PORT OF PASCAL FPGA-LOGARITHMIC-UNIT SIMULATOR TO SIMULINK/RTW

Jiri Kadlec (1), Rudolf Matoušek (1), Christian Vialatte (1), Nicholas Coleman(2)

(1) UTIA AV CR, Prague, CZ, (2) University of Newcastle, Newcastle, UK

Introduction

Implementation of floating point algorithms in FPGA (Field Programmable Gate Arrays) creates an open problem. One of possible solutions is the representation of floating point numbers as an integer (fixed point) logarithm (32 bit) [1]. Basic arithmetical operations can be performed in the logarithm numbering system which is suitable for FPGA and ASIC implementation. Simulator of such system has been designed originally in Pascal. The Pascal simulator has been utilizing special 64-bit integer format of the Turbo-Pascal compiler.

This paper describes port of the Pascal simulator [1],[3-7] to C code Simulink S-function, compatible with Matlab 5.3 and RTW. Special attention is given to the solution of the C code, working identically for the 64-bit integer variables under (a) SIMULINK S-function; (b) under W95/NT as EXE code created by the RTW from Simulink (compiled by MSCV 6) and (c) on 64-bit Alpha AXP platform (Alpha Data Parallel Systems, Ltd.) as the application created by the Alpha AXP target for the RTW (compiled by 3L Parallel C).

32bit logarithmic arithmetic by correction of interpolation error

In the presented Logarithmic Numbering System (LNS), is real number *x* is represented as the 32bit fixed point value *i=log2(x).* LNS multiplication, division and square roots can be implemented fast as the integer plus, minus and shift operations. Effective solution to the LNS addition (subtraction) has been recently proposed by Dr. J.N. Coleman. LNS addition is based in the formulae *log2(x+y)= i + log2(1+2^(i-j))* where *i=log2(x); j=log2(y) and j<=i.*

The term *log2(1+2^(i-j))* has to be approximated by a look-up tables. Dr. Coleman has proposed innovative, patented approach, leading to drastic reduction of the size of look-up tables by parallel evaluation of partial tables and joint correction term. This approach leads to a solution, which is suitable to VLSI and FPGA implementation, works with integer operations and logic cells, avoids the need of barrel shift.

The 32 bit NLS addition can be performed in time comparable to floating-point 32bit addition without loss of the precision. It needs approximately 32k byte of look-up tables with the word-length 35 bytes. The corresponding VLSI adder can be clocked by 500MHz clock and the latency is just 4 clock cycles.

Therefore, the LNS add ALU is a valid candidate for the development of the IP (Intellectual Property) Cores for the FPGA based designs, which need to operate with 32-bit precision range of floating point numbers. The predicted maximal clock frequency is 100 to 200 MHz for the XILINX Virtex devices. The latency 4 clock cycles again. This research is performed under the EU ESPRIT 33544 HSLA Long-term research project, coordinated by the University of Newcastle, UK [1].

Pascal Simulator

The LNS unit was simulated under Turbo Pascal. This widely used compiler is recognizing the COMP data type. It is 80bit wide integer, used for exact precision integer operations. The simulator was using this data type to model the operation of the logarithmic unit on the 35 bite wide words. The simulator declares functions like

```
FUNCTION quotient (dividend, divisor: COMP): COMP;
  BEGIN
    quotient := INT (dividend / divisor);
  END;
```

This function declares quotient operation of 2 operands, returning 32 bit INT result. Call to such function is used to simulate the shift of the first operand to the right for *k* bits by setting the second operand to *2^(k)*.

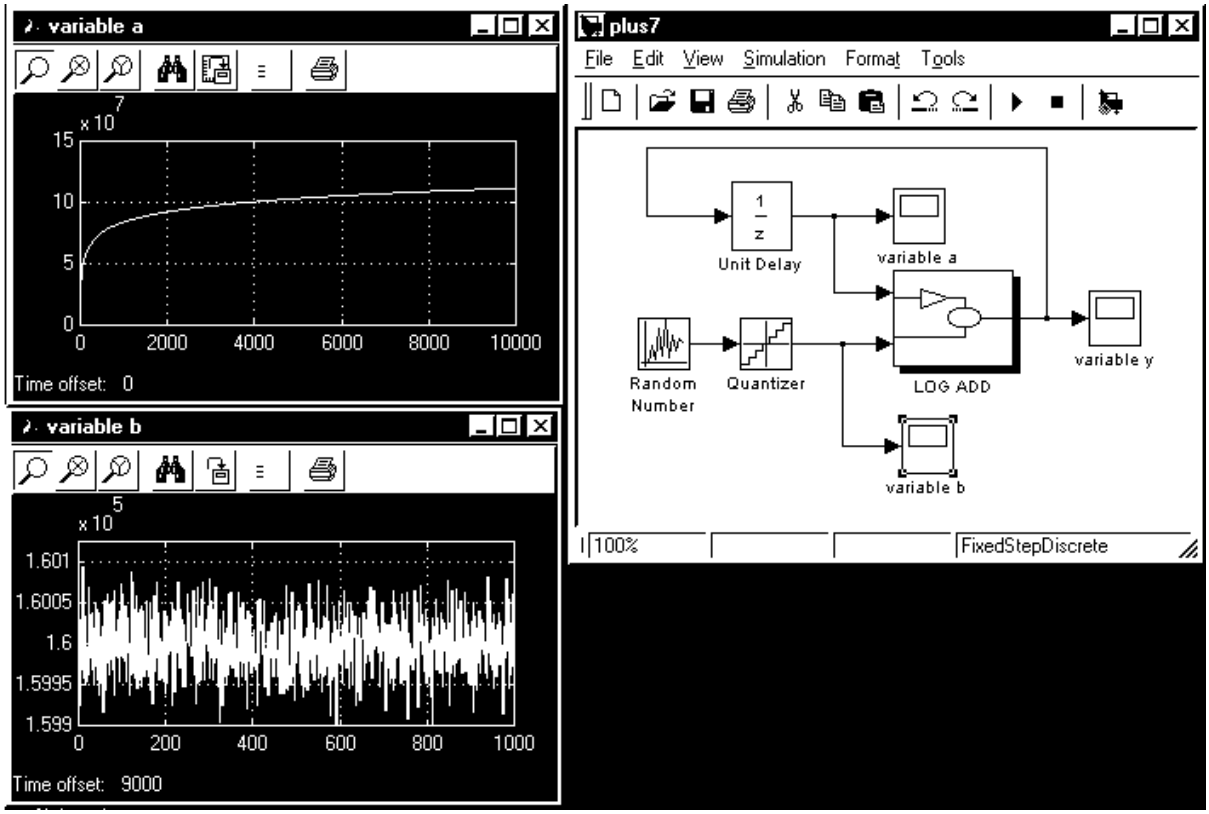

Fig 1. Logarithmic Number System Add under Simulink. LNS ADD block in C with 64-bit int tables.

The presence of integers and look-up tables wider than 32 bit presents special problem for the port to C, and Simulink. Next section of the paper describes our experiences with Matlab 5.3 support for the wider Integer, namely the 64-bit.

C implementation

Our objective was to design C code which would be working identically for the 64-bit integer variables under:

- SIMULINK 2 as the DLL S-function and Matlab 5.3 MEX-function
- W95/NT as the EXE code created by the RTW (Real Time Workshop) from Simulink for the generic target compiled by (Microsoft Visual C, version 6)
- 3L Parallel C on 64-bit Alpha AXP platform as APP application created by RTW, Alpha AXP target [2],[8].

The final solution was simple and proved to work **without modification of the source code** for all above given targets. We have declared the 35-bit wide tables as: **static __int64** data types like:

```
static __int64 table[256];
```

This data type is supported by MSVC version C as well as the Alpha AXP compiler.

The initial section of Pascal simulator code creates the 35-bite wide tables and operates in the floating point format with the extended precision (80bit). We have considered port of this proprietary code to C double as dangerous. The port can introduce unpredictable rounding errors. Instead of port, we left the table generation in Pascal, and added export of 35-bite wide data-tables into temporary data files.
We have designed dedicated Matlab M-file which reads these data files (as sequence of 8-bit integers) and creats the source-code of .h headers. These headers declare tables as static __int64 data types for the Simulink simulator.

Any change in the table-definition (made in Pascal) is reflected after the automated creation of C headers and Re-compilation of the S-functions for the Simulink and the RTW code. **Fig. 1** presents the screen with the HSLA simulator.

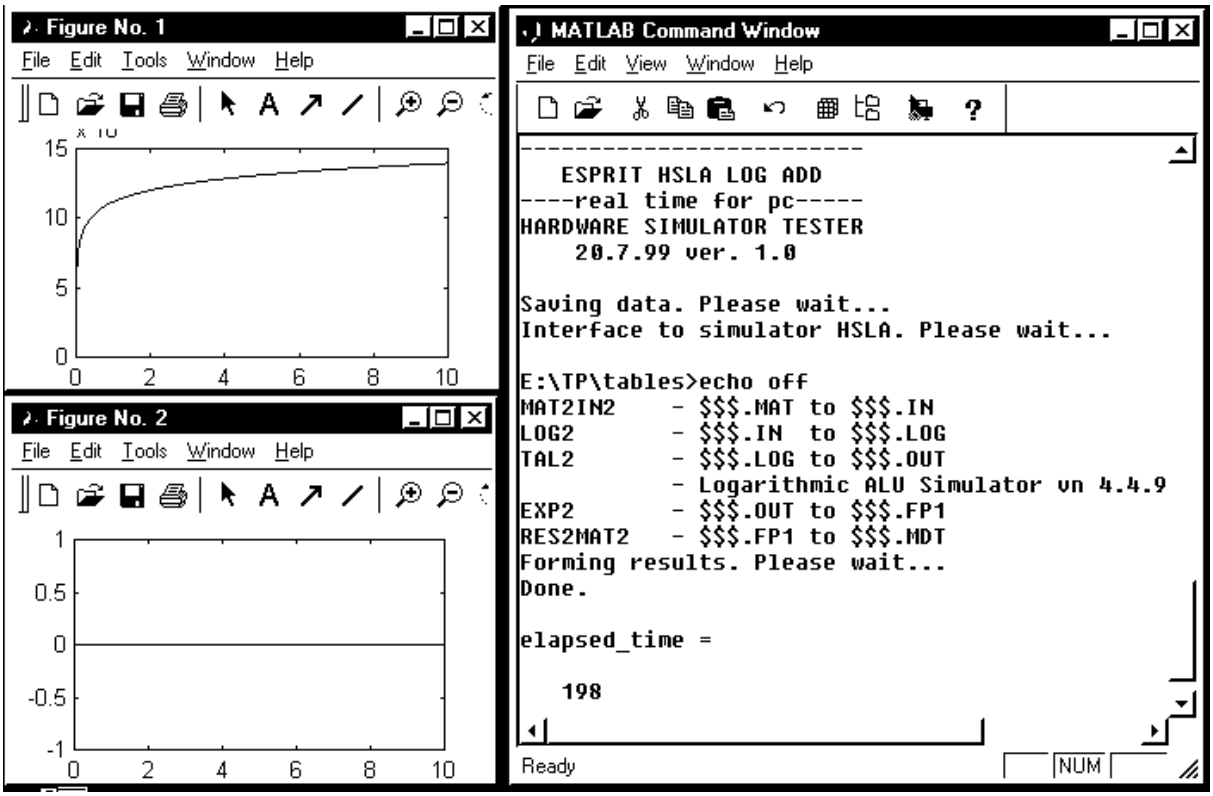

Fig 2. Pascal simulator LNS Add executed from Matlab. Input data are taken from simulink generator. Figures present the output of accumulated ADDs and the zero line proves that the Pascal and Simulink S-function generate bit-exact, identical results. The elapsed time is measured for 100000 ADDs on Pentium 160MHz.

The measured execution time is given in **Tab 1**.

P 160MHz Pascal simulator + all conversions, 100K LOG additions	198 s	1.98 ms/add
P 160MHz Pascal simulator only, 100K LOG add.	22 s	**0.22 ms/add**
P 160 MHz Simulink total execution time 100K LOG add. + noise generation	55 s	0.55 ms/add
PC 160MHz RTW compiled code, total exec. Time, 10M LOG add. + noise	326 s	0.032 ms/add
PC 160MHz RTW compiled code 10M LOG add only	50 s	**0.005 ms/add**
Alpha 233MHz AXP RTW compiled code, 10M LOG add only	40 s	**0.0025 ms/add**

Tab. 1. Execution times

First line includes time needed to convert Matlab matrices to the Pascal-compatible binary code. The transformations are implemented by saving of data as ascii mat-file from matlab, executing of set of calls to Pascal utility programs, including the LNS LOG adder and final return conversion of results back to ascii mat-file back to Matlab. See **Fig. 2.**

Second line presents only the time consumed by the Pascal LNS LOG simulator.

The next 2 lines document the gain of RTW compilation. RTW compiled code is **17 times faster** in comparison to the identical Simulink diagram (Fig. 1), with the Simulink execution time measured with closed scopes.

The final 2 lines present the execution time of a single LNS ADD block after the RTW compilation. The execution time was measured by adding second LOG ADD block to the diagram from Fig 1 and measuring the increase of the execution time after compilation for PC and Alpha AXP.

Notice, that the original Pascal simulator execution time for a single LOG Add operation was **reduced 44 times** In the case of PC and **88 times** for the Alpha board. This significant reduction is mainly due-to the C style of coding, which is close to the processor register shifts. Second significant source of saving is the reduction of the function-calls inside of the simulator.

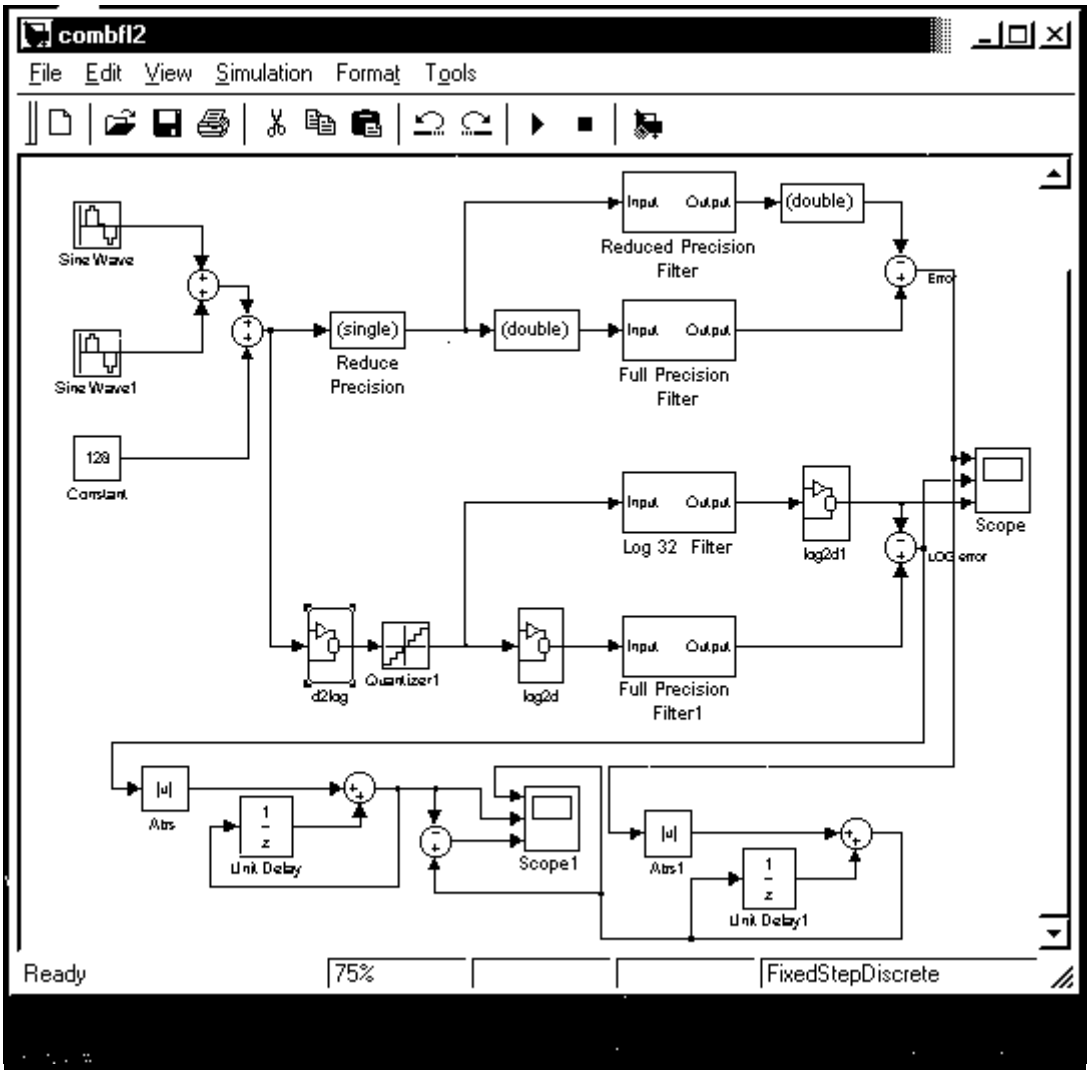

Fig. 3 Use of the LOG ADD block in a simple FIR filter.

The Input and output data of the LOG ADD block are represented in Simulink double, but hold 32 bit integer values, representing the logarithm representation used in the real FPGA circuit. That is why we have to include the Quantizer with 32 bit resolution after the Random number generator.

Fig. 3 presents the use of the LOG ADD block in a simple FIR filter. The diagram compares the FIR filter executed in Matlab 32 bit float format in comparison with LNS 32 bit arithmetic LOG adder. We this 2 implementations with the double-precision alternatives, working on data which are rounded to the 32bit float or 32 bit LOG grid.

Fig. 4 presents the results. The absolute value of the arithmetic error is accumulated and plotted in time.
The left column plots describe

- (upper) arithmetic error 32 bit float
- (middle) arithmetic error 32 bit LOG LNS
- (bottom) output of the filter

The right column plots describe

- (upper) accumulated absolute value of the arithmetic error for the 32 bit float
- (middle) accumulated absolute value of the arithmetic error 32 bit LOG LNS
- (bottom) difference of both, 32 bit LNS outperforms 32 bit floating point

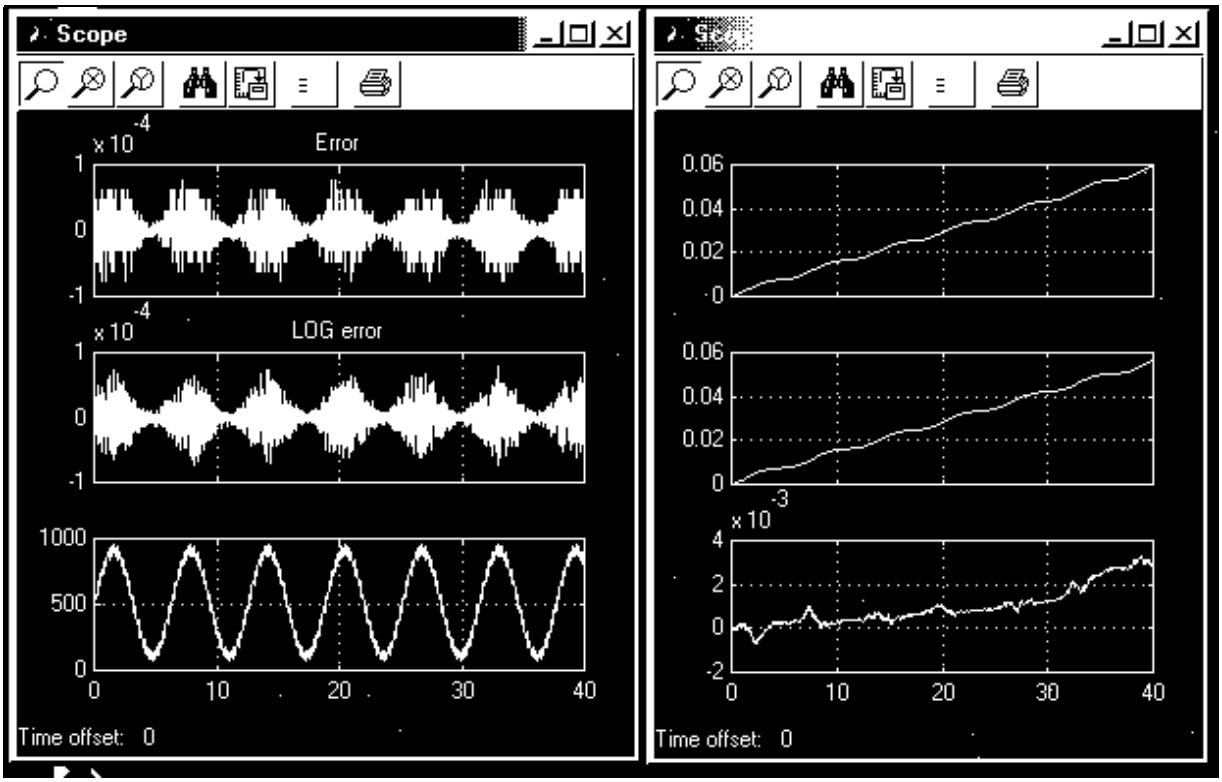

Fig. 4 The absolute value of the arithmetic error is accumulated and plotted in time. Left: (upper) arithmetic error 32 bit float; (middle) arithmetic error 32 bit LOG LNS; (bottom) output of the filter. Right: (upper) accumulated absolute value of the arithmetic error for the 32 bit float (middle); accumulated absolute value of the arithmetic error 32 bit LOG LNS; (bottom) difference of both, 32 bit LNS outperforms 32 bit floating point.

Fig. 5 is the snapshot of the 3-rd order FIR filters used in the simulations. The LNS ADD block replaces the classical add-block.

Conclusion

We have managed to accelerate original Pascal code 44 times on identical machine. On top of it, the Simulink gives us the flexibility and portability we need. Tested design will have to be extended for all elementary operations (minus, multiply, divide and square root) for Simulink and Matlab. This will be the basic foundation for creation of an LNS arithmetic toolbox.

Our next objective is to identify DSP algorithms gaining from LNS in terms of speed and precision. But the ultimate objective is the integration of the LNS unit as FPGA module, compatible with the RTW for the Alpha AXP boards.

Contact

UTIA AV CR, (2) Prague 8, Pod vodarenskou vezi 4, 182 08,
e-mail: kadlec@utia.cas.cz, matousek@utia.cas.cz, cv@utia.cas.cz

University Newcastle, Newcastle upon Tyne, NE17RU, UK, e-mail: j.n.coleman@ncl.ac.uk.

Grants

This work was partially supported by the EU Esprit LTR project No. 33544 “HSLA” and grants of the Ministry of Education of the Czech Republic OK314-99 and OK317-99.

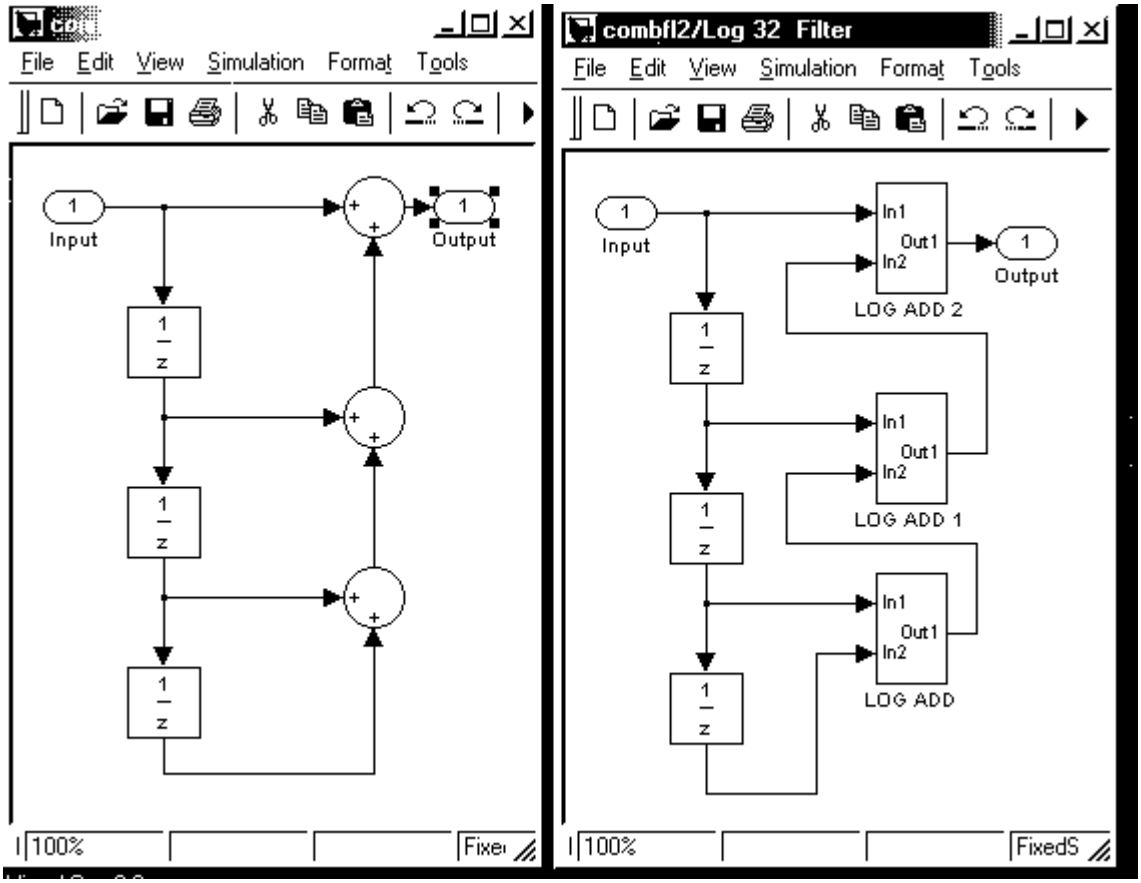

Fig. 5 Snapshot of the 3-rd order FIR filters used in the simulations.

References

[1] J.N. Coleman, “A high Speed Logarithmic Arithmetic Unit. ESPRIT Long-term research project No. 33544 “HSLA”.

[2] Kadlec J.: Acceleration of computation-intensive algorithms on parallel Alpha AXP processors. In: Preprints of the 3rd European IEEE Workshop on Computer-Intensive Methods in Control and Data Processing. (Rojicek J., Valeckova M., Karny M., Warwick K. eds.). UTIA AV CR, Praha 1998, pp. 89-98.

[3] Kadlec J., Schier J.: HSLA 3D Monitor Package. (Research Report No. 1925). UTIA AV CR, Praha 1998, 51 pp.

[4] Kadlec J., Schier J.: HSLA DSP Package. (Research Report No. 1924). UTIA AV CR, Praha 1998, 12 pp .
[5] Kadlec J., Schier J.: Numerical Analysis of a Normalized QR Filter Using Probability Description of Propagated Data. (Research Report No. 1923). UTIA AV CR, Praha 1998, 23 pp.
[6] Kadlec J., Schier J.: Rapid prototyping of adaptive control algorithms on parallel multiprocessors. In: Signal Processing Symposium. IEEE, Leuven 1998, pp. 115-118.
[7] Kadlec J., Schier J.: Results of the Global Probability Analysis Approach. (Research Report No. 1926). UTIA AV CR, Praha 1998, 89 pp.
[8] Kadlec J., Vialatte C.: Rapid prototyping and parallel processing under MATLAB 5. In: MATLAB Conference 1997. Kimhua Technology, Seoul 1997, pp. 120-125.