UTILIZATION OF THE HSLA TOOLBOX FOR THE FPGA PROTOTYPING

Zdeněk Pohl
UTIA

Miroslav Ličko
CAK

Abstract

An innovative design-flow for DSP algorithms using high-speed logarithmic arithmetic (HSLA) toolbox is introduced on a demo RLS lattice application. Utilization of toolbox provides effective design time shortening and it makes designer work easier. Firstly, scripts in Matlab are written and checked. From working scripts, design is decomposed in Simulink to cycle-exact simulation and rewritten in Celoxica DK1 tool. Finally, hardware is targeted from DK1 and results are compared with simulations. Our design flow was successfully tested on a noise canceller demo and it was proofed, that it leads to shortening of design-flow time.

INTRODUCTION

Implementation of large designs containing millions of gates is going to be more important with growing size of FPGA chips. For this reason, more abstract front-end tools like Celoxica DK1 have to be employed. This tool enables user to write "C-like" code that can be translated to VHDL. Developer, who is using DK1 tool, can write and debug complicated algorithms such as DSP applications. Floating-point arithmetic operations are crucial for DSP algorithms. One solution is to develop fixed-point versions of algorithms. The other solution is to use a floating-point library. The *high-speed logarithmic arithmetic unit* (HSLA) and equivalent Matlab library was developed as an alternative solution to floating and fixed point solutions. HSLA is a set of basic arithmetic operations in *logarithmic number system* (LNS) format. It is available as modules in DK1 design tool and as bit-exact modules in Matlab.

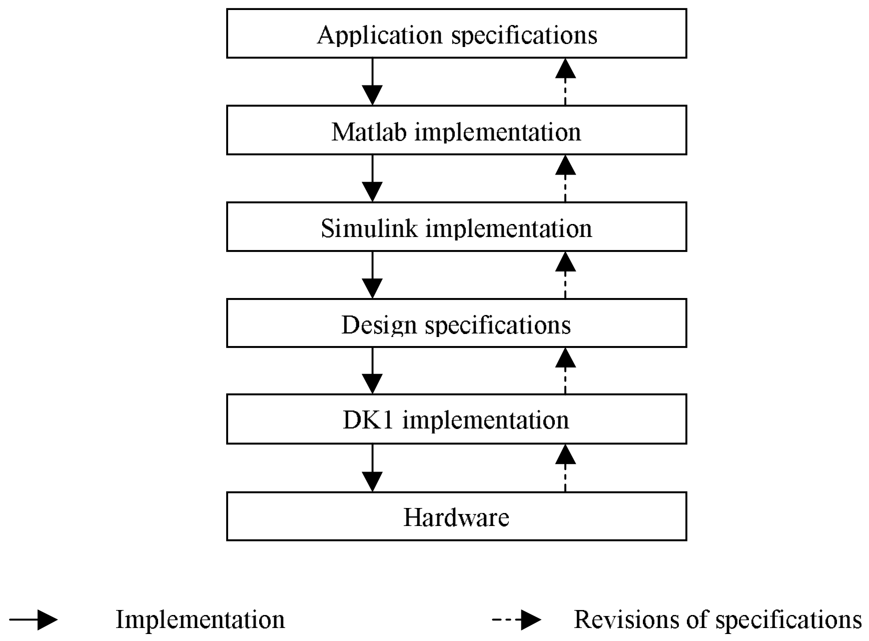
This paper is concerned in design-flow from first specification to final hardware. Firstly, suggested design flow is explained. In the second part, capabilities of HSLA library are explained and its utilization in Matlab is illustrated. In the next, toolbox utilization and debugging steps are described on the example of implementation of noise canceller based on RLS lattice algorithm. Finally, some results from implemented example are introduced.

DESIGN FLOW

An innovative design flow is described in this part. It can be shown that the waterfall model on fig. 1 is a suitable solution for designs with HSLA toolbox.

In the first step, goals of project are specified. Decompositions to lower levels of design and other detail specifications are also created. In the second step, solution is checked by computing test scripts in Matlab. In the next step, bit-exact and cycle exact model is created in Simulink. Test vectors for further debugging are then generated. Algorithm is after that rewritten to DK1 design suite environment. Debug result from DK1 must match with test-vectors generated in Simulink. In the final step, hardware is created directly from DK1 environment. Results from hardware are also checked with generated test-vectors.

Figure 1: Waterfall design-flow for design with HSLA library



Implementation flow is going step-by-step downto hardware. Each step is checked whether final and intermediate results are still matching results from previous steps. New discovers and restrictions, previously not known, may affect previous design specifications. All previous steps have to be revisited.

HSLA LIBRARY

HSLA library was created as a Matlab toolbox modeling bit-exact hardware modules of Logarithmic Arithmetic Unit (LAU). It is providing standard operations as multiplication, division, addition and subtraction etc. Logarithmic numbers are held in a special format as 32bit or 19bit integers, see fig. 2. Two versions of library are thus available. The 19bit precision LNS format maintains the same range as 32bit but has precision reduced to 10 fractional bits. Logarithmic operations can be divided into the two categories (see tab. 1). There are conversions to logarithmic domain and back to integers in the first category (using Matlab logarithms). The second category can be further divided to *standard* and *extended* precision operations expects input and output numbers in the range <–1,1> of input and output numbers is expected. Higher precision of used arithmetic is reached without any change of arithmetic unit.

Figure 2: Format for 32bit and 19bit logarithmic numbers

Sign	Integer part	Fractional part
1bit	8bit	23bit (10 bit for 19bit LNS)

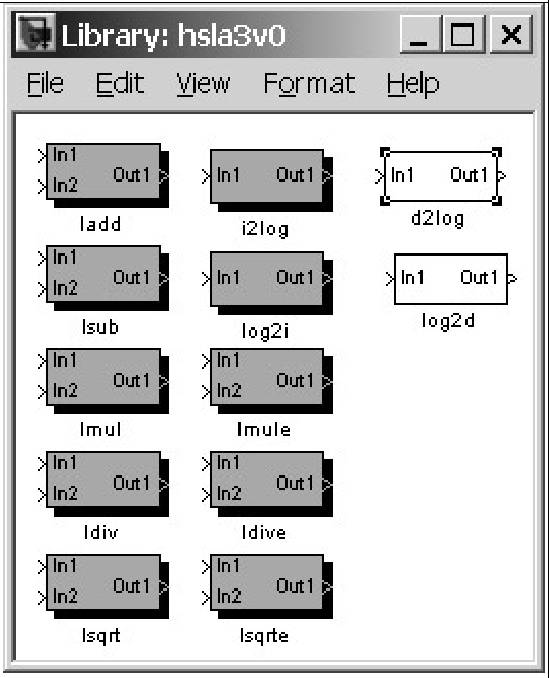
LATTICE IMPLEMENTATION

RLS Lattice filter is advanced DSP algorithm. It has faster convergence than LMS (least mean squares) and better numerical stability than pure RLS (recursive least squares).

Algorithm was implemented as script in Matlab and its functionality was tested. Consequently algorithm was implemented in Simulink as well. Simulink library blocks were used for *bit-exact* and *cycle-exact* lattice algorithm modeling, see table 1. On the figure 3, main simulation is shown. Demo sound of Fiona Kennedy is loaded and disturbed by band-limited white noise transformed by unknown FIR filter with variable parameters. The result is then connected into the RLS lattice filter input and it is also stored for debugging of hardware. Signal going from lattice output is stored as wavefile. It can be displayed in scopes and further analyzed (FFT). Results from simulations are stored and they are used for debugging in hardware implementation steps.

Table 1: Usage of HSLA library operations in Matlab scripts and in Simulink

MATLAB USAGE	SIMULINK USAGE

HARDWARE BIT-EXACT (blue)
Standard:
`z = ladd(a,b) % addition`
`z = lsub(a,b) % subtraction`
`z = lmul(a.b) % multiply`
`z = ldiv(a,b) % division`
`z = lsqrt(a)  % square root`
`z = i2log(a)  % conversion`
`z = log2i(a)  % conversion`
Extended precision:
`z = lmule(a,b)% multiply`
`z = ldive(a,b)% division`
`z = lsqrte(a) % square root`
MATLAB (white)
`z = d2log(a)  % conversion`
`z = log2d(a)  % conversion`
	 Library: hsla3v0
	File Edit View Format Help
	ladd
	i2log
	d2log
	lsub
	log2i
	log2d
	lmul
	lmule
	ldiv
	ldive
	lsqrt
	lsqrte
	In1
	In2
	Out1

Decomposition of the inner structure of lattice filter can be seen on fig. 4. There are conversions from integer to logarithmic domain at the input and output. RLS lattice algorithm is separated to four stages. Pipelining is created from these stages in the real hardware, because they can run in parallel. Situation is modeled by additional delays interlaced between stages. Initial setting for each stage can be seen in the figure 4.

Described lattice configuration is caused by pre-defined hardware restrictions and by restrictions discovered during design time. LAU provides only two add-sub operations in parallel. Two stages of lattice are thus sharing the same add-sub module. Sharing conflicts are avoided by time shift in stage pair that share same add-sub module. Structure of the hardware design must be adapted to parameters of LAU. For example, 19-bit lattice filter requires 28 BRAMs. 6 BRAMs are used in the LAU, 2 BRAMs are consumed by the input output conversions and 20 BRAMs are employed in four stages of the lattice filter. Hardware must be thus adapted for specific hardware resources available in each FPGA device. In our case Virtex XCV-800 or larger device must be used for our lattice macro, because the sum of blockrams required (28) is the same as number of available blockrams in this FPGA (28).

IMPLEMENTATION RESULTS

Noise cancellation scheme was implemented. The block diagram can be seen on the fig. 3. Noise cancellation is an application of adaptive filter, where desired signal is disturbed by a noise transformed by an unknown FIR filter. For example, a passenger is using a mobile phone in a car. The car noise on the other side of the phone line is disturbing, but it can be effectively suppressed by an adaptive filter. An additional microphone (reference sensor) is required to acquire the car noise. The task of adaptive filter is to estimate the transfer function of the noise coming inside of the passenger cabin. When the outside noise is known and the transfer estimated the disturbance could be easily suppressed by subtracting the transformed car noise from the signal acquired by the primary sensor.

Working lattice design in Celoxica DK1 can be converted to VHDL target and compiled downto hardware. Some additional changes are necessary to be made in the design such as interface to audio codec and to other external hardware.

Evaluation board XSV-800 from XESS Corporation was used for implementation of our noise cancellation scheme. The main device on the board is XCV-800-4. Audio input is connected through stereo audio codec to FPGA. Some LEDs are used as signals from design on evaluation board. One pushbutton is used as a switch between lattice noise canceller and direct input to output interconnection. External SRAMs are firstly used for initial lattice settings and when design starts, they are filled by input output data, until maximal capacity is reached. Data stored in external memories are read and tested in Simulink.

Full configuration and implementation results of the lattice noise canceller design are summarized in table 2.

Table 2: XESS evaluation board XSV-800. Used part are marked and numbered. Virtex XCV-800-4 occupation marked on right side

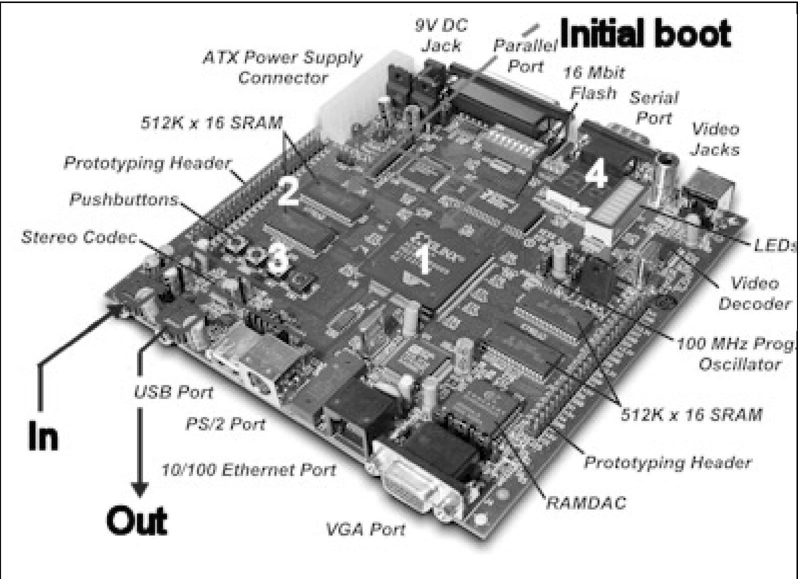
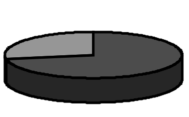
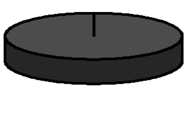
 Initial boot 9V DC Jack Parallel Port 16 Mbit Flash Serial Port Video Jacks ATX Power Supply Connector 512K x 16 SRAM Prototyping Header Pushbuttons Stereo Codec 1 2 3 4 LEDs Video Decoder 100 MHz Prog. Oscillator 512K x 16 SRAM Prototyping Header RAMDAC USB Port PS/2 Port 10/100 Ethernet Port VGA Port In Out	XCV800-4 Design Summary
	Slices  72 %
	Block RAMs  100 %
1) XESS Lattice Design 2) Startup configuration in SRAM 3) Lattice on/off 4) Info LEDs	Frequency 33.492 MHz

Figure 3: Lattice main project in Simulink

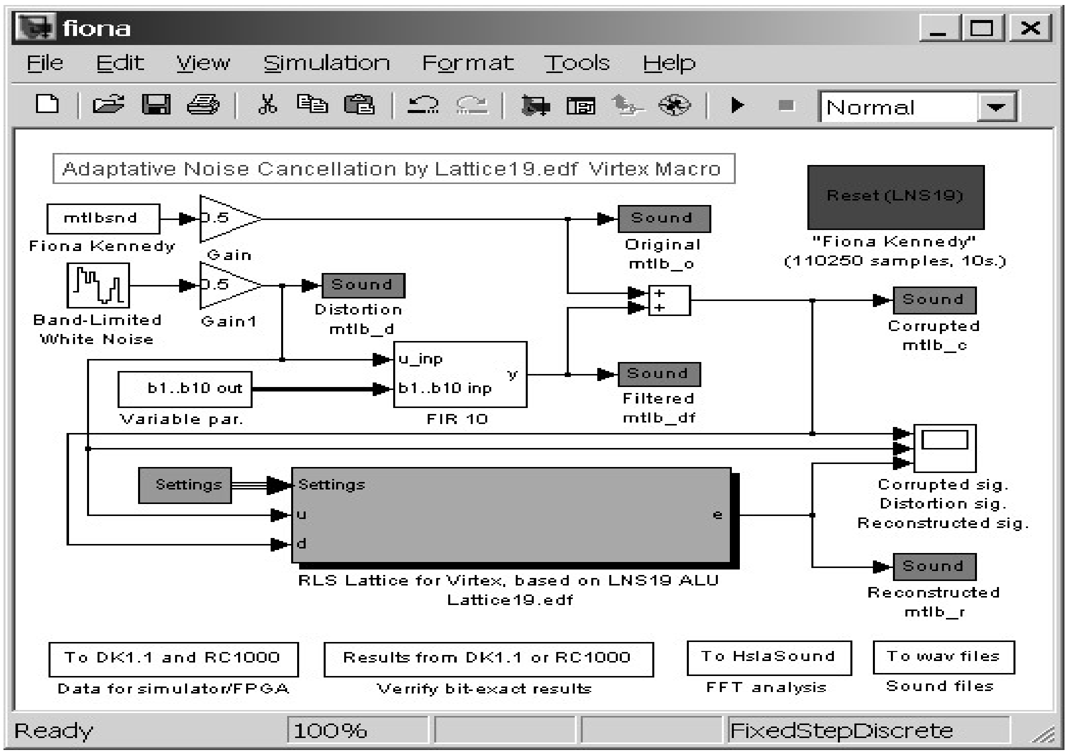
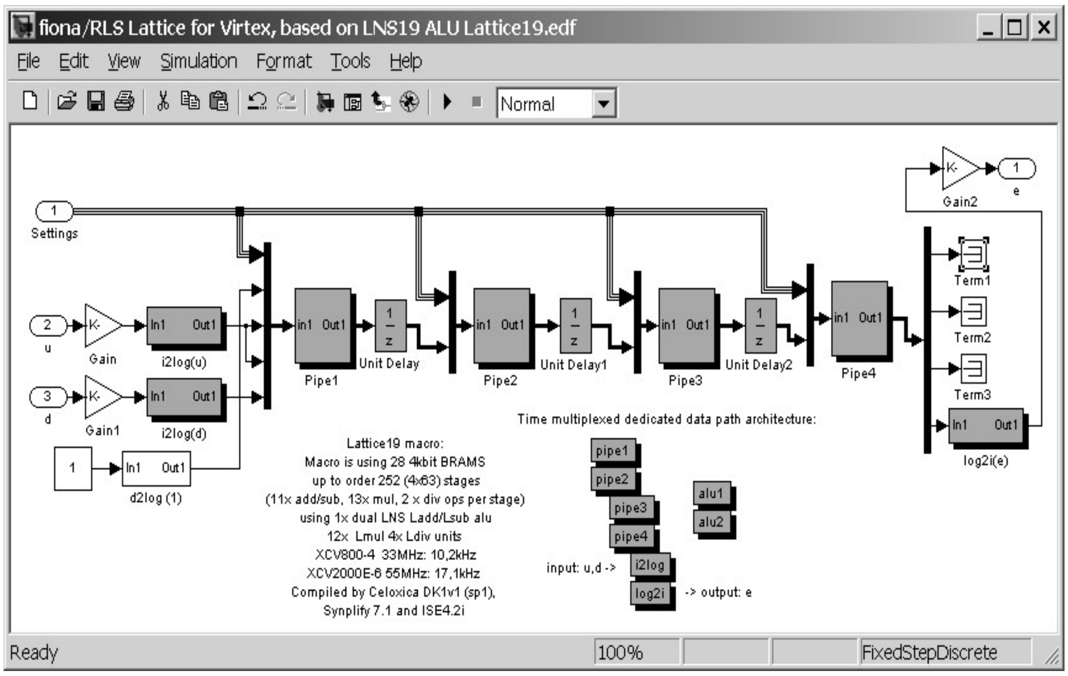


Figure 4: Lattice filter detail



Lattice19 macro:
Macro is using 28 4kbit BRAMS
up to order 252 (4x63) stages
(11x add/sub, 13x mul, 2 x div ops per stage)
using 1x dual LNS Ladd/Lsub alu
12x Lmul 4x Ldiv units
XCV800-4 33MHz: 10,2kHz
XCV2000E-6 55MHz: 17,1kHz
Compiled by Celoxica DK1v1 (sp1),
Synplify 7.1 and ISE4.2i

Audio input is composed from a *disturbed signal in the right channel* and from *a measured noise in the left channel*. Fast furrier transforms (FFT) of these signals can be seen on fig. 5. Transformation of the noise is time varying and in FFT it can be seen as a dark wave in time. Original input sound FFT is in the bottom right corner. Finally FFT of the result from the hardware lattice filter is drawn in the bottom left corner. Note, that the reconstructed sound does not match the original sound exactly. It is caused by the A/D and D/A conversions, different input and output signal gain and finally by the parameter estimation error.

Performance of the implemented filter together with the performance of arithmetic unit is illustrated in table 3. Design is synchronized by a given audio sampling period. This way of synchronization makes time limit for computations. Lattice filter maximal orders for different sampling frequencies are in the table. Order limitation is implied by the given time or by the maximal size of internal variables.

Figure 5: FFT analysis of inputs and output. Left top corner – noise, right top corner – transformed noise mixed with the input signal. Right bottom corner – original input signal, left bottom corner – output from the evaluation board (reconstructed input signal)

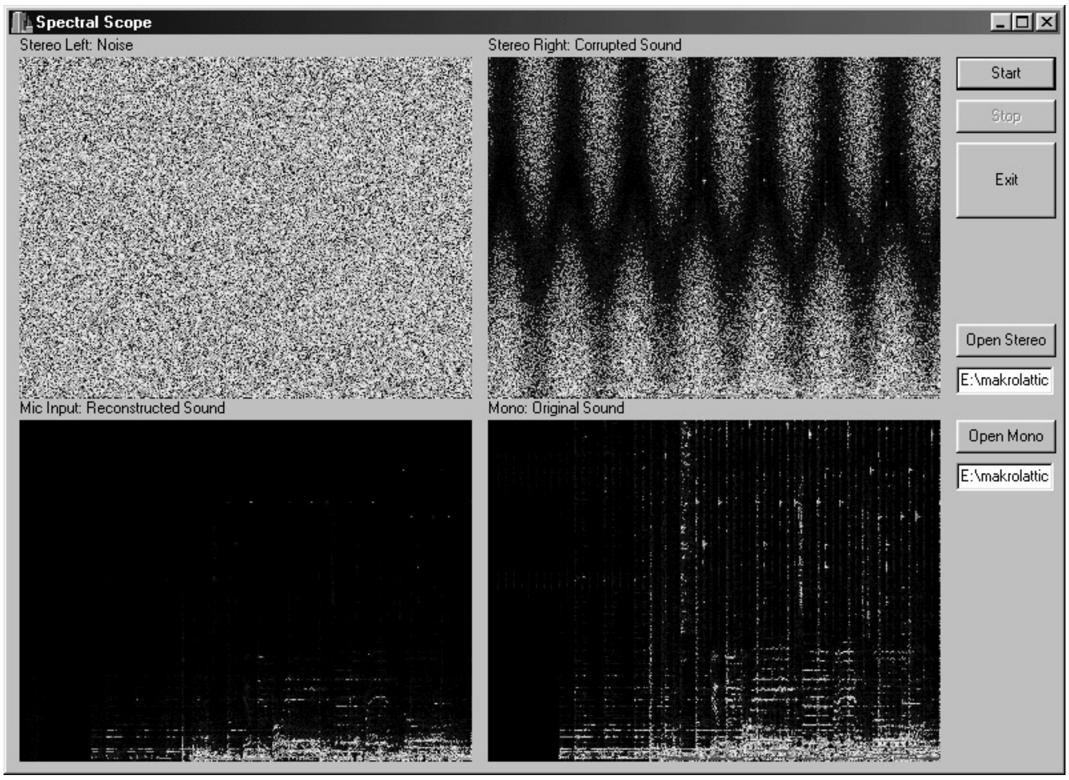


Table 3: Maximal orders and performance for RLS lattice noise canceller demo. Clock speed 33MHz.

Audio Sampling Period [Hz]	Maximal Pipe Order	Maximal Lattice Order	Mflops
31680	21	84	74.5
15840	43	172	76.3
7920	63*	252*	55.9

* Maximal degree allowed by Block RAM size reached

CONCLUSIONS

Matlab library designed for the bit-exact simulation of LNS operations is easy to use tool capable to help to develop various algorithms. The library helps, in cooperation with Matlab, Simulink and Celoxica DK1, with debugging and implementation of designs, where “cascade-like” design flow can be easily used. Implementation of the computationally intensive RLS lattice algorithm was done successfully by methods described above. The lattice algorithm was integrated into the noise cancellation scheme and into the XESS evaluation board as a demonstration of the real-time DSP application. There were discovered some possible lattice filter improvements that will result in a higher performance.

ACKNOWLEDGEMENTS

Work on the HSLA library was supported by the ESPRIT Long-term research project No. 33544 “HSLA” and by the Ministry of Education of the Czech Republic under Project LN00B096.

REFERENCES

[1] Friedlander B.: Lattice Filters for Adaptive Processing. In: Proceedings IEEE, August 1982, no. 8, vol. 70, pp. 829-867.

[2] Matoušek R., Tichý M., Pohl Z., Kadlec J., Softley C.: Logarithmic number system and floating-point arithmetics on FPGA. In: Field-Programmable Logic and Applications: Reconfigurable Computing Is Going Mainstream. (Glesner M., Zipf P., Renovell M. eds.). (Lecture Notes in Computer Science. A 2438). Springer, Berlin 2002, pp. 627-636.

[3] CELOXICA, "Homepage", http://www.celoxica.com

[4] XESS Corporation, "Homepage", http://www.xess.com

Zdeněk Pohl, UTIA, xpohl@utia.cas.cz

Institute of Information Theory and Automation of the Academy of sciences of The Czech Republic

Pod vodárenskou věží 4,
182 08 Praha 8
Czech Republic

Miroslav Líčko, CAK, licko@utia.cas.cz

Department of Control Systems
Center for Applied Cybernetics

Faculty of Electrical Engineering
Czech Technical University

Karlovo nám. 13,
121 35 Prague 2
Czech Republic