LIGO Laboratory / LIGO Scientific Collaboration

LIGO-T070240-00-D *advanced LIGO* 9/24/2007

An improved low-pass filter for the feedback controls system of advanced LIGO

Marzia Colombini

Distribution of this document:
LIGO Science Collaboration

This is an internal working note
of the LIGO Project.

California Institute of Technology
LIGO Project – MS 18-34
1200 E. California Blvd.
Pasadena, CA 91125
Phone (626) 395-2129
Fax (626) 304-9834
E-mail: info@ligo.caltech.edu

Massachusetts Institute of Technology
LIGO Project – NW22-295
185 Albany St
Cambridge, MA 02139
Phone (617) 253-4824
Fax (617) 253-7014
E-mail: info@ligo.mit.edu

LIGO Hanford Observatory
P.O. Box 159
Richland WA 99352
Phone 509-372-8106
Fax 509-372-8137

LIGO Livingston Observatory
P.O. Box 940
Livingston, LA 70754
Phone 225-686-3100
Fax 225-686-7189

http://www.ligo.caltech.edu/

September 24, 2007

VIRGO-LIGO EXCHANGE FELLOWSHIP

AN IMPROVED LOW-PASS FILTER FOR THE FEEDBACK CONTROLS SYSTEM OF ADVANCED LIGO

Marzia Colombini
(Mentor: Daniel Sigg)

ABSTRACT
This project is part of a larger effort to develop new analog-to-digital converters and new filters for the next generation LIGO interferometers: Advanced LIGO. In particular we plan to incorporate an IIR filter (infinite impulse response filter) on a FPGAs (field programmable gate arrays) to process the acquired signals of the feedback system. Our work consists adding 17 bits to an existing filter to improve its precision and to mitigate quantization effects.
A hardware simulation showed that the new filter indeed has a higher precision. The performance is maintened even at lower cut-off frequency filters.

All gravitational wave interferometers implement complex feedback control systems to stay within acceptable operating parameters; both mirror positions and angle inclinations have to be adjusted continuously[1].
Consider the simplified scheme of an interferometer in figure 1 with a laser, the test masses (the mirrors) and the photodetector. The data coming from the photodiode are analog but they need to be stored and processed in a computer. Due correction signals are then sent along a fiber cable back to the mirrors, where they are applied to the actuators.

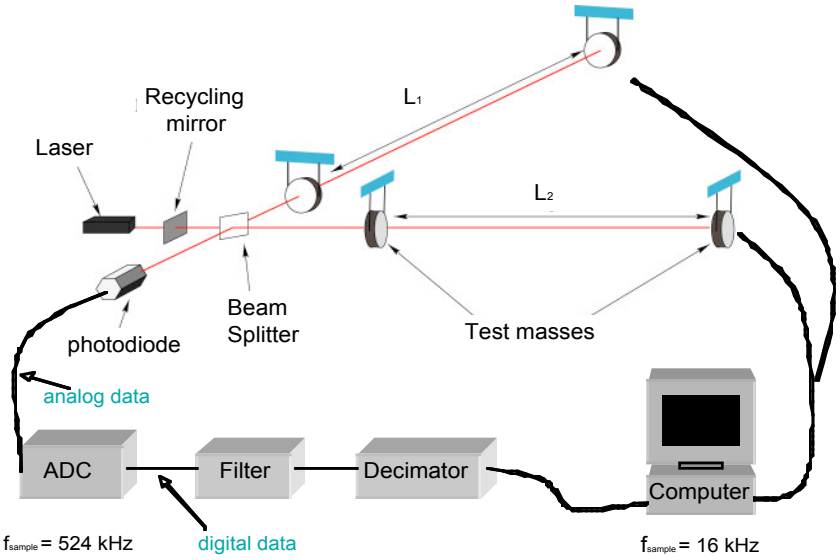

Figure 1: a simplified scheme of an interferometer and a feedback controls system

Between detector and computer we need to have an analog-digital converter (ADC). This ADC has a very high sampling frequency (2^{19} Hz, 524,288 Hz) which allows us to reduce the noise level. In fact higher sampling frequency means that the noise is spread out over a wider frequency range. On the other hand, computer works at slower sampling frequency, 2^{14} Hz (16,384 Hz), because we can't keep and we don't need the high frequency data. So, we need a low-pass filter to connect these two parts and to prevent aliasing effects. After the filter there is a decimator to reduce the data rate by simply dropping data samples.
The interferometer is optimized for low frequencies: it has a sensitivity minimum at 150 Hz. We take data from 50 Hz to a few kHz because interesting astrophysical sources such as Supernovae and millisecond pulsars are in this band.

The choice of the filter is very important: in fact, we want a low-pass filter which completely cuts frequencies higher than a certain cut-off frequency, and leaves lower frequencies unchanged.
The filter we use is an IIR filter on a FPGA[2]. This device contains programmable logic blocks and interconnets. An IIR calculates the result using one or more of its previous input and output data samples. This is the reason it is called "infinite": there are infinite samples which could be used. The number of used previous samples indicates the filter order. The formula that describes a second order filter for a single Second Order Section (SOS) is[3]:

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

where the x_i are input values from different times, now, one clock cycle before and two clock cycles ago; the y_i are output signals at different times, too. The a and b coefficients depend on the chosen filter (in fact there could be different kind of filter, with different cut-off frequencies and more terms to calculate), while c_0 is an individual gain factor, as we will explain later.

An IIR filter can be written as a product of SOSs:

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

In this formula there is also the overall gain factor g to correct for whatever the c_0 don't account for. Note that N_s indicates the number of terms to calculate for each kind of filter.
Basically we need a multiplier and an accumulator[4]: figure 2 shows a block diagram of the filter engine.

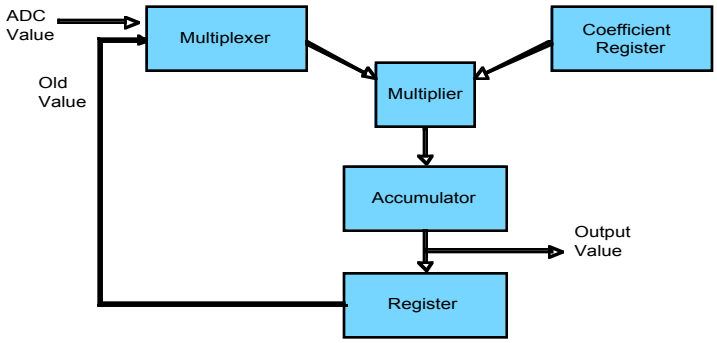

Figure 2: block diagram of the filte engine

There is an initial multiplexer to choose between the input ADC value and the old filter values kept in a register set: the selected number is sent to the multiplier together with a coefficient which is taken from a different memory. After multiplication, the accumulator calculates the result and sends it to the register set so it can be reused.
In reality, schematic is more complicated. First, because the muliplier works with only 18 bit numbers and because we need a more accurate filter. In the old filter we were using 35 bits, whereas in the new filter we are using 52 bits.
So we have to split each number in multiple parts. Each combination has to be multiplied and shift in the right position: this means that we need another component, a shifter. For the old filter, we have only two parts, each of 17 bits, the most significant bits (MSB) and the least significant bits (LSB), plus a bit for the sign. For the new filter we have three parts: MSB, LSB and the intermediate significant bits (ISB).
This split is in the initial multiplexer where we choose which part to send to the multiplier. This operation for the new filter is shown in figure 3.

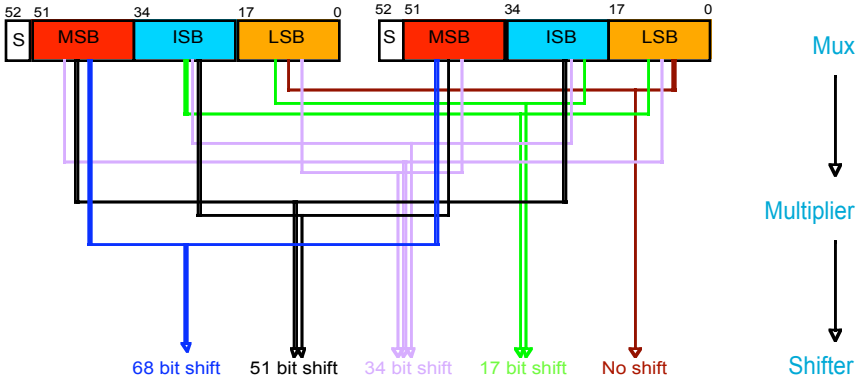

Figure 3: number splitting, combination and shifting

In the Table 1 we show the bit number for each quantity, confronting the old and the new filters.

Parameter	bits		decimal point		extended at front		added at end	
	old	new	old	new	old	new	old	new
ADC Value		18		0		--		--
input value		32		9		5		9
history/filter value	35	(35+17) 52	12	(12+17) 29		0	3	(3+17) 20
filter/gain value	35	(35+17) 52	33	(33+17) 50		--		--
multiplier result	70	(52+52)104	45	(50+29) 79		0		0
accumulator	48	64	20	36		3	-25	-43
filter/history value	35	(35+17) 52	12	(12+17) 29		-5	-8	-7
final output		32		9		0	-3	(3+17) -20

Table 1: bit number table

The ADC value has 18 bits but the input number of both the filters has 32 bits: we have added 9 all-zero bits at the end after the decimal point, and 5 sign-extended bits in the front. Sign extension means that we repeat the sign, 1 for negative numbers and 0 for positive numbers. We split this number into two or three parts, depending on the kind of filter, adding as many zero bits as we need to reach 18 bit numbers for each part. The first difference is in the coefficient values (history values and gain factors) and in the multiplier result, where we have longer numbers. Before the accumulator we cut the least significant 25 and 43 bits, respectively. We need more accurate numbers only during the calculations, not in the final result. In fact the output has again 32 bits, exactly like the input value.

The new filter schematic is shown in figure 4. The Multishifter shifts the multiplier result in its correct position. The Output Register stores the output value. The Reg52 before the History Memory keeps the accumulator results for 4 clock cycles before writing them into the History Memory, since they don't want to overwrite the old values while they are still in use. The Overflow Detector checks it there isn't any overflow after the summation; the two Reg2 allow us to reach the correct delay during multiplications and sums. The Barrel Shifter implements the individual gain factor, c_0. The problem is when we calculate the formula we obtain very big output numbers; in order not to overflow, we divide or, as in this case, shift the input number by the factor c_0.
Note, that the Multishifter should contain the long 104 bit number but its output has only 64 bits. So, since we can throw away the last 40 digits to obtain a result with the correct number of bits, we don't need to calculate product of the two LSB.
To better understand how each part of the schematics work look at the filter pipeline in Table 2. This table lists what values are present in each clock cycle in the different components of the schematics. It is used to define the microcode: a series of18 bit numbers which control the multiplexer, the multishifter and the history memory. It determines what numbers to choose, how many bits to shift and when to write and when to read in the History Memory. At each clock cycle a new microcode value is loaded, repeating itself for each filter calculation. There are 128 clock cycles in a filter calculation in which we calculate three cascading SOSs.

The old filter is less sensitive than we need. In fact, we can look at the power spectrum plots of the filter output signal with different cut-off frequencies (7400 Hz, figure 5, and 900 Hz, figure 6, for a 4^{th} order Butterworth filter): the blue line represents the ADC input signal, a 1 kHz sine wave stimuli which lasts for 17 s, while the red line represents the filter output signal.We notice that for the lower cut-off frequency filter the noise level is higher, 10^{-7} instead 10^{-8}. This level is caused by the filter.The ADC quantization noise due to the fact we are converting an analog signal with infinite precision into a finite number is about 10^{-8}. We also obtain a higher noise level with peaking if we

choose a higher order filter. For example a 12^{th} order Elliptic with more terms to calculate is shown in figure 7.

We can see the effect of bit resolution in figure 8 and figure 9: in the first plot we have a 6^{th} Elliptic filter calculated with 22 bits, and in the second plot we have the same filter calculated with 32 bits. It is evident that we have gained a factor 10 in the noise level. We can expect that adding another 17 bits in the calculations, as done by the new filter, will lower this level again.

The results are shown in the next figures: one can see the power spectrum of a 6^{th} order Elliptic filter with different cut-off frequencies (7400 Hz, 900 Hz, 110 Hz and 14 Hz) using the same stimulus, a 1 kHz sine wave which lasts for 17 s. The blue curve is calculated with 27 bits of precision, whereas the red curve is calculated with 18 bits. In every case, even with the lower cut-off frequency filter the noise level is just below 10^{-8}, as we expect. If we had an ADC with more digits, we could lower this level as far as 10^{-11}.
Looking at plots with different cut-off frequency filters, we see that the noise level calculated with 27 bits is increasing: it is around 10^{-11} for the 7400 Hz filter, whereas it is just below 10^{-8} for the 14 Hz filter.

CONCLUSIONS

We conclude that adding 17 bits of resolution is enough to reach the noise level of the ADC. Furthermore, our filter could lower noise level again, if we used an ADC with more than 18 bits. Our filter can implement only low order filters, up to the sixth oder because we can calculate only three SOSs. So, if we need additional filtering, we either need to add another multiplier or we have to cascade multiple filter engines.

[1] B. Abbott et al, Nucl. Instrum. and Meth. A 517 (2004) 154-179
[2] http://www.xilinx.com/
[3] A.V. Oppenheim, R.W. Schafer, Discrete-Time Signal Processing, Prentice Hall
[4] D. Sigg, Implementig an IIR Filter in Hardware, LIGO T050060-00

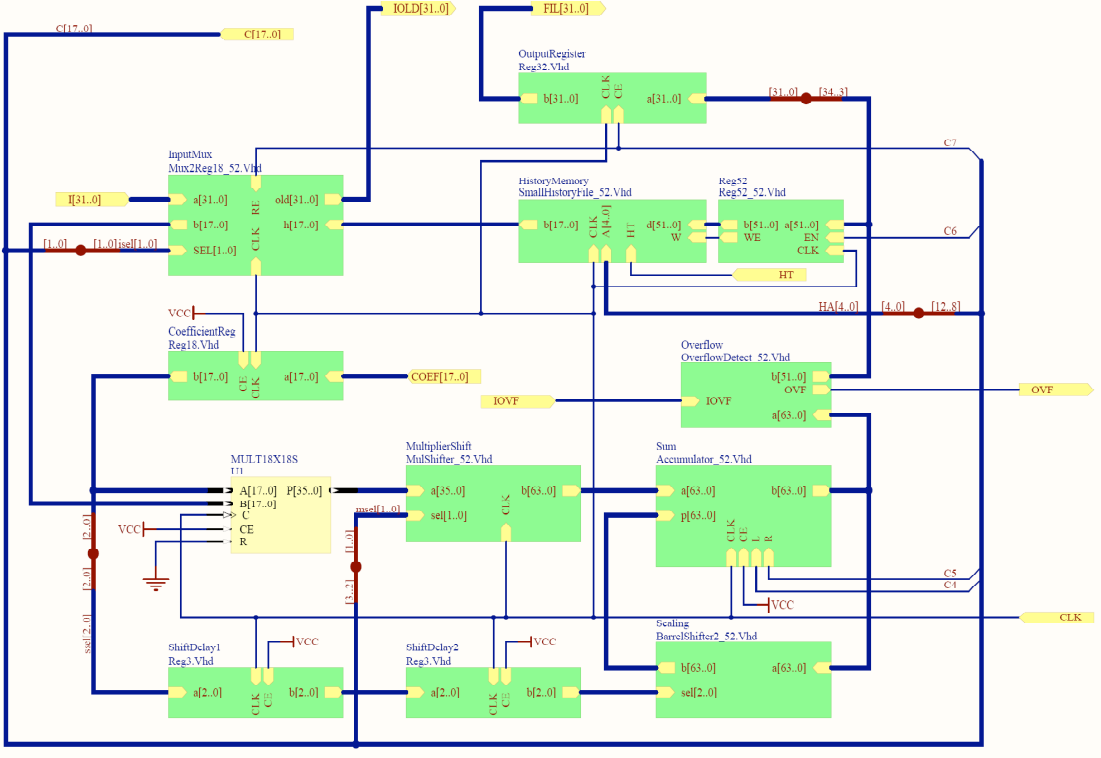

figure 4: New corrected schematic for 52 bit Filter

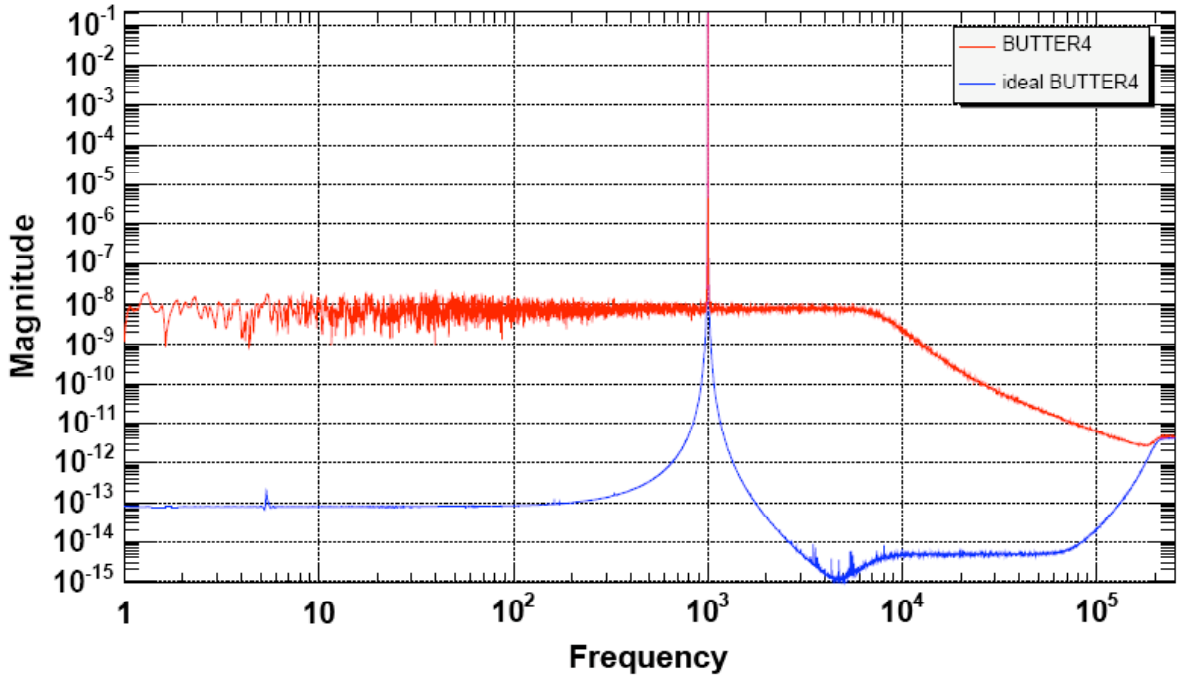

figure 5: Old 35 bit Filter, power spectrum plot of 7400 Hz cut-off Butter 4 output signal

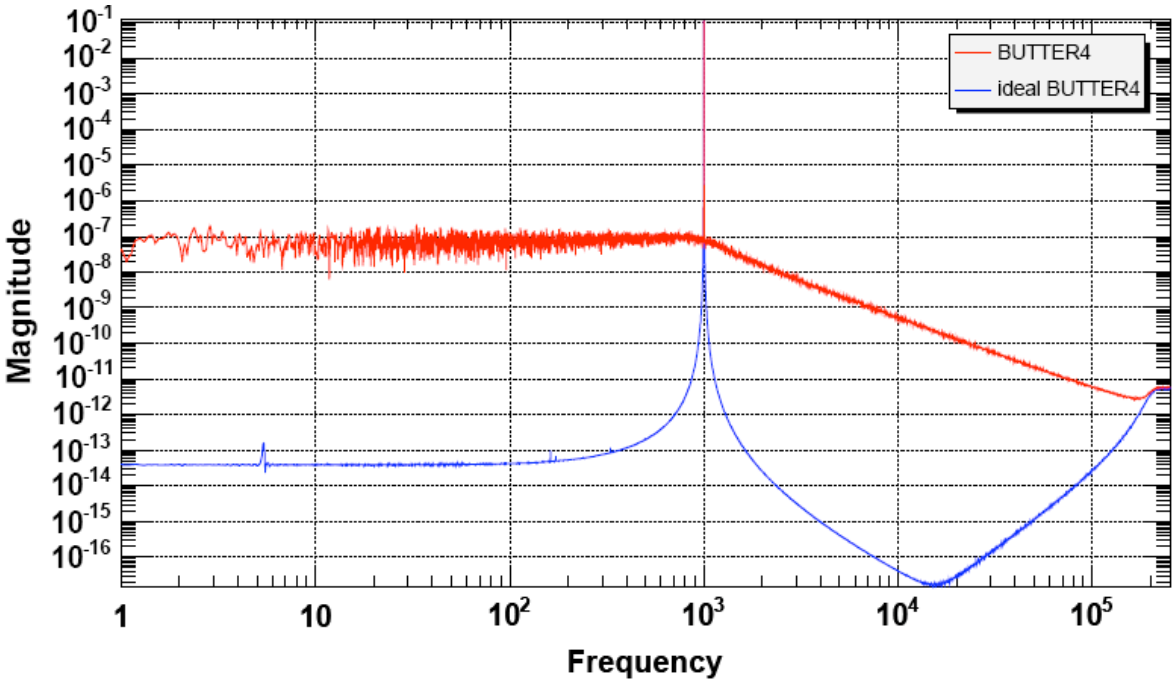

figure 6: Old 35 bit Filter, power spectrum plot of 900 Hz cut-off Butter 4 output signal

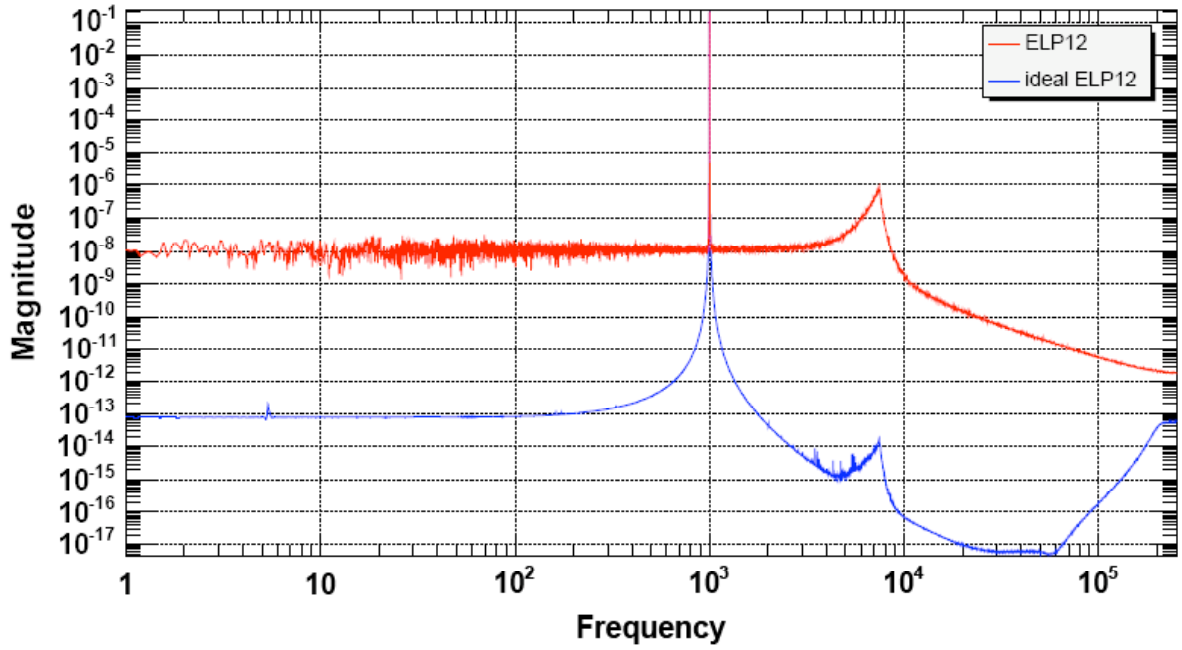

figure 7: Old 35 bit Filter, power spectrum plot of 7400 Hz cut-off Elliptic 12 output signal

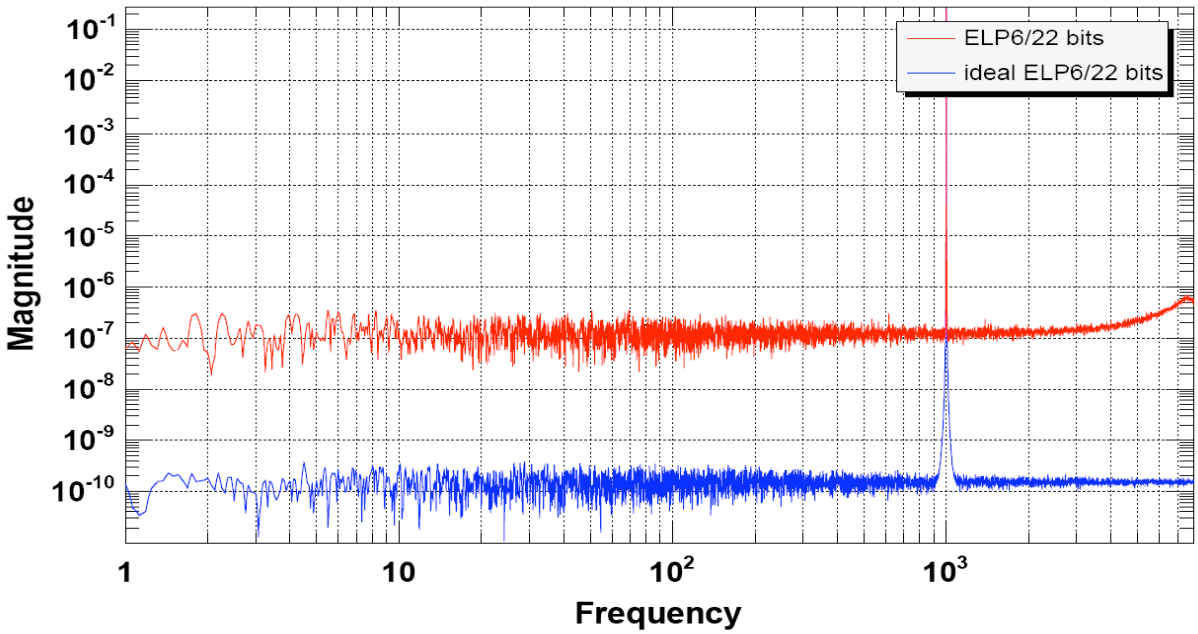

figure 8: Old 35 bit Filter, effect of bit resolution, Elliptic 6 calculated with 22 bits

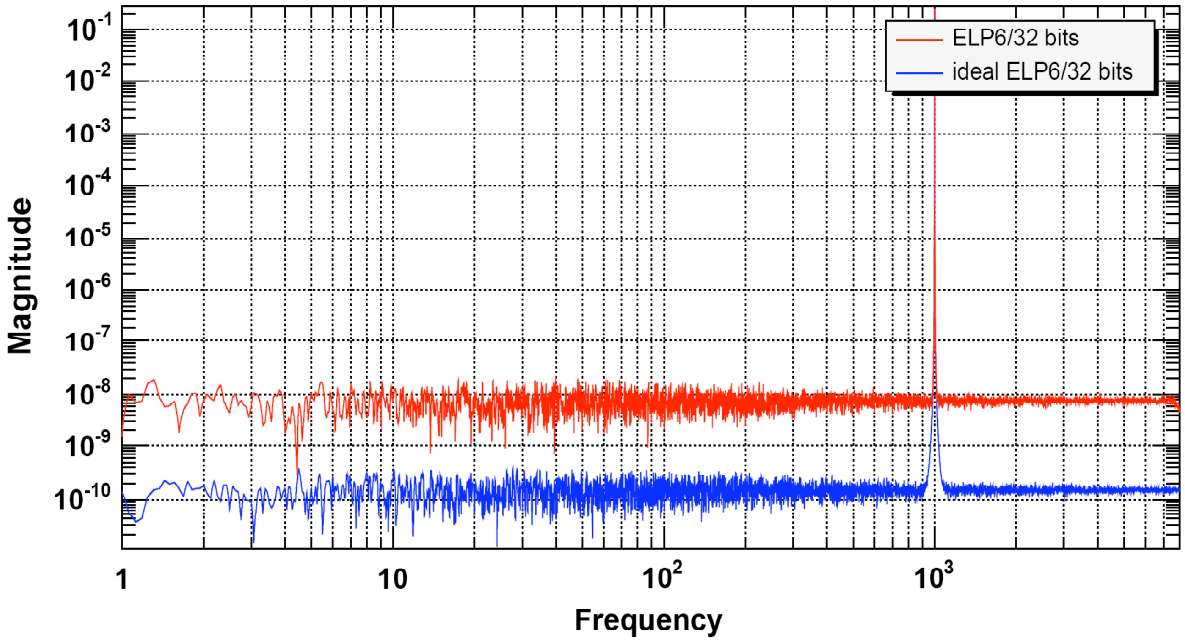

figure 9: Old 35 bit Filter, effect of bit resolution, Elliptic 6 calculated with 32 bits

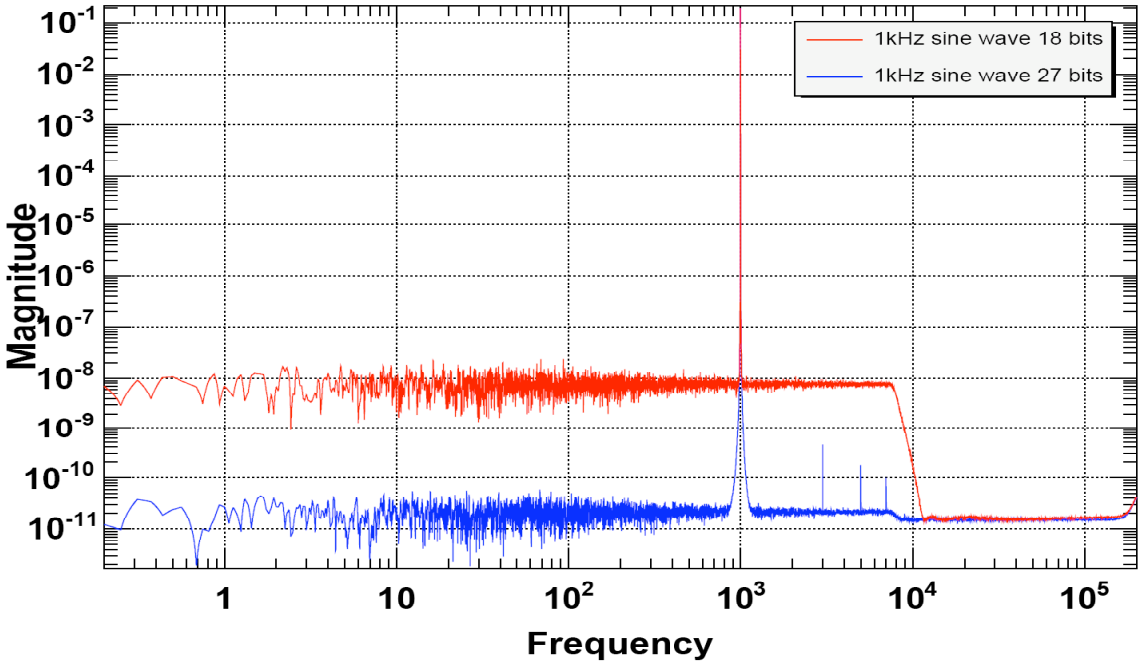

figure 10: New 52 bit Filter, power spectrum plot of 7400 Hz cut-off Elliptic 6 output signal

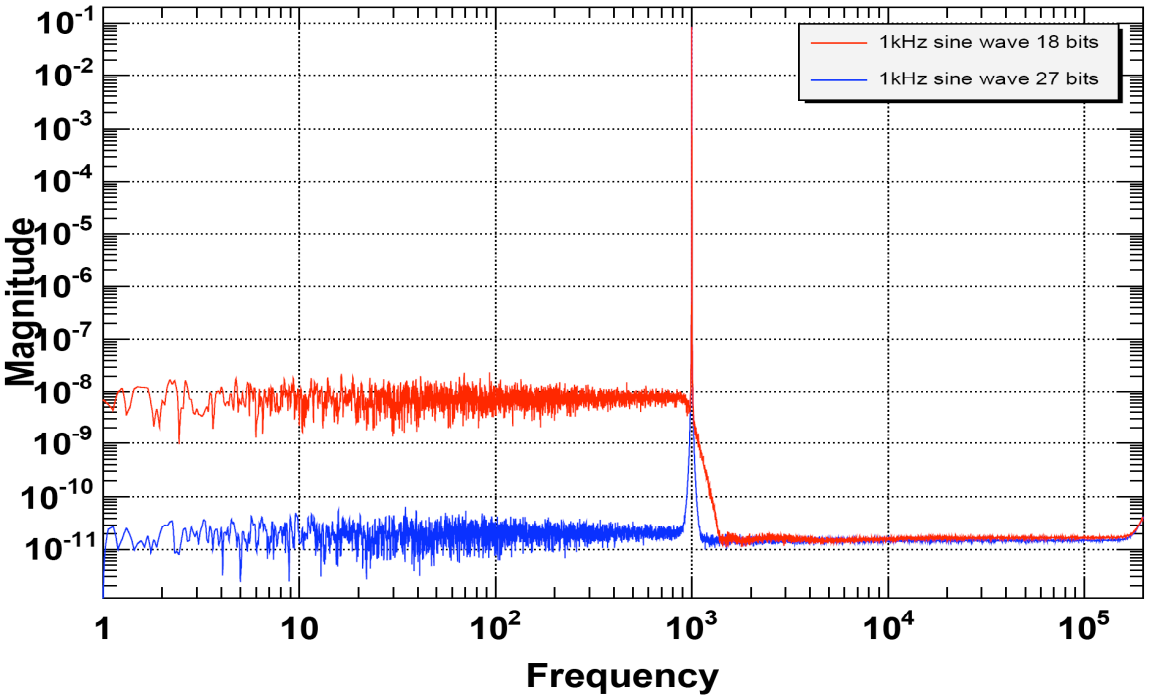

figure 11: New 52 bit Filter, power spectrum plot of 900 Hz cut-off Elliptic 6 output signal

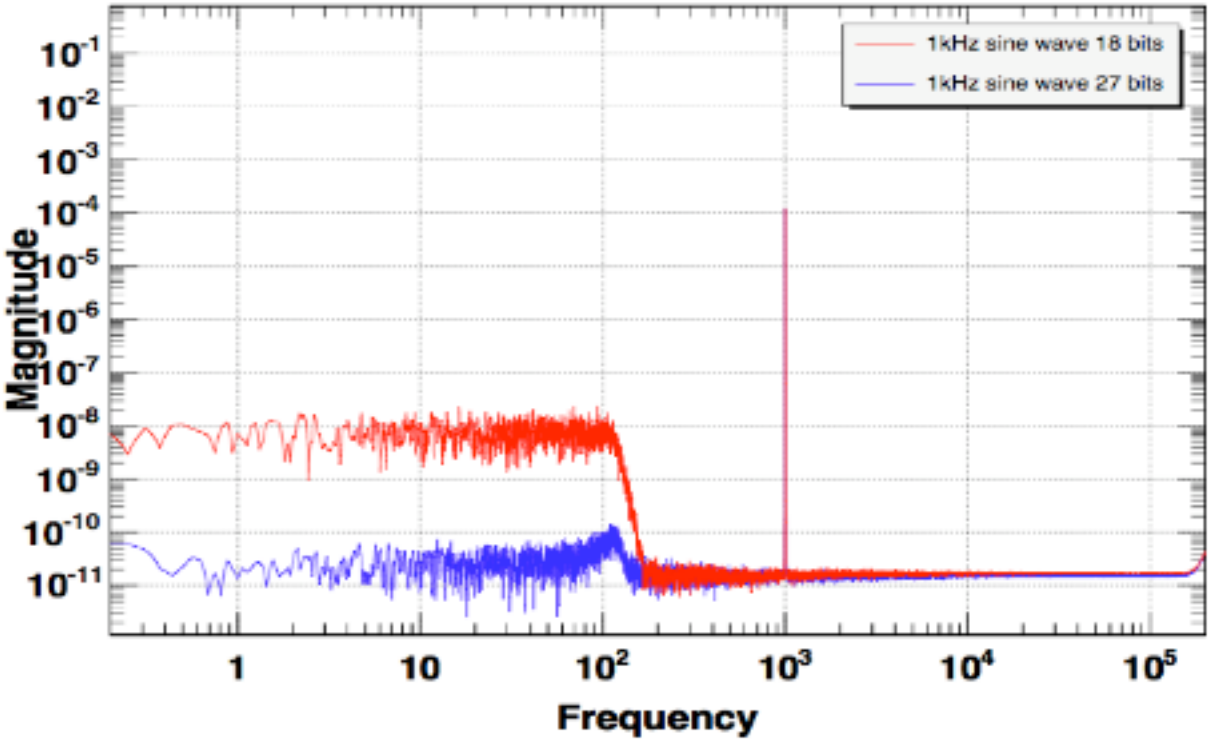

figure 12: New 52 bit Filter, power spectrum plot of 110 Hz cut-off Elliptic 6 output signal

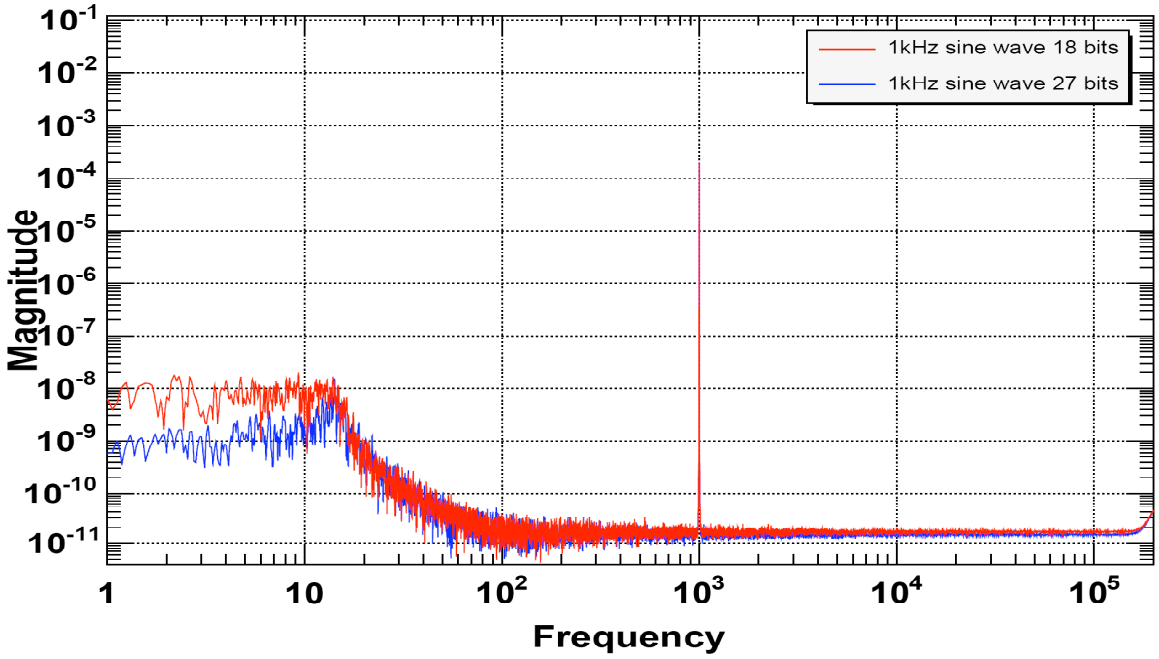

figure 13: New 52 bit Filter, power spectrum plot of 14 Hz cut-off Elliptic 6 output signal

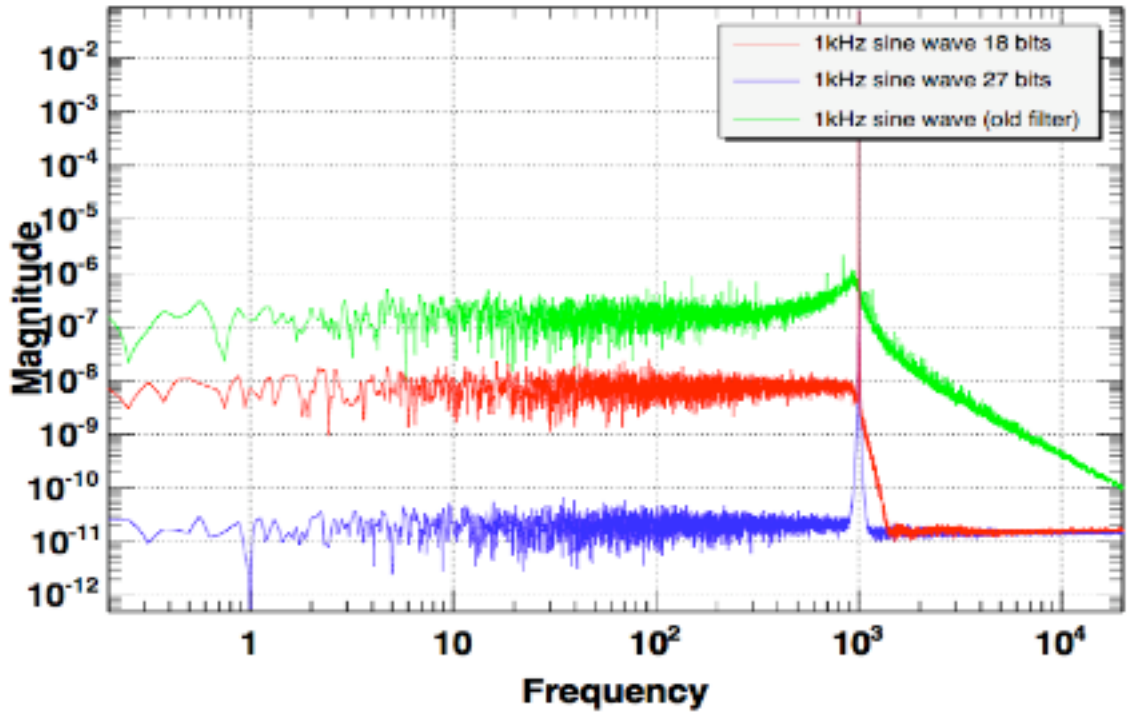

figure 14: New 52 bit Filter and Old 35 bit Filter results, power spectrum plot of 900 Hz cut-off Elliptic 6 output signal

Table 2: Filter Pipeline Table

Clock	Hist Mem	InputMux	Coeff	Coeff Reg	Multip	Shifter	Accumulat
0	-	-	g,i	-	-	-	-
1	-	X00,l	g,m	g,i	-	-	-
2	-	X00,l	g,l	g,m	X00,l*g,i	-	-
3	-	X00,i	g,i	g,l	X00,l*g,m	X00,l*g,i	0
4	-	X00,i	g,m	g,i	X00,i*g,l	X00,l*g,m	...+X00,l*g,i
5	-	X00,i	g,l	g,m	X00,i*g,i	X00,i*g,l	...+X00,l*g,m
6	-	X00,m	g,i	g,l	X00,i*g,m	X00,i*g,i	...+X00,i*g,l
7	-	X00,m	g,m	g,i	X00,m*g,l	X00,i*g,m	...+X00,i*g,i
8	X-20,l	X00,m	b20,i	g,m	X00,m*g,i	X00,m*g,l	...+X00,i*g,m
9	X-20,l	X-20,l	b20,m	b20,i	X00,m*g.m	X00,m*g,i	...+X00,m*g,l
10	X-20,i	X-20,l	b20,l	b20,m	X-20,l*b20,i	X00,m*g.m	...+X00,m*g,i
11	X-20,i	X-20,i	b20,i	b20,l	X-20,l*b20,m	X-20,l*b20,i	...+X00,m*g.m=g*X0
12	X-20,i	X-20,i	b20,m	b20,i	X-20.i*b20,l	X-20,l*b20,m	...+X-20,l*b20,i
13	X-20,m	X-20,i	b20,l	b20,m	X-20,i*b20,i	X-20.i*b20,l	...+X-20,l*b20,m
14	X-20,m	X-20,m	b20,i	b20,l	X-20,i*b20,m	X-20,i*b20,i	...+X-20.i*b20,l
15	X-20,m	X-20,m	b20,m	b20,i	X-20,m*b20,l	X-20,i*b20,m	...+X-20,i*b20,i
16	X-10,l	X-20,m	b10,i	b20,m	X-20,m*b20,i	X-20,m*b20,l	...+X-20,i*b20,m
17	X-10,l	X-10,l	b10,m	b10,i	X-20,m*b20,m	X-20,m*b20,i	...+X-20,m*b20,l
18	X-10,i	X-10,l	b10,l	b10,m	X-10,l*b10,i	X-20,m*b20,m	...+X-20,m*b20,i
19	X-10,i	X-10,i	b10,i	b10,l	X-10,l*b10,m	X-10,l*b10,i	...+X-20,m*b20,m=g*X00+b
20	X-10,i	X-10,i	b10,m	b10,i	X-10.i*b10,l	X-10,l*b10,m	...+X-10,l*b10,i
21	X-10,m	X-10,i	b10,l	b10,m	X-10,i*b10,i	X-10.i*b10,l	...+X-10,l*b10,m
22	X-10,m	X-10,m	b10,i	b10,l	X-10,i*b10,m	X-10,i*b10,i	...+X-10.i*b10,l
23	X-10,m	X-10,m	b10,m	b10,i	X-10,m*b10,l	X-10,i*b10,m	...+X-10,i*b10,i
24	-	X-10,m	c00	b10,m	X-10,m*b10,i	X-10,m*b10,l	...+X-10,i*b10,m
25	Y-20,l	-	a20,i	c00	X-10,m*b10,m	X-10,m*b10,i	...+X-10,m*b10,l
26	Y-20,l	Y-20,l	a20,m	a20,i	-	X-10,m*b10,m	...+X-10,m*b10,i
27	Y-20,i	Y-20,l	a20,l	a20,m	Y-20,l*a20,i	-	...+X-10,m*b10,m=g*X00+b20*X·
28	Y-20,i	Y-20,i	a20,i	a20,l	Y-20,l*a20,m	Y-20,l*a20,i	c00*X
29	Y-20,i	Y-20,i	a20,m	a20,i	Y-20,i*a20,l	Y-20,l*a20,m	c00*X+Y-20,l*a20,i
30	Y-20,m	Y-20,i	a20,l	a20,m	Y-20,i*a20,i	Y-20,i*a20,l	...+Y-20,l*a20,m
31	Y-20,m	Y-20,m	a20,i	a20,l	Y-20,i*a20,m	Y-20,i*a20,i	...+Y-20,i*a20,l
32	X-20,m	Y-20,m	a20,m	a20,i	Y-20,m*a20,l	Y-20,i*a20,m	...+Y-20,i*a20,i
33	Y-10,l	X-20,m	a10,i	a20,m	Y-20,m*a20,i	Y-20,m*a20,l	...+Y-20,i*a20,m
34	Y-10,l	Y-10,l	a10,m	a10,i	Y-20,m*a20,m	Y-20,m*a20,i	...+Y-20,m*a20,l
35	Y-10,i	Y-10,l	a10,l	a10,m	Y-10,l*a10,i	Y-20,m*a20,m	...+Y-20,m*a20,i
36	Y-10,i	Y-10,i	a10,i	a10,l	Y-10,l*a10,m	Y-10,l*a10,i	...+Y-20,m*a20,m=c00*X+a
37	Y-10,i	Y-10,i	a10,m	a10,i	Y-10.i*a10,l	Y-10,l*a10,m	...+Y-10,l*a10,i
38	Y-10,m	Y-10,i	a10,l	a10,m	Y-10,i*a10,i	Y-10.i*a10,l	...+Y-10,l*a10,m
39	Y-10,m	Y-10,m	a10,i	a10,l	Y-10,i*a10,m	Y-10,i*a10,i	...+Y-10.i*a10,l
40	Y-10,m	Y-10,m	a10,m	a10,i	Y-10,m*a10,l	Y-10,i*a10,m	...+Y-10,i*a10,i
41		Y-10,m		a10,m	Y-10,m*a10,i	Y-10,m*a10,l	...+Y-10,i*a10,m
42					Y-10,m*a10,m	Y-10,m*a10,i	...+Y-10,m*a10,l

Table 2: Filter Pipeline Table

43						Y-10,m*a10,m	...+Y-10,m*a10,i
44							...+Y-10,m*a10,m=c00*X+a20*Y-
89	-		-				
90	Y-22,l	-	a22,i	-			
91	Y-22,l	Y-22,l	a22,m	a22,i	-		
92	Y-22,i	Y-22,l	a22,l	a22,m	Y-22,l*a22,i	-	
93	Y-22,i	Y-22,i	a22,i	a22,l	Y-22,l*a22,m	Y-22,l*a22,i	c02*X
94	Y-22,i	Y-22,i	a22,m	a22,i	Y-22,i*a22,l	Y-22,l*a22,m	...+Y-22,l*a22,i
95	Y-22,m	Y-22,i	a22,l	a22,m	Y-22,i*a22,i	Y-22,i*a22,l	...+Y-22,l*a22,m
96	Y-22,m	Y-22,m	a22,i	a22,l	Y-22,i*a22,m	Y-22,i*a22,i	...+Y-22,i*a22,l
97	X-22,m	Y-22,m	a22,m	a22,i	Y-22,m*a22,l	Y-22,i*a22,m	...+Y-22,i*a22,i
98	Y-12,l	X-22,m	a12,i	a22,m	Y-22,m*a22,i	Y-22,m*a22,l	...+Y-22,i*a22,m
99	Y-12,l	Y-12,l	a12,m	a12,i	Y-22,m*a22,m	Y-22,m*a22,i	...+Y-22,m*a22,l
100	Y-12,i	Y-12,l	a12,l	a12,m	Y-12,l*a12,i	Y-22,m*a22,m	...+Y-22,m*a22,i
101	Y-12,i	Y-12,i	a12,i	a12,l	Y-12,l*a12,m	Y-12,l*a12,i	...+Y-22,m*a22,m=c02*X+a
102	Y-12,i	Y-12,i	a12,m	a12,i	Y-12,i*a12,l	Y-12,l*a12,m	...+Y-12,l*a12,i
103	Y-12,m	Y-12,i	a12,l	a12,m	Y-12,i*a12,i	Y-12,i*a12,l	...+Y-12,l*a12,m
104	Y-12,m	Y-12,m	a12,i	a12,l	Y-12,i*a12,m	Y-12,i*a12,i	...+Y-12,i*a12,l
105	Y-12,m	Y-12,m	a12,m	a12,i	Y-12,m*a12,l	Y-12,i*a12,m	...+Y-12,i*a12,i
106	-	Y-12,m	-	a12,m	Y-12,m*a12,i	Y-12,m*a12,l	...+Y-12,i*a12,m
107	-	-	-	-	Y-12,m*a12,m	Y-12,m*a12,i	...+Y-12,m*a12,l
108	-	-	-	-	-	Y-12,m*a12,m	...+Y-12,m*a12,i
109	-	-	-	-	-	-	...+Y-12,m*a12,m=c028X+a22*Y-
110	-	-	-	-	-	-	-
111	-	-	-	-	-	-	-
112	-	-	-	-	-	-	-
113	-	-	-	-	-	-	-
114	-	-	-	-	-	-	-
115	-	-	-	-	-	-	-
116	-	-	-	-	-	-	-
117	-	-	-	-	-	-	-
118	-	-	-	-	-	-	-
119	-	-	-	-	-	-	-
120	-	-	-	-	-	-	-
121	-	-	-	-	-	-	-
122	-	-	-	-	-	-	-
123	-	-	-	-	-	-	-
124	-	-	-	-	-	-	-
125	-	-	-	-	-	-	-
126	-	-	-	-	-	-	-
127	-	-	-	-	-	-	-