A hardware implementation of the Levinson routine in the radio detector of cosmic rays to improve a suppression of non-stationary RFI

Zbigniew Szadkowski[1] IEEE Member, for the Pierre Auger Collaboration

[1] *University of Łódź, Poland*

ABSTRACT

The radio detector system for ultra high-energy cosmic rays in the Pierre Auger Observatory operates in the frequency range 30-80 MHz, which is often contaminated by human-made RFI. Several filters were used to suppress the RFI: based on the FFT, IIR notch filter and FIR filter based on the linear prediction. The last refreshes the FIR coefficients calculating either in the external ARM processor, internal soft-core NIOS® processor implemented inside the FPGA or hard-core embedded processors (HPS) being a silicon part of the FPGA chip. Refreshment times significantly depend on the type of calculation process. For stationary RFI, the FIR coefficients can be refreshed each minute or rarer. However, an efficient suppression of non-stationary short-term contaminations requires a much faster response. Calculations of FIR coefficients in an external ARM take 1-2 seconds, by the soft-core virtual NIOS® processor on the level of hundreds milliseconds. The HPS allows a reduction of refreshment time to $\sim$20 ms (for 32-stage FIR filter). A symmetry of covariance matrix allows one to use the much faster Levinson procedure instead of typical Gauss routine for solving a set of linear equations. The Levinson procedure calculated even in the HPS takes relatively a lot of time. A hardware implementation of this procedure inside the FPGA fabric as specialized a micro-controller requires only ~53 800 clock cycles. We used 64- or 48-bit floating-point representations to calculate FIR coefficients. Resources occupation is relatively high, as the design was optimized for a maximal register performance. However, the RFI suppression is very efficient. We expect significant suppression of even short-term non-stationary RFI.

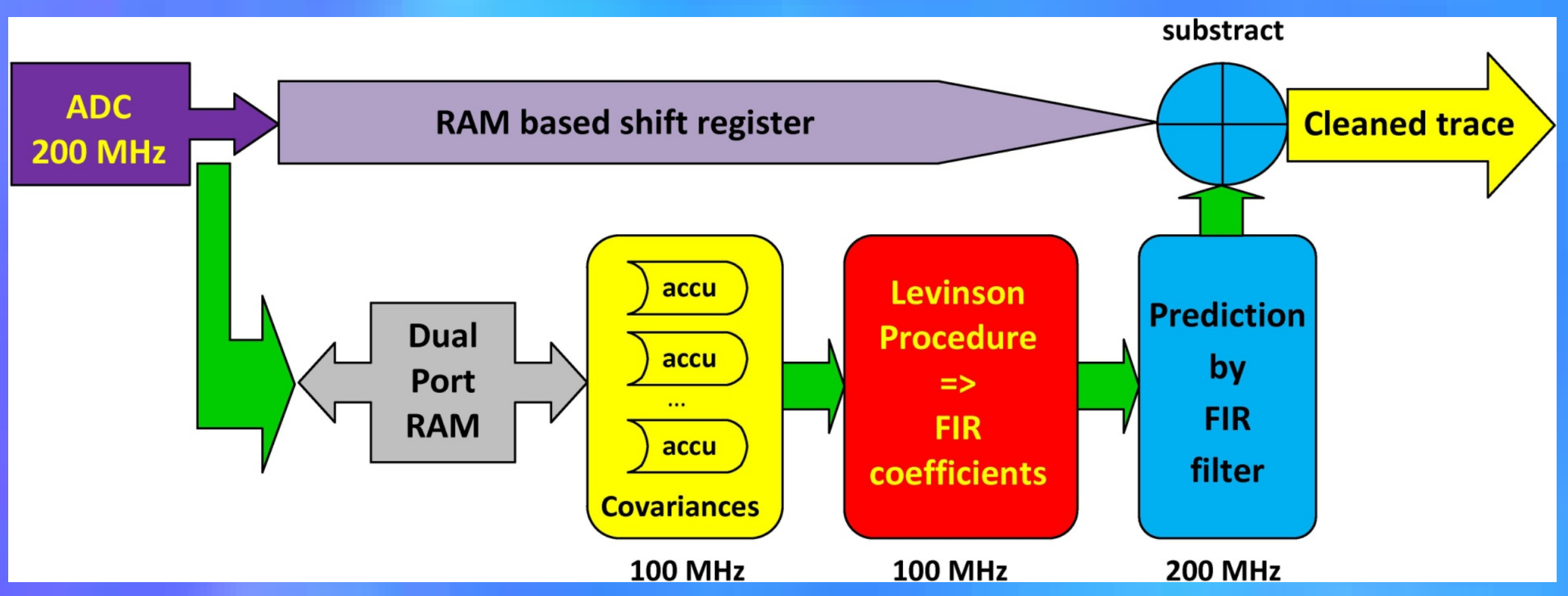


Fig. 1 – The structure of data flow for the hardware implementation of the Levinson algorithm.

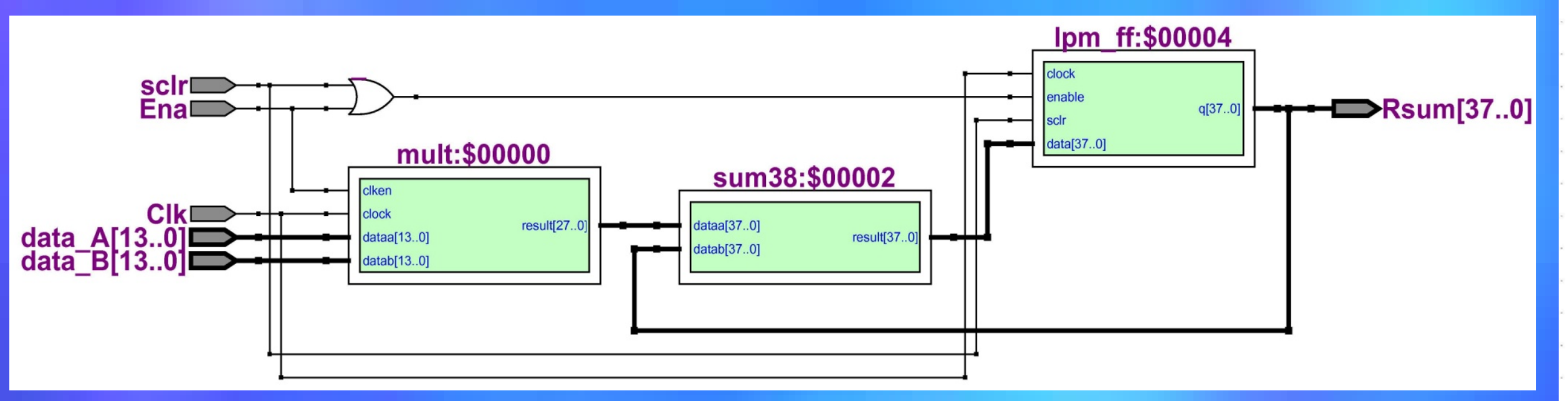


Fig. 2 – A structure of 32 fast logic blocks calculating covariances (accu in Fig. 1. All data are the fixed-point representation.

	Loop A	Loop B	Loop C	Loop D

n=1	xi = 0;
	xi -= r[1] * a[0];
	xi /= e;
		a[1] = 0;
		a[1] += a[0] * xi;
			z = y[1];
			z -= r[1] * x[0];
			pm = z/e;
				x[1] = 0;
				x[0] += a[1] * pm;
				x[1] += a[0] * pm;

n=2	xi = 0;
	xi -= r[2] * a[0];
	xi -= r[1] * a[1];
	xi /= e;
		a[2] = 0;
		a[2] += a[0] * xi;
		a[1] += a[1] * xi;
			z = y[2];
			z -= r[2] * x[0];
			z -= r[1] * x[1];
			pm = z/e;
				x[2] = 0;
				x[0] += a[2] * pm;
				x[1] += a[1] * pm;
				x[2] += a[0] * pm;

n=3	xi = 0;
	xi -= r[3] * a[0];
	xi -= r[2] * a[1];
	xi -= r[1] * a[2];
	xi /= e;
		a[3] = 0;
		a[3] += a[0] * xi;
		a[2] += a[1] * xi;
		a[1] += a[2] * xi;
			z = y[3];
			z -= r[3] * x[0];
			z -= r[2] * x[1];
			z -= r[1] * x[2];
			pm = z/e;
				x[3] = 0;
				x[0] += a[3] * pm;
				x[1] += a[2] * pm;
				x[2] += a[1] * pm;
				x[3] += a[0] * pm;

n=4	xi = 0;
	xi -= r[4] * a[0];
	xi -= r[3] * a[1];
	xi -= r[2] * a[2];
	xi -= r[1] * a[3];
	xi /= e;
		a[4] = 0;
		a[4] += a[0] * xi;
		a[3] += a[1] * xi;
		a[2] += a[2] * xi;
		a[1] += a[3] * xi;
			z = y[4];
			z -= r[4] * x[0];
			z -= r[3] * x[1];
			z -= r[2] * x[2];
			z -= r[1] * x[3];
			pm = z/e;
				x[4] = 0;
				x[0] += a[4] * pm;
				x[1] += a[3] * pm;
				x[2] += a[2] * pm;
				x[3] += a[1] * pm;

etc				

Fig. 4 – Four first steps in the Levinson algorithm implementation

```
void levinson(double *r, double *y,
              double *x, double *a,
              unsigned int dim)
   {
   unsigned int    n,i;
   double          e,z,xi,temp;
// initialization step //
   e = r[0];
   a[0] = 1;
   x[0] = y[0]/e;
// main loop //
   for (n = 1; n < dim; ++n)
       {
// calculate xi //
           xi = 0;
           for (i = 0; i < n; ++i)
               { // loop A //
               xi -= r[n-i] * a[i];
               }
           xi = xi/e;
// update a //
           a[n] = 0;
           for (i = 0; i <= (n-1)/2; ++i)
               { // loop B //
               temp = a[i];
               a[i]   = temp + a[n-i] * xi;
               a[n-i] = a[n-i] + temp * xi;
               }
           if (n div 2 == 0)
               {
               a[i] = a[i] + a[n-i] * xi;
               }
// calculate e //
           e = e * (1 - xi*xi);
// calculate l //
           z = y[n];
           for (i = 0; i < n; ++i)
               { // loop C //
               z -= r[n-i] * x[i];
               }
// update x //
           x[n] = 0;
           for (i = 0; i <= n; ++i)
               { // loop D //
               x[i] += a[n-i] * z/e;
               }
       }
   return;
   }
```

Fig. 3 – C code of the Levinson algorithm.

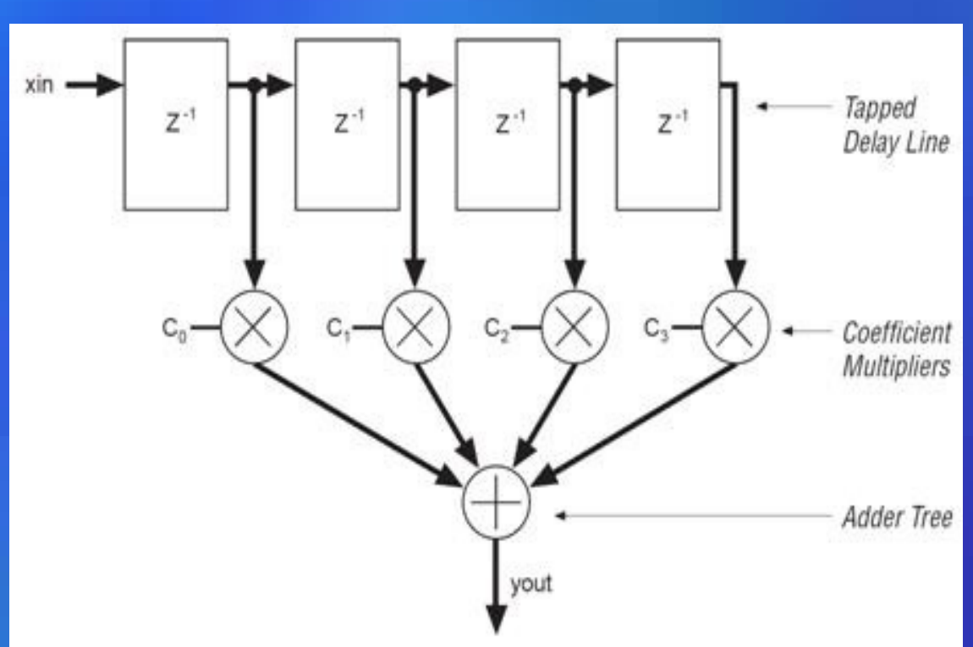
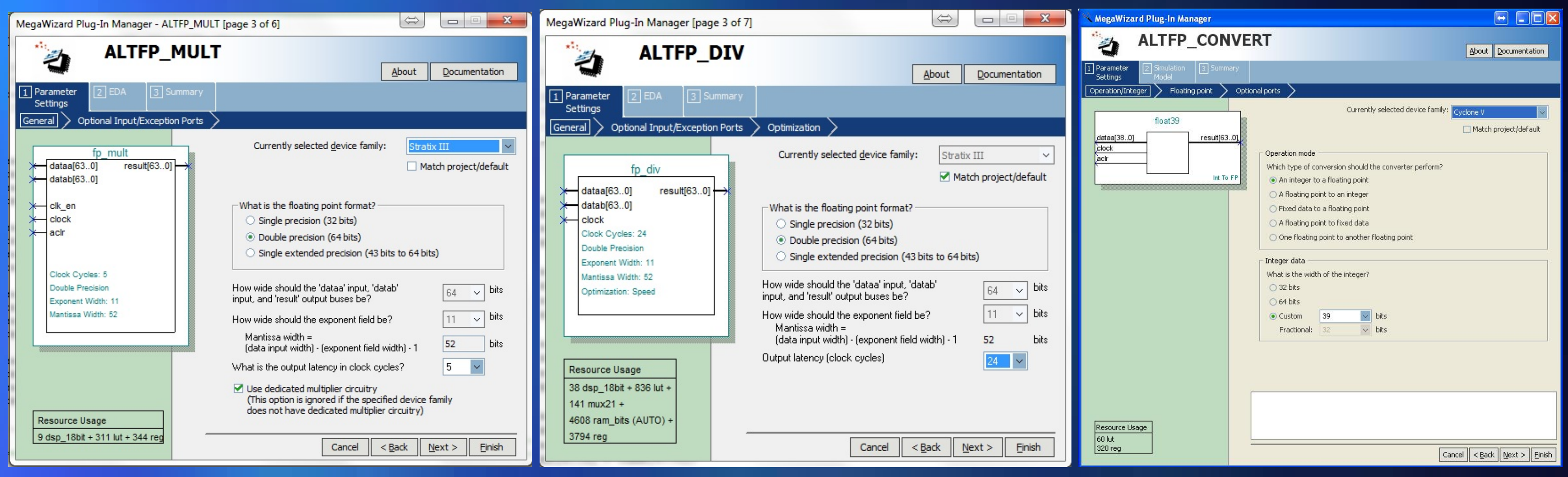


Fig. 5 – The structure of the FIR filter.



MegaWizard Plug-In Manager - ALTFP_MULT [page 3 of 6]

ALTFP_MULT

Clock Cycles: 5
Double Precision
Exponent Width: 11
Mantissa Width: 52

Currently selected device family: Stratix III

☐ Match project/default

What is the floating point format?
○ Single precision (32 bits)
● Double precision (64 bits)
○ Single extended precision (43 bits to 64 bits)

How wide should the 'dataa' input, 'datab' input, and 'result' output buses be? 64 bits

How wide should the exponent field be? 11 bits

Mantissa width = (data input width) - (exponent field width) - 1 52 bits

What is the output latency in clock cycles? 5

☑ Use dedicated multiplier circuitry (This option is ignored if the specified device family does not have dedicated multiplier circuitry)

Resource Usage
9 dsp_18bit + 311 lut + 344 reg

MegaWizard Plug-In Manager [page 3 of 7]

ALTFP_DIV

Clock Cycles: 24
Double Precision
Exponent Width: 11
Mantissa Width: 52
Optimization: Speed

Currently selected device family: Stratix III

☑ Match project/default

What is the floating point format?
○ Single precision (32 bits)
● Double precision (64 bits)
○ Single extended precision (43 bits to 64 bits)

How wide should the 'dataa' input, 'datab' input, and 'result' output buses be? 64 bits

How wide should the exponent field be? 11 bits

Mantissa width = (data input width) - (exponent field width) - 1 52 bits

Output latency (clock cycles) 24

Resource Usage
38 dsp_18bit + 836 lut +
141 mux21 +
4608 ram_bits (AUTO) +
3794 reg

MegaWizard Plug-In Manager

ALTFP_CONVERT

Currently selected device family: Cyclone V

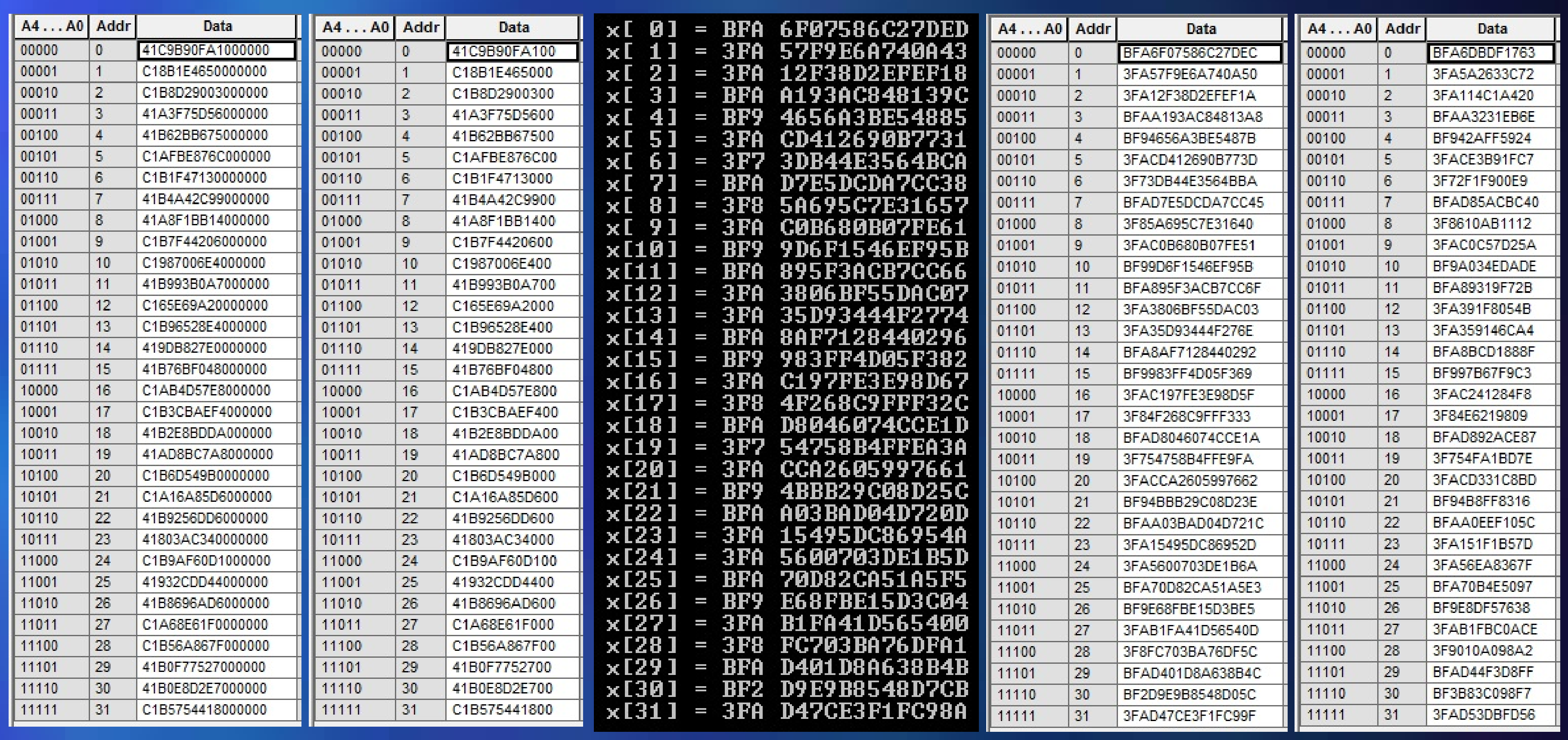
☐ Match project/default

Operation mode
Which type of conversion should the converter perform?
● An integer to a floating point
○ A floating point to an integer
○ Fixed data to a floating point
○ A floating point to fixed data
○ One floating point to another floating point

Integer data
What is the width of the integer?
○ 32 bits
○ 64 bits
● Custom 39 bits
Fractional: 32 bits

Resource Usage
60 lut
320 reg

Fig. 8 – Altera® mega-functions multiplying, dividing and converting floating-point 64-bit variables.



A4...A0	Addr	Data	A4...A0	Addr	Data
00000	0	41C9B90FA1000000	00000	0	41C9B90FA100
00001	1	C18B1E4650000000	00001	1	C18B1E465000
00010	2	C1B8D29003000000	00010	2	C1B8D2900300
00011	3	41A3F75D56000000	00011	3	41A3F75D5600
00100	4	41B62BB675000000	00100	4	41B62BB67500
00101	5	C1AFBE876C000000	00101	5	C1AFBE876C00
00110	6	C1B1F47130000000	00110	6	C1B1F4713000
00111	7	41B4A42C99000000	00111	7	41B4A42C9900
01000	8	41A8F1BB14000000	01000	8	41A8F1BB1400
01001	9	C1B7F44206000000	01001	9	C1B7F4420600
01010	10	C1987006E4000000	01010	10	C1987006E400
01011	11	41B993B0A7000000	01011	11	41B993B0A700
01100	12	C165E69A20000000	01100	12	C165E69A2000
01101	13	C1B96528E4000000	01101	13	C1B96528E400
01110	14	419DB827E0000000	01110	14	419DB827E000
01111	15	41B76BF048000000	01111	15	41B76BF04800
10000	16	C1AB4D57E8000000	10000	16	C1AB4D57E800
10001	17	C1B3CBAEF4000000	10001	17	C1B3CBAEF400
10010	18	41B2E8BDDA000000	10010	18	41B2E8BDDA00
10011	19	41AD8BC7A8000000	10011	19	41AD8BC7A800
10100	20	C1B6D549B0000000	10100	20	C1B6D549B000
10101	21	C1A16A85D6000000	10101	21	C1A16A85D600
10110	22	41B9256DD6000000	10110	22	41B9256DD600
10111	23	41803AC340000000	10111	23	41803AC34000
11000	24	C1B9AF60D1000000	11000	24	C1B9AF60D100
11001	25	41932CDD44000000	11001	25	41932CDD4400
11010	26	41B8696AD6000000	11010	26	41B8696AD600
11011	27	C1A68E61F0000000	11011	27	C1A68E61F000
11100	28	C1B56A867F000000	11100	28	C1B56A867F00
11101	29	41B0F77527000000	11101	29	41B0F7752700
11110	30	41B0E8D2E7000000	11110	30	41B0E8D2E700
11111	31	C1B5754418000000	11111	31	C1B575441800

```
x[ 0] = BFA 6F07586C27DED
x[ 1] = 3FA 57F9E6A740A43
x[ 2] = 3FA 12F38D2EFEF18
x[ 3] = BFA A193AC848139C
x[ 4] = BF9 4656A3BE54885
x[ 5] = 3FA CD412690B7731
x[ 6] = 3F7 3DB44E3564BCA
x[ 7] = BFA D7E5DCDA7CC38
x[ 8] = 3F8 5A695C7E31657
x[ 9] = 3FA C0B680B07FE61
x[10] = BF9 9D6F1546EF95B
x[11] = BFA 895F3ACB7CC66
x[12] = 3FA 3806BF55DAC07
x[13] = 3FA 35D93444F2774
x[14] = BFA 8AF7128440296
x[15] = BF9 983FF4D05F382
x[16] = 3FA C197FE3E98D67
x[17] = 3F8 4F268C9FFF32C
x[18] = BFA D8046074CCE1D
x[19] = 3F7 54758B4FFEA3A
x[20] = 3FA CCA2605997661
x[21] = BF9 4BBB29C08D25C
x[22] = BFA A03BAD04D720D
x[23] = 3FA 15495DC86954A
x[24] = 3FA 5600703DE1B5D
x[25] = BFA 70D82CA51A5F5
x[26] = BF9 E68FBE15D3C04
x[27] = 3FA B1FA41D565400
x[28] = 3F8 FC703BA76DFA1
x[29] = BFA D401D8A638B4B
x[30] = BF2 D9E9B8548D7CB
x[31] = 3FA D47CE3F1FC98A
```

A4...A0	Addr	Data	A4...A0	Addr	Data
00000	0	BFA6F07586C27DEC	00000	0	BFA6DBDF1763
00001	1	3FA57F9E6A740A50	00001	1	3FA5A2633C72
00010	2	3FA12F38D2EFEF1A	00010	2	3FA114C1A420
00011	3	BFAA193AC84813A8	00011	3	BFAA3231EB6E
00100	4	BF94656A3BE5487B	00100	4	BF942AFF5924
00101	5	3FACD412690B773D	00101	5	3FACE3B91FC7
00110	6	3F73DB44E3564BBA	00110	6	3F72F1F900E9
00111	7	BFAD7E5DCDA7CC45	00111	7	BFAD85ACBC40
01000	8	3F85A695C7E31640	01000	8	3F8610AB1112
01001	9	3FAC0B680B07FE51	01001	9	3FAC0C57D25A
01010	10	BF99D6F1546EF95B	01010	10	BF9A034EDADE
01011	11	BFA895F3ACB7CC6F	01011	11	BFA89319F72B
01100	12	3FA3806BF55DAC03	01100	12	3FA391F8054B
01101	13	3FA35D93444F276E	01101	13	3FA359146CA4
01110	14	BFA8AF7128440292	01110	14	BFA8BCD1888F
01111	15	BF9983FF4D05F369	01111	15	BF997B67F9C3
10000	16	3FAC197FE3E98D5F	10000	16	3FAC241284F8
10001	17	3F84F268C9FFF333	10001	17	3F84E6219809
10010	18	BFAD8046074CCE1A	10010	18	BFAD892ACE87
10011	19	3F754758B4FFE9FA	10011	19	3F754FA1BD7E
10100	20	3FACCA2605997662	10100	20	3FACD331C8BD
10101	21	BF94BBB29C08D23E	10101	21	BF94B8FF8316
10110	22	BFAA03BAD04D721C	10110	22	BFAA0EEF105C
10111	23	3FA15495DC86952D	10111	23	3FA151F1B57D
11000	24	3FA5600703DE1B6A	11000	24	3FA56EA8367F
11001	25	BFA70D82CA51A5E3	11001	25	BFA70B4E5097
11010	26	BF9E68FBE15D3BE5	11010	26	BF9E8DF57638
11011	27	3FAB1FA41D56540D	11011	27	3FAB1FBC0ACE
11100	28	3F8FC703BA76DF5C	11100	28	3F9010A098A2
11101	29	BFAD401D8A638B4C	11101	29	BFAD44F3D8FF
11110	30	BF2D9E9B8548D05C	11110	30	BF3B83C098F7
11111	31	3FAD47CE3F1FC99F	11111	31	3FAD53DBFD56

Fig. 9 – Comparison of 64- and 48-bit input covariances (two left columns), FIR coefficients calculated in the PC (middle column) and in the FPGA 64-bit (4th column) and 48-bit (right column). Some discrepancy on least significant bits for 64-bit calculations are visible. However, the relative accuracy is on a level of 10^{-13}. The accuracy for 48-bit calculations dramatically drops down to a level of 10^{-4}.

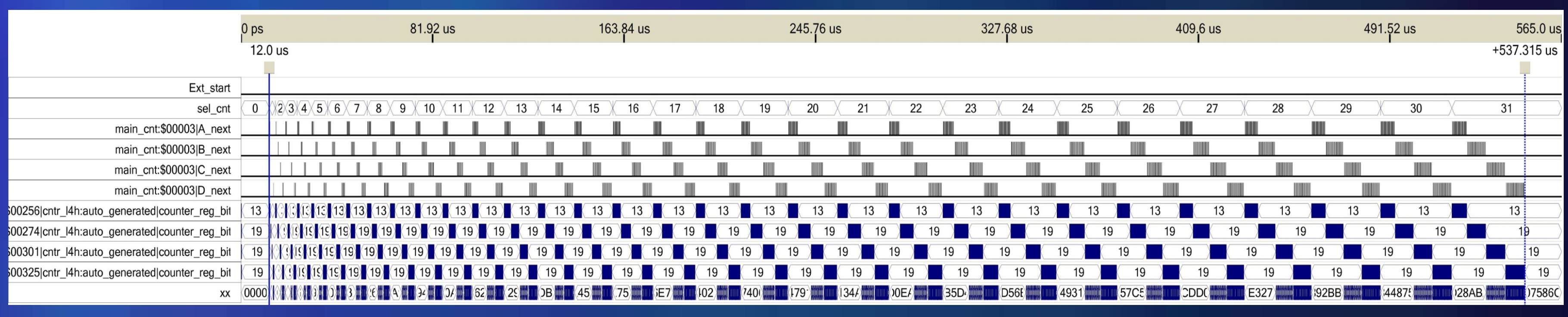


Fig. 10 – Quartus® simulation of FIR coefficients calculation by the hardware Levinson procedure. Calculation of **r** and **y** covariances takes 12 µs, calculation of 64-bit floating-point coefficients takes next ~538 µs. Simulations show sequences of processing in the consecutive A, B, C and D loops.

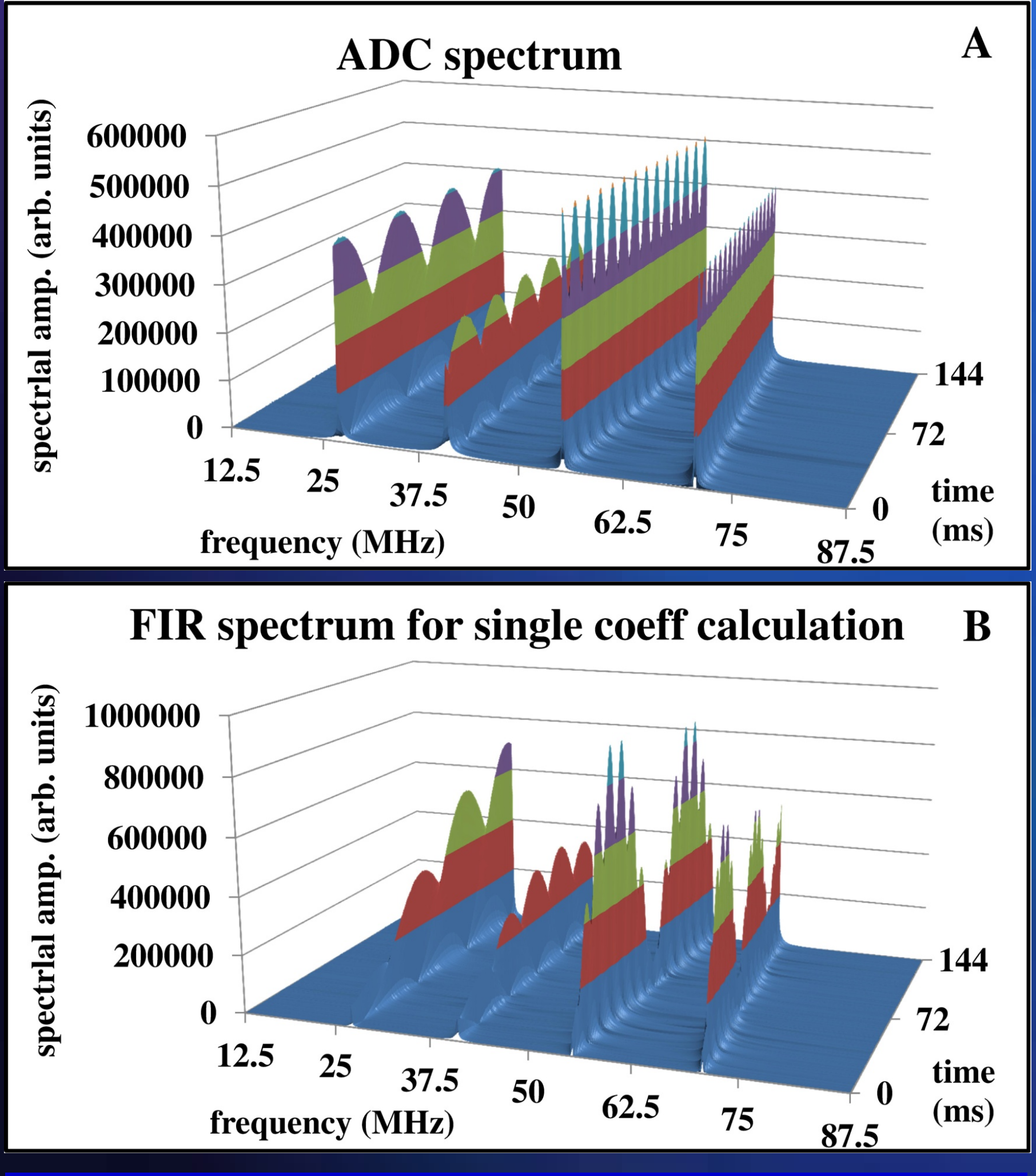


Fig. 6 – Fourier spectra for the original signal (A) and for a single calculation of FIR coefficients (B) in the beginning of trace cleaning.

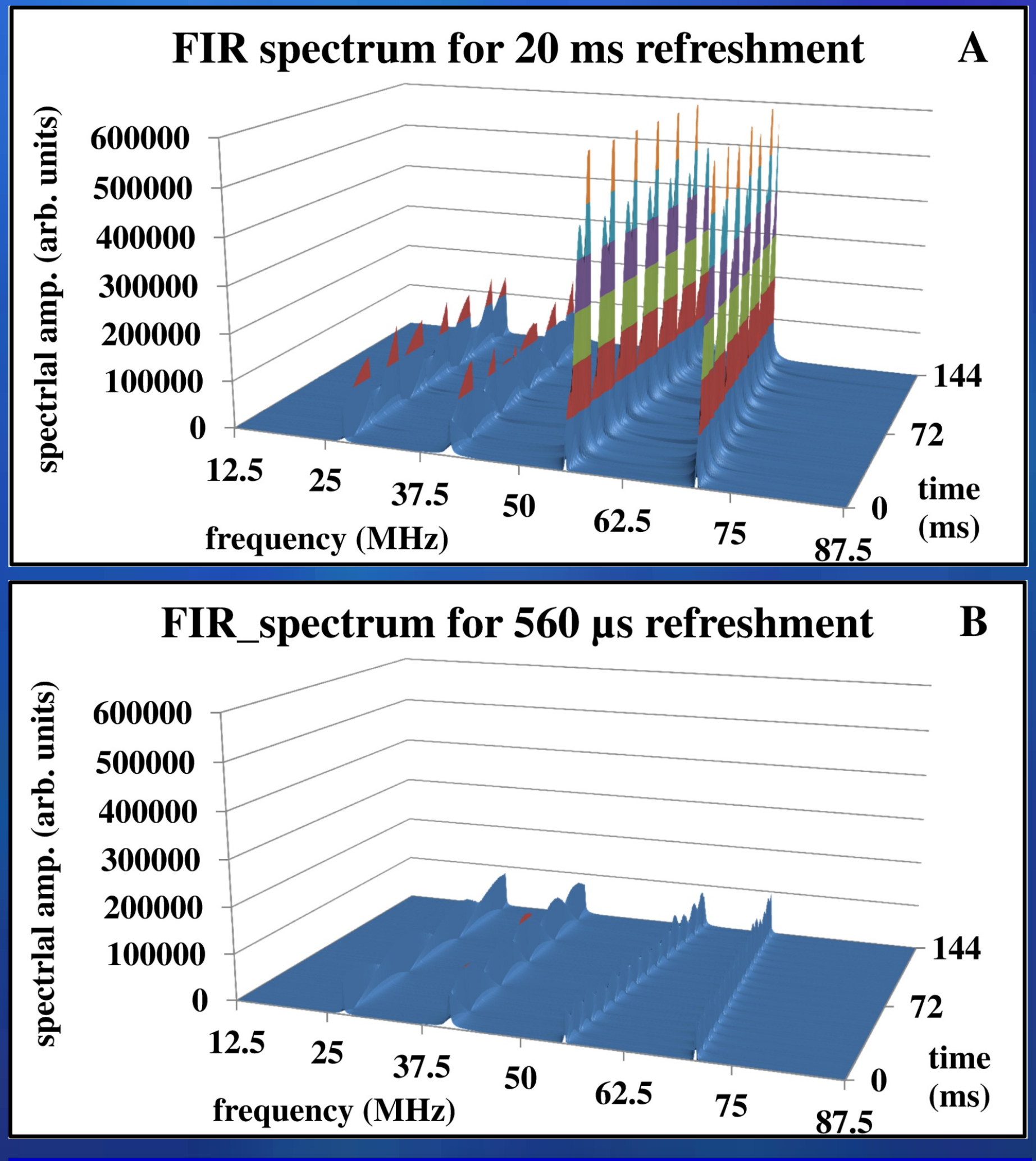


Fig. 7 – Modules of Fourier spectra for a refreshment of FIR coefficients each 20 ms (A) and for each 560 µs (B).

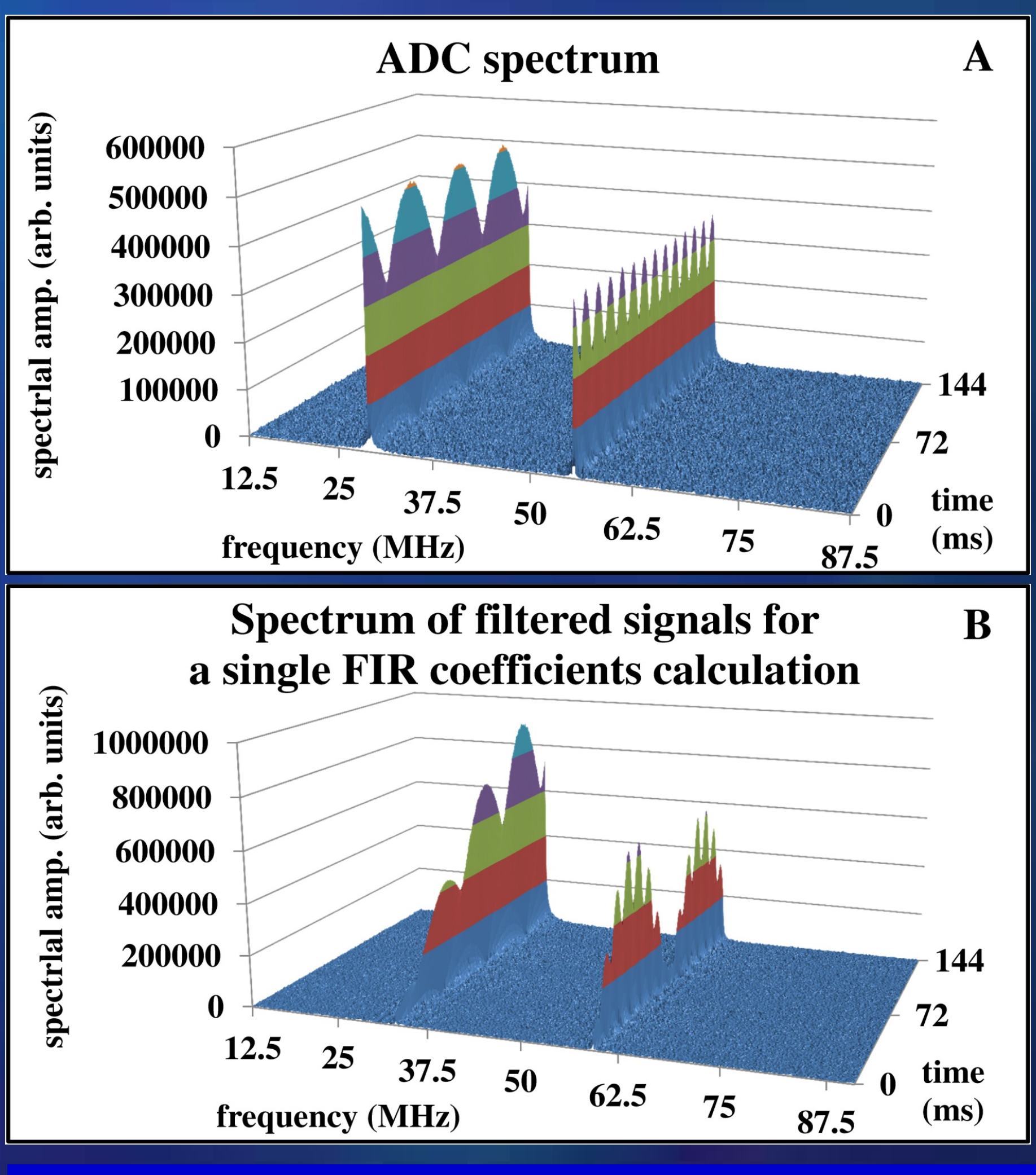


Fig. 11 – Fourier spectra for the original signal (A) and for a single calculation of FIR coefficients in the beginning of trace cleaning (B).

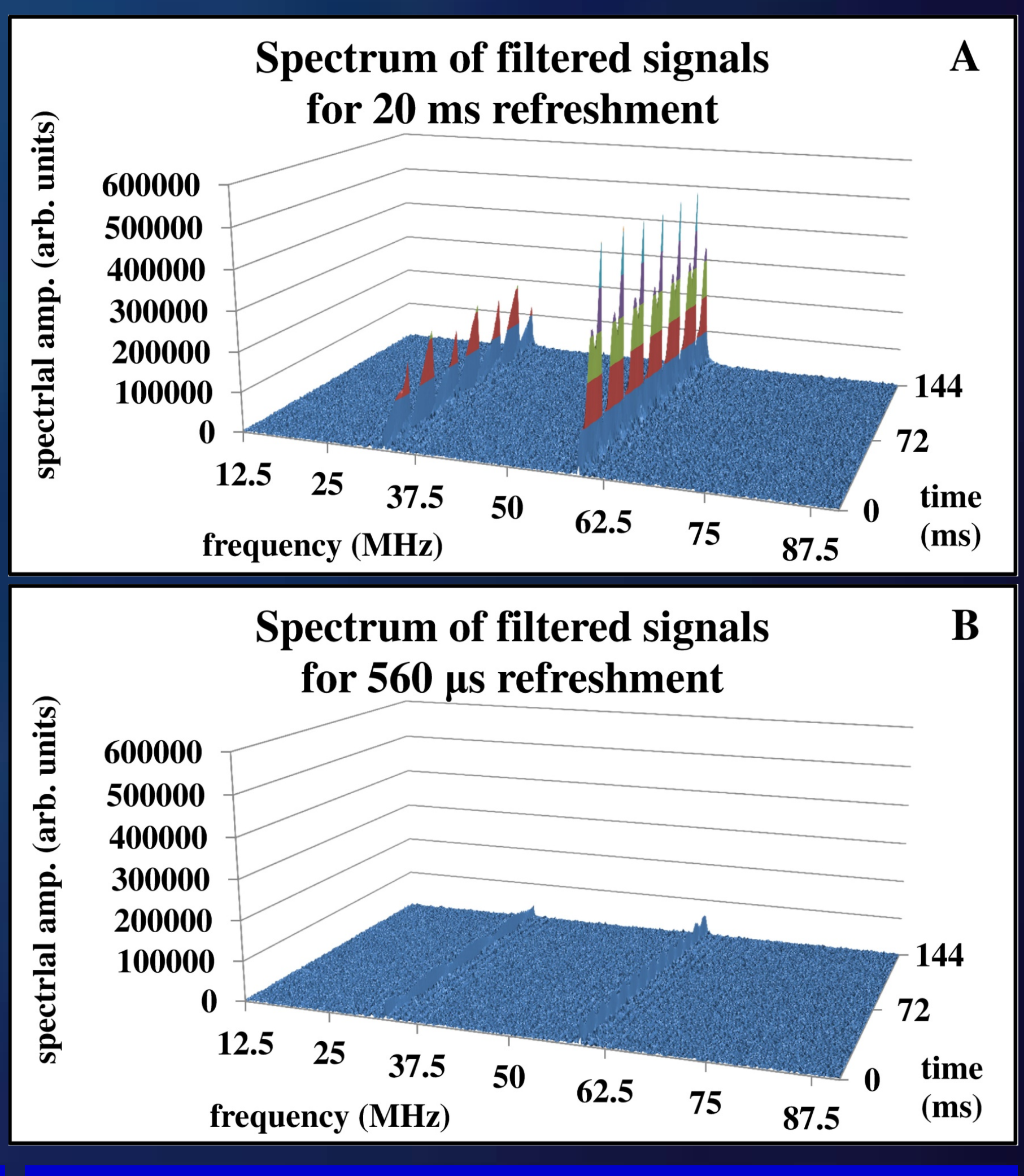


Fig. 12 – Modules of Fourier spectra for the cleaned measured signal 20 ms (A) and 560 µs refreshment (B), respectively.

[1] K. Greisen, *End to the cosmic-ray spectrum?, Phys. Rev. Lett.* **16** (1966) 748, G. T. Zatsepin, V. A. Kuzmin, *Upper limit of the spectrum of cosmic rays", JETP Letters* **4** (1966) 78.

[2] J. K. Becker, *High-energy neutrinos in the context of multi-messenger astrophysics, Phys. Reports* **458** no. 4-5 (2008) 173.

[3] P. Bhattacharjee and G. Sigl, *Origin and propagation of extremely high-energy cosmic rays, Phys. Reports* **327** no. 3-4 (2000) 109.

[4] V. S. Berezinsky, A.Yu. Smirnov, *Cosmic neutrinos of ultrahigh energies and detection possibility, Astrophys. and Space Sci.* **32** no. 2 (1975) 461.

[5] Z. Szadkowski, *A spectral 1st level FPGA trigger for detection of very inclined showers based on a 16-point Discrete Cosine Transform for the Pierre Auger Observatory, Nucl. Instrum. Meth.* **A 606** (2009) 330.

[6] Z. Szadkowski, *Optimization of the Detection of Very Inclined Showers Using a Spectral DCT Trigger in Arrays of Surface Detectors, IEEE Trans. on Nucl. Science* **60** (2013) 3647.

[7] Z. Szadkowski, K. Pytel, *Artificial Neural Network as a FPGA Trigger for a Detection of Very Inclined "Young" Showers, IEEE Trans. on Nucl. Science* **62** Issue 3 (2015) 1002.

CONCLUSIONS

The hardware Levinson procedure implemented into FPGA fabric significantly shortens the refreshment time in the linear predictor approach used for RFI suppression. With Cyclone® IV (FPGA used at present in the Dutch AERA digitizers) and in Cyclone® V E (considered as FPGA for the future upgrade) the refreshment time is on a level of 560 µs (with 100 MHz clock for covariances calculation and the Levinson recursion). The 200 MHz speed has been achieved in the Stratix ® III FPGAs (speed grade - 2) and allowed a reduction of refreshment time to 275 µs, however, the Stratix® series are too power consuming and too expensive. We plan to use the above algorithm for tests of RFI suppression: a) in Łódź environment using the standard AERA antenna and b) in real radio stations on Argentinean pampas.

This work was supported by
the National Science Centre (Poland)
under NCN Grant No.
2013/08/M/ST9/00322