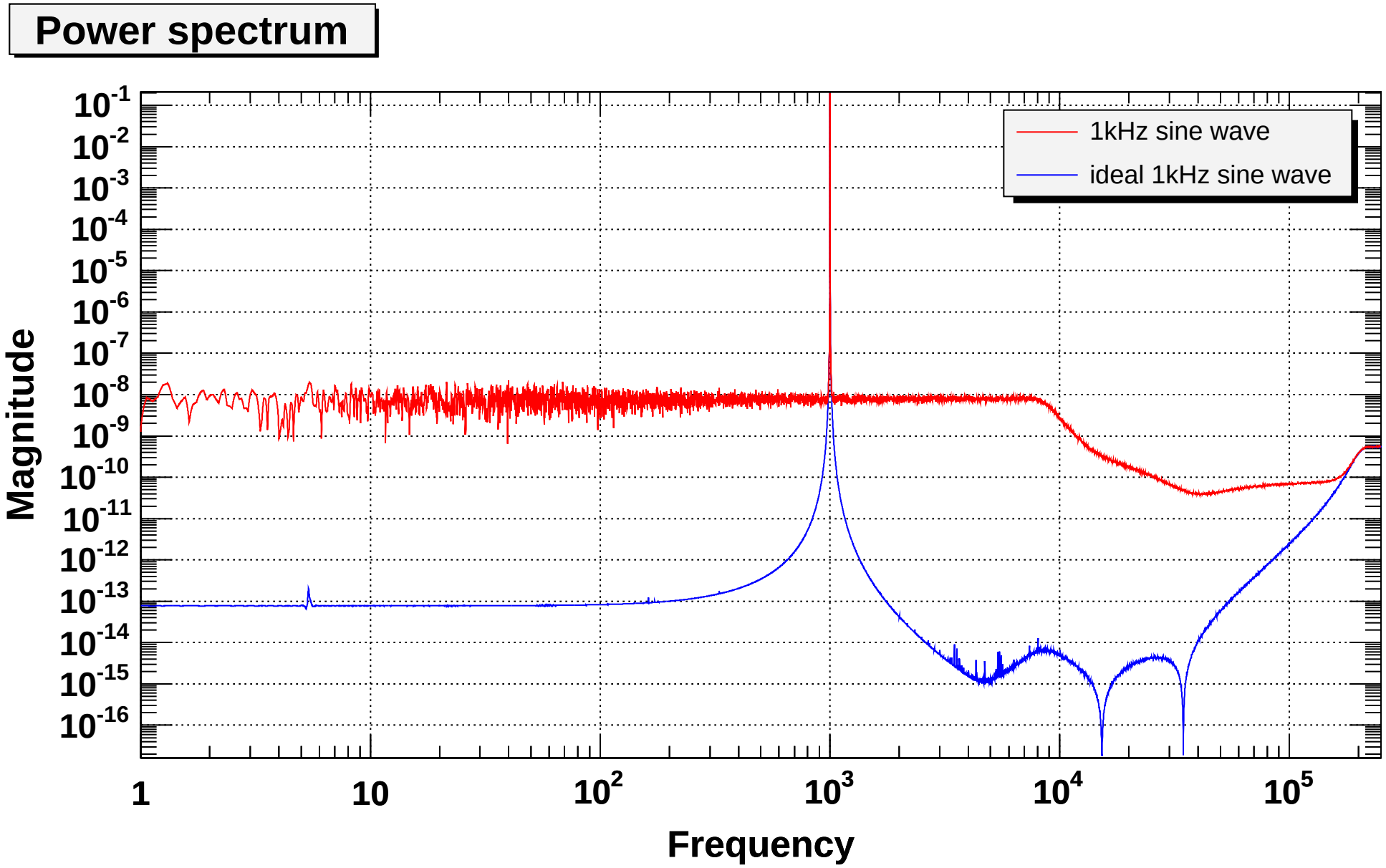
*T0=08/04/2005 16:37:36

iir_coeff.txt

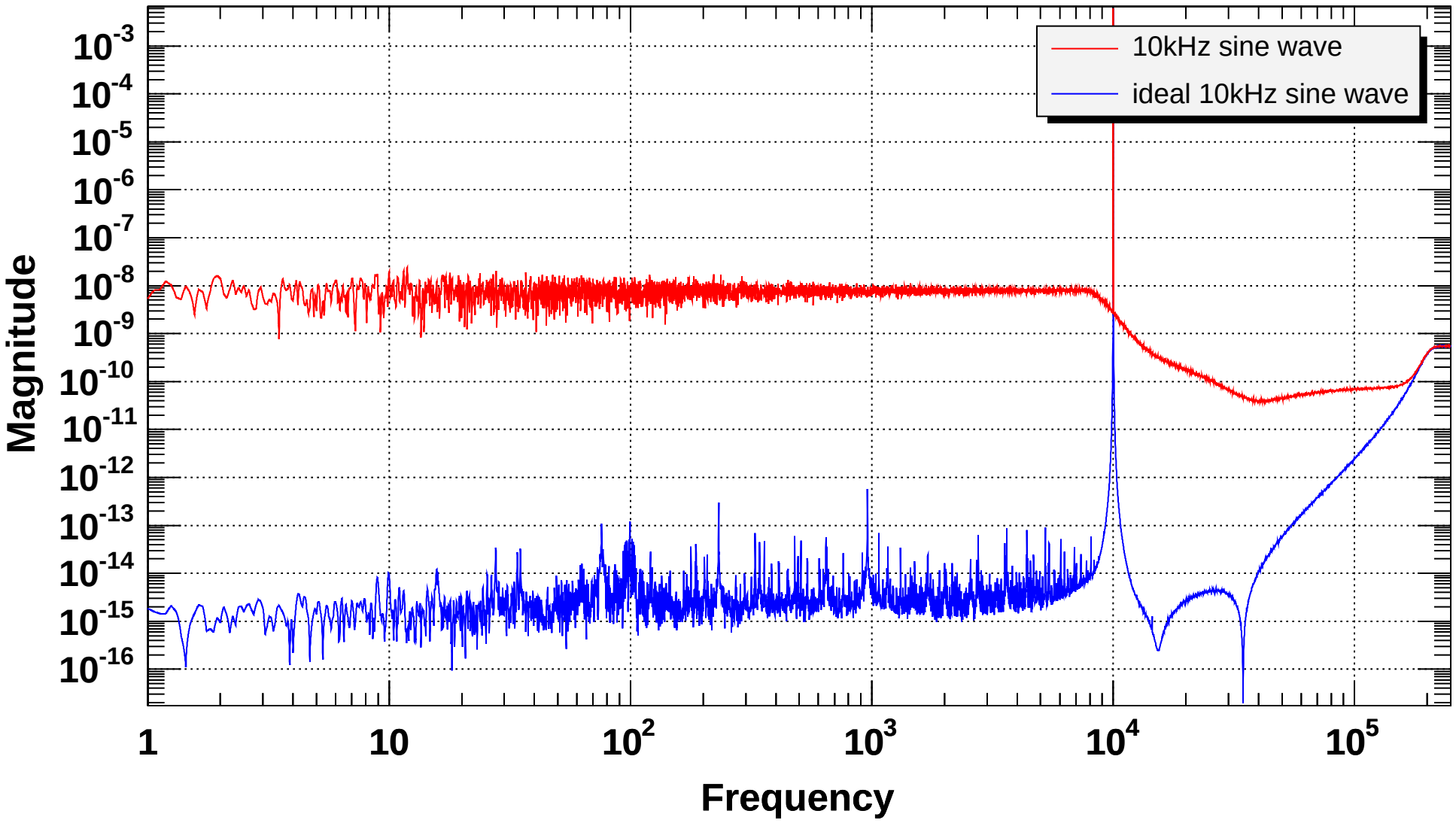
APPENDIX D SPECTRA OF DIFFERENT STIMULI


Power spectrum
10Hz sine wave
ideal 10Hz sine wave
Magnitude
1
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
Frequency
T0=06/01/1980 00:00:00
Bin=419L


Power spectrum
100Hz sine wave
ideal 100Hz sine wave
Magnitude
1
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
Frequency
T0=06/01/1980 00:00:00
Bin=419L

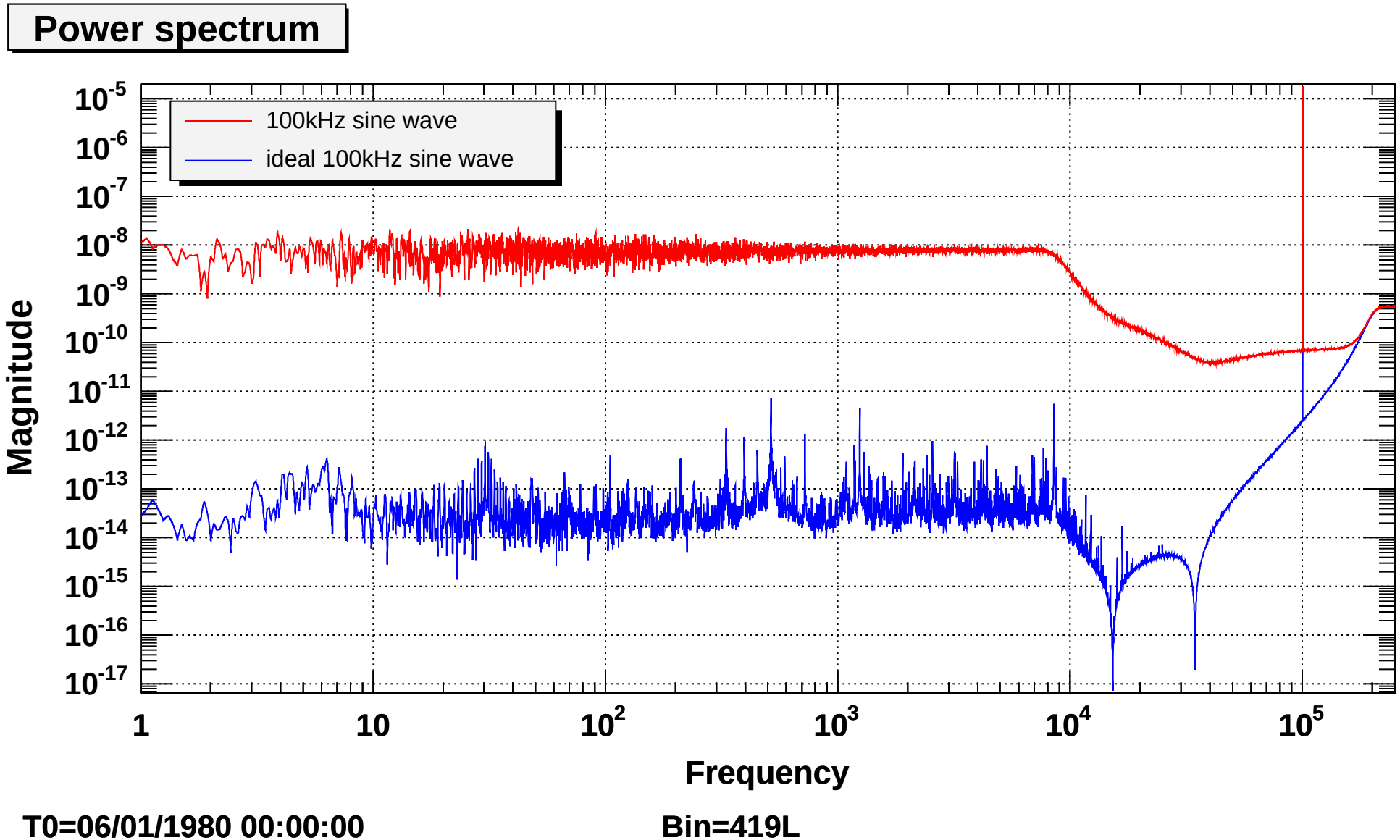

Power spectrum
1kHz sine wave
ideal 1kHz sine wave
Magnitude
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
10^{-7}
10^{-8}
10^{-9}
10^{-10}
10^{-11}
10^{-12}
10^{-13}
10^{-14}
10^{-15}
10^{-16}
1
10
10^{2}
10^{3}
10^{4}
10^{5}
Frequency
T0=06/01/1980 00:00:00
Bin=419L

Power spectrum

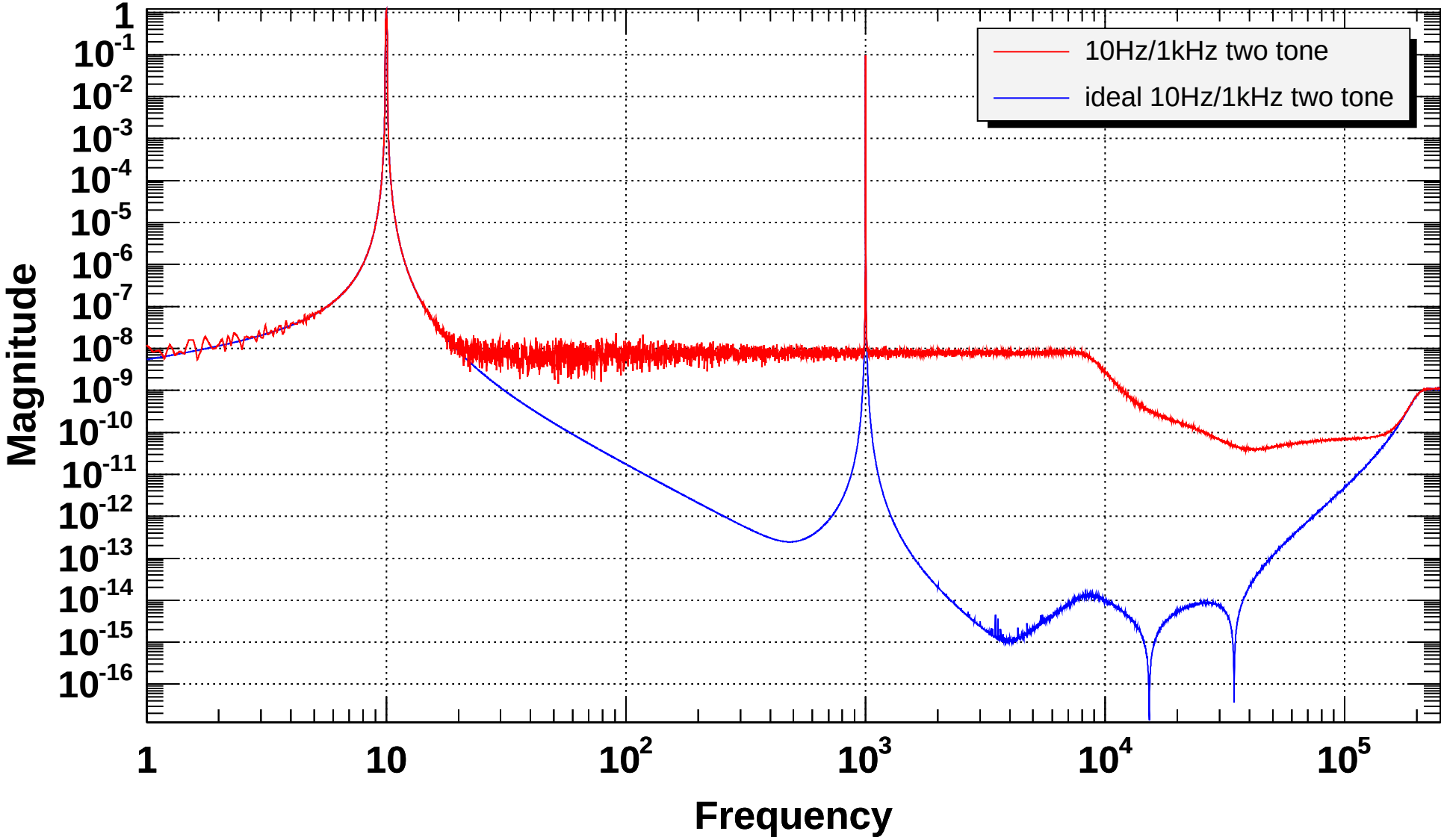


T0=06/01/1980 00:00:00

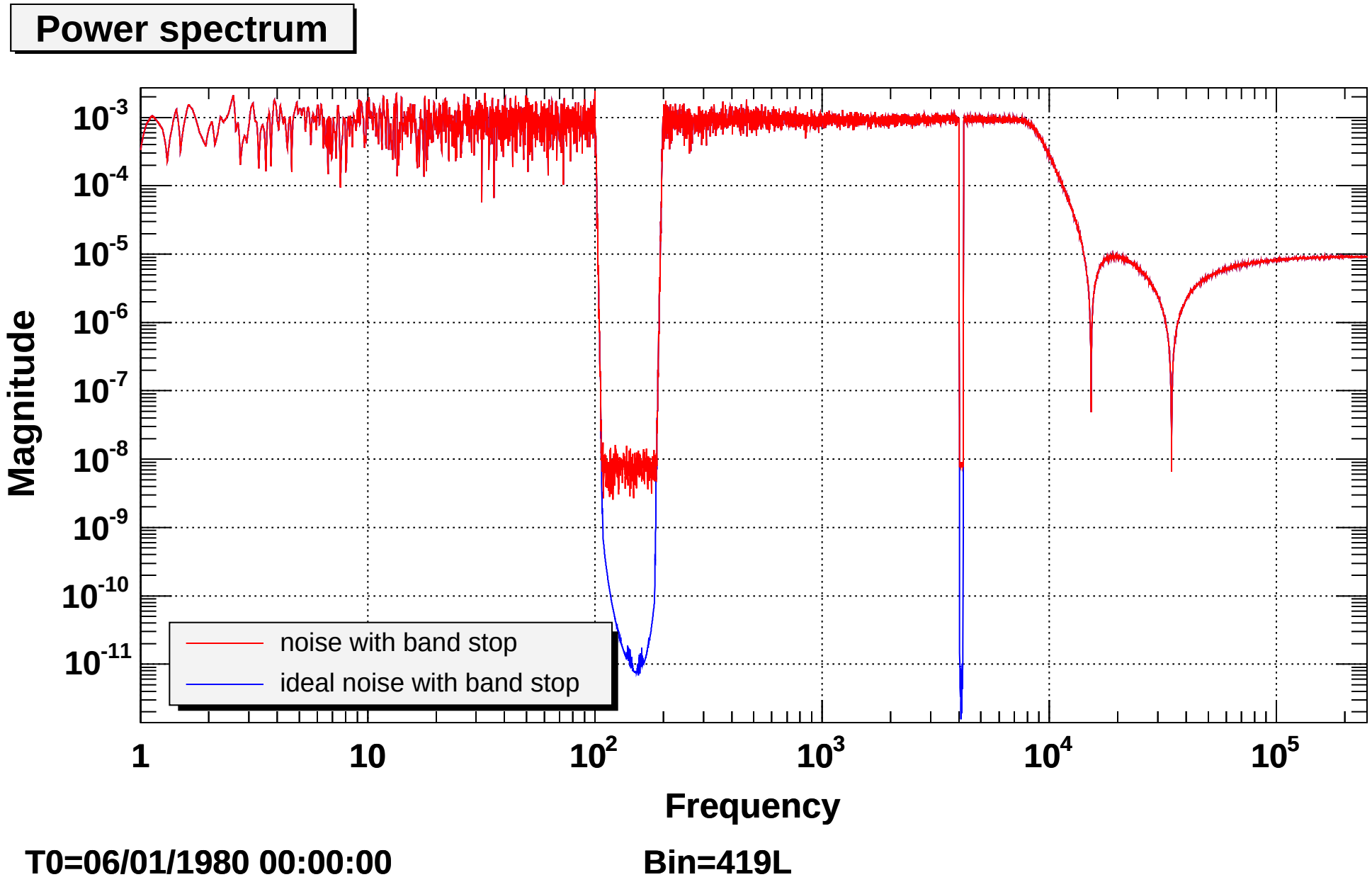
Bin=419L

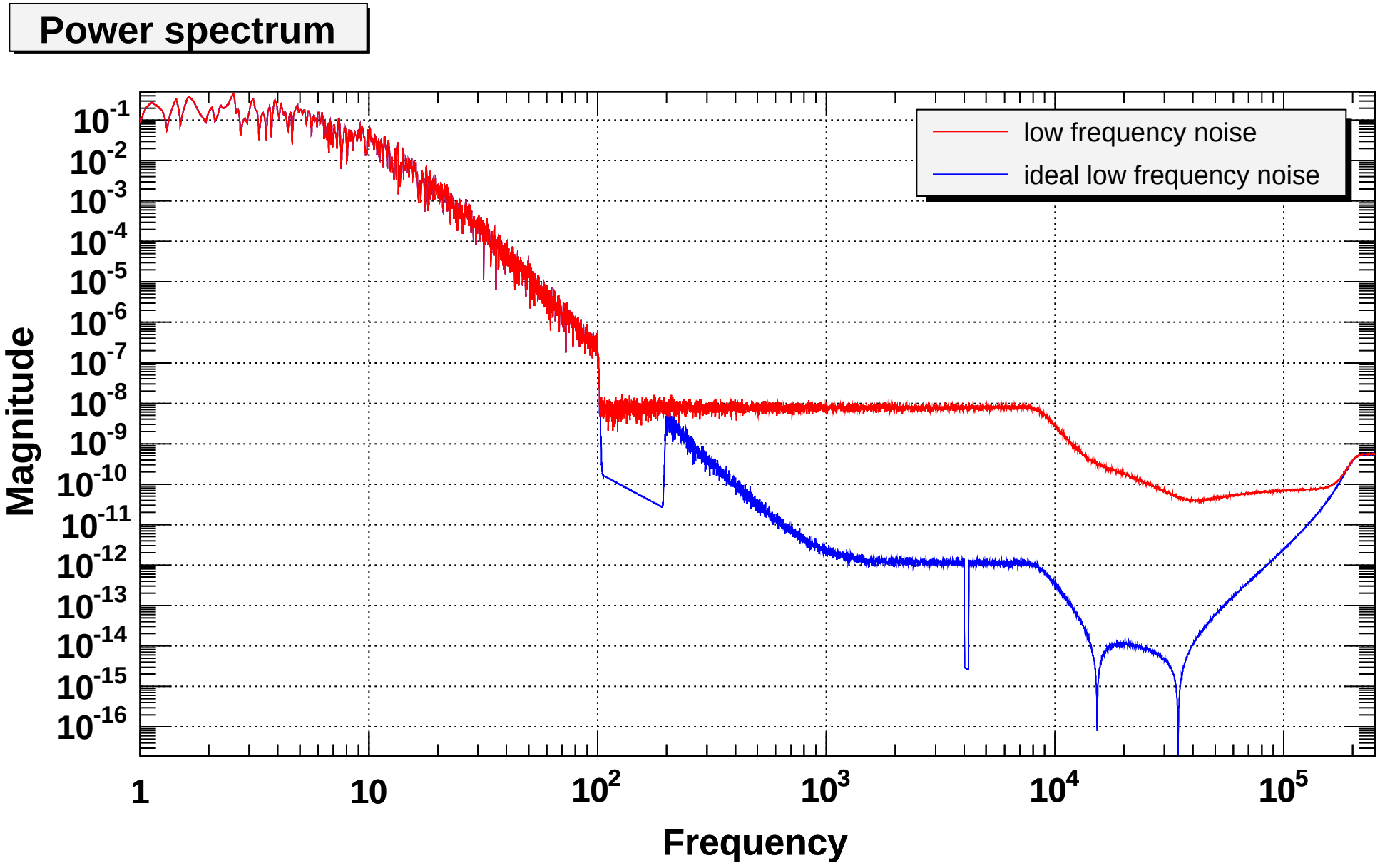

Power spectrum
100kHz sine wave
ideal 100kHz sine wave
Magnitude
10^-5
10^-6
10^-7
10^-8
10^-9
10^-10
10^-11
10^-12
10^-13
10^-14
10^-15
10^-16
10^-17
1
10
10^2
10^3
10^4
10^5
Frequency
T0=06/01/1980 00:00:00
Bin=419L

Power spectrum

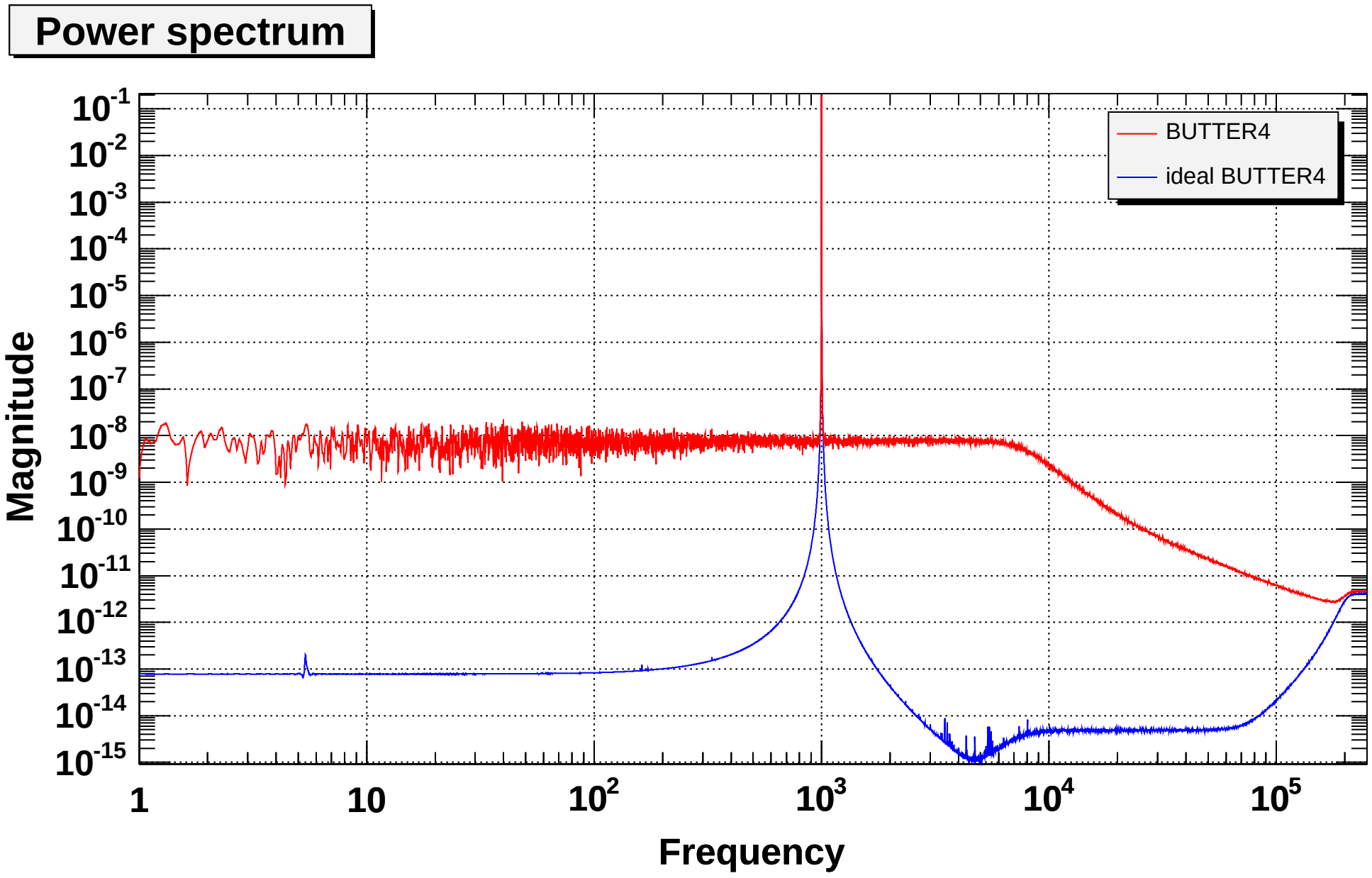


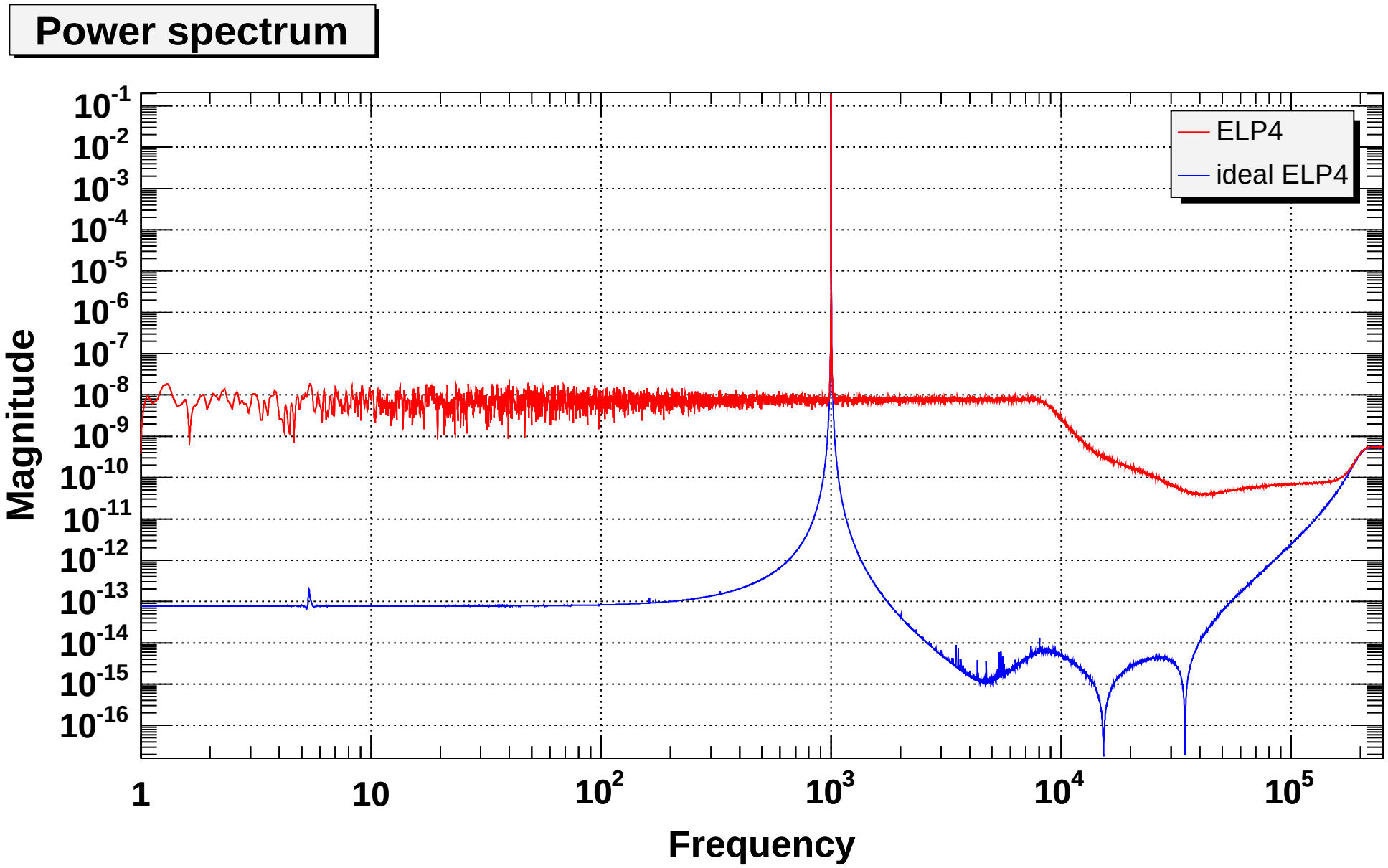
T0=06/01/1980 00:00:00 Bin=419L


Power spectrum
Magnitude
10-3
10-4
10-5
10-6
10-7
10-8
10-9
10-10
10-11
noise with band stop
ideal noise with band stop
1
10
10^2
10^3
10^4
10^5
Frequency
T0=06/01/1980 00:00:00
Bin=419L

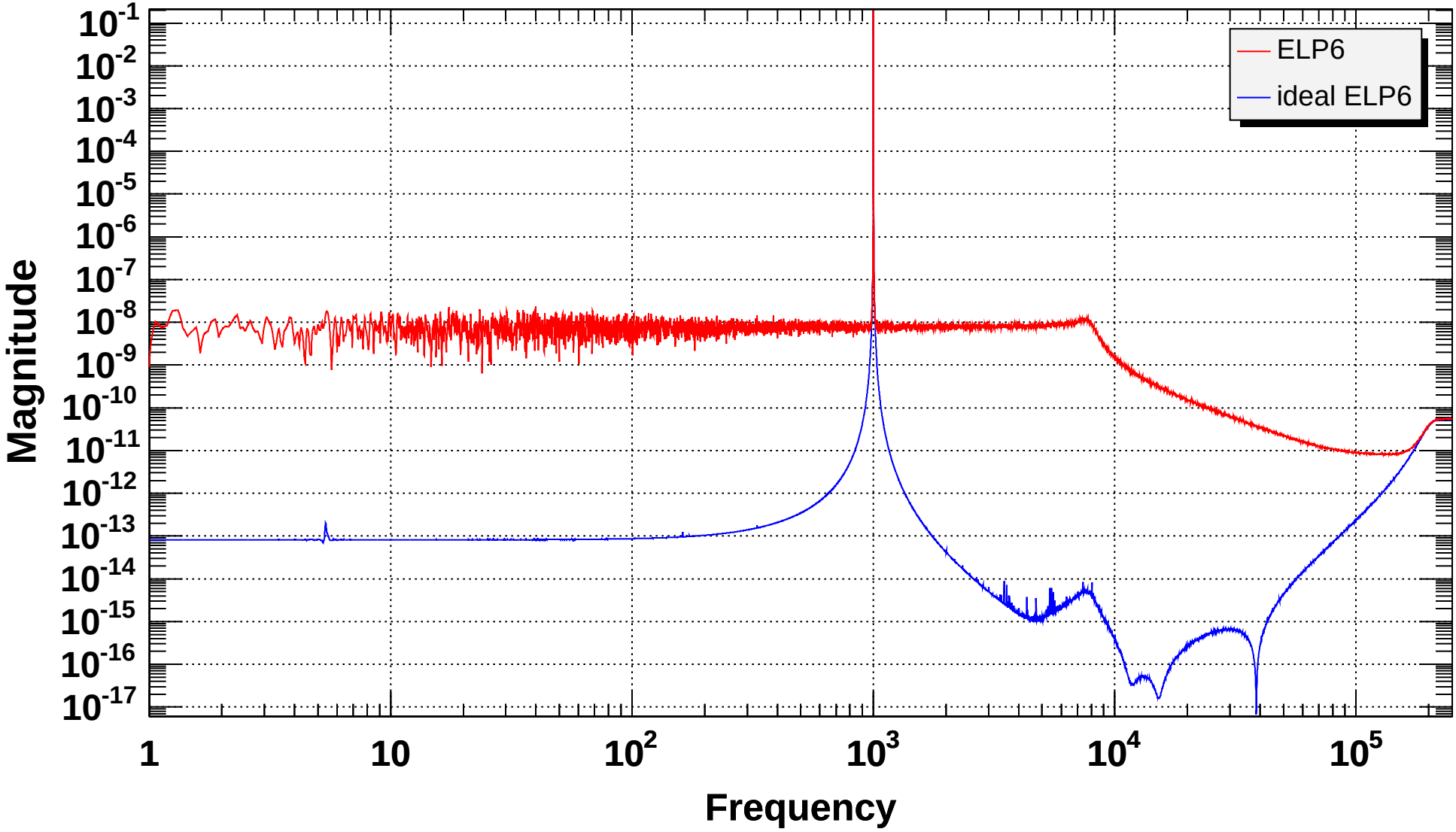

Power spectrum
low frequency noise
ideal low frequency noise
Magnitude
10^-1
10^-2
10^-3
10^-4
10^-5
10^-6
10^-7
10^-8
10^-9
10^-10
10^-11
10^-12
10^-13
10^-14
10^-15
10^-16
1
10
10^2
10^3
10^4
10^5
Frequency
T0=06/01/1980 00:00:00
Bin=419L

APPENDIX E SPECTRA OF DIFFERENT FILTERS

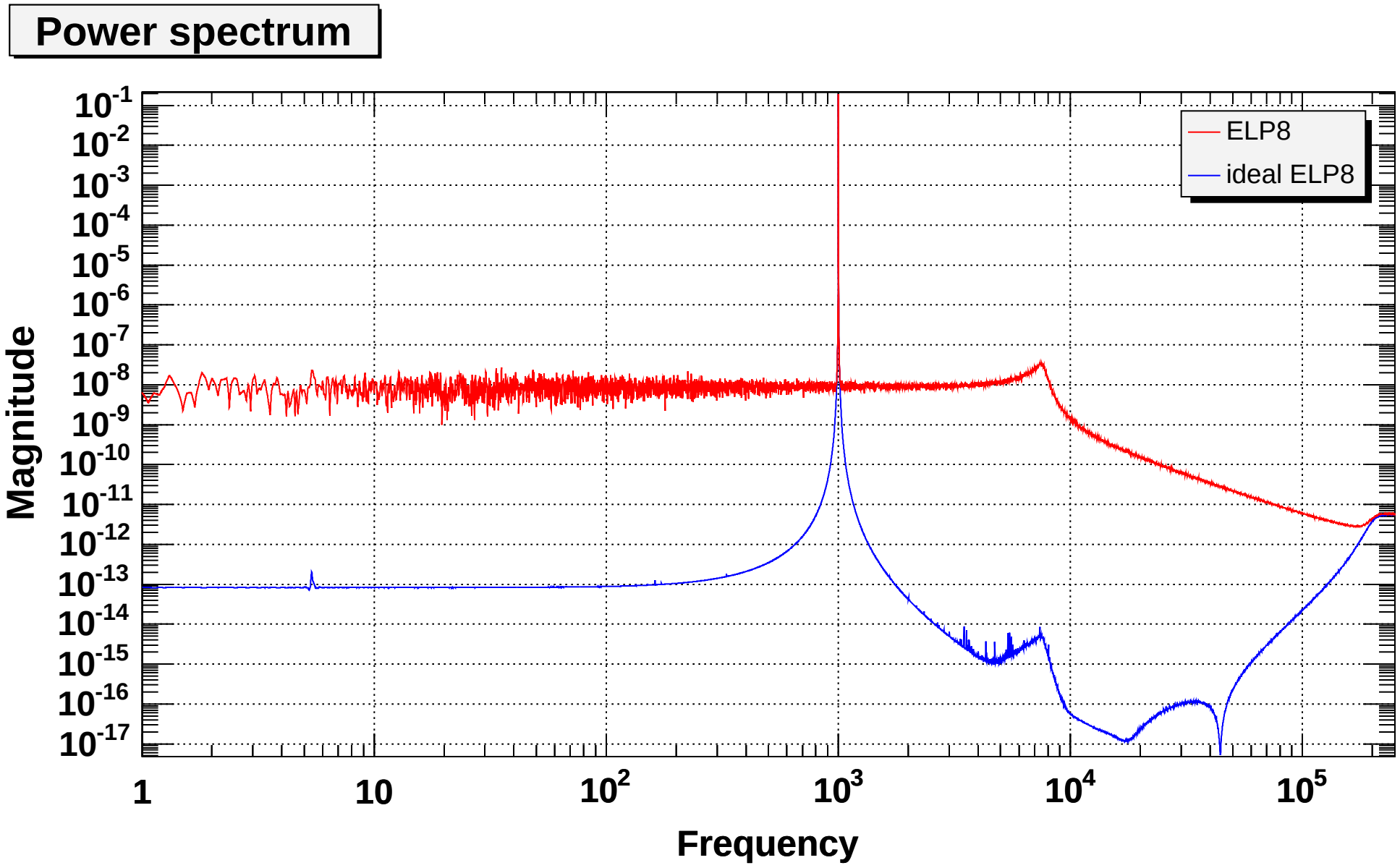

Power spectrum
BUTTER4
ideal BUTTER4
Magnitude
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
10^{-7}
10^{-8}
10^{-9}
10^{-10}
10^{-11}
10^{-12}
10^{-13}
10^{-14}
10^{-15}
1
10
10^{2}
10^{3}
10^{4}
10^{5}
Frequency
T0=06/01/1980 00:00:00
Bin=419L


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

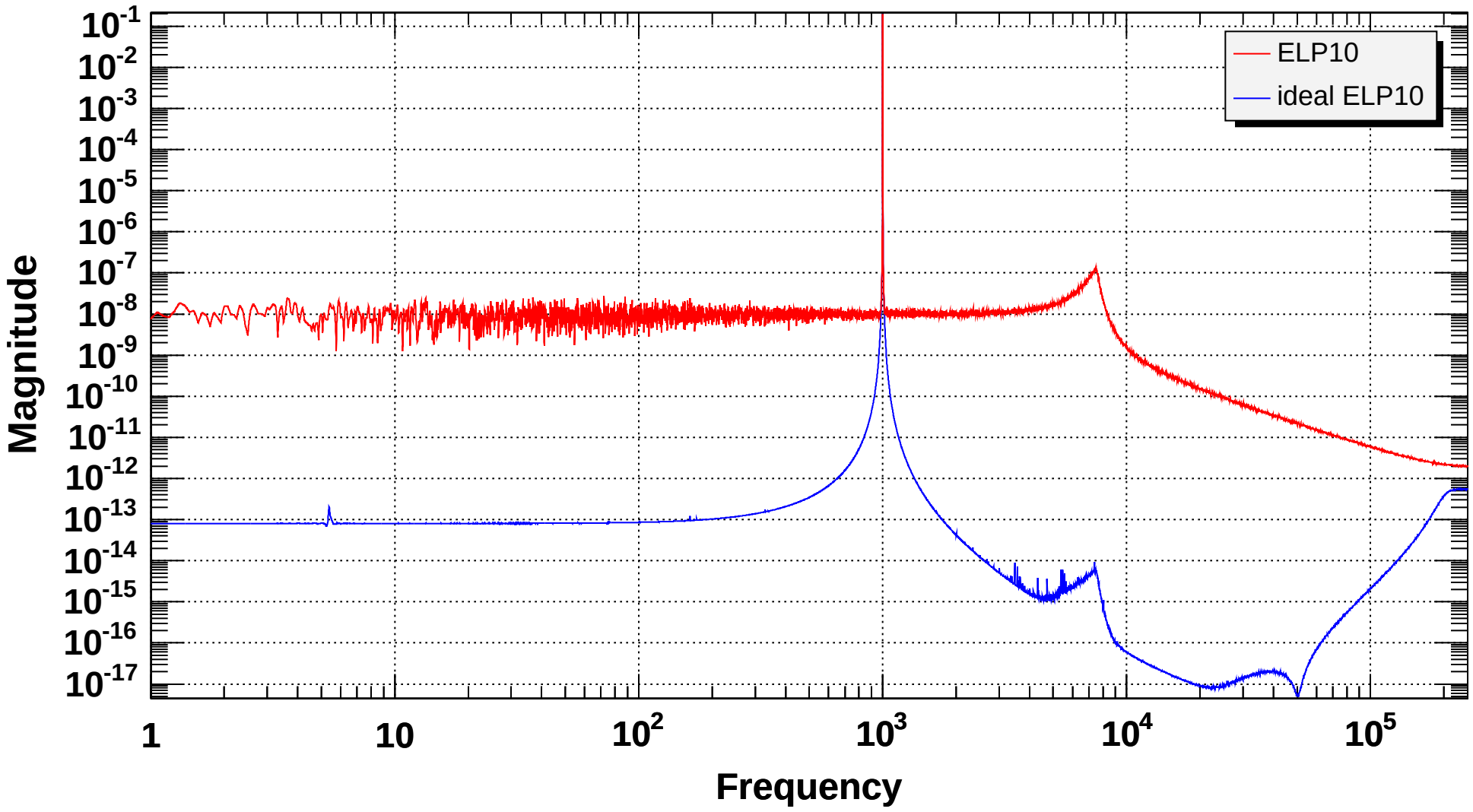
Power spectrum



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

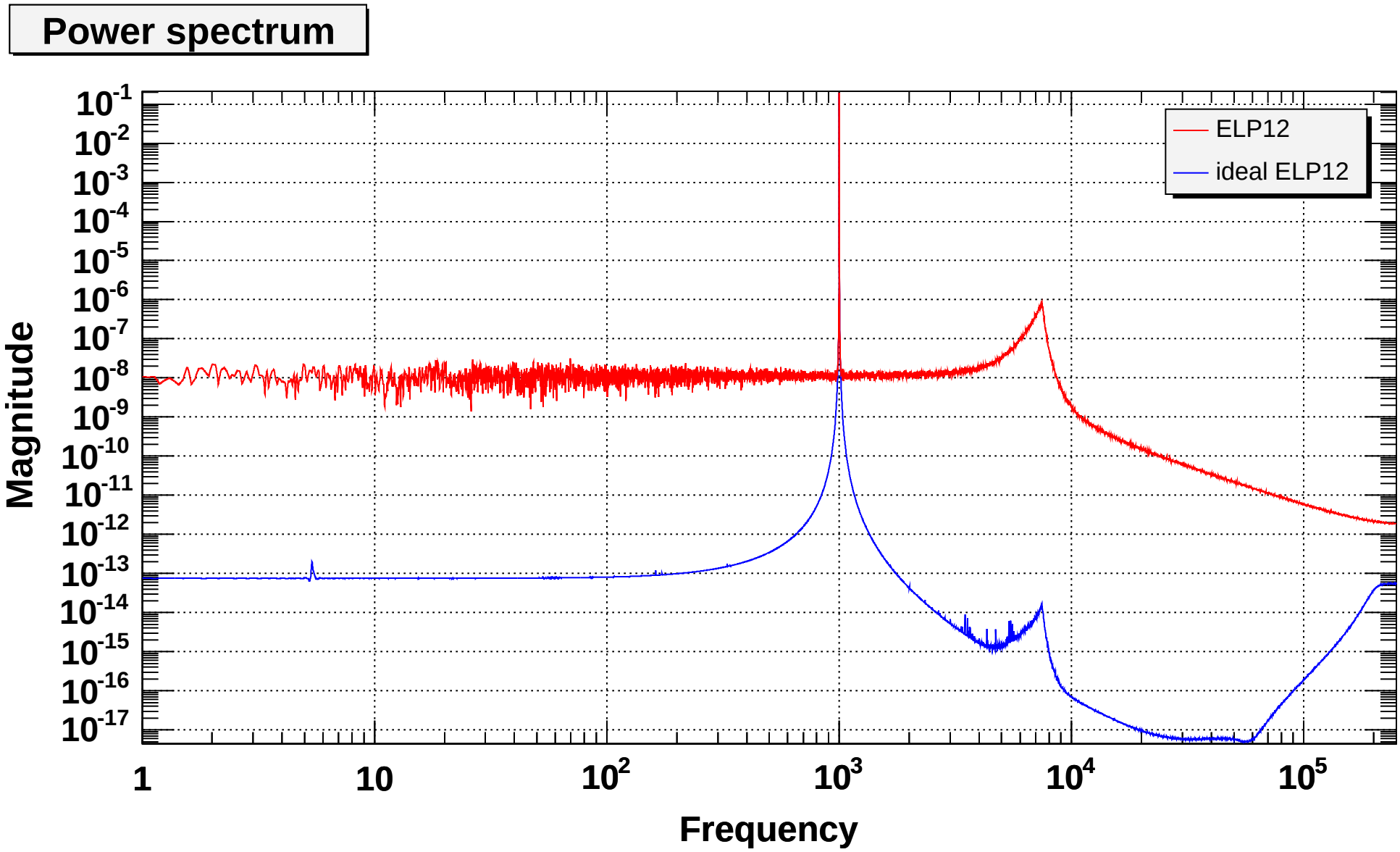

Power spectrum
ELP8
ideal ELP8
Magnitude
Frequency
T0=06/01/1980 00:00:00
Bin=419L

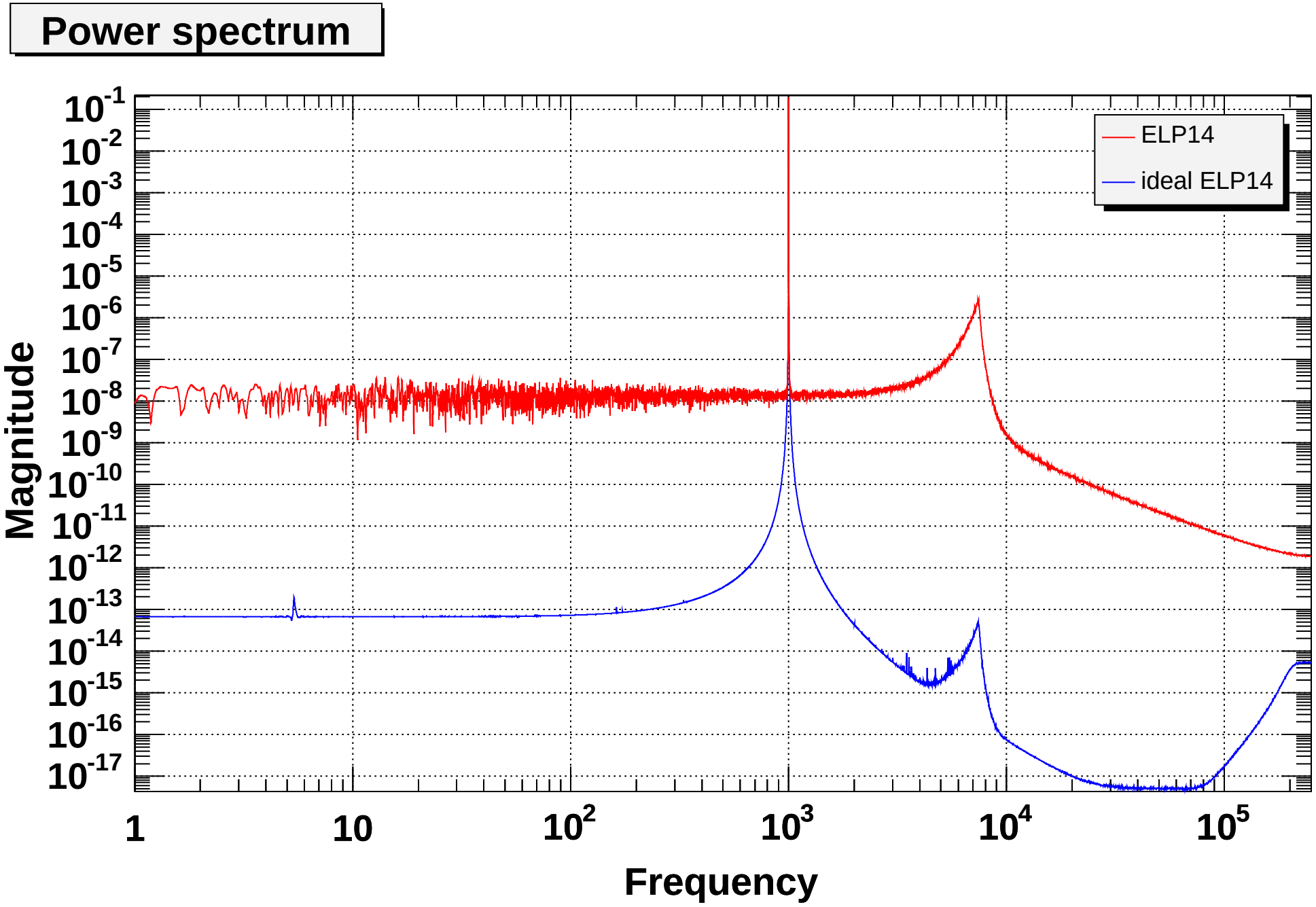
Power spectrum



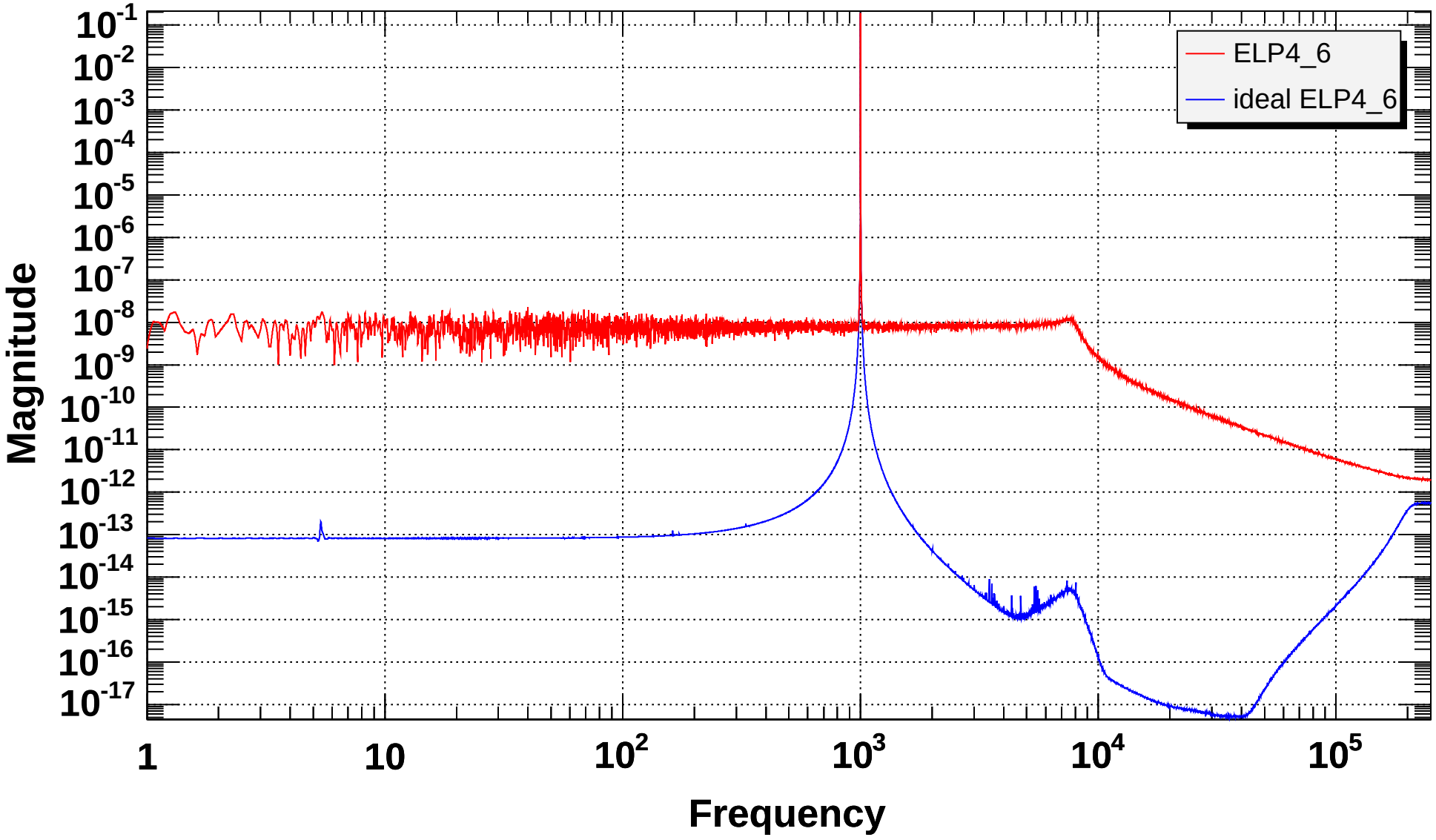
T0=06/01/1980 00:00:00

Bin=419L

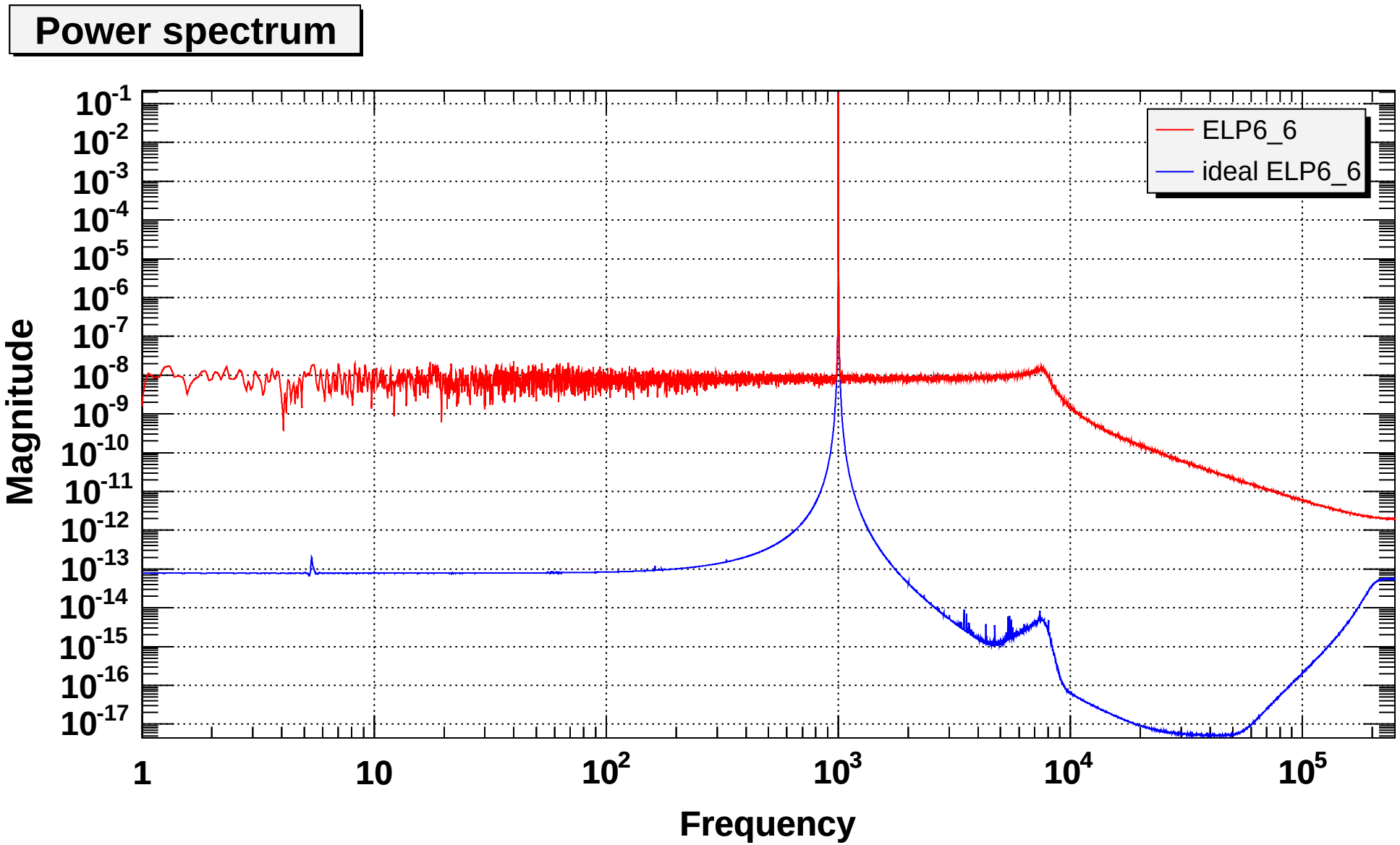

Power spectrum
ELP12
ideal ELP12
Magnitude
Frequency
1
10
10^2
10^3
10^4
10^5
10^-1
10^-2
10^-3
10^-4
10^-5
10^-6
10^-7
10^-8
10^-9
10^-10
10^-11
10^-12
10^-13
10^-14
10^-15
10^-16
10^-17
T0=06/01/1980 00:00:00
Bin=419L

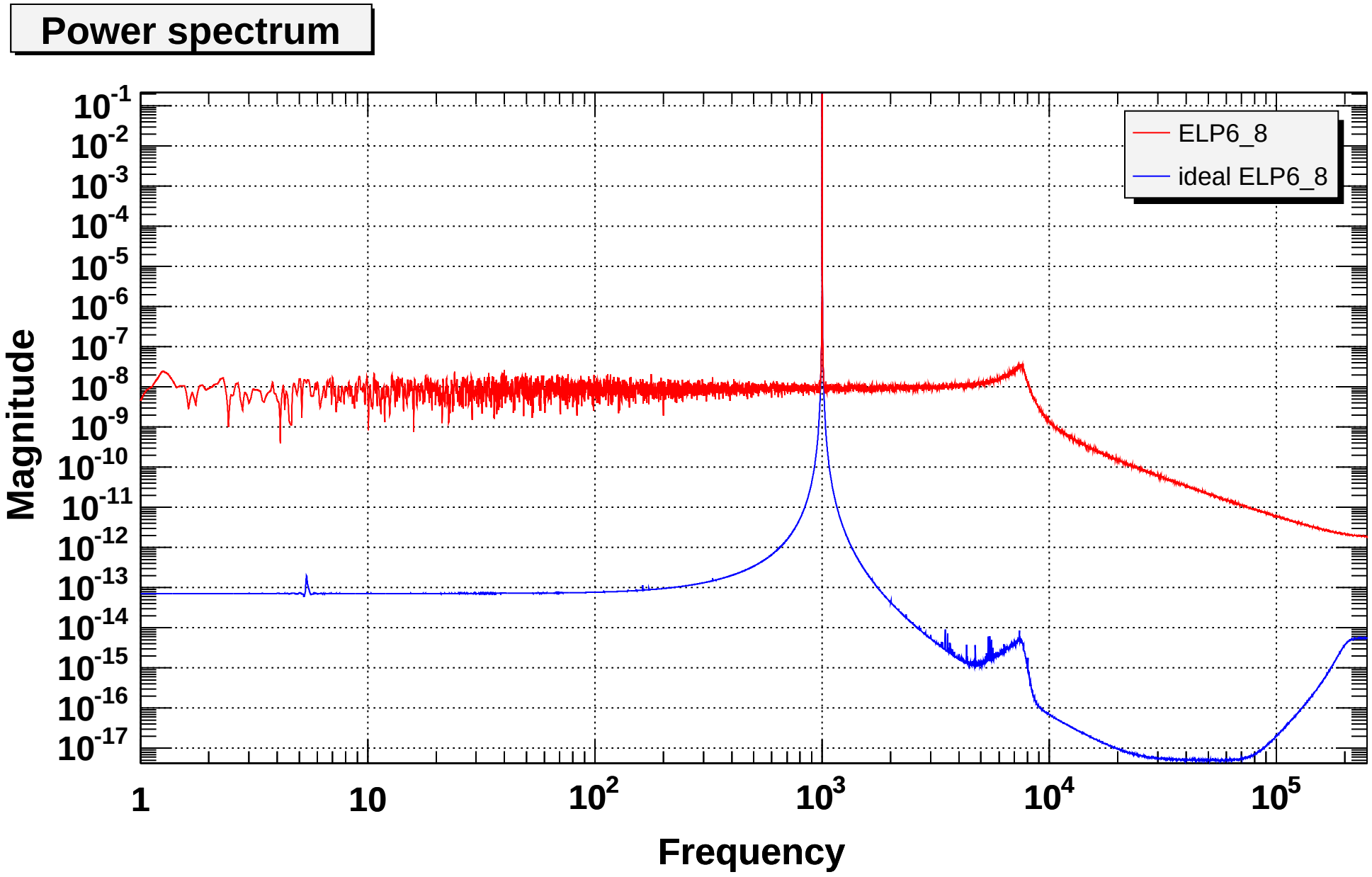

Power spectrum
ELP14
ideal ELP14
Magnitude
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
10^{-7}
10^{-8}
10^{-9}
10^{-10}
10^{-11}
10^{-12}
10^{-13}
10^{-14}
10^{-15}
10^{-16}
10^{-17}
1
10
10^{2}
10^{3}
10^{4}
10^{5}
Frequency
T0=06/01/1980 00:00:00
Bin=419L

Power spectrum

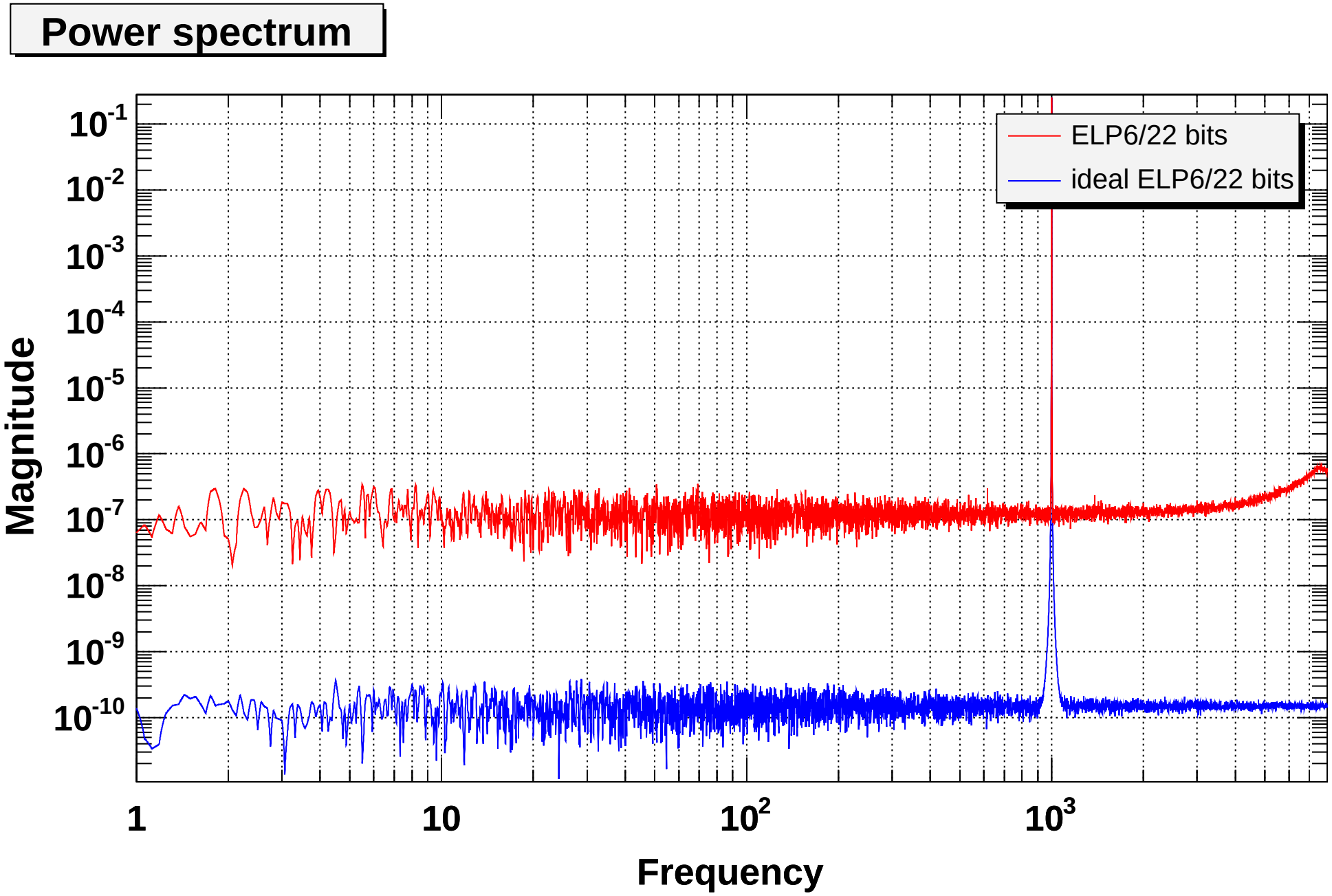


T0=06/01/1980 00:00:00 Bin=419L

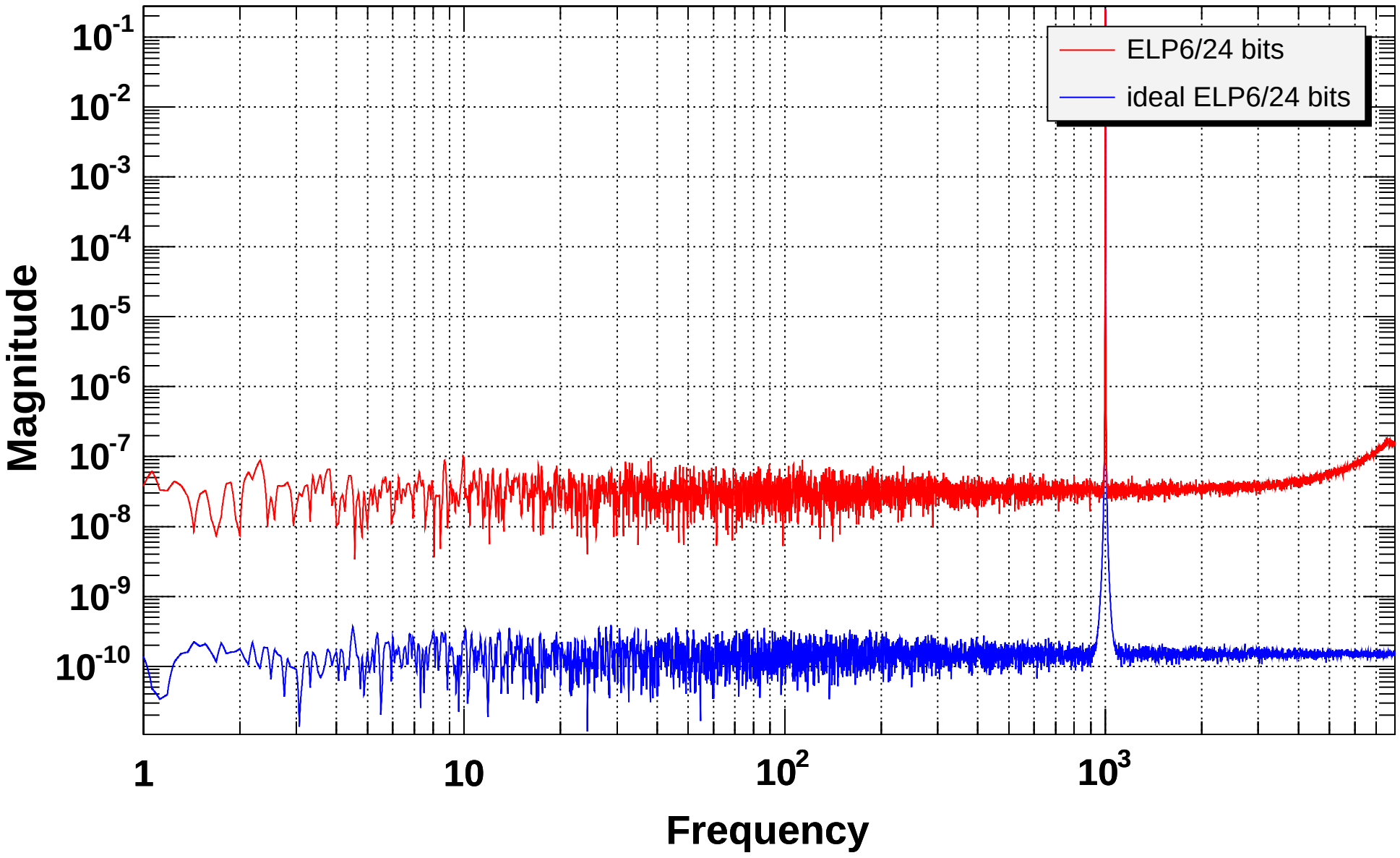

Power spectrum
ELP6_6
ideal ELP6_6
Magnitude
Frequency
T0=06/01/1980 00:00:00
Bin=419L


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

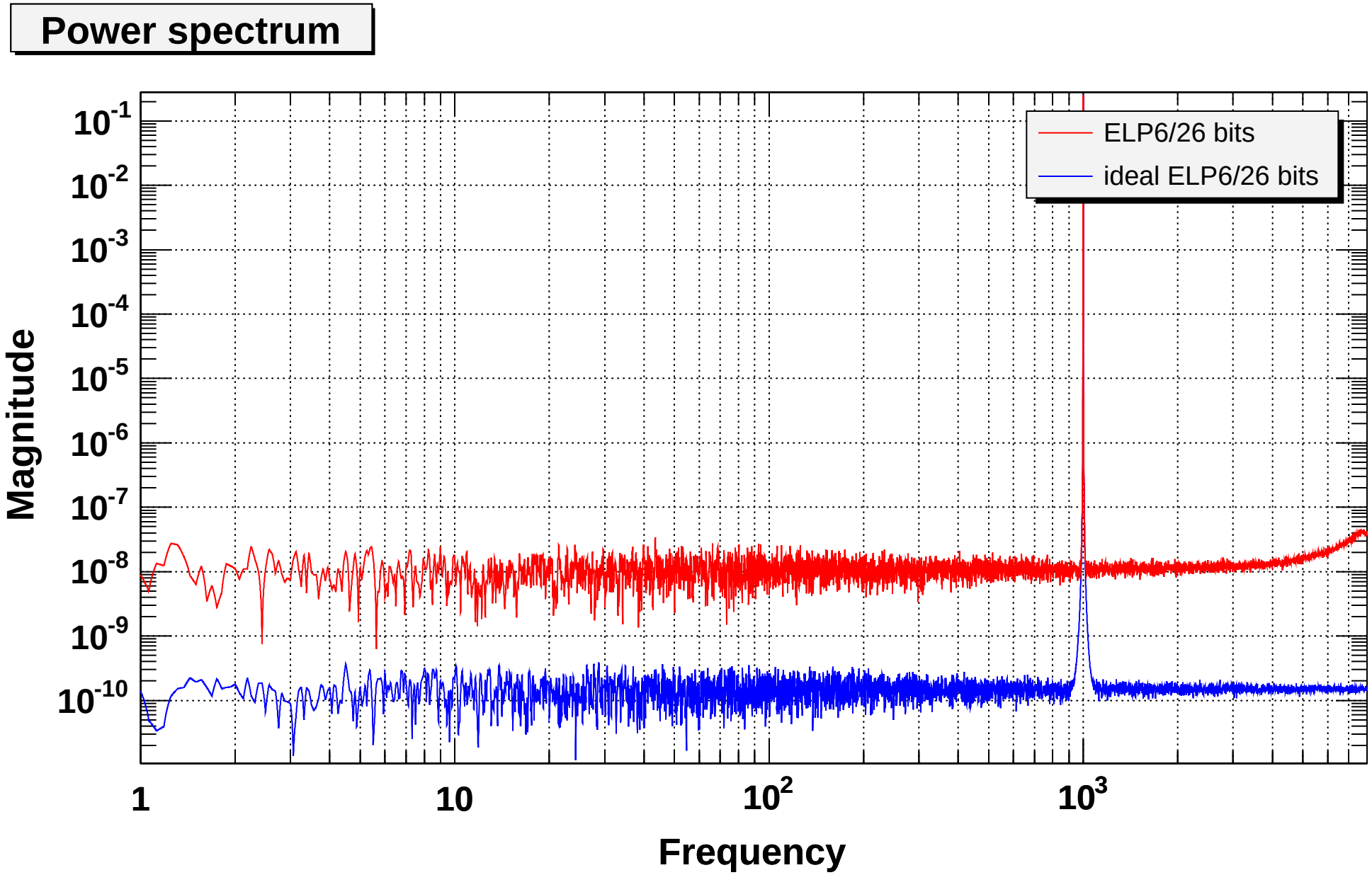
APPENDIX F EFFECT OF BIT RESOLUTION

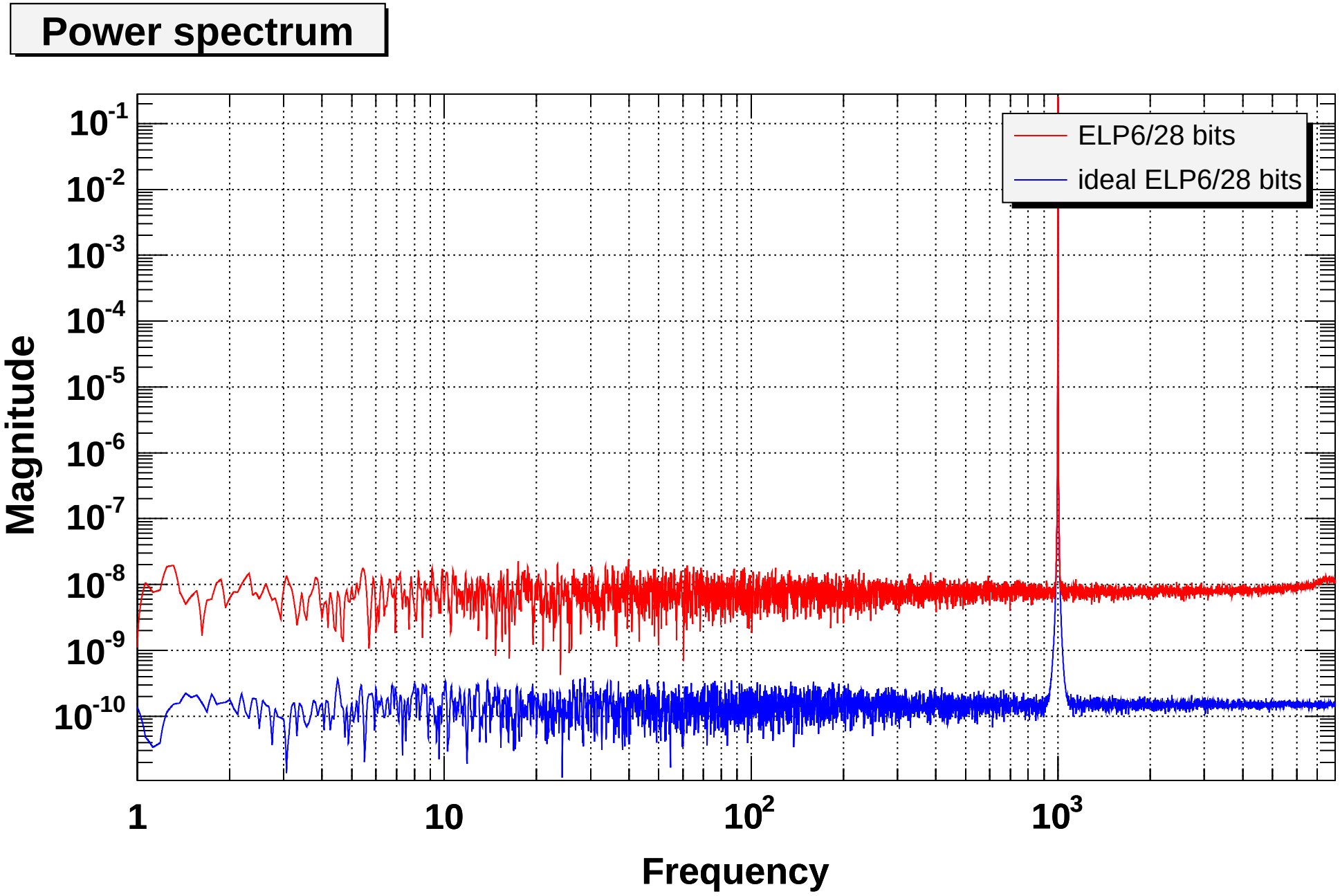

Power spectrum
ELP6/22 bits
ideal ELP6/22 bits
Magnitude
10-1
10-2
10-3
10-4
10-5
10-6
10-7
10-8
10-9
10-10
1
10
10^2
10^3
Frequency
T0=06/01/1980 00:00:00
Bin=13L

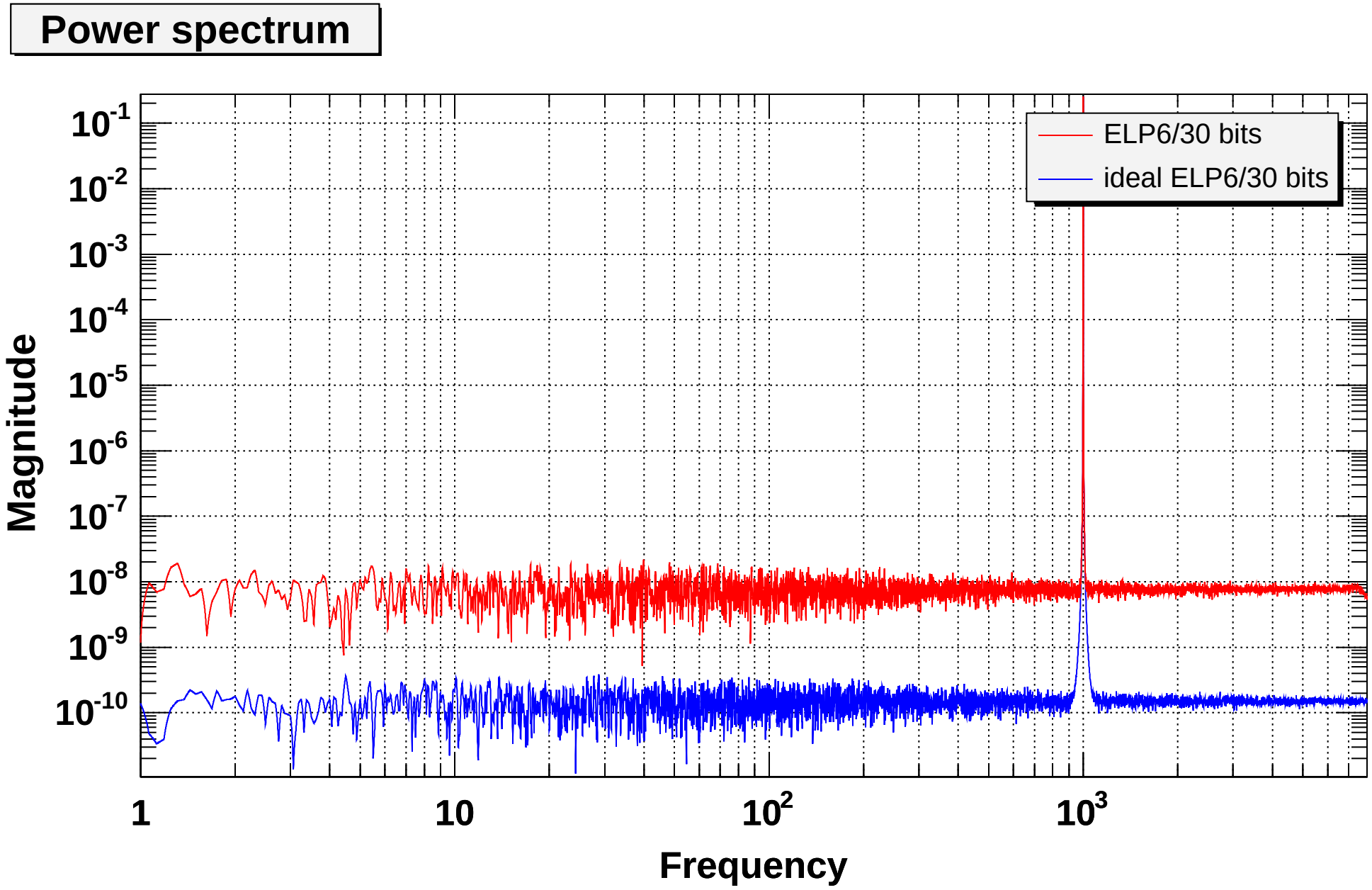
Power spectrum

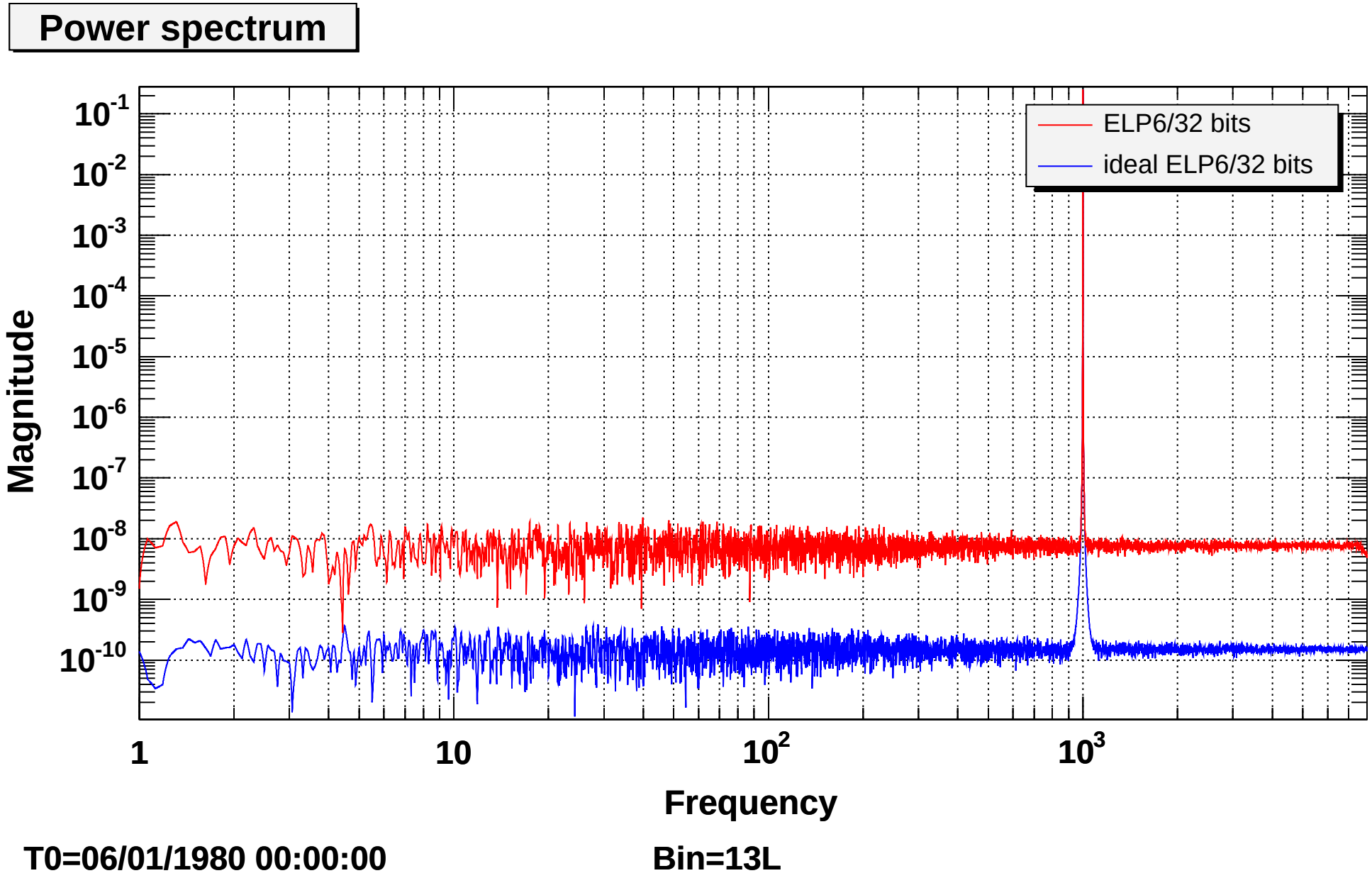


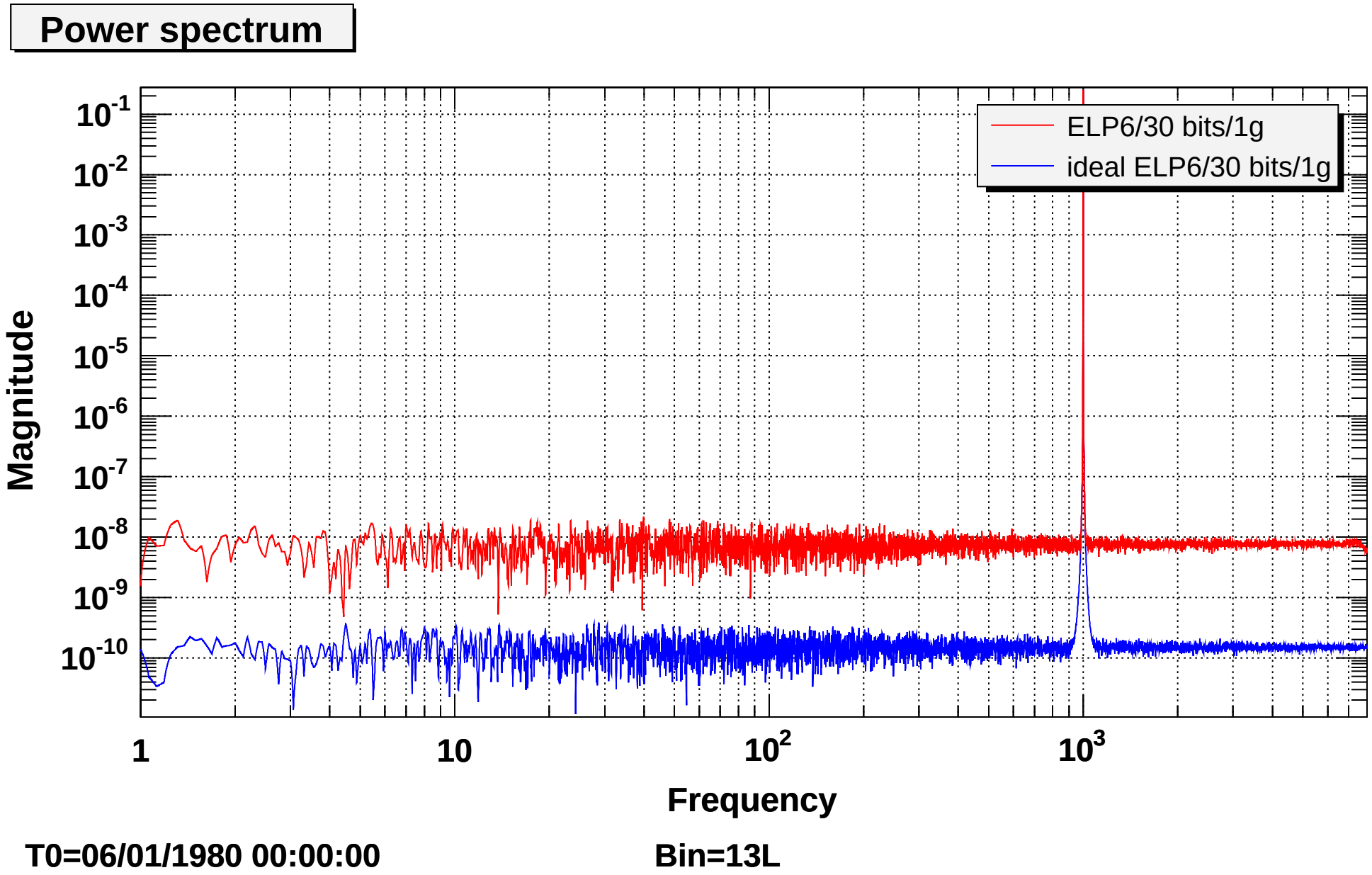
T0=06/01/1980 00:00:00 Bin=13L

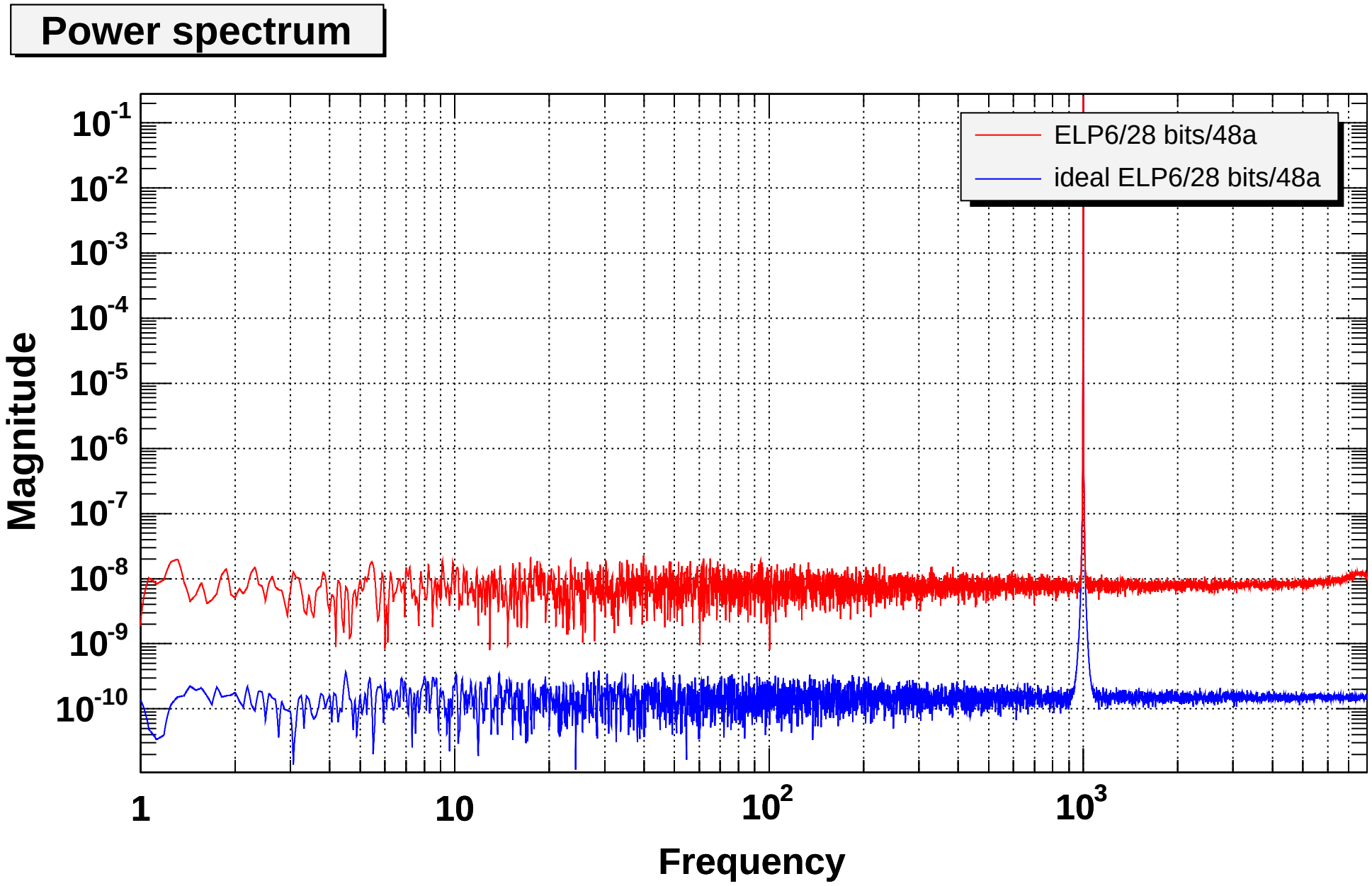

Power spectrum
ELP6/26 bits
ideal ELP6/26 bits
Magnitude
10-1
10-2
10-3
10-4
10-5
10-6
10-7
10-8
10-9
10-10
1
10
10^2
10^3
Frequency
T0=06/01/1980 00:00:00
Bin=13L

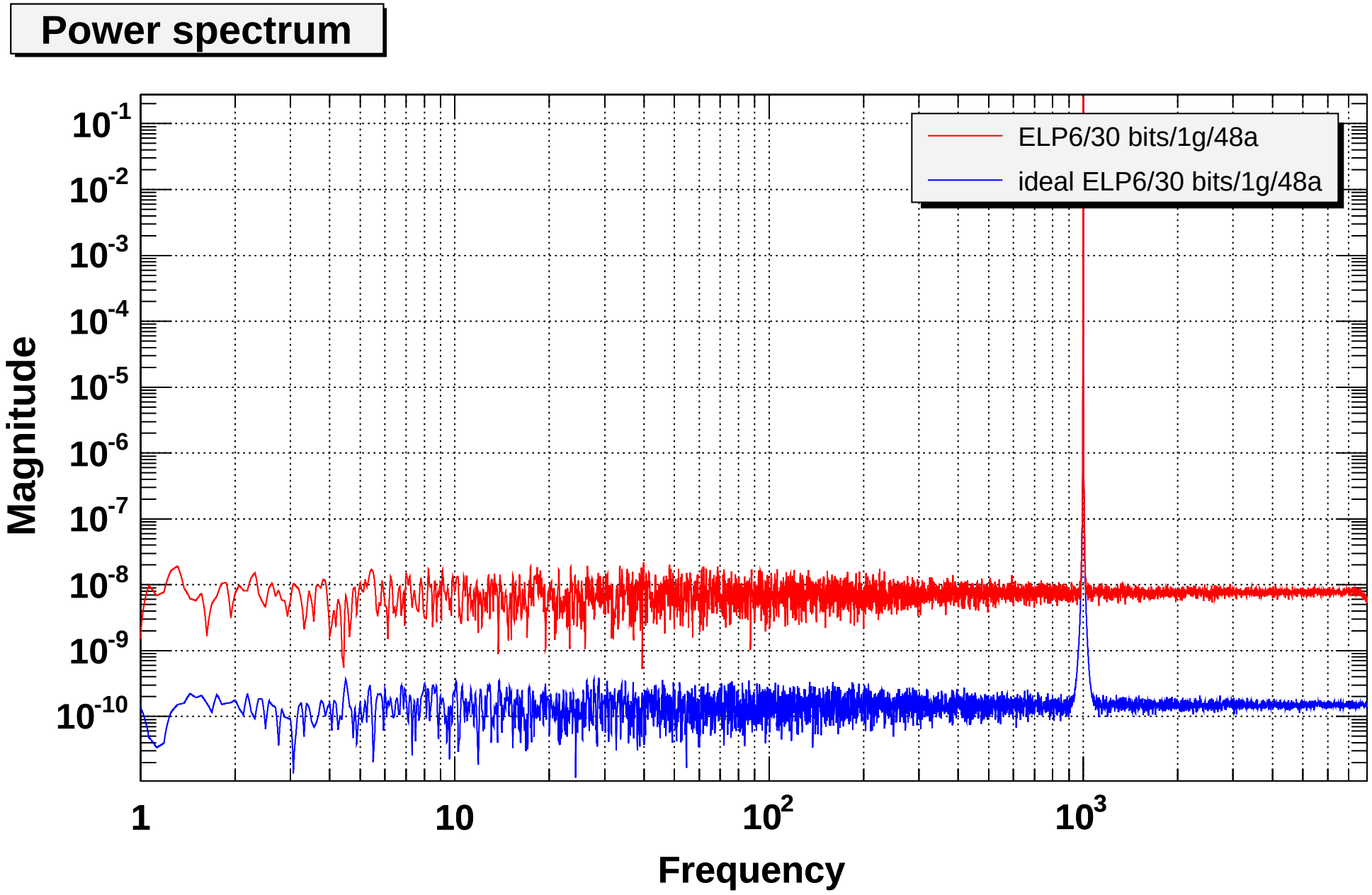

Power spectrum
ELP6/28 bits
ideal ELP6/28 bits
Magnitude
10-1
10-2
10-3
10-4
10-5
10-6
10-7
10-8
10-9
10-10
1
10
10^2
10^3
Frequency
T0=06/01/1980 00:00:00
Bin=13L


Power spectrum
ELP6/30 bits
ideal ELP6/30 bits
Magnitude
10-1
10-2
10-3
10-4
10-5
10-6
10-7
10-8
10-9
10-10
1
10
10^2
10^3
Frequency
T0=06/01/1980 00:00:00
Bin=13L

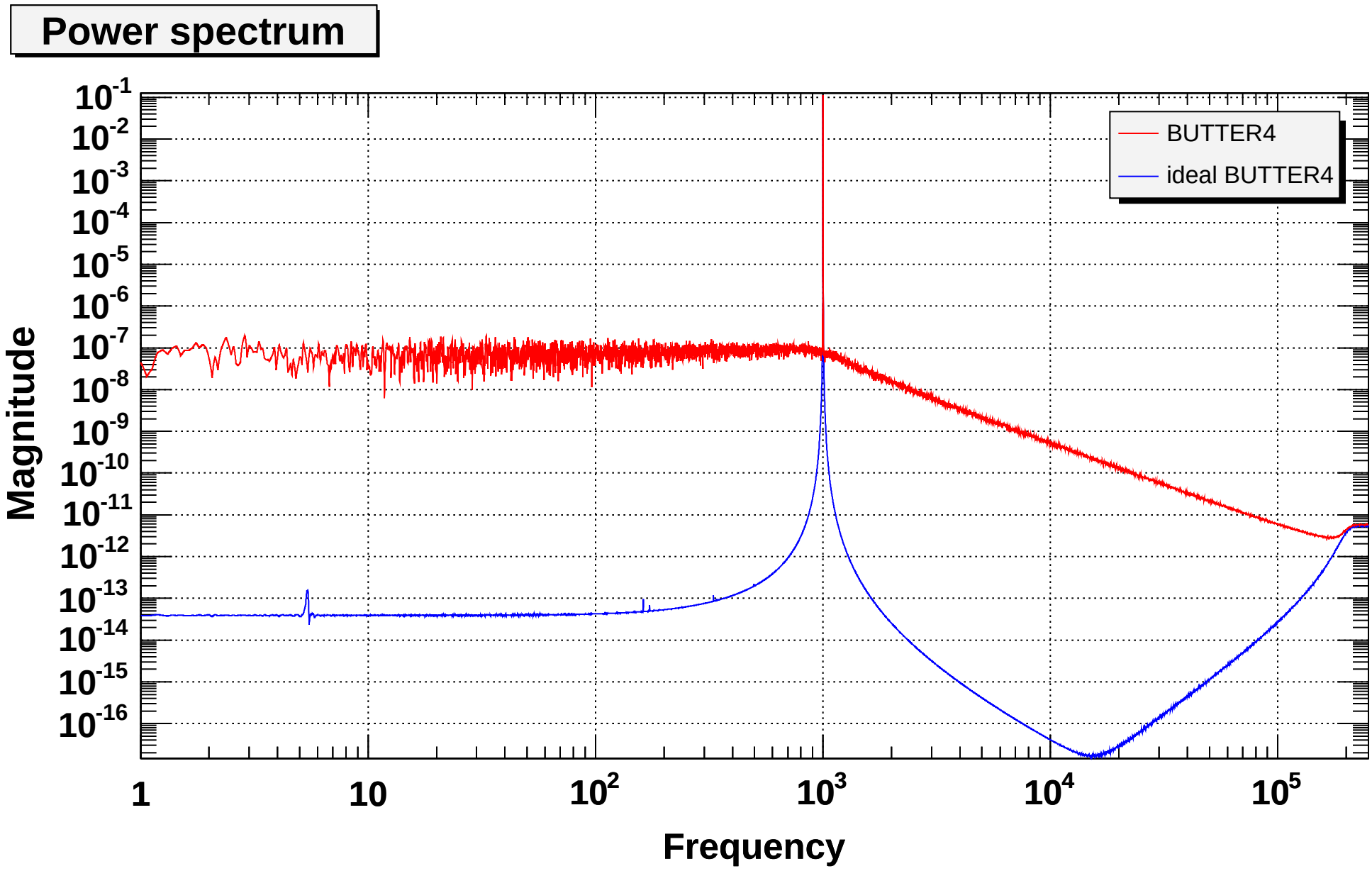

Power spectrum
ELP6/32 bits
ideal ELP6/32 bits
Magnitude
10-1
10-2
10-3
10-4
10-5
10-6
10-7
10-8
10-9
10-10
1
10
10^2
10^3
Frequency
T0=06/01/1980 00:00:00
Bin=13L

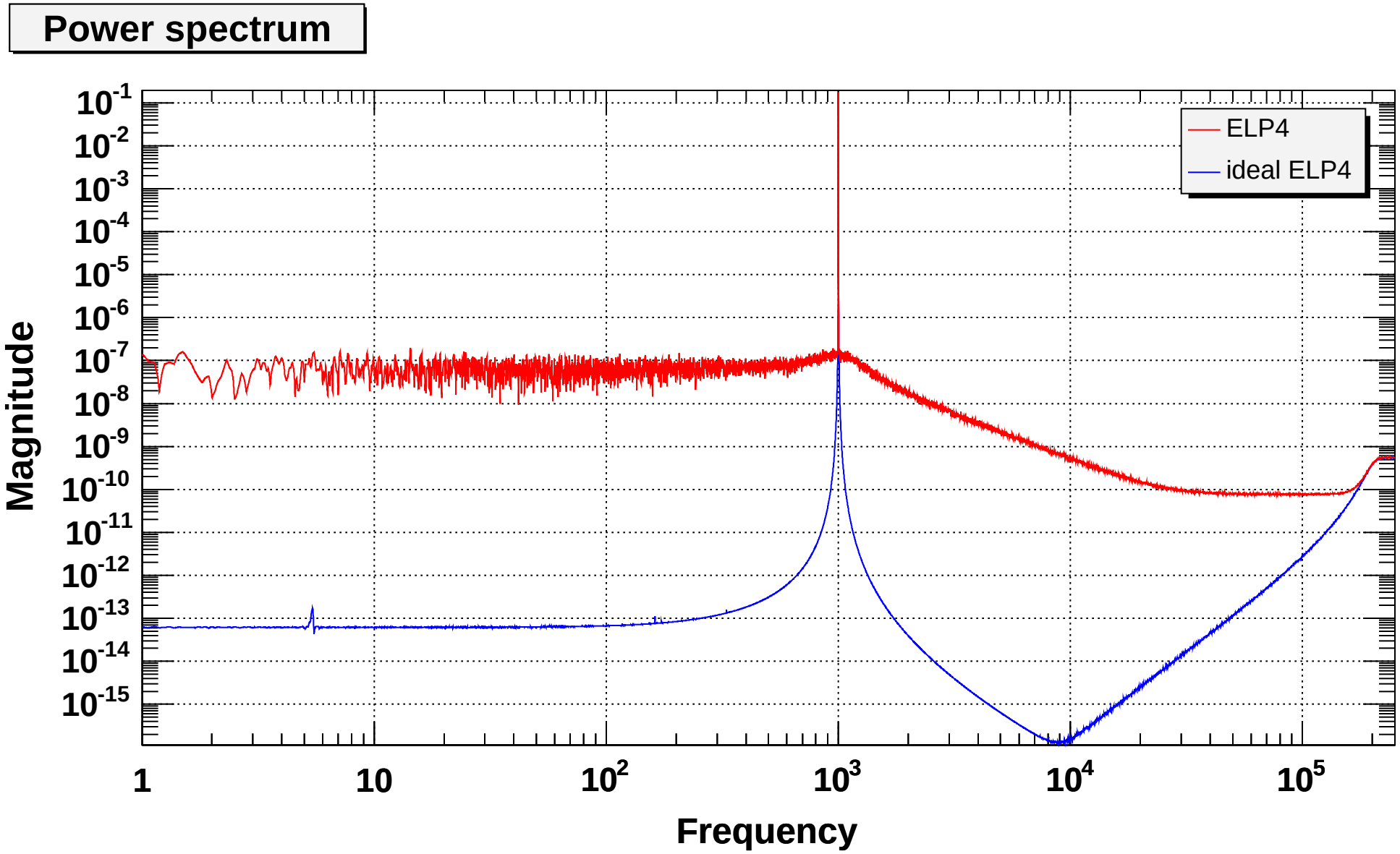

Power spectrum
ELP6/30 bits/1g
ideal ELP6/30 bits/1g
Magnitude
10-1
10-2
10-3
10-4
10-5
10-6
10-7
10-8
10-9
10-10
1
10
10^2
10^3
Frequency
T0=06/01/1980 00:00:00
Bin=13L

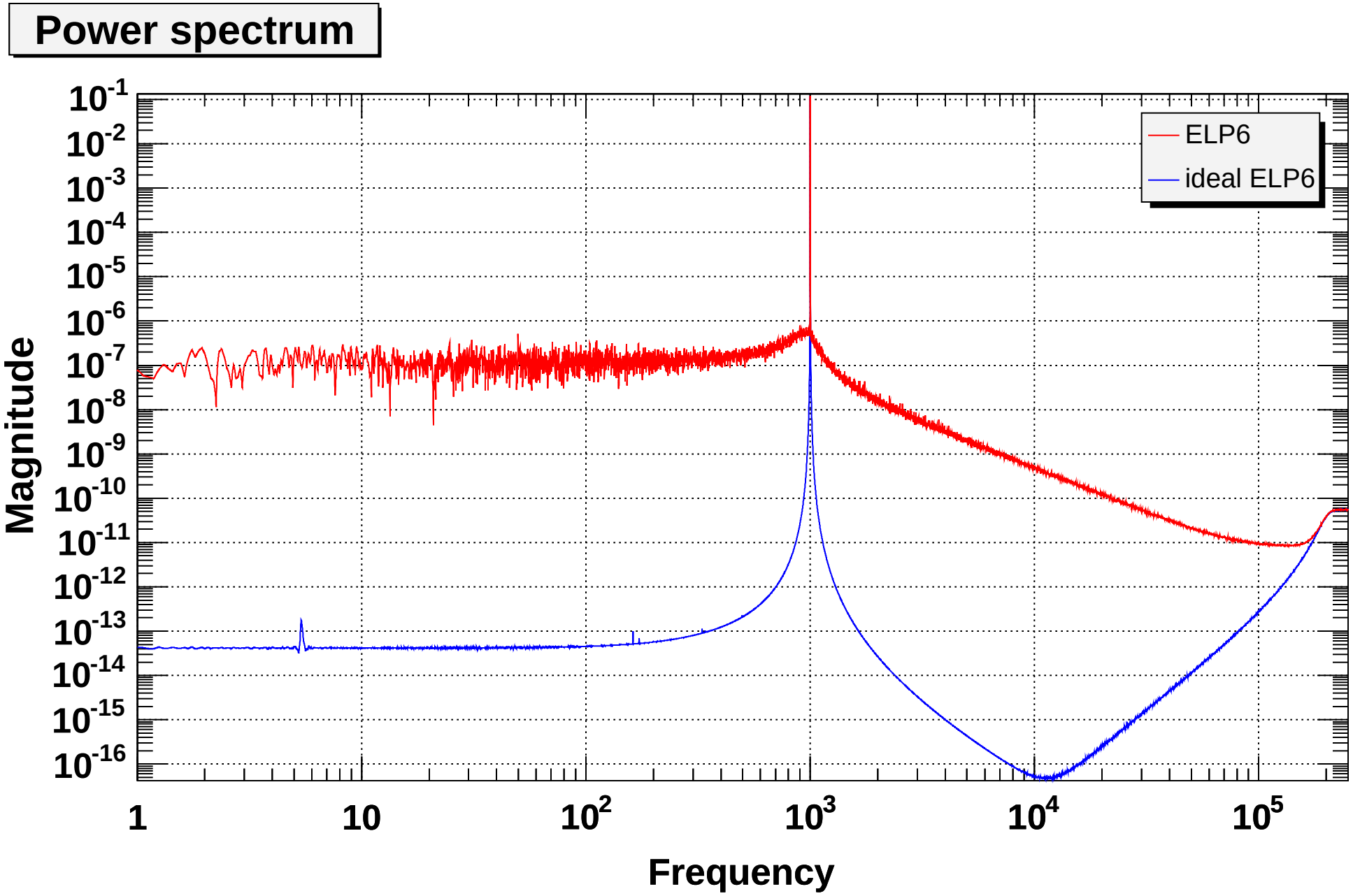

Power spectrum
ELP6/28 bits/48a
ideal ELP6/28 bits/48a
Magnitude
10-1
10-2
10-3
10-4
10-5
10-6
10-7
10-8
10-9
10-10
1
10
10^2
10^3
Frequency
T0=06/01/1980 00:00:00
Bin=13L


Power spectrum
ELP6/30 bits/1g/48a
ideal ELP6/30 bits/1g/48a
Magnitude
10-1
10-2
10-3
10-4
10-5
10-6
10-7
10-8
10-9
10-10
1
10
102
103
Frequency
T0=06/01/1980 00:00:00
Bin=13L

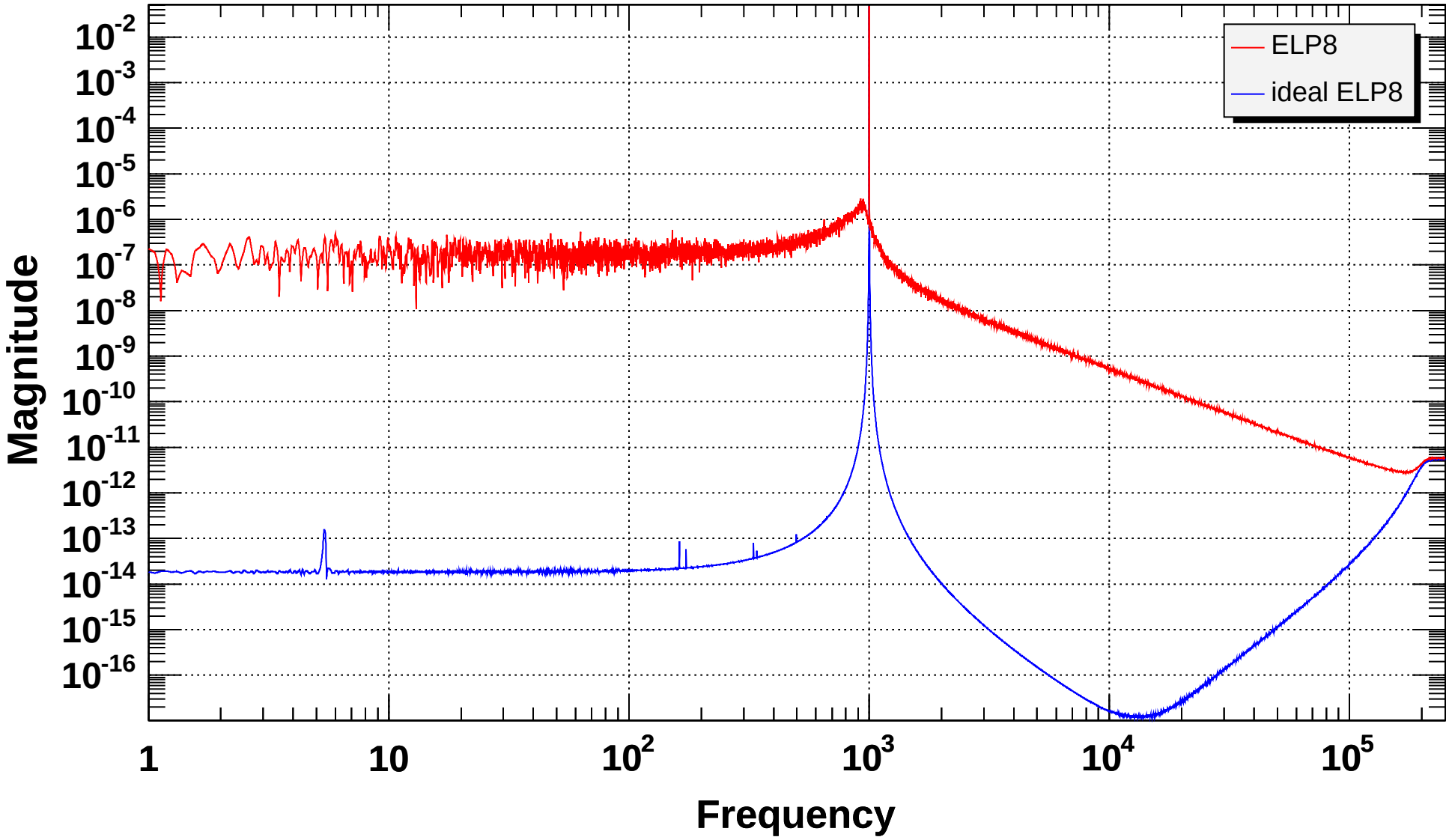
APPENDIX G 2 KHZ SAMPLING RATE


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


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

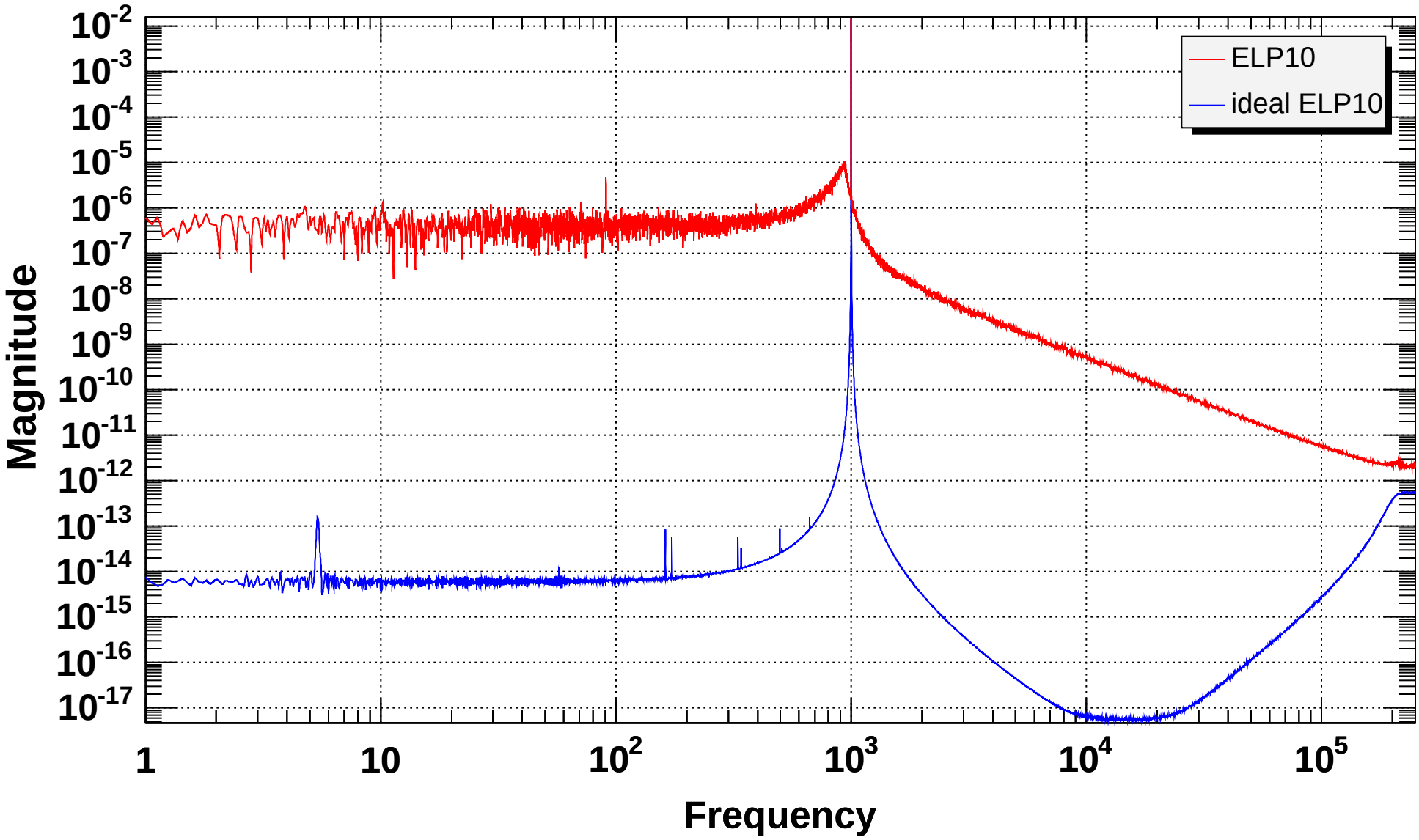

Power spectrum
ELP6
ideal ELP6
Magnitude
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
10^{-7}
10^{-8}
10^{-9}
10^{-10}
10^{-11}
10^{-12}
10^{-13}
10^{-14}
10^{-15}
10^{-16}
1
10
10^{2}
10^{3}
10^{4}
10^{5}
Frequency
T0=06/01/1980 00:00:00
Bin=419L

Power spectrum

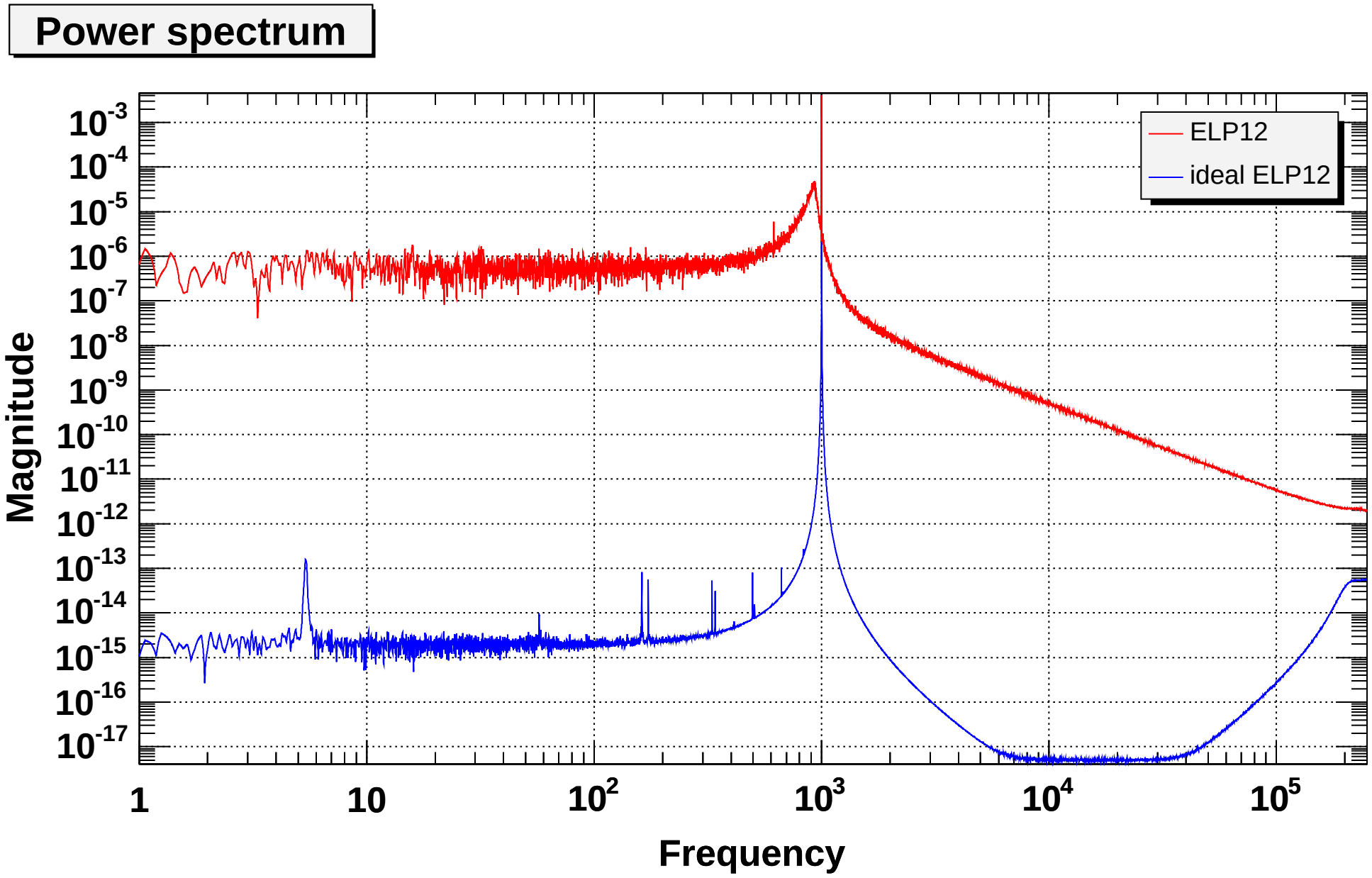


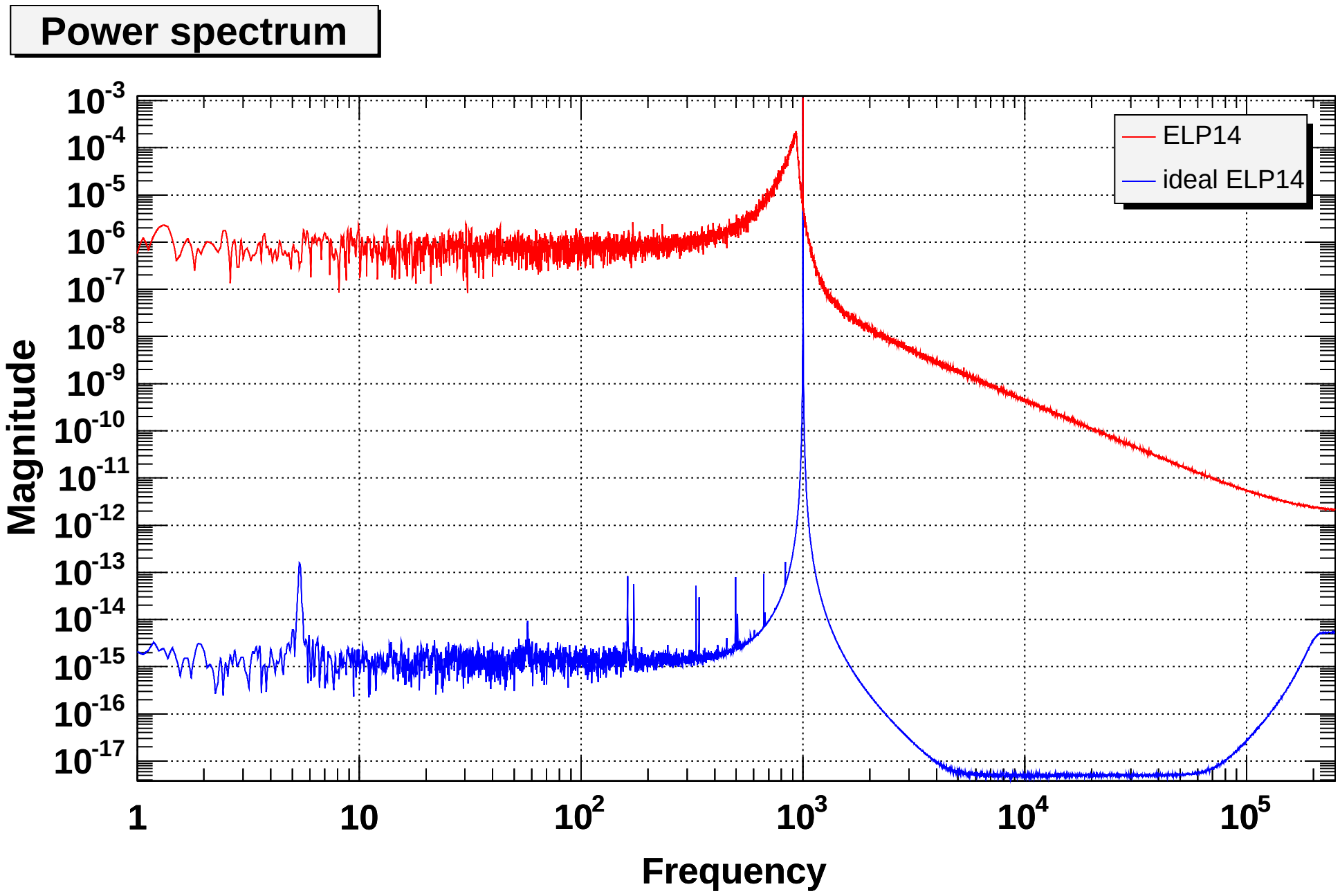
T0=06/01/1980 00:00:00 Bin=419L

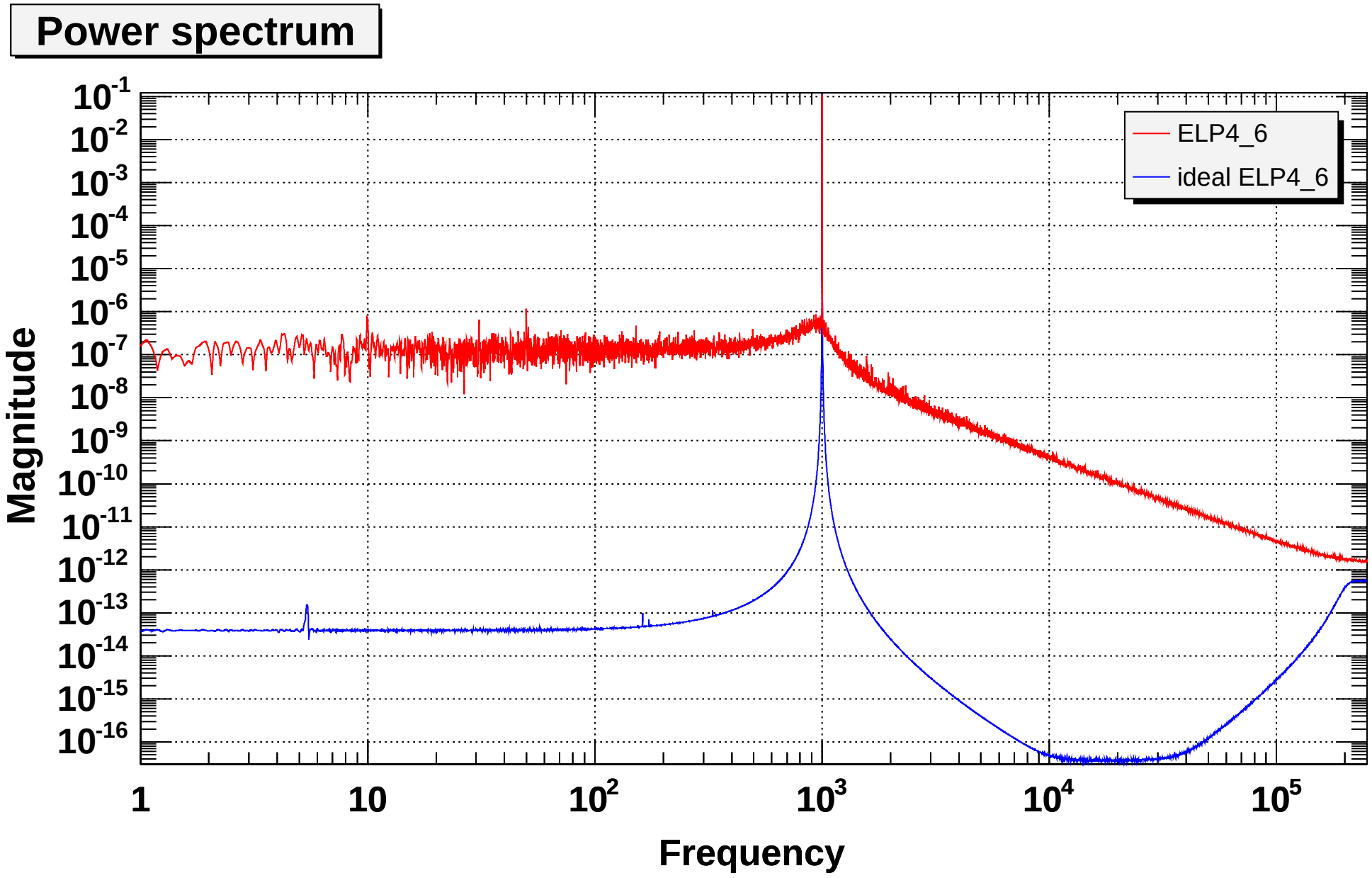
Power spectrum

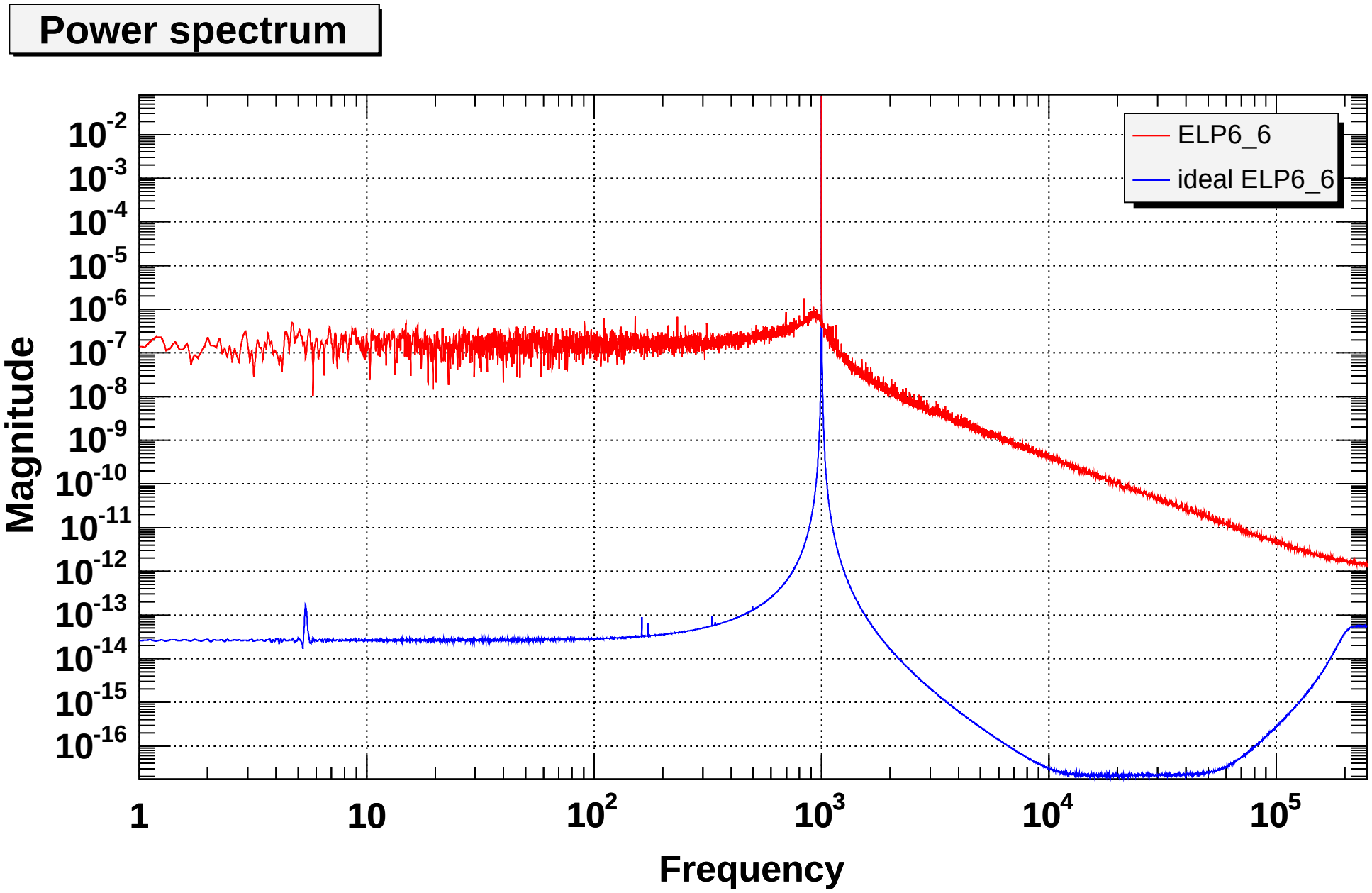


T0=06/01/1980 00:00:00 Bin=419L

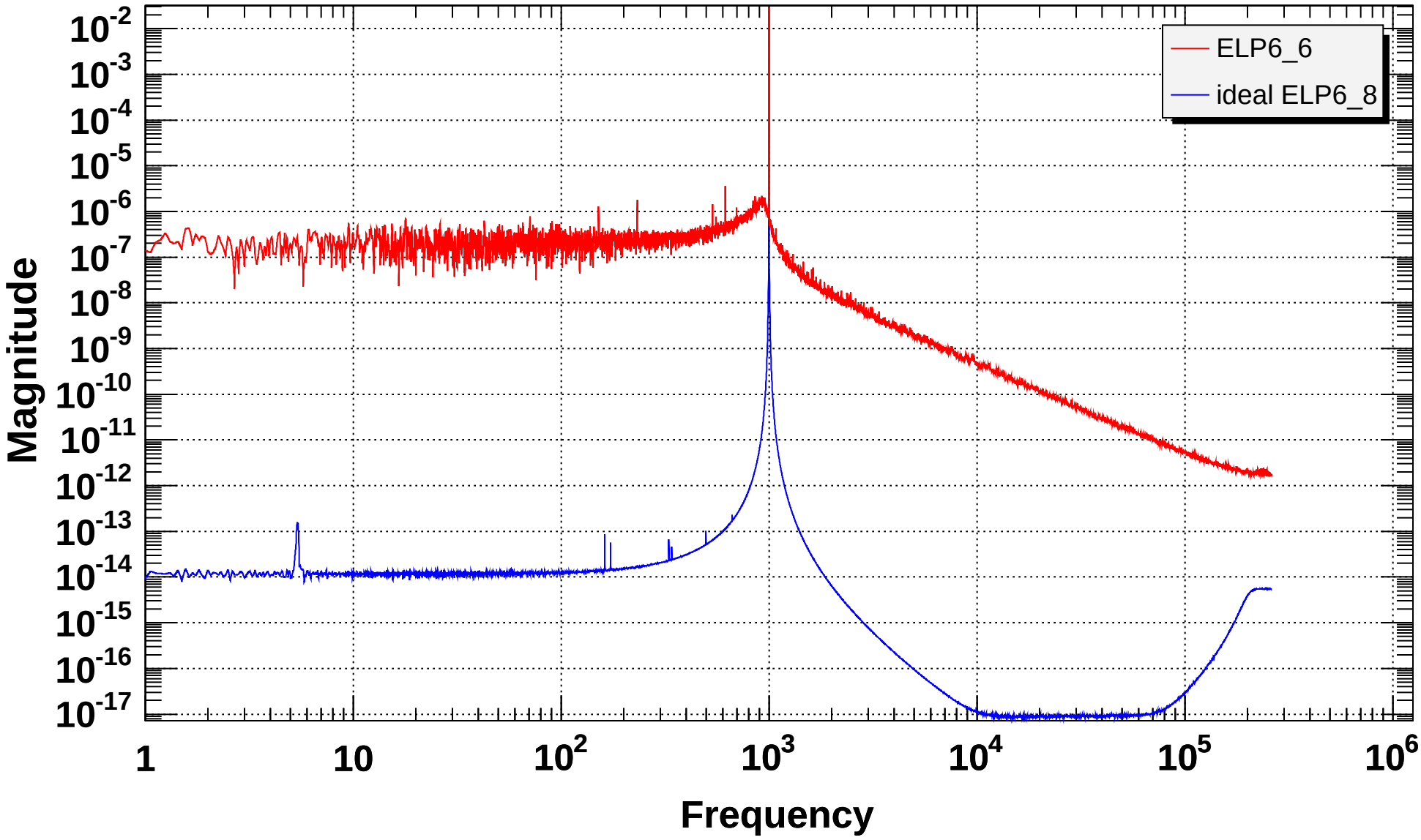

Power spectrum
ELP12
ideal ELP12
Magnitude
10^{-3}
10^{-4}
10^{-5}
10^{-6}
10^{-7}
10^{-8}
10^{-9}
10^{-10}
10^{-11}
10^{-12}
10^{-13}
10^{-14}
10^{-15}
10^{-16}
10^{-17}
1
10
10^{2}
10^{3}
10^{4}
10^{5}
Frequency
T0=06/01/1980 00:00:00
Bin=419L


Power spectrum
ELP14
ideal ELP14
Magnitude
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
10-17
1
10
10^2
10^3
10^4
10^5
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
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
Frequency
T0=06/01/1980 00:00:00
Bin=419L

Power spectrum



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