VILNIUS GEDIMINAS TECHNICAL UNIVERSITY

Tomyslav SLEDEVIČ

AN EFFICIENT IMPLEMENTATION OF LATTICE-LADDER MULTILAYER PERCEPTRONS IN FIELD PROGRAMMABLE GATE ARRAYS

DOCTORAL DISSERTATION

TECHNOLOGICAL SCIENCES,
ELECTRICAL AND ELECTRONIC ENGINEERING (01T)

VGTU
LEIDYKLA TECHNIKA
Vilnius 2016

Doctoral dissertation was prepared at Vilnius Gediminas Technical University in 2012–2016.

Scientific supervisor

Prof. Dr Dalius NAVAKAUSKAS (Vilnius Gediminas Technical University, Electrical and Electronic Engineering – 01T).

The Dissertation Defense Council of Scientific Field of Electrical and Electronic Engineering of Vilnius Gediminas Technical University:

Chairman

Prof. Dr Vytautas URBANAVIČIUS (Vilnius Gediminas Technical University, Electrical and Electronic Engineering – 01T).

Members:

Prof. Dr Habil Arūnas LUKOŠEVIČIUS (Kaunas University of Technology, Electrical and Electronic Engineering – 01T),

Prof. Dr Jurij NOVICKIJ (Vilnius Gediminas Technical University, Electrical and Electronic Engineering – 01T),

Prof. Dr Habil Adolfas Laimutis TELKSNYS (Vilnius University, Informatics Engineering – 07T),

Dr Pawel WAWRZYNSKI (Warsaw University of Technology, Electrical and Electronic Engineering – 01T).

The dissertation will be defended at the public meeting of the Dissertation Defense Council of Electrical and Electronic Engineering in the Senate Hall of Vilnius Gediminas Technical University at **2 p. m. on 8 June 2016**.

Address: Saulėtekio al. 11, LT-10223 Vilnius, Lithuania.
Tel. +370 5 274 4956; fax +370 5 270 0112; e-mail: doktor@vgtu.lt

A notification on the intend defending of the dissertation was send on 6 May 2016.
A copy of the doctoral dissertation is available for review at VGTU repository http://dspace.vgtu.lt and at the Library of Vilnius Gediminas Technical University (Saulėtekio al. 14, LT-10223 Vilnius, Lithuania).

VGTU leidyklos TECHNIKA 2371-M mokslo literatūros knyga

Parengta LaTeX2$_\varepsilon$ sistema

ISBN 978-609-457-933-2

tomyslav.sledevic@vgtu.lt

VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS

Tomyslav SLEDEVIČ

PYNUČIŲ-KOPĖTĖLIŲ DAUGIASLUOKSNIŲ PERCEPTRONŲ EFEKTYVUS ĮGYVENDINIMAS LAUKU PROGRAMUOJAMOMIS LOGINĖMIS MATRICOMIS

MOKSLO DAKTARO DISERTACIJA

TECHNOLOGIJOS MOKSLAI,
ELEKTROS IR ELEKTRONIKOS INŽINERIJA (01T)

VGTU
LEIDYKLA TECHNIKA
Vilnius 2016

Disertacija rengta 2012–2016 metais Vilniaus Gedimino technikos universitete.

Vadovas

prof. dr. Dalius NAVAKAUSKAS (Vilniaus Gedimino technikos universitetas, elektros ir elektronikos inžinerija – 01T).

Vilniaus Gedimino technikos universiteto Elektros ir elektronikos inžinerijos mokslo krypties disertacijos gynimo taryba:

Pirmininkas

prof. dr. Vytautas URBANAVIČIUS (Vilniaus Gedimino technikos universitetas, elektros ir elektronikos inžinerija – 01T).

Nariai:

prof. habil. dr. Arūnas LUKOŠEVIČIUS (Kauno technologijos universitetas, elektros ir elektronikos inžinerija – 01T),

prof. dr. Jurij NOVICKIJ (Vilniaus Gedimino technikos universitetas, elektros ir elektronikos inžinerija – 01T),

prof. habil. dr. Adolfas Laimutis TELKSNYS (Vilniaus universitetas, informatikos inžinerija – 07T),

dr. Pawel WAWRZYNSKI (Varšuvos technologijos universitetas, elektros ir elektronikos inžinerija – 01T).

Disertacija bus ginama viešame Elektros ir elektronikos inžinerijos mokslo krypties disertacijos gynimo tarybos posėdyje **2016 m. birželio 8 d. 14 val.** Vilniaus Gedimino technikos universiteto senato posėdžių salėje.

Adresas: Saulėtekio al. 11, LT-10223 Vilnius, Lietuva.
Tel. +370 5 274 4956; fax +370 5 270 0112; el. paštas: doktor@vgtu.lt

Pranešimai apie numatomą ginti disertaciją išsiųsti 2016 m. gegužės 6 d..

Disertaciją galima peržiūrėti VGTU talpykloje http://dspace.vgtu.lt ir Vilniaus Gedimino technikos universiteto bibliotekoje (Saulėtekio al. 14, LT-10223 Vilnius, Lietuva).

Abstract

The implementation efficiency of electronic systems is a combination of conflicting requirements, as increasing volumes of computations, accelerating the exchange of data, at the same time increasing energy consumption forcing the researchers not only to optimize the algorithm, but also to quickly implement in a specialized hardware. Therefore in this work, the problem of efficient and straightforward implementation of operating in a real-time electronic intelligent systems on field-programmable gate array (FPGA) is tackled. The object of research is specialized FPGA intellectual property (IP) cores that operate in a real-time. In the thesis the following main aspects of the research object are investigated: implementation criteria and techniques.

The aim of the thesis is to optimize the FPGA implementation process of selected class dynamic artificial neural networks. In order to solve stated problem and reach the goal following main tasks of the thesis are formulated: rationalize the selection of a class of Lattice-Ladder Multi-Layer Perceptron (LLMLP) and its electronic intelligent system test-bed – a speaker dependent Lithuanian speech recognizer, to be created and investigated; develop dedicated technique for implementation of LLMLP class on FPGA that is based on specialized efficiency criteria for a circuitry synthesis; develop and experimentally affirm the efficiency of optimized FPGA IP cores used in Lithuanian speech recognizer.

The dissertation contains: introduction, four chapters and general conclusions. The first chapter reveals the fundamental knowledge on computer-aided-design, artificial neural networks and speech recognition implementation on FPGA. In the second chapter the efficiency criteria and technique of LLMLP IP cores implementation are proposed in order to make multi-objective optimization of throughput, LLMLP complexity and resource utilization. The data flow graphs are applied for optimization of LLMLP computations. The optimized neuron processing element is proposed. The IP cores for features extraction and comparison are developed for Lithuanian speech recognizer and analyzed in third chapter. The fourth chapter is devoted for experimental verification of developed numerous LLMLP IP cores. The experiments of isolated word recognition accuracy and speed for different speakers, signal to noise ratios, features extraction and accelerated comparison methods were performed.

The main results of the thesis were published in 12 scientific publications: eight of them were printed in peer-reviewed scientific journals, four of them in a Thomson Reuters Web of Science database, four articles – in conference proceedings. The results were presented in 17 scientific conferences.

Reziumė

Elektroninių sistemų įgyvendinimo efektyvumas yra prieštaringų reikalavimų derinys, nes didėjančios skaičiavimų apimtys, spartėjantys duomenų mainai, o tuo pačiu didėjantis energijos suvartojimas verčia ne tik optimizuoti algoritmus, bet ir juos greitai įgyvendinti specializuotoje aparatūroje. Todėl disertacijoje sprendžiama realiuoju laiku veikiančių elektroninių intelektualiųjų sistemų efektyvaus įgyvendinimo lauku programuojama logine matrica (LPLM) problema. Tyrimo objektas yra specializuoti LPLM intelektinės nuosavybės (IN) moduliai veikiantys realiuoju laiku. Disertacijoje tiriami šie, su tiriamuoju objektu susiję, dalykai: įgyvendinimo kriterijai ir metodas.

Disertacijos tikslas – optimizuoti pasirinktos dinaminių dirbtinių neuronų tinklo klasės įgyvendinimo procesą LPLM. Siekiant išspręsti nurodytą problemą ir pasiekti tikslą suformuluojami šie pagrindiniai disertacijoje sprendžiami uždaviniai: Pagrįsti įgyvendinimui ir tyrimams atrinktų pynučių-kopėtėlių daugiasluoksnio perceptrono (PKDP) klasės ir jos elektroninės intelektualiosios testavimo aplinkos – neįgaliesiems skirto priklausomo nuo kalbėtojo lietuviškos šnekos atpažintuvo, pasirinkimą; sukurti specializuotais grandynų sintezės kriterijais paremtą PKDP klasės įgyvendinimo LPLM metodą; sukurti optimizuotus LPLM IN modulius ir taikant lietuvių šnekos atpažintuve eksperimentiškai patvirtinti jų efektyvumą.

Disertaciją sudaro įvadas, keturi skyriai, bendrosios išvados. Pirmajame skyriuje pateikiamos esminės žinios apie kompiuterizuotą grandinių, dirbtinių neuronų tinklų ir kalbos atpažinimo įgyvendinimą LPLM. Antrajame skyriuje LPLM IN modulių įgyvendinimui siūlomi efektyvumo kriterijai ir metodas daugiakriteriniam optimizavimui pagal greitaveiką, resursų naudojimą ir PKDP sudėtingumą. PKDP skaičiavimams optimizuoti yra taikomi duomenų srauto grafai. Pasiūloma optimizuota neuroninio apdorojimo elemento struktūra. Požymių išskyrimo ir palyginimo IN moduliai įgyvendinti lietuvių šnekos atpažintuve ir yra analizuojami trečiajame skyriuje. Įgyvendinti PKDP IN moduliai yra eksperimentiškai patikrinti ketvirtajame skyriuje. Tiriamas lietuvių šnekos pavienių žodžių atpažinimo sistemos tikslumas ir greitaveika priklausomai nuo kalbėtojo, signalo ir triukšmo santykio, požymių išskyrimo ir palyginimo metodų. Tiriamos PKDP IN modulio taikymo galimybės triukšmui šalinti kalbos signale.

Pagrindiniai disertacijos rezultatai paskelbti 12-oje mokslinių straipsnių, iš kurių 8 atspausdinti recenzuojamuose mokslo žurnaluose. Iš jų 4 straipsniai paskelbti mokslo žurnaluose, įtrauktuose į *Thomson Reuters Web of Science* duomenų bazę. Rezultatai viešinti 17 mokslinių konferencijų.

Notations

In General[1]

Text	– emphasis;
Text	– a foreign (depends on context) word or abbreviation;
Text	– computer program, chip familly and number;
a	– scalar (number);
$\boldsymbol{a}$	– vector (column of numbers);
$\boldsymbol{A}$	– matrix (multidimensional array of numbers);
$\mathcal{A}$	– set of elements (dataset), graph;
$\mathfrak{a}$	– class (criteria);
$\mathtt{A}$	– program (in VHDL) variable;
$\blacksquare_{\text{label}}$, $\blacksquare^{\text{label}}$	– main and supplement labels;
$\blacksquare^{(i)}$, $\blacksquare_i$	– main and supplement index;
$\blacksquare^{\star}$	– important (to be specified) aspect;
$\blacksquare^{\bigstar}$	– value achieved by authors implementation;
$\blacksquare^{\top}$, $\blacksquare^{\bot}$	– maximum and minimum values.

[1] All units of measurement related to digital electronics and computing complies with a standard IEEE 1541-2002.

Symbols

a_i	– LPC coefficients;
$\mathtt{A}, \mathtt{B}, \mathtt{C}, \mathtt{D}, \mathtt{P}$	– DSP block input/output terminals;
$\boldsymbol{c}_t(n)$	– vector of cepstrum coefficient at time moment n of type t: "LF" – LFCC, "MF" – MFCC, "LP" – LPCC;
$\delta_h^{(l)}(n)$	– instantaneous error of hth neuron in lth layer;
$\epsilon_t(f_\mathrm{b}), \epsilon_t(f_\mathrm{b})$	– relative to bandwidth f_b (normalized bandwidth f_b) absolute (normalized) errors of t: "b" – bandwidth, "c" – central frequency;
$e_h(n)$	– instantaneous output error of hth neuron;
E	– mean square error;
$E(n)$	– instantaneous mean square error;
E_i	– an ith edge of directed graph (subgraph);
$\mathcal{E}$, $\mathcal{E}_t^{(l)}$	– a set of edges of directed graph and a set of edges of l-th sub-graph of type t: "⋆" – all subgraphs, "C" – covered;
$\mathscr{E}_\mathrm{MA}$, $\mathscr{E}_\mathrm{MA}$	– mean absolute error (MAE) and normalized MAE;
$\mathscr{E}_\mathrm{MM}$	– maximum magnitude error (MME);
$\mathscr{E}_\mathrm{RMS}$	– root mean square error (RMSE);
$f^\top$	– maximal frequency of (circuit) operation;
f_s	– signal sampling frequency;
f_t, $\widetilde{f}_t$, f_t	– float-point, fixed-point and normalized fixed-point designs t: "h" – cut-off high frequency, "l" – cut-off low frequency, "c" - central frequency, "b" – bandwidth;
$f_{ij}^{(l)}(n)$, $b_{ij}^{(l)}(n)$	– lattice-ladder feedforward and feedback signals;
$\Phi_t^{(l)}(\cdot)$	– lth layer neuron activation function of type t: "htan" – hyperbolic tangent, "step" – binary step, "ident" – identity;
$\mathcal{F}_i$	– ith female speaker set of word records;
g	– neuron activation function gain coefficient;
$\mathcal{G}$, $\mathcal{G}_t^{(l)}$	– directed graph and l-th sub-graph (sets) of type t: "⋆" – all subgraphs, "DSP" – supported by DSP, "C" – covered, "L" – on schedule list, "S" – scheduled;
i, j, h	– indices for inputs, synapse weights (coefficients) and outputs;
K_m	– the transfer function of the mel-scaled triangular mth filter in a filter bank;
l	– index of ANN layer;
L	– a total number of ANN layers;
M, $M^{(l)}$	– an order of: a model and lth layer filters;
$\mathcal{M}_i$	– ith male speaker set of word records;

n	– time (sample) index;
N, $N^{(l)}$	– a total number of: samples and lth layer neurons;
N_Π, N_Σ, N_Φ	– a total number of multiplication, addition and trigonometric operations;
$N_t^\star$	– a total number of mathematical operations (t: Π – multiplications, Σ – additions) performed by regressor of type "$\star$": $\mathfrak{t}_3$, $\mathfrak{t}_4$, $\mathfrak{t}_{11}$ and $\mathfrak{t}_{12}$;
P	– an input port of directed graph (subgraph);
$\mathcal{P}_{\text{xx}}(f)$, $\mathcal{P}_{\text{xy}}(f)$	– auto and cross power spectral densities;
$\mathcal{Q} \equiv \{\mathfrak{q}_t\}$	– a quality of efficient implementation dependent on a set of quality criteria of type t: "Thr" – throughput, "Acc" – accuracy, "Com" – complexity, "Res" – resource, "Pow" – power, "Price" – price;
r_i	– autocorrelation coefficients;
R_t, R_t^i, R_p^t	– an amount of FPGA resources of type t ("DSP", "BRAM", "LUT", "≡LUT", "Lat") that needs to be used by implementation techniques i ("HLS", "$\star$") in order to implement LLMLP defined by parameters p;
$\tilde{s}_{ih}^{(l)}(n)$	– output signal of synapse connecting ith with $h$tj neuron;
$\mathring{s}_h^{(l)}(n)$, $s_h^{(l)}(n)$	– signal before and after hth neuron activation function;
$T_{\text{LLN}}(f; g)$	– floating point implementation LLN (with gain g) estimated transfer function;
$\widetilde{T}_{\text{LLN}}\left(f; g, R_{\text{BRAM}}\right)$	– fixed-point implementation LLN (with gain g and BRAM size R_{BRAM}) estimated transfer function;
τ, τ^{CP}	– latency of FPGA implementation and its critical path;
τ_i^o	– latency of implemented in FPGA object (o: synapse, neuron) at specified junction i indexed by: l – layer, n – neuron, m – order, $\mathfrak{t}$ – training;
$\mathcal{T} \in \{\mathfrak{t}_i\}$	– a set of LLMLP training algorithms $\mathfrak{t}_i$;
Θ_j	– rotation angle of lattice coefficients;
$v_{ijh}^{(l)}$	– ladder coefficients;
V_i	– an ith vertex of directed graph (subgraph);
$\mathcal{V}$, $\mathcal{V}_t^{(l)}$	– a set of vertices of directed graph and a set of vertices of l-th sub-graph of type t: "$\star$" – all subgraphs, "DSP" – supported by DSP, "C" – covered;
$\omega(n)$	– white noise sample;
w, $\boldsymbol{w}$	– weights or system parameters;
W, W_{i}, W_{f}	– word length in bits: total, integer and fractional parts.

Operators and Functions

$\equiv$	– equivalence;
$\Rightarrow$	– implication;
$\rightarrow$	– substitution;
$\triangleq$	– definition;
$\approx$	– approximation;
$\wedge$, $\vee$	– conjunction and disjunction;
$\lvert . \rvert$	– absolute value (modulus);
$\lVert . \rVert$	– norm (length);
$\widehat{.}$	– estimate;
$\widetilde{.}$	– fixed-point calculations;
$\overline{.}$	– normalized value;
$\underline{.}$	– mean;
$.^{\hookrightarrow t}$, $.^{\hookleftarrow t}$	– feedforward and feedback processing order in parts t: 's' – synapse; 'n' – node
$\nabla_x(n)$	– instantaneous gradient w.r.t. x;
z, z^{-1}	– delay and advancement;
$\mathcal{C}(.)$	– discrete cosine transform;
$\mathcal{F}(.)$, $\mathcal{F}^{-1}(.)$	– direct and inverse fast Fourier transform;
$\mathcal{W}_{\mathrm{DT}}(.)$	– Dynamic Time Warping transform;
$\mathrm{addr}(a)$	– address of element a;
$\mathrm{init}(.)$, $\mathrm{ter}(.)$	– initial and terminal nodes of an edge;
$\max(.)$, $\min(.)$	– maximum and minimum;
$\mathrm{mel}(.)$	– mel-scale;
$\mathrm{pri}(.)$	– priority;
$\mathrm{reg}(.)$	– registry;
$\mathrm{Re}(.)$, $\mathrm{Im}(.)$	– real and imaginary parts of complex number.

Abbreviations

ANN	– artificial neural network;
ARM	– advanced RISC machines;
ASAP	– as soon as possible;
ASIC	– application-specific integrated circuit;
BRAM	– block random access memory;
CAD	– computer-aided design;
CPM	– critical path method;
DFG	– data-flow graph;
DFS	– depth first search;

DSP – digital signal processor;
DTW – dynamic time warping;
FFT – fast Fourier transform;
FIR – finite impulse response;
FPGA – field programmable gate array;
HDL – hardware description language;
HLS – high level synthesis;
IIR – infinite impulse response;
IP – intellectual property;
LFCC – linear frequency cepstral coefficient;
LL – lattice-ladder;
LLF – lattice-ladder filter;
LLMLP – lattice-ladder multilayer perceptron;
LLN – lattice-ladder neuron;
LPC – linear predictive coding;
LPCC – linear prediction cepstral coefficient;
LSR – Lithuanian speech recognizer;
LUT – look-up table;
MAC – multiply-accumulate;
MAE – mean absolute error;
MFCC – mel frequency cepstral coefficient;
MLP – multilayer perceptron;
MME – maximum magnitude error;
NPE – neuron processing element;
PAR – place and route;
PWL – piecewise linear;
RAM – random access memory;
RMSE – root mean square error;
ROM – read-only memory;
RTL – register-transfer level;
VLSI – very large scale integration.

Keywords[2]

IEEE Taxonomy	Part(s)
Circuits and systems	
Circuits	
Programmable circuits – Field programmable gate arrays	**2 3 4**
Computational and artificial intelligence	
Artificial intelligence	
Intelligent systems	**1.2 1.3 3.4 4.1 4.2**
Neural networks	
Neural network hardware	**1.2 2.1 2.2 3.2 4.1**
Computers and information processing	
Computer science	
Programming – Performance analysis	**2 4**
Pattern recognition	
Speech recognition – Automatic speech recognition	**1.3 3 4.2**
Software	
Software packages – MATLAB	**1.1 2.2**
Mathematics	
Algorithms	
Backpropagation algorithms	**2.2 4.1**
Computational efficiency	**2 4**
Signal processing	
Acoustic signal processing – Speech processing	**3.1 3.2 3.3 4.2**
Systems engineering and theory	
System analysis and design	
System performance	**1 2 4**

[2]Keywords (assigned to parts) are structured and in-line with the latest IEEE taxonomy (see <https://www.ieee.org/documents/taxonomy_v101.pdf>).

Contents

INTRODUCTION 1
Problem Formulation 1
Relevance of the Thesis 2
The Object of the Research 3
The Aim of the Thesis 4
The Objectives of the Thesis 4
Research Methodology 4
Scientific Novelty of the Thesis 4
Practical Value of the Research Findings 5
The Defended Statements 5
Approval of the Research Findings 6
Structure of the Dissertation 7
Acknowledgements 7

1. REVIEW OF ELECTRONIC SYSTEMS IMPLEMENTATION IN FIELD PROGRAMMABLE GATE ARRAYS 9
1.1. Computer-Aided Design for Implementation in Field Programmable Gate Array 9
1.1.1. Specifics of Field Programmable Gate Array Architectures 11
1.1.2. High Level Tools for Hardware Description Language Generation 13

1.2. Implementation of Artificial Neural Networks in Field Programmable Gate Array 15
1.2.1. Specifics of Artificial Neural Networks 16
1.2.2. Artificial Neuron Structures 19
1.2.3. Dynamic Neuron 21
1.2.4. Training of Dynamic Neuron 23
1.2.5. Indirect (High Level) Approach 28
1.2.6. Direct (Low Level) Approach 29
1.2.7. Artificial Neural Network Chips 31
1.2.8. Efficiency Criteria of Neural Network Hardware 32
1.3. Implementation of Speech Recognition in Field Programmable Gate Array 34
1.3.1. Linear Frequency Cepstral Analysis 37
1.3.2. Mel-Frequency Cepstral Analysis 38
1.3.3. Linear Predictive and Linear Predictive Cepstral Analysis . 39
1.3.4. Features Classification 41
1.4. Conclusions of the 1st Chapter and Formulation of the Thesis Objectives 42

2. EFFICIENT IMPLEMENTATION OF LATTICE-LADDER MULTI-LAYER PERCEPTRON 45
2.1. Implementation Quality 46
2.2. Introduction to Implementation Technique 49
2.3. Neuron Processing Element Optimization 52
2.3.1. The Data Flow Graphs Generation 52
2.3.2. Subgraph Matching 53
2.3.3. Graph Covering and Merging 54
2.3.4. Critical Path Search 56
2.3.5. Resource Constrained Scheduling 58
2.3.6. Design Description 60
2.4. Neuron Layers Optimization 63
2.4.1. Accuracy Optimization 63
2.4.2. Throughput Optimized Implementation Strategy 68
2.4.3. Resource Optimized Implementation Strategy 71
2.5. Conclusions of the 2nd Chapter 73

3. IMPLEMENTATION OF LITHUANIAN SPEECH RECOGNIZER IN FIELD PROGRAMMABLE GATE ARRAY 75
3.1. Speech Recognition System Overview 76
3.2. Features Extraction Implementations 78

3.2.1. Linear Frequency Cepstral Analysis Intellectual Property Core . . . 78
3.2.2. Mel-Frequency Cepstral Analysis Intellectual Property Core 79
3.2.3. Linear Predictive Cepstral Analysis Intellectual Property Core . . . 80
3.3. Word Recognition Implementations . . . 81
3.3.1. Dynamic Time Warping Intellectual Property Core . . . 81
3.3.2. Accelerated Pattern Matching Intellectual Property Core . . 85
3.4. Iterative Voice Response Interface . . . 88
3.5. Conclusions of the 3rd Chapter . . . 90

4. EXPERIMENTAL VERIFICATION OF DEVELOPED INTELLECTUAL PROPERTY CORES . . . 91
4.1. Investigation of Lattice-Ladder Multilayer Perceptron and its Implementation Technique . . . 92
4.1.1. Word Length Selection . . . 92
4.1.2. Neuron Activation Function Implementation . . . 93
4.1.3. Lattice-Ladder Neuron Implementation . . . 96
4.1.4. Single Layer of Lattice-Ladder Multilayer Perceptron Implementation . . . 102
4.1.5. Qualitative Lattice-Ladder Multilayer Perceptron Implementation . . . 103
4.2. Investigation of Lithuanian Speech Recognizer . . . 105
4.2.1. Comparison with Initial Developments . . . 106
4.2.2. Recognition Accuracy Tune-Up . . . 108
4.2.3. Execution Speed Determination . . . 113
4.3. Conclusions of the 4th Chapter . . . 115

GENERAL CONCLUSIONS . . . 117

LIST OF SCIENTIFIC PUBLICATIONS BY THE AUTHOR ON THE TOPIC OF THE DISSERTATION . . . 133

SUMMARY IN LITHUANIAN . . . 135

SUBJECT INDEX . . . 151

ANNEXES[3] . . . 155
Annex A. Created Intellectual Property Cores . . . 156
Annex B. The Co-authors' Agreement to Present Publications Material in the Dissertation . . . 157

[3]The annexes are supplied in the enclosed compact disc

Annex C. The Copies of Scientific Publications by the Author on the Topic of the Dissertation 162

Introduction

Problem Formulation

Rapid development of mobile, multimedia, 3D, and virtualization technology is the foundation of the modern electronic world development (Spectrum 2014). The fast spreading of intelligent technologies – the world's 16.8 % annual increase in smart phone sales in 2016 predicting 458 millions tablet computer sales and their predominance makes the problem of electronic systems efficient implementation more relevant (YANO 2015). During the implementation of intelligent electronic system process a combination of conflicting demands must be satisfied: accommodation of increasing computational complexity, catch-up of acceleration in the data exchange, deal with increasing energy consumption and computational resource requirements. All the mentioned reasons forces the researchers not only to optimize the processing algorithms as such, but to speed-up the whole implementation process, too.

The application of Field Programmable Gate Array (FPGA) technology enables electronic systems to reduce development and testing time. Moreover, due to FPGA array like structure, the parallelization and pipelining principles in conjunction with pre-optimized customizable arithmetic, logic and memory resources can be used in a way that the significant speed-up in the computations becomes possible. Therefore, in the following, the problem of efficient and straightforward implementation of operating in a real-time electronic intelligent systems on field programmable gate array is tackled.

In order to solve this problem such main hypothesis was raised and proven: an intrinsic structural features of electronic intelligent system class can be used for the formulation of specialized criteria for circuitry synthesis and the development of dedicated implementation technique, such that optimize the whole system implementation on field programmable gate array.

Relevance of the Thesis

According to Disability and Working Capacity Assessment Office, the need for permanent care is recognized for 17,731 persons and need for permanent help is assigned for 16,694 persons in Lithuania in 2015 (NDNT 2016). Such persons require a nursing staff to perform elementary actions: turn over in bed, turn the lights on or off, handle the household appliances, call for medical assistant, and so on. The ability to initiate actions by voice enables to increase the independence of humans with physical disabilities, improve their quality of life and reduce the need for nursing staff.

According to Ministry of Social Security and Labour of Lithuania 142,200 persons are registered with 30–40 % working capacity and 32,400 persons with 0–25 % working capacity in 2014 (Socmin 2015). The voice interface for those people can be an additional control interface, which enables to improve work efficiency and speed. Over the last 50 years the developed speech recognition techniques allow to implement the services based on speech recognition in personal computers and mobile phones. Despite the declared naturalness of voice interface and predicted growth, the spread of voice technology was not great. Just few software tools, which allow to type text by voice, control programs running on computer or phone can be found in a market.

The ability to produce voice controlled devices allows to create new services not only for entertainment, or for small tasks, but also for social tasks. One of the potential application areas of speech technology is people with physical disabilities. Voice control devices would be integrated into specialized functional units (handicapped beds, wheelchairs, hoists or communication with nursing systems), and other home devices (TV, lighting, window blinds, room heating and air conditioning systems). The ability to manage by voice would compensate a part of the lost skills and enhance the quality of life for people with disabilities. Excellent example of the application of speech technology is a speech synthesis which is used in the work with computer for the blind and weak-sighted people.

Over the last decade, the increased alternative of hardware equipment and application of specialized electronic devices form the opportunities to create speech recognition systems in a hardware. In particular a major boost

to speech technology has provided the recently launched smart phones with voice-controlled functions. Successful and well advertised product has recovered the forgotten idea of voice controlled devices. Over the last couple of years, Google, Nuance announced the creation of a voice controlled TV, offered a voice-operated GPS navigators and voice technology re-evaluated as one of the most promising forms of the human-machine interface (CES 2016). The essential difference between these solutions – voice operated stand alone devices, unrelated with computer and without online access to the network services. This shows a needfully growth of hardware tools for speech recognition and embedded systems requirements.

With the growing amount of data being processed, and in order to work on real-time it is necessary to increase the processing speed of the algorithms implemented on speech recognizers. For this purpose the accelerated intellectual property (IP) cores are developed for FPGAs. The increasing use of FPGA lies in the ability for algorithms parallelization, therefore, FPGA based devices work more efficiently in comparison with modern processors, even if the clock frequency is only in range of 10–300 MHz.

Due to the noisy environment, the creating of speech filtering IP core is important in FPGA based recognizer. The lack of scientific exploration of artificial neural network (ANN) application in hardware recognizers for noise reduction leads to the new investigations and development of ANN IP cores dedicated for enhancement of speech recognition accuracy. The known class of ANN is the lattice-ladder multilayer perceptron (LLMLP) and its FPGA implementation is investigated insufficiently. The implementation of LLMLP IP cores on FPGA is first time investigated in this thesis. The application of optimized LLMLP IP core in speech recognizer for word recognition rate enhancement is appropriate due to the stability in lattice part. The fast evaluation of the amount of required resources and proper FPGA chip selection is an important step in a system design. The creation of the automatic LLMLP IP generation tool based of proposed technique is needful in order to fast verification of design performance and suitable learning circuit selection at different LLMLP complexity.

The Object of the Research

The object of the research is specialized field programmable gate array intellectual property (FPGA IP) cores that operate in a real-time. The following main aspects of the research object are investigated in the present thesis: implementation criteria and techniques.

The Aim of the Thesis

The aim of the thesis is to optimize the field programmable gate array implementation process of lattice-ladder multi-layer perceptron and provide its working example in Lithuanian speech recognizer for disabled persons' use.

The Objectives of the Thesis

In order to solve stated problem and reach the aim of the thesis the following main objectives are formulated:

1. To rationalize the selection of a class of lattice-ladder multi-layer perceptron (LLMLP) and its electronic intelligent system test-bed – a speaker dependent Lithuanian speech recognizer for the disabled persons, to be created and investigated.
2. To develop technique for implementation of LLMLP class on FPGA that is based on specialized criteria for a circuitry synthesis.
3. To develop and experimentally affirm the efficiency of optimized FPGA IP cores used in Lithuanian speech recognizer.

Research Methodology

The following theories are applied in this work: digital signal processing, spectral and cepstral analysis, speaker dependent word recognition, artificial neural networks, optimization, data flow graph, and statistical analysis. Techniques of linear prediction, mel-frequency scaling, dynamic time-warping, lattice-ladder multilayer perceptron recall and training, graph covering, subgraph search, instruction scheduling, are adopted and implemented.

Original Lithuanian speech data sets for the experiments are recorded. The simulations are carried out with the use of Matlab 7 and ModelSim 6.5 software packages. The intelligent electronic systems are implemented on Virtex-4 and ZynQ-7000 FPGA family. For their development and experimental investigation Xilinx ISE Design Suite 14.7, Vivado HLS 2015.4 together with originally developed software tools are used.

Scientific Novelty of the Thesis

1. The new technique for LLMLP implementation, which takes into account FPGA specifics and generates IP core more efficiently in comparison with general purpose commercial tool, is created.

2. Pareto frontiers estimation for the LLMLP specialized criteria of circuitry synthesis, which supports optimal decision of FPGA chip selection according to given requirements for LLMLP structure, sampling frequency and other resources, is proposed.
3. The new accelerated pattern matching and double random seed matching algorithms, which accelerate word recognition process in a working prototype of Lithuanian speech recognizer in FPGA, are developed.
4. The noise filtering by lattice-ladder neuron, which improves the recognition accuracy, is proposed.

Practical Value of the Research Findings

Based on the proposed technique, a new compiler is created for LLMLP efficient implementation in FPGA. The results presented by Pareto frontiers enable us to choose a proper FPGA chip according to given requirements for LLMLP complexity, sampling frequency and resources.

The implemented Lithuanian speech recognition system is based on a new architecture ZynQ-7000 chip with integrated Artix-7 family FPGA and dual-core ARM Cortex A9 MPCore processor that allow to control devices by voice. The recognition system and implementation have been investigated and applied:

- in the scientific group project for technological development "Development and validation of control by Lithuanian speech unit model for the disabled", supported by Research Council of Lithuania (No. MIP-092/2012, 2012–2014);
- in the scientific work of VGTU "Research of the digital signal processing for real-time systems" (No. TMT 335, 2013–2017).

The device is able to communicate with human by voice, identify Lithuanian language given commands in a real-time and form a specified control signals. For each controlled device the list of the commands can be customized, thus increasing the efficiency of voice control. In this respect, the FPGA based recognizer does not have analogues in Lithuania and is one of a few in the world.

The Defended Statements

1. The IP cores created by use of proposed LLMLP implementation in FPGA technique are at least 3 times more efficient in comparison with the Vivado HLS tool.

2. Pareto frontiers estimation for the LLMLP specialized criteria of circuitry synthesis enables to make an optimal choice of FPGA chip according to given requirements for LLMLP structure, sampling frequency and other resources.
3. The optimized FPGA IP cores developed for Lithuanian speech recognizer are suitable for real-time isolated word recognition achieving 7800 word/s comparison speed.
4. The application of lattice-ladder neuron for 15 dB SNR records preprocessing improves the recognition rate by 4 %.

Approval of the Research Findings

The research results are published in 12 scientific publications:

- four articles are printed in a peer-reviewed scientific journals listed in a Thomson Reuters Web of Science list and having impact factor (Sledevič, Navakauskas 2016, Tamulevičius *et al.* 2015, Serackis *et al.* 2014, Sledevič *et al.* 2013);
- two articles are printed in a peer-reviewed scientific journal listed in Index Copernicus database (Sledevič, Stašionis 2013, Stašionis, Sledevič 2013);
- two articles are printed in a peer-reviewed scientific journal listed in SCImago database (Tamulevičius *et al.* 2014, Sledevič *et al.* 2013);
- four publications are printed in other scientific works: two – in international conference proceedings listed in Thomson Reuters Web of Science list ISI Proceedings category (Serackis *et al.* 2013, Sledevič, Navakauskas 2013) and two – in international conference proceedings listed in IEEEXPLORE (INSPEC) database (Sledevič, Navakauskas 2015, Sledevič, Navakauskas 2014).

The main results of the thesis are presented in the following 17 scientific conferences:

- 13th international "Biennial Baltic Electronics Conference (BEC)", 2012, Estonia, Tallinn;
- international conference on "Communication, Control and Computer Engineering (ICCCCE)", 2013, Turkey, Istanbul;
- 7th international European Modelling Symposium (EMS), 2013, England, Manchester;
- international conference on "Computer as a Tool (EUROCON)", 2013, Croatia, Zagreb;

- 16th international conference on "Image, Signal and Vision Computing (ICISVC)", 2014, France, Paris;
- national conference "Multidisciplinary Research in Natural and Technology Sciences" 2014, Lithuania, Vilnius;
- 3rd international workshop on "Bio-Inspired Signal and Image Processing (BISIP)", 2014, Lithuania, Vilnius;
- 6th international seminar "Data Analysis Methods for Software Systems", 2014, Lithuania, Druskininkai;
- 2nd international workshop on "Advances in Information, Electronic and Electrical Engineering (AIEEE)", 2014, Lithuania, Vilnius;
- international conferences "Electronics", 2013–2015, Lithuania, Palanga;
- 3rd international workshop on "Advances in Information, Electronic and Electrical Engineering (AIEEE)", 2015, Latvia, Riga;
- annual national conferences "Science – Future of Lithuania", 2013–2016, Lithuania, Vilnius.

The technical presentation of Lithuanian speech isolated word recognizer was recognized as one of the best in the conference "Multidisciplinary Research in Natural and Technology Sciences". This research was certificate awarded by Lithuanian Academy of Sciences together with "Infobalt" association established scholarship.

Structure of the Dissertation

The dissertation contains: introduction, four chapters, general conclusions, summary in Lithuanian, 3 annexes, list of references with separately presented list of publications by the author. The list of symbols, abbreviations, keywords and subject index are presented. The dissertation consists of 156 pages, where: 87 displayed equations, 73 figures, 12 tables, 6 algorithms and 1 example are presented. In total 152 references are cited in the thesis.

Acknowledgements

I would like to express my deepest gratitude to my supervisor Prof. Dr Dalius Navakauskas, for his excellent guidance, caring, patience, enthusiasm, and providing me with an excellent atmosphere for doing research. I would also like to thank all the staff at the Department of Electronic Systems of Vilnius Gedi-

minas Technical University for guiding my research for the past several years and helping me to develop my background in a field of electronics, artificial intelligence and speech recognition.

Special thanks goes to Assoc. Prof. Dr Artūras Serackis, who have first introduced me to FPGA and inspired to new research directions. I am grateful to Liudas Stašionis for the cooperation and assistance in recognizer implementation. I am thankful to Assoc. Prof. Dr Gintautas Tamulevičius, who gave me an understanding of methods applied in theory of speech recognition. I also want to thank Assoc. Prof. Dr Vacius Mališauskas, who gave me, when I was a young student, great insights into the nature of scientific work. I have greatly enjoyed the opportunity to work with Dovilė Kurpytė, Darius Plonis, Dalius Matuzevičius, Raimond Laptik, Andrius Katkevičius, Audrius Krukonis, Edgaras Ivanovas, and Ricardo Henrique Gracini Guiraldelli. Thank you for the fun and many motivating discussions about life, Ph.D.s, computer science, and all that we had over the last several years. I was very lucky to have crossed paths with Andrius Gudiškis, Darius Kulakovskis, Eldar Šabanovič, Vytautas Abromavičius, and Aurimas Gedminas. Thank you for your support and encouragement.

I am especially grateful for all the members of the Biometrics and Machine Learning Laboratory at Warsaw University of Technology for providing a great work environment, for sharing their knowledge and for their help during my internship.

I would also like to thank Lithuanian Academy of Sciences, Research Council of Lithuania, Education Exchanges Support Foundation and “Infobalt” association for research funding, foreign internship support, and scholarship for conference.

Finally, I would also like to thank my parents and my brother. They were always supporting me and encouraging me with their best wishes.

1

Review of Electronic Systems Implementation in Field Programmable Gate Arrays

In this chapter we give an overview of the aspects for efficient electronic systems implementation on FPGA. Firstly, we will cover the specifics of FPGA architecture and the most essential design steps and tools in order to find out an appropriate one for efficient design implementation. Afterwards, we will go through overview of the artificial neural network (ANN) and they hardware implementation issues. The specifics of neuron structure and training are overviewed in order to select suitable one for speech recognition rate enhancement. The advantages of high and low level approaches for ANN implementation are presented. Lastly, the aspects for speech recognition implementation in FPGA are described with the emphasis on features extraction and classification methods. At the end of this chapter the tasks for future work are formulated.

1.1. Computer-Aided Design for Implementation in Field Programmable Gate Array

In the last two decades the FPGA became very popular between engineers and scientists because of the opportunity to accelerate the traditional CPU-based algorithms. This speed-up would be impossible without the appropriate digital circuit design tools usually called as computer-aided design (CAD) tools for FPGA. These tools are constantly improved together with more complex architecture of FPGA, as well as considering the demand of the programmers (Chen

et al. 2006). From the view of the consumers the FPGA developing suite must be friendly, easily understandable and quickly verifiable. The main practical purpose of the FPGA chips is a relatively short developing time comparing to the application specific integrated circuit (ASIC) devices. Despite that ASICs utilizes 20–30 times less area, works 3–4 times faster and consumes approximately 10 times less power, the FPGAs have advantages in cost, reconfiguration and time to market (Kuon, Rose 2007).

Usually the CAD flow has a generalized and mostly vendor independent structure, as shown in Fig. 1.1 (Czajkowski 2008). It is used for generation of the uploadable bitstream for FPGA configuration.

Nowadays the high level synthesis (HLS) contains the implementation of algorithm in a fast and consumer convenient way using tools, that works at a high level of abstraction, e.g., Matlab, Simulink (Alecsa *et al.* 2012; Arias-Garcia *et al.* 2013), LabView (Laird *et al.* 2013; Ursutiu *et al.* 2013). This approach does not require a deep knowledge about FPGA specifics and make the test of desired part of the system easier. The general-purpose behavior description languages, e.g., C or SystemC can describe the design, however not allows to evaluate the cycle-accurate behavior (Chen *et al.* 2006).

The more tradition way to design the logic circuit in a register transfer level is the application of hardware description language (HDL), e.g., Verilog or VHDL. These languages describe the behavior of the logic circuit at each clock cycle applying strictly defined templates of logic, arithmetic and memory blocks. At the circuit synthesis stage the circuitry at Register Transfer Level (RTL) is transformed into a netlist. The netlist stores an information about the

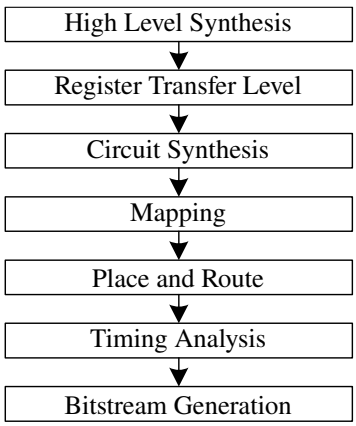


Fig. 1.1. The implementation stages of bitstream for field programmable gate array

description of logic gates and their interconnections. At the mapping stage one or more netlists are combined into groups with the aim to fit the design efficiently in the look-up tables (LUT). The mapper needs information about the FPGA target architecture. At the place and route stage the mapped components are assigned to the physical LUT, DSP, RAM resources at various locations in the FPGA. These resources are routed together using available routing wires and switching matrices. The optimization of LUT location takes the most time because it must satisfy the user defined timing constraints and placement requirements. The bitstream generation starts after the successful done of the timing and placement. The final FPGA programming file contains the connection settings for the programmable switch matrix and the true-tables, that will be loaded in the RAMs and LUTs (Vandenbout 2013).

The HLS and RTL are mainly used by researchers and engineers for the circuit implementation on FPGA. Any high level tool accelerates the design process, however it not allows to deny or skip the RTL. Moreover, comparing to other levels the reasonable performance of the design is achievable working in RTL (Misra, Saha 2010; Ronak, Fahmy 2014). The deeper we come in the implementation chain, the more information about certain FPGA internal architecture and configurable block interconnection must be known. Any editing of placed and routed design at the lower level is hardly realizable, since the control options are hidden by FPGA manufacturer. Furthermore, designing at the lowest possible level do not ensures optimal performance. The known RapidSmith tool works at place and route (PAR) level and is based on already mapped hard macro primitives placement (Lavin *et al.* 2013). It reduces the compile time by 50 times, however it 3 times slows down a clock rate of final design. Therefore, with respect to above statements, we prefer that created compiler implements an efficient circuits at RTL. The FPGA primitives essential for LLMLP implementation in RTL are discussed below.

1.1.1. Specifics of Field Programmable Gate Array Architectures

Any FPGA is composed of a finite number of predefined resources with programmable interconnections. There can be distinguished three main groups of resources: arithmetic, logic and memory. The LLMLP is most hungry for arithmetic resources, as is shown in Table 1.4. The dedicated DSP slices are used for a fast arithmetic operation implementation. DSP slices can be dynamically reconfigured to process a piece of the equation suitable for instantaneous DSP configuration. The DSP may accept the configuration according to the equation:

$$\mathtt{P} = \mathtt{C}|\mathtt{P} \pm (\mathtt{D} \pm \mathtt{A}) \times \mathtt{B}, \tag{1.1}$$

where A, B, C, D are the values of the data at the DSP input; P is the data value on the output of DSP; C|P means that data from input C or previous output value P is used.

The DSP is purposely designed for pipelined data processing with maximum performance. This is achieved inserting flip-flops (storage elements) and minimizing critical path in a circuit, as is shown in Fig. 1.2. Sequentially connected flip-flops introduces latency of 4 clock cycles in the dataflow processing independent on the selected instantaneous DSP configuration.

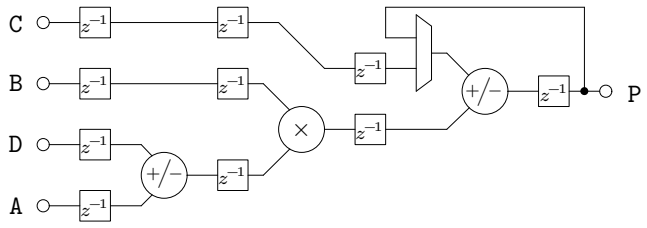


Fig. 1.2. The simplified structure of digital signal processing slice in 7th series field programmable gate array

If the DSP pattern do not contains any multiplication operation, then it can be directly instantiated in the RTL. It is efficient only for patterns with addition and subtraction, because of single clock cycle latency for addition/subtraction operations instead of 4 clock cycles latency for any kind of pattern implemented on DSP. List of the useful DSP patterns for LLMLP implementation is shown in Table 1.1.

The logic resources on the FPGA can perform logic functions. Logic resources are grouped in slices to create configurable logic blocks. Dependent on the FPGA manufacturer each slice contains various number of basic elements like *k*-input look-up tables, storage elements – flip-flops (FF), multiplexers, arithmetic and carry chain, as is shown in simplified slice structure in Fig. 1.3. There are two types of slices called SLICEM and SLICEL (Xilinx 2012a). Only SLICEM supports additional configurations to shift registers and distributed RAM.

Table 1.1. List of the pattern for digital signal processing slice configuration

DSP configuration	DSP configuration or direct instantiation
$C\|P \pm (D \pm A) \times B$	$C\|P \pm (D \pm A)$
$C\|P \pm D \times B$	$C\|P \pm D$
$(D \pm A) \times B$	$D \pm A$
$D \times B$	

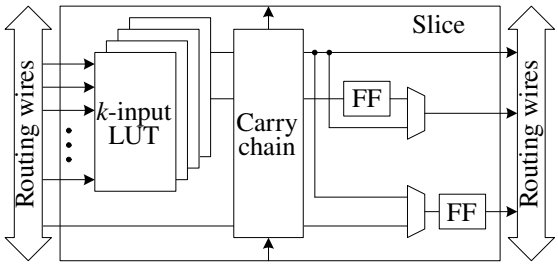


Fig. 1.3. The reconfigurable slice structure with look-up tables, carry chain and flip-flops in field programmable gate array

Due to the DSP sharing the intermediate calculation results must be stored in FPGA for fast access. To form small distributed memories in the latest FPGA each LUT in the SLICEM can be configured to the distributed 64 b RAM or ROM. All four LUT in a single SLICEM can form 256 b memory formatted in various ways as single, dual or quad port RAM with asynchronous or registered output. To create a large buffers, the block RAM (BRAM) is used. BRAM can be configured as single or dual port RAM or ROM (Fig. 1.4).

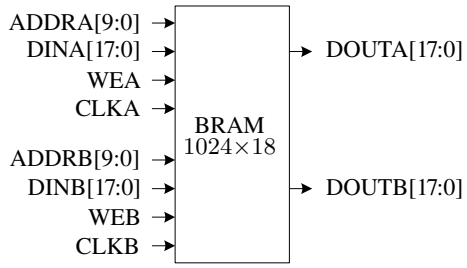


Fig. 1.4. The interface of the configurable dual port block random access memory in field programmable gate array

The modern Xilinx FPGAs contains a hundreds of BRAM each 1024×18 b size (Xilinx 2013d). Such blocks will be used to store synapse weights, forward/backward signals and trigonometric function values.

1.1.2. High Level Tools for Hardware Description Language Generation

One of the main challenges in designs efficient implementation in FPGA is the lack of the direct connection between algorithm design and hardware implementation. There can be hard to verify operation on chip against algorithm

specifications as hand coded HDL can bb error prone and hard to debug. There can also be differences between a fixed-point implementation and a floating-point specification of the algorithm. With different elements of the tool chain often sourced from different vendors requiring multiple purchase request which can be difficult to budget. To address these challenges hardware designers often looks at model-based design like Matlab and Simulink. HDL Coder allows to convert the Simulink models and Matlab code (written in embedded Matlab form with restrictions) to HDL synthesizable code and testbenches (MathWorks 2013). Better performance is achieved using Xilinx System Generator (XSG) for DSP (Xilinx 2013b), because each block is a pre-optimized IP core for Xilinx FPGAs. However, the cycle accurate scheduling nor resource sharing is hard to apply with XSG. Moreover, the XSG generated HDL code is rather long and complex (Van Beeck *et al.* 2010).

The StateCAD allows to design of the state machines through graphical interface automatically generating synthesizable HDL code direct from the diagram (Xilinx 2007). Despite that StateCAD reduces product development cost, this tool was removed since ISE Design Suite 11 version, because lack of popularity and support only on Windows platform.

The Matlab AccelDSP tool was used to map high-level described application to DSP slices for Xilinx FPGAs. It allows to generate synthesizable HDL source with automated float to fixed-point conversion and testbench verification (Xilinx 2012b). Unfortunately, AccelDSP synthesis tool has been discontinued and is no longer available.

The commercial HLS tools C-to-Silicon, Catapult C, Synphony or Vivado HLS take a software description and turn it into effective hardware. Their objective is to generate hardware whose performance rivals with hand-written designs. The advantage is that same design described in software can be reused for different FPGA (Cadence 2013). However, these tools have limitations when transforming sequential software to hardware. The high performance is achievable only for easy-to-pipeline algorithms with regular structure trivial filters. As a proof, the favourite example is still the FIR filter (Xilinx 2015b). The performance of more irregular design with feedback loops and dynamic memory allocations is rather poor.

First four HLS tools in Table 1.2 are model-based. It means, that design is assembled by hand from pre-compiled blocks and each of them has straight defined reflection in the following RTL code. The required blocks must be manually entered into the design. Therefore, designer must still worry about proper sequence of the block and compatibility of data type.

The HLS tools are very useful for quickly verification and validation of the algorithm in a hardware, when the design time is prior to the final design performance. But till nowadays, for a greater design optimization we have to

Table 1.2. Comparison of high level synthesis tools

Tool	Programming way/language	Generated code
HDL Coder	Blocks	RTL
XSG	Blocks/Embedded-M	RTL
LabView	Blocks	RTL
AccelDSP	Blocks	RTL
StateCAD	State diagrams	RTL/Abel
C-to-Silicon	C/C++/SystemC	RTL
Catapult C	C	RTL
Synphony	C	RTL
Vivado HLS	C/C++	RTL/SystemC

hand code, despite the large choice of HLS tools, which significantly reduces FPGA design time at the cost of increasing latency or resource utilization. The main advantages of the HLS design tools: fast implementation and verification, user friendly. Disadvantage: weak performance control, hard to meet of the timing or resource constraints, complicated code readability.

Regarding the mentioned disadvantages the researchers are developing a custom HLS tool for a specific design implementation mainly based on data and control flow graphs exploiting the design transformations (pipelining, retiming, folding) for circuit performance optimization (Cong, Jiang 2008; Kazanavičius, Venteris 2000; Ronak, Fahmy 2012). The created tools are used for efficient mapping of mathematical expressions into DSP slices (Ronak, Fahmy 2014), mapping the Kahn processor network into DSP and FPGA architecture (Žvironas, Kazanavičius 2006) or targeting multiple application domains (Cong *et al.* 2011). Therefore, in the next section we will go through the specifics of artificial neural network implementation on FPGA with the aim to distinguish suitable structures for further application in speech recognition.

1.2. Implementation of Artificial Neural Networks in Field Programmable Gate Array

Speech recognition systems work better if they are allowed to adapt to a new speaker, the environment is quiet, and the user speaks relatively carefully. Any deviation from these conditions will result in significantly increased errors. In most of the practical applications the speech is corrupted by a background noise (Chan *et al.* 2013; Yiu *et al.* 2014). This strongly degrades the accuracy of speech recognizers and makes it unpractical to use in applications that are

working in real conditions. Therefore, to make the FPGA-based recognizer more robust, many methods for noise cancellation are proposed, e.g., independent component analysis (Kim *et al.* 2003), spectral subtraction (Firdauzi *et al.* 2013), adaptive filtering (Hadei, Lotfizad 2011; Thilagam, Karthigaikumar 2015) or artificial neural networks (ANN) (Bosque *et al.* 2014; Er *et al.* 2005).

The adaptive filters in conjunction with multilayer perceptron (MLP) seems to be attractive in speech enhancement application (Navakauskas 1999). Such MLP can have finite impulse response (FIR), infinite impulse response (IIR) or lattice-ladder (LL) filters in a place of the neuron weights. These filters can be employed for speech signal preprocessing and adaptive noise reduction. The filter coefficients updating algorithm is well known and tested on PC applications. But since the IIR MLP (Back, Tsoi 1993) or lattice-ladder multilayer perceptron (LLMLP) (Navakauskas 2003) was firstly proposed, till nowadays they are not implemented on FPGA. Therefore, our interest in this thesis is the efficient LLMLP implementation on FPGA for the improvement of speech recognition. As LLMLP has same hardware implementation specifics as other ANN, thus the basic principles for mapping ANN algorithms to hardware structures are described below.

The FPGA has a suitable structure for pipelined and parallel ANN implementation. The main advantages of mapping ANN to FPGA are following (Misra, Saha 2010):

- The hardware offers very high computational speed at limited price.
- Provides reduced system cost by lowering the total component count and decreasing power requirements.
- The parallel and distributed architectures allow applications to continue functioning with slightly reduced performance even in the presence of faults in some components.

To obtain successful implementation of ANN, it is essential to have an understanding of the properties of the ANN circuit to be implemented. For example, sensitivity to coefficient errors affects on signal quality, the final arithmetic behavior affects the robustness of the algorithm and thereby its usefulness. The computational properties of the ANN are also important, e.g., parallelism and maximal sample rate affects the performance and implementation cost. These and other important specifics for efficient ANN implementation in FPGA are considered in following subsections.

1.2.1. Specifics of Artificial Neural Networks

Because of limited hardware resources the implementable ANN size and speed highly depends on efficient implementation of the single neuron (Muthurama-

lingam *et al.* 2008). Therefore, the implementation of a large network on FPGA with resource and speed trade-off is a challenging task. The main problem is in the replacing of floating-point numbers to the fixed-point. The input data must be normalized in a range from 1 to -1 with restricted number of levels. Therefore, the limited accuracy plays significant role. The minimum 16 b for weights and 8 b for activation function inputs is good enough for applications with feed-forward ANN (Holt, Hwang 1993). The higher precision gives lower quantization error. While higher speed, lower area and power consumption are available with lower bit precision. One of the disadvantage is a saturation, because the range of weight$\times$data product has fixed length (Moussa *et al.* 2006). Before implementation of any type of ANN in FPGA the issues presented in Fig. 1.5 must be solved.

Precision. The recurrent part in the LLF alerts about growing error if the number of bits assigned to the signal is too low. Using the Xilinx IP core generator the maximum 64 b signal width is available. 16 b width connections in feed-forward pass of ANN are acceptable. The ANN with feed-back loops requires rather wide, i.e., 24–32 b, width signals and depends on accuracy demand for specific application.

Latency. The performance of whole ANN depends on the combinational and sequential paths delay of the longest route in the synthesized circuit. The bad HDL practice as asynchronous resets, adder threes instead of adder chains, incorrect synthesis tool settings, gated clocks, high number of nests within sequential statements has become a bottleneck in aspiration of maximum frequency. The pipelined implementation gives the possibility to put a data to the input of ANN synchronously at each clock. It is not always possible to implement the ANN with feed-back wires in a pipelined way. Using of the variables helps to merge a long arithmetic chain to easily be completed in a single clock cycle. But the longer the unregistered path in a chain, the lower the maximum frequency in a final system. For the reasonable design at least 200–300 MHz frequency is expected and is possible with some serious scrutiny.

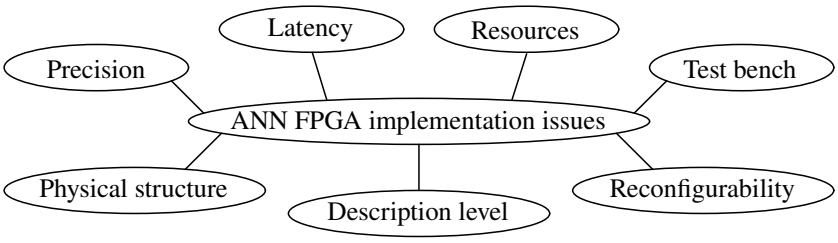


Fig. 1.5. The hardware implementation issues of artificial neural network

Resources. The ANN requires a lot of arithmetic operations like summation, multiplication and activation function stored in ROM. One DSP slice performs a multiplication of 18×25 b signals. The increase of precision requires higher DSPs utilization rate, e.g., 32 b data multiplication utilizes 4 DSP, 64 b – 16 DSPs. It is possible to use multipliers created from logic cells. In such case 32×32 b multiplier uses 1088 LUT6 and it is more than 2 % of total number of LUTs in e.g., ZynQ-7000 family xc7z020 FPGA. The limited number of resources makes it impossible to physically implement large ANN. In example, 10-th order LLF utilizes all DSP slices in ZynQ-7000 Artix-7 FPGA. Therefore, physically implemented small part of ANN must share the resources to create larger network. Since the hardware complexity of a multiplier is proportional to the square of the word-length, a filter with more multipliers and smaller word-length can sometimes be implemented with less hardware, than a filter with less multipliers and larger word-length. Therefore, the hardware complexity of a digital filter should be determined carefully depending upon the frequency specifications for LLF (Chung, Parhi 1996).

Test bench. To test the accuracy of the ANN various test bench files must be evaluated on the simulations first of all. It is important to check the network precision and weight dependence on different parameters as input signal, LLF order, number of layers, bit width of the data. All the results must be compared with the floating point precision ANN to get a relative error.

Physical structure. Limited resources do not allows physically implement whole network. For example, the M-th order LLF with their learning algorithm utilizes at least $36\times M + 4$ DSP slices if 32 b signals are used. Therefore, it is more convenient to have few physical neurons and share them through larger virtual ANN (Misra, Saha 2010).

Description level. It is considered that VHDL and Verilog are the main hardware description languages. They describe a circuit at low level, at that time FPGA primitives are manually arranged in a design. The HDL gives us full performance and resource control. For inexperienced designers or for those, who want to implement design quickly, there are a lot of high level synthesis tools, that will be described in next section. The HLS accelerates FPGA design and verification time, but it hides the options to precisely manage the resource allocation and timing constraints.

Reconfigurability. The partial reconfiguration allows the modification of an operating FPGA design by loading partial reconfiguration file, thus modifying ANN topology (Finker *et al.* 2013). The reconfiguration time is directly related to the size of partial configuration file and usually takes less than a second (Xilinx 2014b). We are interested not in the reconfigurability based on supplementary file loading to FPGA, however on synchronous reconfiguration of DSP for new operation.

1.2.2. Artificial Neuron Structures

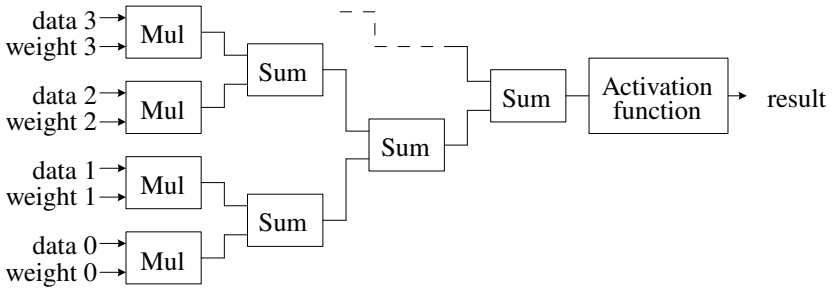
The artificial neuron is a mathematical model of biological neuron. The body of biological neuron is based on soma, which acts as the summation function. The input signals arrive in the soma from the dendrites (synapses). The output of the artificial neuron is analogous to the axon of biological neuron. Each time when the soma reaches a certain potential threshold, the axon transmits output signal to synapses of other neurons. Therefore, the axon is analogous to the activation function in artificial neuron.

There are two different neuron configuration in a hardware (Omondi *et al.* 2006): tree (Fig. 1.6a) and chain (Fig. 1.6b). The trade-off is between processing latency and efficient use of logic resources. With a tree configuration the use of slice logic is quite uneven and less efficient than with a chain. If number of inputs is large, then a combination of these two approaches is used. Such a reservation of a single DSP for one synapse allows to implement parallel ANN structures, which size is highly limited by FPGA internal interconnection resources. The number of available synapses is also restricted by the number of DSP. In Fig. 1.6 presented neuron is suitable for small and fast ANN structures with a pipelined input. The activation functions can be implemented as polynomial approximations, CORDIC algorithm, table-driven method. For hardware implementation the trade-off between accuracy, performance and cost are all important.

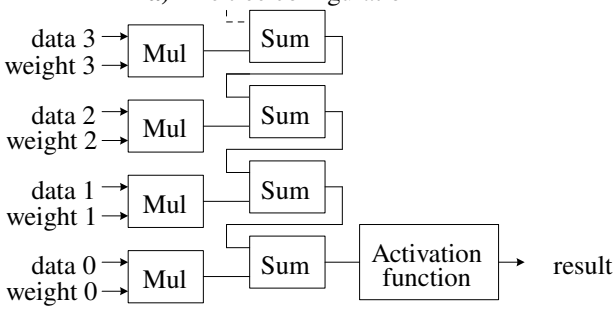
It is often unnecessary to have such ANN, that captures data at each rising edge of clock. Therefore, it is possible to use one multiplication (DSP slice) to compute the product and summation of each weight$\times$data operation in a sequential manner (Fig. 1.7). In this case, the latency is proportional to the number of synapses in each neuron. Such a model of neuron is popular in a hardware implementation of the feed-forward neural network, i.e., MLP (Ferreira *et al.* 2007; Muthuramalingam *et al.* 2008). The structure of MLP with parallel neural unit that calculates 32 products at each clock cycle is proposed (Lotrič, Bulić 2011).

The precision of the activation functions plays a significant role especially if the number of hidden layer in the MLP is high. It is possible to implement the sigmoid function in a simplified manner replacing e^{-x} with 1.4426×2^{-x} and splitting the x into whole number and fractional part (Muthuramalingam *et al.* 2008). Such approach needs a 100 cycles to implement an activation function. Therefore, the look-up tables are proposed to use as a faster way for nonlinearity estimation.

An important problem for designers of FPGA-based ANN is a proper selection of the ANN model suitable for limited hardware resources. A comparative analysis of hardware requirement for implementing MLP in Simulink and



a) The tree configuration



b) The chain configuration

Fig. 1.6. Two distinct implementation of the neuron: the tree a) and chain b) configurations

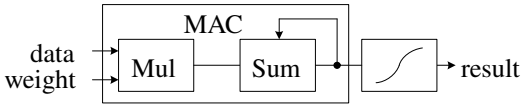


Fig. 1.7. Digital neuron model based on multiply-accumulate (MAC) unit

translating to hardware indicates that the not complex spiking neural network (SNN) is a most appropriate model for non-linear task solving in a FPGA (Johnston *et al.* 2005). The reason is a type of neuron. The spiking neuron do not uses multipiers in a synapses, however only addition and threshold operations are required. Any FPGA always has more configurable logic blocks with summation and comparison operators than DSPs. Therefore, SNN is more area efficient and relatively easier to construct than any other ANN. However, it is much harder to develop a model of SNN with stable behavior that computes a specific function (Carrillo *et al.* 2012). Moreover, current FPGA routing structures cannot accommodate the high levels of interneuron connectivity inherent in complex SNNs (Harkin *et al.* 2009).

1.2.3. Dynamic Neuron

Regardless of the type of ANN, the synapse of a neuron always plays a meaning role, since its ability to adapt. For the processing of a time-varying signals usually it is useful to replace a single weight synapse with a band-pass filter of the desired order. Therefore, associated dynamic neuron training algorithm becomes more sophisticated especially if infinite impulse response filters are used. When the direct structure of such filter is selected the number of parameters to be adapted becomes proportional to $2M + 1$, where M is the filter order (usually M is set the same for the synapses of particular layer). However the main disadvantage of such dynamic neuron training implementation lies in the elaborated control of filter stability during the learning process. Thus the rearrangement of the filter to the lattice-ladder structure with the simplest stability check (Regalia 1995) is appealing for the implementation.

The presented neurons (Fig. 1.6 and 1.7) will be dynamic if the synapses are replaced by filters. One of the possible filter is the LLF presented in Fig. 1.9. Each section of the LLF consists of a normalized lattice autoregressive-moving average filter. The local flow of information in the lattice part of filter for j section is defined by:

$$\begin{bmatrix} f_{j-1}(n) \\ b_j(n) \end{bmatrix} = \begin{bmatrix} \cos\Theta_j & -\sin\Theta_j \\ \sin\Theta_j & \cos\Theta_j \end{bmatrix} \begin{bmatrix} f_j(n) \\ z^{-1}b_{j-1}(n) \end{bmatrix}; \tag{1.2}$$

$$s_{\text{out}}(n) = \Phi\left(\sum_{j=0}^{M} b_j(n)v_j\right), \tag{1.3}$$

with initial and boundary conditions $b_0(n) = f_0(n)$ and $f_M(n) = s_{\text{in}}(n)$, where $s_{\text{in}}(n)$ – input signal; $v_j(n)$ – weights of the ladder; Θ_j – rotation angles of the lattice; $f_j(n)$ and $b_j(n)$ – forward and backward signals of the lattice, correspondingly; $s(n)$ – the output of the LLN with N inputs and Mth order LLF as its synapses; $\Phi\{\cdot\}$ – neuron activation function.

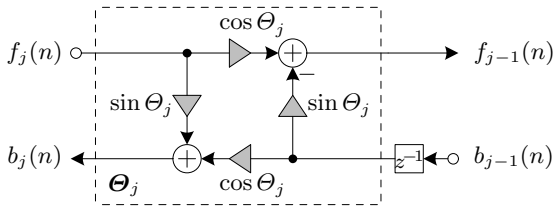


Fig. 1.8. The structure of normalized lattice section used in Fig. 1.9 presented lattice-ladder filter

In a time-varying environment the normalized lattice filter is superior over other lattice forms with one, two or three multipliers (Gray, Markel 1975). Contrary each normalized lattice section requires four times more multiplication operations when compared to a single multiplier lattice for the cost of stability. It could be shown that equality:

$$\left|f_{j-1}(n) + ib_j(n)\right| = \left|f_j(n) + ib_{j-1}(n-1)\right|, \tag{1.4}$$

holds for all j sections (Gray, Markel 1975). Therefore, the normalized lattice filter requires only the starting point to be stable and any truncation of the filter coefficient don't moves poles outside the unit circle. Thus there is no need to check lattice stability for normalized lattice filter.

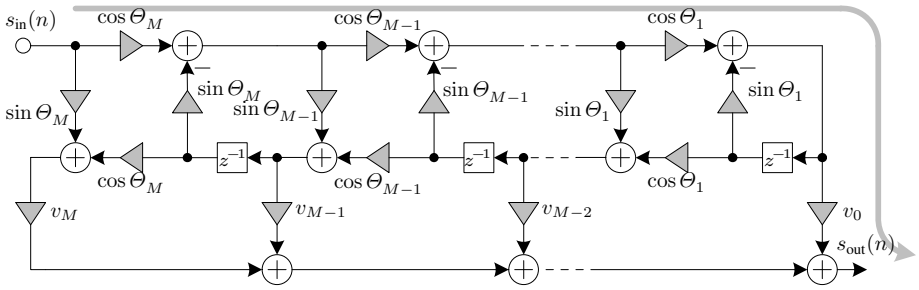


Fig. 1.9. The critical path in the M-th order lattice-ladder filter

The unregistered critical path in the LLF is marked by a gray color in Fig. 1.9. In the LLF the computation time delay in critical path is proportional to the filter order M and equal to $(M+1)\times 2$. The longer the critical path, the higher delay is between the synapse input $s_{\text{in}}(n)$ and the synapse output $s_{\text{out}}(n)$. If this filter will be implemented as is, its maximal clock frequency degrades drastically, because of the growing filter order M. In order to achieve higher performance the LLF can be pipelined through retiming and interleaving procedures (Chung, Parhi 2012; Parhi 2013). Such a circuit manipulation procedures are indeed helpful for pipelining a design without feedbacks, if there is no input $s_{\text{in}}(n) = f\big(s_{\text{out}}(n)\big)$ dependence on the output $s_{\text{out}}(n)$. The Mth order LLFs (Fig. 1.9) are used as synapses in artificial neurons. The neurons distributed in parallel construct a single layer of MLP presented in Fig. 1.10. The MLP consists of multiple layers, with each layer fully connected to the next one. A lattice-ladder multilayer perceptron (LLMLP) is defined by number of layers L, number of neurons in each layer $\{N^{(0)}, N^{(1)}, \ldots, N^{(L)}\}$ and filter orders $\{M^{(0)}, M^{(1)}, \ldots, M^{(L)}\}$. The input layer denotes the number of features/dimensions in the input signals. The number of outputs in LLMLP

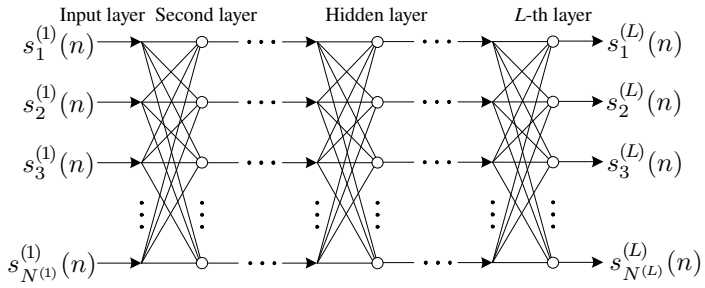


Fig. 1.10. Representation of multilayer perceptron

is directly related with number of neurons in the last Lth output layer. All the layers arranged between input and output layers are hidden.

1.2.4. Training of Dynamic Neuron

Development of the lattice version of the gradient descent algorithm follows the same methodology as for the direct form. The output error δ is differentiated with respect to the filter parameters $\sin\Theta_M$, $\cos\Theta_M$, to obtain negative gradient signals ∇_{Θ_M}. The main advantage of the lattice form over the direct form is related with LLF stability. The LLF is inherently stable for time-varying signals (due to the fact that $\sin\Theta_M \leq 1$, $\cos\Theta_M \leq 1$), while the direct form is not (Regalia 1995). Synapse presented in Fig. 1.9 has to learn Θ_j and v_j parameters using simplified gradient descent training algorithm, when weight updates are expressed:

$$\Theta_j(n+1) = \Theta_j(n) + \mu\delta\nabla_{\Theta_j}(n); \tag{1.5}$$

$$v_j(n+1) = v_j(n) + \mu\delta b_j(n); \tag{1.6}$$

$$\delta = \big(1 - s_{\text{out}}(n)^2\big)e(n), \tag{1.7}$$

here μ – a training step; δ – an instantaneous output error; $e(n) = s^{\star}_{\text{out}}(n) - s_{\text{out}}(n)$ – error between desired $s^{\star}_{\text{out}}(n)$ and output signal $s_{\text{out}}(n)$; $\nabla_{\Theta_{i,j}}$ – gradients of lattice weights; $b_j(n)$ – backward signal inside of lattice.

There are in total 14 possible ways how to exactly implement order recursion required for gradient training algorithm (Navakauskas 2003). And only four of them (t_3, t_4, t_{11}, t_{12}) have a straightforward implementation, as is shown in Fig. 1.11 and Fig. 1.12. That is justified by the fact that the other ten remaining cases have one or several un-implementable advanced operators. The dashed boxes separates the "main order" calculations from the calculations involved with the delays and additional signals going from lattice part of the LLF. The gradients ∇_{Θ_j} of lattice weights are estimated by cir-

cuits presented in Fig. 1.13–1.16 for four useful cases of the order recursions $\mathfrak{t}_i \ \forall \ i \in \{3, 4, 11, 12\}$. Each regressor lattice is formed cascading M sections of the order recursion blocks and taking in account boundary conditions. Gradients ∇_{Θ_M} for lattice parameters are calculated using extra addition and multiplication with constant $\cos^{-1}\Theta_M$ operations.

The selection of certain regressor lattice (Fig. 1.11, Fig. 1.12) for practical implementation influences the amount of operations. The number of arithmetic and trigonometric operations required to train the LLMLP can be expressed:

$$N_{\Pi} = \sum_{l=1}^{L} N^{(l)} N^{(l-1)} \left(5M^{(l)} + 2 + N_{\Pi}^{\star(l)}\right) + 2\sum_{l=1}^{L-1} N^{(l)} + \sum_{l=2}^{L} N^{(l)} N^{(l-1)} N_{\Pi}^{\star(l)}, \tag{1.8}$$

$$N_{\Sigma} = \sum_{l=1}^{L} N^{(l)} N^{(l-1)} \left(9M^{(l)} + 3 + N_{\Sigma}^{\star(l)}\right) + \sum_{l=1}^{L-1} N^{(l)} N^{(l+1)} + \sum_{l=2}^{L} N^{(l)} N^{(l-1)} N_{\Sigma}^{\star(l)} + N^{(L)}, \tag{1.9}$$

$$N_{\Phi} = 3\sum_{l=1}^{L} N^{(l)} N^{(l-1)} M^{(l)} + \sum_{l=1}^{L-1} N^{(l)}, \tag{1.10}$$

here N_{Π}, N_{Σ}, N_{Φ} are the numbers of multiplication, addition and trigonometric operation; $N^{(l)}$ – number of neurons in lth layer; $M^{(l)}$ – the LLF order in lth layer; L – total number of layers; $N_{\Pi}^{\star(l)}$ – number of multiplications in a regres-

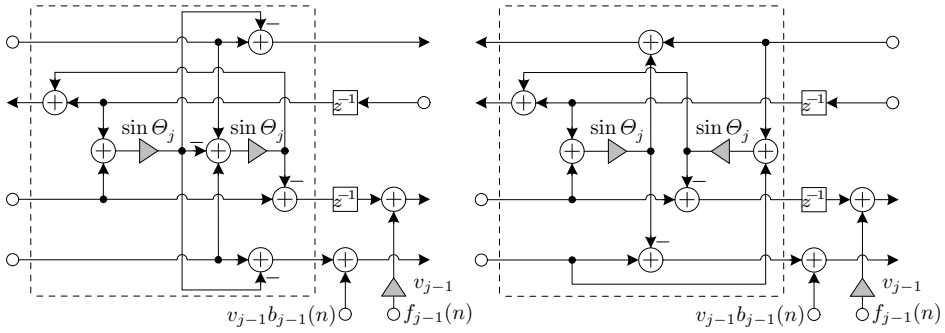


Fig. 1.11. Simplified order recursions of type: a) – $\mathfrak{t}_3$ and b) – $\mathfrak{t}_4$

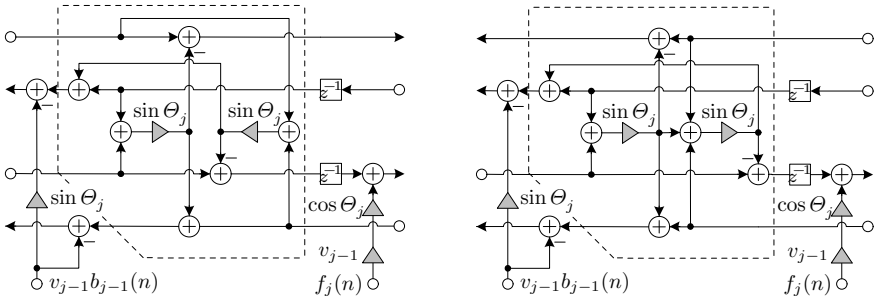


Fig. 1.12. Simplified order recursions of type: a) – $\mathfrak{t}_{11}$ and b) – $\mathfrak{t}_{12}$

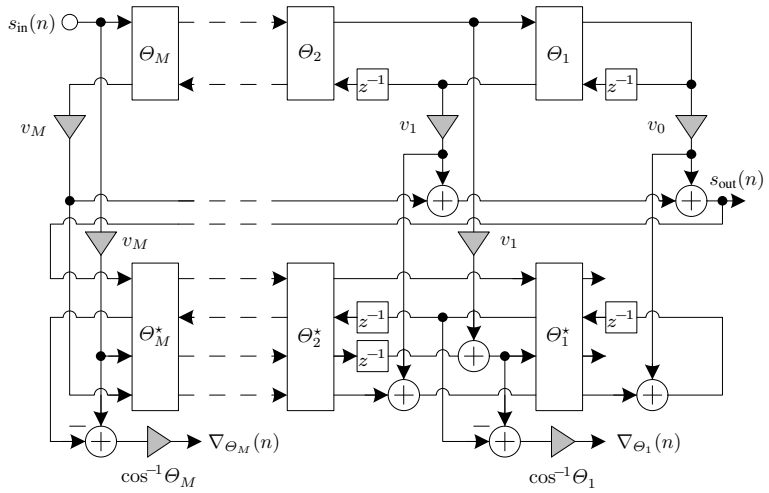


Fig. 1.13. The connected circuit of lattice-ladder filter and $\mathfrak{t}_3$ type regressor lattice for $\nabla_{\Theta_j}(n)$ gradients estimation

sor in the lth layer; $N_{\Sigma}^{\star(l)}$ – number of additions in a regressor in the lth layer. Both parameters $N_{\Pi}^{\star(l)}$ and $N_{\Sigma}^{\star(l)}$ depend on the regressor type as summarized in Table 1.3. Despite that regressor lattices $\mathfrak{t}_3$ and $\mathfrak{t}_4$ need $\sin\Theta_j$, $\cos^{-1}\Theta_j$ and alike $\mathfrak{t}_{11}$ and $\mathfrak{t}_{12}$ need $\sin\Theta_j$, $\cos\Theta_j$, $\cos^{-1}\Theta_j$ trigonometric functions to be calculated for each order recursion, the total number of trigonometric operations O_{Φ} is same for all regressor lattice types $\mathfrak{t}_i$ (last row in Table 1.3). Each additional order recursion in $\mathfrak{t}_i$ requires only $\cos^{-1}\Theta_j$ calculation, since $\sin\Theta_j$, $\cos\Theta_j$ are computed in primary lattice and results are shared with regressor lattice. The total number of operation necessary for LLMLP implementation in conjunction with weight adaptation is summarized in Table 1.4.

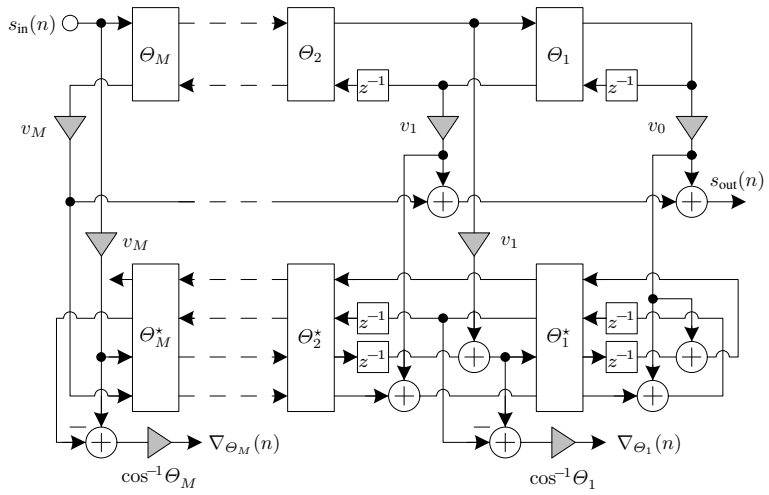


Fig. 1.14. The connected circuit of lattice-ladder filter and type t_4 regressor lattice for $\nabla_{\Theta_j}(n)$ gradients estimation

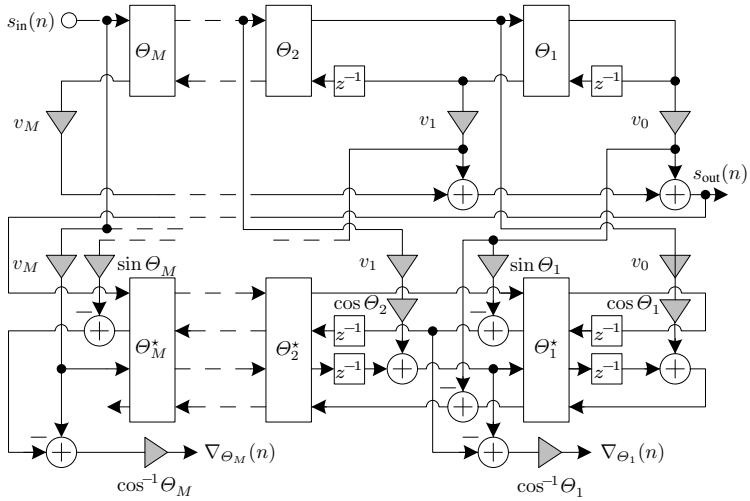


Fig. 1.15. The connected circuit of lattice-ladder filter and type t_{11} regressor lattice for $\nabla_{\Theta_j}(n)$ gradients estimation

The newest FPGA contains up to 3600 DSP blocks dedicated for multiplication and addition (Xilinx 2014a). The parallel implementation of large LLMLP is restricted by limited arithmetic resources. Therefore, the LLMLP with only few neurons and low order LLF can be implemented in single FPGA

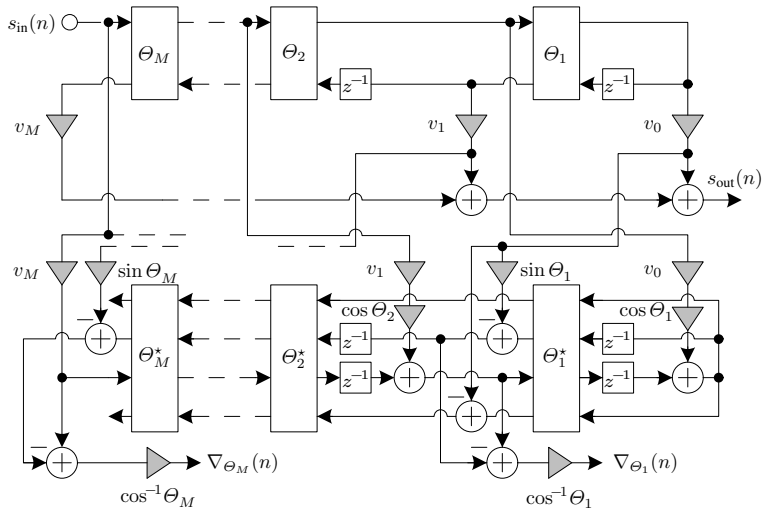


Fig. 1.16. The connected circuit of lattice-ladder filter and type t_{12} regressor lattice for $\nabla_{\Theta_j}(n)$ gradients estimation

Table 1.3. The total number of operations for lattice-ladder filter and regressor

Mathematical calculations	**Single LLF**	**Single regressor of type**			
		t_3	t_4	t_{11}	t_{12}
Multiplications, $N_\Pi^\star$	$5M+1$	$4M$	$4M$	$6M+1$	$6M$
Additions, $N_\Sigma^\star$	$3M$	$10M-1$	$9M-1$	$10M-2$	$11M-3$
Trig. functions, $N_\Phi^\star$	$2M$	$1M$	$1M$	$1M$	$1M$

Table 1.4. The total number of operations for different lattice-ladder multilayer perceptron sizes

$N^{(l)}$	$M^{(l)}$	N_Π	N_Σ	N_Φ
2-2	2	220	230	48
1-10-1	4-6	1620	1961	310
2-10-1	4-6	2170	2531	430
1-10-10-1	2-4-6	8480	10981	1460
5-10-10-1	2-4-6	9640	12141	1700

without resource sharing. Obviously the DSP slice sharing through time is necessary for large LLMLP implementation. To solve the issues of large ANN implementation on restricted FPGA, the researchers invoke the high or low level approaches.

1.2.5. Indirect (High Level) Approach

The implementation of desired ANN in a high level is a fastest way from the programmer point of view to code the ANN equations and take less care about physical ANN structure. Many researchers applied their own written tools in C, Matlab or others high-level languages to implement desired ANN on FPGA in a convenient way. In most cases the tools generate VHDL or RTL output files, that exactly describe the ANN in a lower level. Afterwards the FPGA vendors synthesis, mapping and PAR tools are used. It is popular to use Simulink tool in combination with Xilinx system generator as a high level description of ANN (Oniga *et al.* 2008). The advantage of Simulink application is a fast description time of ANN at the expense of final circuit performance.

A special program is developed in C++ to automatic generation of VHDL code of the ANN (Dinu *et al.* 2010), while each neuron is represented as special case of Boolean function. The parameters of ANN are feed to the C++ program as a matrices in text file. The Matlab is used to develop MLP VHDL IP core which can run a network with 2 hidden layers, 128 neurons and 31 inputs for each neuron (Rosado-Munoz *et al.* 2008). The generated VHDL files contains the *generate* instruction and *generic* parameters, which are customizable after generation of IP core. Every single neuron have a multiply-accumulate (MAC) structure (Fig. 1.7) and makes calculation in pipeline.

The support vector machine (SVM) IP core compiler was developed in a C# language to generate the VHDL files (Anguita *et al.* 2011). In a created GUI it is able to configure the SVM to desired precision, choose a number of support vectors and many others parameters. To solve the quadratic programming problem the sequential minimal optimization was used. The SVM uses fixed-point arithmetic, therefore can be implemented on resource-constrained hardware (Anguita *et al.* 2008). The SVM achieves 6 % classification rate while using 12 b for coefficient and kernel precision. The lowest 6 cycles latency is achieved using linear piecewise kernel architecture. The highest 15 cycles latency gives a CORDIC-like kernel.

The automatic VHDL files generator for self-organizing features map was developed in (Onoo *et al.* 2009). The network size problem is solved introducing physical and logical neuron concept when the size of desired neural network is larger than available FPGA resources (Yamamoto *et al.* 2011). The physical neurons sequentially executes the logical neurons. The advantage is in smaller circuit size, but the execution time is longer. The Simulink based approach for self-organizing features map with on-chip learning implementation was developed (Tisan, Cirstea 2012). Two DSP slices are hardwired to unique neuron, so the maximal network size is determined by available FPGA resources.

All the mentioned ANN implementation approaches reduce the design time, because network structure is described in a high-level language. But it is still unknown, whether the network has an optimal composition in the physical circuit. This issue is considered in next subsection.

1.2.6. Direct (Low Level) Approach

Towards the direct ANN implementation in FPGA at the RTL level the care must be taken firstly about limited arithmetic resources. Therefore, instead of implementing a complete network it is advisable to implement only the single largest layer. Afterwards, combine the entire ANN using the layer multiplexing and some control logic (Himavathi *et al.* 2007). Instead of dedicating a multiplication operator for each synapse, a DSP slice configured to MAC can process all the inputs of neuron in serial (Savich *et al.* 2007). The use of FPGA partial reconfigurable areas reduce resource usage by dynamically inserting and removing neuron blocks on the network and this way configure MLP with different topologies (Alberto de Albuquerque Silva *et al.* 2015).

If multiplication operation has to be implemented in DSP48E1 slices on Xilinx FPGA, then the maximum 25×18 b width input product can be achieved utilising single DSP slice Xilinx (2014a). The higher precision requirements can be satisfied in two ways: by time-multiplexing of a single DSP slice (however that will increase a total latency of the system) or by the use of two DSP slices (however that will consume DSP slices very fast).

One of the most important issue in the hardware-based ANN is the implementation of nonlinear activation functions such as sigmoid (Nambiar *et al.* 2014), logarithmic sigmoid (Vaitkus *et al.* 2014) or hyperbolic tangent (Bahoura 2014). Such functions have easily computed derivative, which is important for training process, because that decreases the computational load. However, the precise implementation of the nonlinearity in FPGA gives cause for concern. The reasonable solutions while solving this issue can be: combinational (Tommiska 2003; Zamanlooy, Mirhassani 2014), piecewise linear (PWL) (Armato *et al.* 2011; Ferreira *et al.* 2007) or quadratic (PWQ) approximations (del Campo *et al.* 2013; Nambiar *et al.* 2014), look-up tables (LUT) synthesized in a logic (Gomperts *et al.* 2011; Lotrič, Bulić 2012) or stored in on-chip memory (Bahoura 2014). The straightforward implementation of nonlinear function in hardware is not a correct approach, because both exponentiation and division are logic and arithmetic resource hungry operations (Gomperts *et al.* 2011; Tommiska 2003). The CORDIC algorithm introduces latency and has limited input domain (Armato *et al.* 2011).

Efficient FPGA implementation of activation function is a multi-criteria task. The balancing of the accuracy, resources and processing speed must be

considered (Armato *et al.* 2011). High precision requires more resource. Signal routing through high amount of logic is a bottleneck to achieve higher clock frequency for the synthesized design, due to the growing delay in a critical path of not compact hardware. Moreover, resource reduction makes circuit faster, but also influences inaccuracy in approximated activation function.

When only few bits are used as the input of activation function, then it makes sense to use combinational approximation based on direct bit level mapping, that doesn't requires any arithmetic operators. The bit level mapping method was proposed in (Tommiska 2003). It is shown that with 6 b precision the maximal absolute error of the activation function is less than 1 %.

The polynomial approximation methods are based on the function input range division into equal parts, where each function subinterval is approximated by line or curve (Armato *et al.* 2011). The PWL approximation of hyperbolic tangent with 256 linear sections provides a maximum error of 0.002 % using 32 b arithmetic (Ferreira *et al.* 2007). The controlled accuracy second order approximation of sigmoid function was implemented on single DSP slice with maximum allowable 1 % error (del Campo *et al.* 2013). The amount of additional logic resources changes depending on the required word-length and error. The PWQ approximation can be implemented by reusing already utilized multiplier and adder units in neuron block proposed in (Nambiar *et al.* 2014). The main disadvantage of polynomial approximation technique is the increased latency due to the growing number of multiplication operations for the higher-order polynomial. The maximum allowable 2 % error with 9 b input is achieved using hybrid PWL and LUT methods in (Meher 2010) implementing hyperbolic tangent function in hardware.

The LUT-based approach works much faster than polynomial approximation, since LUT consumes memory. Small LUT is usually implemented in distributed memory, which does not require delay units, but has a limited size (Bahoura 2014). Newest FPGA chips have at least 0.5 MB of on-chip block RAM (BRAM) with one clock cycle latency to access stored data (Xilinx 2013c). Therefore, large LUT is generally stored in BRAM. The accuracy of approximated hyperbolic tangent under various precision input signal and LUT size was investigated in (Gomperts *et al.* 2011). Depending on the application and required accuracy for the activation function, different sizes LUTs from 28 samples (Lotrič, Bulić 2012) to 215 samples (Bahoura 2014) are used.

The description in the literature of limits for accuracy of activation function is fragmented and it lacks commonly acceptable values. The average ϵ_{avg} and maximum $\epsilon_{\max}$ error values usually depend on the precision requirements for the application and in many cases maximum error 0.4 % (Sledevič, Navakauskas 2016), 1.5 % (Armato *et al.* 2011) or even 2 % (Meher 2010; Zamanlooy, Mirhassani 2014) is tolerable.

The more common way to implementing the nonlinear function in current FPGA-based neural network is to have a LUT. Using nowadays FPGA there is not much concern about random access memory, therefore LUT stored in BRAM is advantageous and will be used in our implementation.

1.2.7. Artificial Neural Network Chips

The neurochip always contains a weight×input multiplication and their summation blocks. The other blocks for neuron state, activation function, weighs loading controllers can be on or off chip. There are several categories of dedicated chip:

- *Bit-like* – first neurochips constructed from cheap building blocks (single neurons) with off-chip learning possibility and from few to dozens of neurons and synapses on single chip (Mauduit *et al.* 1992; MD1220 Neuro Bit Slice; NLX-420 Datasheet).
- *Single instruction multiple data* (SIMD) neurochip run the same instruction simultaneously on multiple processing elements (PE) with different data for each PE. The number of PEs in single chip varies from 16 to 64 (Hammerstrom 1990; Kim *et al.* 2005; Means, Lisenbee 1991).
- In the *systolic array* each PE process instructions synchronously with other PEs and the result from first layer is passed to each next layer in the pipelined manner. The array structure is well matched to ANN having potentially high utilization ratio of the processing unit because the multiplier in each synapse has maximal load (Amin *et al.* 1999; Chung *et al.* 1992; MA16 Technical Report). The main disadvantages of the systolic PE stucture are: the complex array interfacing and synchronous weight update control.

Different ANN design evaluation in a short time, reasonable cost and reduced hardware development cycle bring FPGA to effective programmable resources for ANN implementation. Partial and online reconfiguration provide software like flexibility and additional advantages. In contrast to VLSI the circuit density is still lower in the FPGA and it restricts the implementation of large scale ANN with thousands of neurons.

A potential issue with implementing a ANN in FPGA is the limited amount of routing resources (Misra, Saha 2010). Unlike the logic resources (flip-flops, look-up tables or memory blocks), routing resources are difficult to quantify. The growing ANN hits the routing limits very quickly. Place and route process takes longer to fit the ANN into FPGA and it creates a connections through logic cells if available routing resources are insufficient. Therefore,

most of the researchers takes this approach and choose on hardware implementation of several neurons only for bigger virtual ANN creation using scheduling and multiplexing in time. Having just one physical layer of neurons reduces hardware requirements (Nedjah *et al.* 2009). Such a technique requires additional amount of memory to store connectivity weights and intermediate results. The big issue is to write a scheduler, which keeps memory buses saturated in order to achieve maximal throughput.

The direct register transfer level approach is more advantageous against ANN implementation in high-level of abstraction. From programmers point of view, one of the fastest and easiest way for efficient implementation of ANN in FPGA is to use well known HDL and describe the net as a set of equations. Then put this set to the circuit design chain and finally only remains to hope that synthesizer, mapper, placer-and-router will give optimal circuit. The vendor standard FPGA design tools are general purpose, therefore have a global resource, power or latency optimization goals. Consequently, we are not guaranteed, that ANN optimally fits constrained hardware after the final PAR stage. The direct low-level ANN implementation gives full control of primitives and their routing, but requires additional knowledge about internal structure of slices and routing resources. In the design chain each additional circuit conversion step brings uncertainty to final structure. The closer the ANN description is to FPGA primitive building block, the more efficient network deploying can be achieved.

The main FPGA vendors provide design software suite with special tools – editors for already placed and routed designs (Altera 2013; Xilinx 2013a). These tools as a rule are dedicated for timing and routing analysis. The manual optimization of already routed circuit is a bad practice, since the timing requirement is still not guaranteed and rather become worse (Clayton 2008).

1.2.8. Efficiency Criteria of Neural Network Hardware

The ANN are generally described in terms of topology, learning algorithm, activation function, number of processing elements, layers, the type and number of inputs/outputs. The ANN implemented in FPGA are additionally specified by data representation, precision, weight storage, synaptic connection (hardwired or programmable), and on-chip learning possibility. Considering the mentioned parameters many ANN performance ratings are proposed and three of them are commonly used (Misra, Saha 2010):

- The *processing speed* measured in connections per second. It is a rate of multiplication, addition and activation function computation and shows how well the chosen network fits the hardware architecture.

- The *learning speed* measured in connection updates per second. It indicates the weights updating rate and measures the amount of samples to learn.
- The *average energy* in watt per synapse needed to compute and update the synapse.

Despite above mentioned main performance measurement rates, more hardware specific criteria exist:

- *Reconfigurability number* defined as the size of the set of all implementable but different net topologies.
- *Effective connection primitives per second* provides improved measure of processing speed, which takes into account input and weight accuracy in bits (van Keulen *et al.* 1994).
- The *algorithmic efficiency* is measured as an impact of the hardware constraints on the convergence of learning error in various ANN structures implemented in neurochip (Cornu, Ienne 1994).
- Nonlinear functions implemented in a hardware consumes time and area resources. The *overhead of the activation function* (AF) suggests to use LUTs (Hammerstrom 1990; Noory, Groza 2003) or piecewise linear functions for the approximation (Amin *et al.* 1997). An experimental study on the precision requirements denotes that 16 b is sufficient for the encoding of AF (Holt, Hwang 1993). The AF interpolation by Taylor series allows to decrease the number of LUT saving relatively high accuracy (Beiu 1998).
- The *synaptic storage density* plays significant role especially in large scale ANN, when trade-off must be achieved between memory size and power consumption. The type of memory. The distributed RAM provides high density, but consumes more power. On the other hand static RAM consumes less power with 4 times worse memory density (Misra, Saha 2010).
- The *Quality of Results* (QoR) is the most often meaning of the maximum clock frequency, at which a given design must operate in a specific FPGA. QoR is an indicator of critical path performance, which is highly related with the timing constraints, coding style and PAR tool.

The mentioned criteria are usually valuated independently on one another. However, e.g., the acceleration of synaptic processing or learning speed through parallelization of ANN structure yields the growth of FPGA resource utilization. Therefore, optimization by one individual objective makes any other individual worse. The lack of multi-objective analysis of ANN implementation in FPGA allows to lead out new criteria taking into account conflicting requirements.

1.3. Implementation of Speech Recognition in Field Programmable Gate Array

Nowadays people spend more and more time interacting with many electronic devices installed at their homes, working places, transportation vehicles, other public places. Therefore, human machine communication is a hot research topic. Speech is a natural and one of the easiest means for humans to use for communication and information exchange. Reliable and fast device control by speech can strongly facilitate everyday life of people. Therefore, a lot of research efforts are directed towards improvement of the speech recognition accuracy and the recognition speed.

Despite of recent software-based Lithuanian speech recognition (Lileikytė, Telksnys 2011) and synthesis (Pyz *et al.* 2012) implementations on personal computers and servers there is an unaddressed need of embedded systems for mobile and stand-alone devices, interactive voice controlled systems, disabled person equipment, etc. Embedded systems bring in their specific requirements for speech recognizers: the limited speed of processing, the limited size of memory, the low power consumption. The recognition of large vocabulary and continuous speech requires complicated algorithms with huge amounts of calculations, large quantities of memory (Choi *et al.* 2010; Veitch *et al.* 2010). This can result in enlarged power consumption, longer recognition time and higher recognition error rate.

Many automatic speech recognition systems for the languages of minor use are now developed. Presented in (Martinčić-Ipšić *et al.* 2011) Croatian speech recognizer uses acoustic models based on context-dependent triphone hidden Markov models (HMM) and phonetic rules. Experimentally it is shown that the system can be used for speech recognition with word error rate below 5 %. In (Sojka *et al.* 2004) a speaker independent speech recognition system for Estonian language is described. Clustered triphones with Gaussian mixture components are used there for acoustic modelling. The error rate of the system is improved to 27.3 %. In (Hirsimaki, Kurimo 2004) Finnish speech recognition based on sub-word decoders is presented. The pursued task was to find the most probable HMM state sequence. The word error rate there is decreased up to 32 % for very large vocabulary. For Czech speech recognition in (Prochazka *et al.* 2011) investigation on usability of publicly available n-gram corpora to create Czech language models is carried out. The experiments on large test data illustrate the impact of Czech as highly inflective language on the perplexity. The best achieved average error rate is 20 %. The multi-lingual Italian – Lithuanian small vocabulary speech recognition is implemented using multilingual transcription in (Maskeliūnas, Esposito 2012). The average recog-

nition accuracy of ten spoken numbers for the Lithuanian language is 93 % and for the Italian – 98 %. It is important to acknowledge that all above analysed speech recognizers are implemented in software.

In some cases the robustness and correctness of recognition, together with the low power consumption are preferred against the size of the vocabulary (Zhang *et al.* 2011). Then the natural choice is an isolated word recognition approach leading to lower hardware requirements: much smaller vocabulary (less memory), simpler classification (lower speed and power consumption), potentially higher recognition rate (correctness), and at the same time an ability to use advanced noise cancellation (robustness) (Stašionis, Serackis 2011).

The FPGA platform lets to employ the parallelization and pipelining technique in speech recognition. It is a flexible architecture to develop systems in comparison to implementations on the ASIC devices. The embedded processor-based solutions on the market have an average 80 % recognition rate and limited size of the dictionary: 32 (EasyVR (Chakravarty 2014)) or 75 (NLP (Sensory 2012)) commands. The main issue in such recognizer is to ensure real-time requirements for the speech recognition (especially at the features comparison stage). Contrary for this approach the implemented features extraction and matching processes can run independently in FPGA (Choi *et al.* 2010), (Veitch *et al.* 2010), (Zhang *et al.* 2011), (Pan *et al.* 2011) or GPU (Sart *et al.* 2010), (Zhang *et al.* 2012).

First soft-core implementations on Virtex-4 family FPGA of Lithuanian isolated word recognizer were done by Electronic Intelligent System group in VGTU Electronic Faculty (Arminas *et al.* 2010). The use of soft-core processor MicroBlaze together with intellectual property cores for signal processing enables to accelerate word recognition process by 1.55 times (Ivanovas 2012; Tamulevičius *et al.* 2010), however it was still not enough for a real-time operation.

The common speech recognition algorithm consists of several steps, as shown in Fig. 1.17. The speech signal can be loaded from memory or captured from microphone in a real-time. The preprocessing step involves the noise reduction and signal partitioning into equal overlapped 10–30 ms frames. For each frame the features vector is formed employing predictive or cepstral analysis. The extracted features are compared with the features stored in dictionary. If both vectors of features match each other a concrete command related with recognized word is executed.

Speech recognition process is quite elaborative, thus it requires a lot of arithmetic, logic and memory access operations. When a speech recognition algorithm primarily implemented on a computer is moved to the embedded system, its execution speed drastically decreases. In this case, the only way to maintain the speed is to use the parallelisation and pipelining techniques.

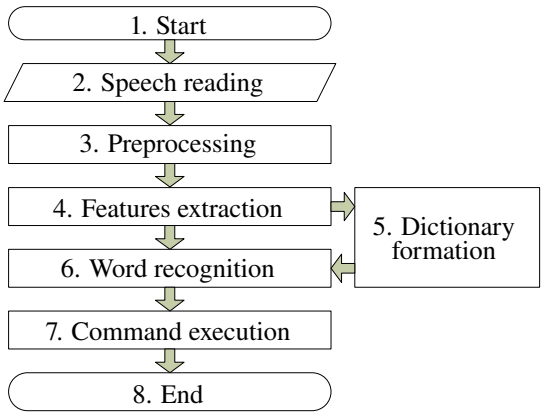


Fig. 1.17. Isolated word recognition algorithm

Therefore, during the last decade researchers are increasingly implementing speech recognizers on FPGA or graphics processing units (Amudha *et al.* 2008; Sart *et al.* 2010; Veitch *et al.* 2011; Zhang *et al.* 2012). FPGA provides an opportunity of parallel signal processing and addressing at relatively low frequencies compared with a central processing unit. If speech recognition application works on a battery powered device, a one logic array in case of size and power consumption is superior to the multi-core processor.

The main issues, which prevent to design a reliable and universal method for speech recognition, are the environmental impacts, poor pronunciation, speech variability, limited vocabulary size, and a similar phonetic transcription of the words (Tamulevičius 2008a). These issues influence the correctness of features extraction in a speech. Therefore, an evaluation of algorithms is important especially for the real-time recognizers. The experimental investigation of factors influencing recognition accuracy shows that setting the same training and testing environments yields improved accuracy (Lileikytė, Telksnys 2011; Čeidaitė, Telksnys 2010). Authors claim that features based on cepstrum coefficients are very sensitive to environment. However the autoregression based features, e.g., linear predictive coefficients (LPC) and linear predictive cepstral coefficients (LPCC), have an average sensitivity. The quality estimation of the speech features shows that linear frequency cepstral coefficients (LFCC), Mel-scale frequency cepstral coefficients (MFCC) are suitable for Lithuanian phonemes recognition. Experiments on small vocabulary Lithuanian speech recognition confirmed that accuracy of 93 % is acceptable for application with multilingual transcriptions engines (Maskeliūnas, Esposito 2012).

In a last few years the MFCC and LPCC features become a reasonable leaders in the recognition systems. A 98.5 % rate was achieved recognizing

isolated words in a small vocabulary using MFCC (Darabkh *et al.* 2012). The MFCC and LPCC features are suitable for classification of speech disfluencies with 92.55 % and 94.51 % rates accordingly (Ai *et al.* 2012). A comparative study of LFCC vs MFCC was performed in (Zhou *et al.* 2011). Results show that LFCC consistently outperforms MFCC. The benefits are visible especially on female speech recognition. There are known hardware-based MFCC and LPCC implementations, which allows to accelerate features extraction process (Staworko, Rawski 2010; Vu *et al.* 2010). The combination of MFCC and LPCC are also appropriate for speaker identification with maximum 97.12 % (Yujin *et al.* 2010). The LPCC always outperforms the LPC features over normal and noisy conditions (Fook *et al.* 2012). The highest 91.4 % recognition rate was achieved using LPC and artificial neural network in a small vocabulary system (Wijoyo 2011). The autoregression based algorithm (e.g., LPC) are more suitable for software implementation rather than hardware due to precision requirements (Xu *et al.* 2005). Therefore, a soft-core processor is popular for recursion implementation (Atri *et al.* 2012).

For features extraction in speech signal in most above mentioned recognizers the spectral, MFCC (Martinčić-Ipšić *et al.* 2011), LFCC (Tamulevičius *et al.* 2010), LPC, LPCC, perceptual linear prediction (Lileikytė, Telksnys 2011) analysis are applied. The brief overview of these methods is given further.

1.3.1. Linear Frequency Cepstral Analysis

A real-time calculation of cepstrum coefficients can be performed (Tamulevičius *et al.* 2010). In most cases, the speech signal is divided in short 10–30 ms half overlapped frames. Then, the LFCC feature vector is estimated for each single frame by:

$$\boldsymbol{c}_{\mathrm{LF}}(n) = \mathrm{real}\Big(\mathcal{F}\Big(\log_2\Big|\mathcal{F}\big(\boldsymbol{s}(n)\big)\Big|\Big)\Big), \tag{1.11}$$

here $\boldsymbol{c}_{\mathrm{LF}}(n)$ is the cepstrum vector; $\boldsymbol{s}(n)$ is the signal frame; $\mathcal{F}$ is the fast Fourier transform operator. The feature vector is composed of several first cepstrum coefficients. The LFCC are estimated according to signal flow-graph in Fig. 1.18.

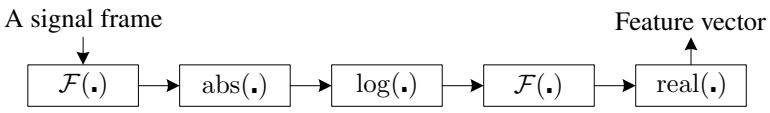


Fig. 1.18. The signal flow-graph of linear frequency cepstral coefficients extraction

The $\mathcal{F}$ operator is provided by FPGA manufacturer as an optimized IP core for certain hardware architecture (Altera 2014; Xilinx 2015a). The $\log_{10}$ operator can be implemented using LUT, multiplexers and single sum operation (Wang *et al.* 2002). The accuracy of such $\log_{10}$ approximation depends linearly on LUT size. However, in FPGA is more convenient to implement the $\log_2$ function as a search of index for first left bit in a binary number. The abs and real operators are trivial and easy implementable. Therefore, generally the LFCC features are appropriate for FPGA implementation and speech analysis in real-time.

1.3.2. Mel-Frequency Cepstral Analysis

The mel-frequency cepstral coefficients extraction process is accomplished in five steps in Fig. 1.19.

Fig. 1.19. The signal flow-graph of mel-frequency cepstral coefficients extraction

The MFCC are defined by:

$$c_{\mathrm{MF}\,i}(n) = \sum_{m=1}^{N_\mathrm{M}} \log S_m(n) \cdot \cos \frac{\pi(m-0.5)i}{N_\mathrm{M}}, \quad \forall\, i \in [0, N_\mathrm{M}-1]; \tag{1.12a}$$

$$S_m(n) = \sum_{j=0}^{N_\mathrm{F}-1} \left|\mathcal{F}(\boldsymbol{s}(n))\right|^2 K_m(j), \quad \forall\, m \in [1, N_\mathrm{M}], \tag{1.12b}$$

here $\boldsymbol{s}(n)$ is a signal frame at time instant n; $K_m(j)$ is the transfer function of the given mth filter bank; j is the frequency bin index; N_F is the number of discrete frequencies; $S_m(n)$ is the mel-filtered energy spectrum (at time instant n) filtered by N_M mel-scaled triangular filters; $c_{\mathrm{LF}\,i}(n)$ is the ith mel-scale frequency cepstral coefficient, when discrete cosine transform $\mathcal{C}(.)$ is used.

The possible implementation of fast Fourier transform $\mathcal{F}$ and $\log$ operator in field programmable gate array have been reviewed in previous subsection. The sum of the squared output of the $\mathcal{F}$ block is taken as energy spectrum utilizing two multipliers for both real and imaginary part and one adder (Bahoura, Ezzaidi 2013; Sarkar, Saha 2010). The mapping of the linear-scale frequency

to the mel-scale frequency in a filterbank block is realized according to logarithmic dependence:

$$\mathrm{mel}(f) = 1127 \ln\left(1 + \frac{f}{700}\right), \tag{1.13}$$

here f is linear frequency; $\mathrm{mel}(f)$ is a mel-scale frequency.

Instead of implementing the (1.13) directly in a hardware, the transfer function coefficients $K_m(j)$ usually are stored in read-only memory and the mel-filtered energy spectrum can by implemented on single multiply-accumulate unit (Bahoura, Ezzaidi 2013; Schmadecke, Blume 2013; Vu *et al.* 2010). The filter banks can be implemented with multiplexer, shifts and additions (Staworko, Rawski 2010). Thus the size of multiplexers and the number of combinational circuits for bit shifts and additions increases proportionally to the number of filter coefficients. The $\mathcal{C}(\cdot)$ calculation can be implemented on single MAC, where the values of the cos function are precomputed and stored in LUT (Sarkar, Saha 2010; Vu *et al.* 2010).

The Simulink or LabView tool can be used to fast calculation of MFCC but such method is not always efficient due to low performance and less than 100 MHz of maximum operating frequency (Bahoura, Ezzaidi 2013; Farjo *et al.* 2012). At least two times higher operating frequency of MFCC circuit can be achieved using hardware description language (Veitch *et al.* 2010). If embedded processor is fast enough to calculate speech features in real-time then it can be used in co-design with FPGA. It becomes popular and one of the fastest way to develop MFCC features extraction in FPGA with soft-core processors (Cheng *et al.* 2011; Manikandan, Venkataramani 2011; Pan, Li 2012; Zhang *et al.* 2011).

1.3.3. Linear Predictive and Linear Predictive Cepstral Analysis

Linear predictive analysis is based on simplified vocal tract model, where the voice production can be modelled by process of passing excitation signals through autoregressive filter. The coefficients of this filter describes the compressed voice and they are obtained by using Levinson-Durbin autoregression. The LPC features by itself are sensitive to quantization errors. Therefore, the LPC features should be converted to less quantization sensitive linear predictive cepstral coefficients (LPCC), which are more suitable for implementation in FPGA with fixed-point precision without (Wu *et al.* 2006) or with hardware accelerators (Liu, Bergmann 2010). These features are extracted in three steps, as is shown in Fig. 1.20.

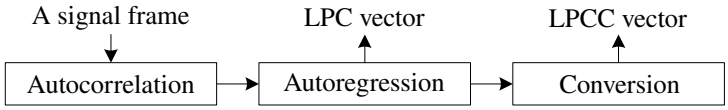


Fig. 1.20. The signal flow-graph of linear predictive and linear predictive cepstral coefficients extraction

The autocorrelation coefficients r_i are estimated by:

$$r_i = \sum_{n=0}^{N_\mathrm{W}-1} s(n)s(n+i), \quad \forall\, i \in [0, M], \tag{1.14}$$

here N_W is the window length of speech signal $s(n)$; M is the autocorrelation length.

The systolic array is often used for autocorrelation implementation, because of suitable arrangement and interconnection of DSP cores in a hardware Fig. 1.21 (Atri *et al.* 2012).

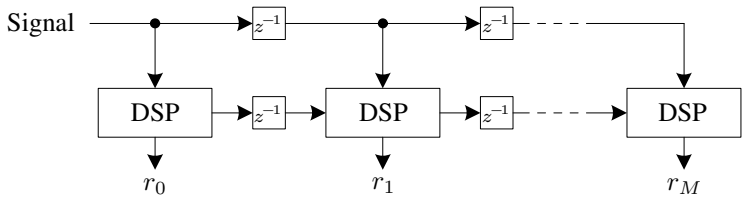


Fig. 1.21. The systolic array for the autocorrelation

The LPC and LPCC features are extracted using Levinsin-Durbin algorithm (Tamulevičius 2008b):

$$e_0 = r_0; \quad k_1 = r_1/e; \quad a_1 = k_1; \tag{1.15}$$

$$e_i = \left(1 - k_i^2\right)e_{i-1}, \quad \forall\, i \in [1, M]; \tag{1.16}$$

$$k_i = \left(r_i - \sum_{j=1}^{i-1} a_j r_{i-j}\right)e_{i-1}, \tag{1.17a}$$

$$a_j = a_j - k_i a_{i-j}, \tag{1.17b}$$

$$c_{\mathrm{LP}\,i} = a_i - \sum_{j=1}^{i-1}\left(\frac{j}{i}\, c_{\mathrm{LP}\,j}\, a_{i-j}\right), \quad \forall\, i \in [1, M] \text{ and } j \in [1, i], \tag{1.17c}$$

here e is the temporal energy value; k is the reflection coefficient; M is the LP analysis order; $\boldsymbol{a}$ and $\boldsymbol{c}_{\text{LP}}$ are the LPC and LPCC vectors.

Despite the advantage of parallelism in signal processing, the quantization error accumulates during the autoregression and conversion stages adversely affecting features accuracy (Xu *et al.* 2005). Therefore, the recursive part of LPCC analysis is more often implemented on soft-core processor with floating-point arithmetic rather than pure in a hardware.

1.3.4. Features Classification

The main difficulties in speech recognition arise from speech variability, incorrect pronunciation, environmental impact, phonetic transcription similarities, or dictionary size limits (Tamulevičius 2008b; Čeidaitė, Telksnys 2010). Therefore, many methods for the speech features classification are developed, e.g., the dynamic time warping (DTW) (Hussain, Rashid 2012; Sart *et al.* 2010; Tamulevičius *et al.* 2010; Zhang *et al.* 2012), hidden Markov models (Gavat *et al.* 2008; Hirsimaki, Kurimo 2004; Prochazka *et al.* 2011), Gaussian mixture models (Sojka *et al.* 2004) and artificial neural networks (Amudha *et al.* 2008).

For precise spoken word recognition with HMM it is preferred to use floating-point numbers due to the specifics of Viterbi algorithm, which operates on probabilities of acoustic models. Another issue is the learning of new word. The phonetic transcription and the statistical model must be created for each new word added to dictionary. It is time consuming process. As well as adapting the ANN, that updates the weights and changes topology, when it is trained for unknown word. Different from above mentioned classifiers, the DTW do not requires any specific training algorithm, because it needs only to store a feature vector of new word.

We are concentrated in the speaker dependent Lithuanian isolated word recognition. Each spoken word can be presented as a finite set of features restricted in a time. The time-limited signals are most appropriate to compare with the DTW algorithm, which is suitable for FPGA due to straightforward arithmetic on fixed-point numbers and pipelined signal processing (Sart *et al.* 2010; Zhang *et al.* 2012). The DTW is already more than four decades old but still an effective method to match two templates varied in the time domain (Ding *et al.* 2008). The length of each template as well as the number of templates stored in a dictionary influences the matching time. A speed-up of the whole process can be achieved by duplication of DTW cores in single FPGA (Hussain, Rashid 2012). Various basic and parallel DTW implementations are proposed. The experimental studies in last mentioned references confirm that using such approach one DTW can be calculated in approximately 0.57 ms at 100 MHz clock with 64×64 size of the error matrix.

Another way of DTW acceleration is the normalization: reduction of the length of all time series from their original length to the predefined length (Sart *et al.* 2010). This method becomes efficient when comparable data series are very long (the duration of comparison process is proportional to the squared length of the time series). In a recent paper the normalization block is employed to normalize the length of input series to exactly 128 samples. The authors claim that normalization of raw data does not affect the accuracy of DTW. The proposed warper uses an error matrix with the 128×128 size. The DTW is based on a systolic array that is primarily described in C language and converted to VHDL via *ROCCC* compiler (Buyukkurt, Najjar 2008). The similarity search algorithm works on 250 MHz clock and requires 128 clock cycles to calculate the DTW distance. The time series with the reduced precision of 8 b integers are used intentionally with the aim to increase the performance.

Even if features extraction algorithm works correct, there is still important to chose proper classification method. The DTW method is an appropriate way to find similarities in two vectors of features. FPGA based pipelined implementation of the classification allows to accelerate speech recognition process (Pan, Li 2012; Zhang *et al.* 2011). A comparative study of DTW implementations on different platforms shows that FPGA-based DTW outperforms the GPU and CPU-based implementations more than 44 and 2100 times accordingly (Sart *et al.* 2010; Zhang *et al.* 2012). These arguments inspires to implement DTW intellectual property core for a real-time isolated word recognition.

1.4. Conclusions of the 1st Chapter and Formulation of the Thesis Objectives

On the basis of the analytical review it is possible to assert that:

1. The higher circuit implementation level, the less guarantee that logic circuit optimally fits to FPGA resources. However, from the designer point of view the system can be described faster with HLS tool.
2. The lower circuit implementation level, the more time is requited to evaluate all possible paths and find an optimal one. Lower level is designer unfriendly, but gives possibility to implement the circuit on an optimal criteria.
3. The FPGA contains optimized DSP, memory and logic blocks suitable for efficient lattice-ladder multilayer perceptron (LLMLP) implementation.
4. The LLMLP neural network was never before implemented on FPGA. A large number of required arithmetic resources suggests to reuse DSP slices for network implementation.

5. The general purpose high level synthesis tools generates design with poor performance and are suitable only for fast concept proof, however not for long-term functioning solution. Therefore, specialized hardware description language compiler must be written for LLMLP implementation with respect to hardware limitations and LLMLP specifics.
6. The trade-off between artificial neural network (ANN) accuracy, FPGA utilization rate and processing speed must be maintained, while implementing parallel ANN with training structures.
7. The speech signal analysis methods requires a lot of arithmetic operations. Some of them have recursive processes, which are not appropriate for implementation in a hardware. Therefore, the soft-core processors must be used.
8. Using few dynamic time warping cores on single FPGA or a systolic FPGA structure it is possible to achieve fast features comparison on a relatively low system clock through increased resource utilization and pipelined design.

The following tasks must be solved:

1. The creation of the technique for efficient LLMLP implementation on FPGA based on specialized quality criteria and specifics of LLMLP.
2. Develop the optimized intellectual property (IP) cores used in Lithuanian speech recognizer.
3. Experimentally verify the efficiency of IP cores.
4. Investigate isolated word recognition accuracy and speed.

2

Efficient Implementation of Lattice-Ladder Multilayer Perceptron

Pursuing the Task 2 of the Thesis the results of LLMLP efficient implementation criteria and technique original development are presented in this chapter. In Section 2.1 LLMLP implementation quality and a set of criteria that determine it are discussed. Idea of Pareto frontier calculation for the implementation quality criteria is raised. A new LLMLP implementation technique is introduced in Section 2.2. Here the specialized criteria for circuitry synthesis is brought in together with a selected way to tailor the implementation technique to the intrinsic LLMLP structural features. Afterwards major stages of the technique are elucidated. In Section 2.3 all the steps for a single neuron processing element (NPE) optimization are in details presented. An emphasis on subgraph matching, covering, merging and scheduling algorithms is done. As a result the NPE is optimized for optional pipeline stages and latency/resources optimal training. In Section 2.4 optimization steps of neuron layers implementation are presented. Accuracy optimization as well as two equivalent implementation strategies, namely throughput optimization (Subsection 2.4.2) and resource optimization (Subsection 2.4.3) are crafted by corresponding algorithms.

The research results are published in author publications (Sledevič, Navakauskas 2016, Sledevič, Navakauskas 2015, Sledevič, Navakauskas 2014). The main results are announced in international: “Electronics” (Palanga, 2015), AIEEE (Vilnius, 2014; Riga, 2015); and national “Science – Future of Lithuania”(Vilnius, 2015, 2016) scientific conferences.

2.1. Implementation Quality

As was discussed in Chapter 1, there is not enough resources for implementation of large LLMLP in parallel manner even on modern FPGA. Therefore, the strategy, that we propose to perform NPE, consists of two steps:

- implement a configurable NPE with maximal throughput maintaining shortest possible processing latency τ_{NPE} and maximal clock frequency $f^{\top}$ of synthesized design;
- place and share the NPEs through whole LLMLP maintaining parallelism of synapse, neuron or layer if resources allow.

The efficient implementation of LLMLP employing multiple optimized NPE in FPGA is a multi-criteria task dependent on the optimization goal. These criteria can split into hardware and network related groups of quality criteria. The amount of used resources, accuracy, power consumption and FPGA chip price are related to hardware quality criteria, which can be improved selecting a correct programming tool or changing FPGA chip. The size of LLMLP, the type of training and latency of the signal propagation from the input layer to the output are all related to the group of network quality criteria improved only by the modifications in LLMLP topology. The total quality $\mathcal{Q}$ covers hardware and network related criteria and is equivalent to set of all mentioned criteria:

$$\mathcal{Q} = \{\mathfrak{q}_{\text{Thr}}, \mathfrak{q}_{\text{Acc}}, \mathfrak{q}_{\text{Res}}, \mathfrak{q}_{\text{Com}}, \mathfrak{q}_{\text{Pow}}, \mathfrak{q}_{\text{Price}}\}, \tag{2.1}$$

here $\mathfrak{q}_t$ is a quality criteria of type t: “Thr” – throughput, “Acc” – accuracy, “Res” – resource, “Com” – complexity, “Pow” – power, “Price” – price.

The performance of the NPE circuit depends on the clock frequency, which is known only after final PAR step in the FPGA implementation chain. Usually the maximum clock frequency $f^{\top}$ is highly related to the coding style. Register insertion to the signal propagation path yields shorter delay of the signal in a critical path and higher clock frequency. Therefore, the ratio $f^{\top}/\tau_{\text{NPE}}$ will be used as a throughput criteria for the performance of NPE. The lower it is, the less time is needed for signal processing using single NPE. The throughput criteria $\mathfrak{q}_{\text{Thr}}$ shows the maximal sampling frequency for data during the LLMLP weights updating in training mode. And in a forward filtering mode the $\mathfrak{q}_{\text{Thr}}$ shows the maximal sampling frequency of the signals on the input of LLMLP. The $\mathfrak{q}_{\text{Thr}}$ criteria covers the *Quality of Results*, *processing* and *learning speed* criteria analysed in Section 1.2.8. The aim is to achieve lowest possible latency, which will increase the highest possible frequency for data sampling. The $\mathfrak{q}_{\text{Thr}}$ depends on the $f^{\top}$ at which the synthesized LLMLP can be clocked and on the latency τ_{NPE} of signal processing by NPE:

$$\mathfrak{q}_{\text{Thr}} = f^{\top}/\tau_{\text{NPE}}. \tag{2.2}$$

The accuracy criteria $\mathfrak{q}_{\text{Acc}}$ evaluates the precision of the LLMLP based on fixed-point arithmetic, since FPGA works efficiently only on integer numbers. The precision of fixed-point design must be compared with reference floating-point design implemented on PC.

$$\mathfrak{q}_{\text{Acc}} \equiv \Psi\left(\text{fixed-point design, floating-point design}\right). \tag{2.3}$$

The workflow for investigation of a fixed-point design implementation is proposed further in this chapter.

The resources criteria $\mathfrak{q}_{\text{Res}}$ evaluates the equivalent amount of arithmetic R_{DSP}, logic R_{LUT} and memory R_{BRAM} resources:

$$\mathfrak{q}_{\text{Res}} \equiv \Psi\left(w^{\text{serie}}_{\text{DSP}} R_{\text{DSP}} + w^{\text{serie}}_{\text{LUT}} R_{\text{LUT}} + w^{\text{serie}}_{\text{BRAM}} R_{\text{BRAM}}\right), \tag{2.4}$$

with respective weights $w^{\text{serie}}_{\text{DSP}}$, $w^{\text{serie}}_{\text{LUT}}$ and $w^{\text{serie}}_{\text{RAM}}$ dependent on FPGA serie. These weights are calculated when the arithmetic, logic and memory resources are expressed in LUT equivalent units. For 7th series Xilinx FPGA it is known that a single DSP can be synthesized using 196 LUT, thus $w^{(7)}_{\text{DSP}} = 196$. One BRAM contains 18432 b of with dual port access and one LUT configured to RAM can store 32 b in this manner $w^{(7)}_{\text{BRAM}} = 576$. The LUT weight takes into account that only 0.7–0.8 part of LUT can be efficiently utilized in FPGA due to limited routing resources. In addition, if all the LUTs are used as RAM, then FPGA dependent 0.3–0.5 coefficient must be applied for $w^{(7)}_{\text{LUT}}$ considering that only part of LUT can be configured as memory block.

$$R_{\equiv\text{LUT}} \equiv R_{\text{LUT}} + 196 \times R_{\text{DSP}} + 576 \times R_{\text{BRAM}}. \tag{2.5}$$

The less resources is utilized, the more complex LLMLP can be implemented on FPGA. Therefore, the resource criteria is an inversion of equivalent resources:

$$\mathfrak{q}_{\text{Res}} = 1/R_{\equiv\text{LUT}}. \tag{2.6}$$

The LLMLP complexity criteria $\mathfrak{q}_{\text{Com}}$ depends on number of layers L, neurons N, synapse order M and an optimal training algorithm from a set $\mathfrak{T}$:

$$\mathfrak{q}_{\text{Com}} \equiv \Psi\left(L, N, M, \mathfrak{T}\right). \tag{2.7}$$

The aim is to fit largest complexity LLMLP to FPGA with minimal resource R utilization and minimal processing latency τ optimizing by training type $\mathfrak{T}$:

$$\mathfrak{q}_{\text{Com}} = \frac{1}{\min_{\mathcal{T}}\left(R(\mathcal{T},L,N,M)\right)\min_{\mathcal{T}}\left(\tau_{\mathcal{T},L,N,M}\right)}. \tag{2.8}$$

The consumed power criteria $\mathfrak{q}_{\text{Pow}}$ depends on resource utilisation and maximal clock frequency of the implemented design:

$$\mathfrak{q}_{\text{Pow}} \equiv \Psi\left(R, f^{\top}\right). \tag{2.9}$$

The FPGA price criteria $\mathfrak{q}_{\text{Price}}$ depends on available resources on FPGA and the chip family:

$$\mathfrak{q}_{\text{Price}} \equiv \Psi\left(R, \text{chip family}\right). \tag{2.10}$$

Definitely all these criteria cannot be optimized simultaneously, because of the conflicts between them. The improvement of the accuracy or throughput criteria yields to the rising resource utilisation and as a result the selection of higher price FPGA. The growing LLMLP complexity leads to the higher power consumption. Therefore, we propose to derive the efficiency criteria of LLMLP implementation on FPGA through perspective of user changeable parameters. It is considered that the developer usually prefer to modify the LLMLP structure, change the FPGA chip and set desired sampling frequency depending on the application requirements. Therefore, from the developer point of view to the LLMLP implementation we propose to distinguish two strategies to design optimization for importance criteria:

- The desired *sampling frequency* of the signals, that are entered to the LLMLP (throughput criteria).
- The specific FPGA *chip* or their resources partial utilisation (resources criteria) for particular structure of LLMLP.

In the multi-criteria optimization problem, there does not exist a single solution that simultaneously optimizes each criteria without making at least one individual worse. In our case, the criteria are conflicting and there exists more that one Pareto efficient solution. Evaluation of the criteria of LLMLP through Pareto frontier will enable us to take optimal decisions in the presence of trade-off between two conflicting throughput and resources objectives:

$$\mathcal{Q}_{\text{Pareto}} \triangleq \max(\mathfrak{q}_{\text{Thr}}, \mathfrak{q}_{\text{Res}}). \tag{2.11}$$

The result of optimization by throughput criteria is a maximal possible sampling frequency checking the solution space through constrained resources and different LLMLP configurations. Having a solution space of throughput $\mathfrak{q}_{\text{Thr}} \equiv \Psi\left(\mathfrak{q}_{\text{Res}}\right)$ it is possible to fast evaluate the LLMLP design for certain

FPGA. During the optimization by resource criteria, the LLMLP design is optimized for minimal resource utilisation giving as an input arguments $\mathfrak{q}_{\text{Thr}}$ and LLMLP structure: $\mathfrak{q}_{\text{Res}} \equiv \Psi(\mathfrak{q}_{\text{Thr}})$. Such approach enables us to select the cost efficient FPGA chip in resources utilisation Pareto optimal surface for certain $\mathfrak{q}_{\text{Thr}}$ criteria and net complexity. The throughput and resource optimal algorithms for desired complexity LLMLP evaluation are elucidated in next subsections.

The power and price criteria are considered to be second importance, since the close relation with resources requirement. The suitability of certain training circuit and accuracy must be investigated before the optimization by first importance criteria. The efficient training circuit depends only on the synapse order, therefore the predefined optimal training type can be hardwired to the desired LLMLP complexity criteria $\mathfrak{q}_{\text{Com}}$. The required precision for the signals in fixed-point design will be estimated in comparison with the same floating-point design. The determined optimal width of the signals will be used in all the prospective LLMLP implementations.

The main difference between proposed criteria for LLMLP implementation and analogous purpose criteria for efficient ANN implementation is that proposed criteria takes into account conflicting requirements as throughput and resources. While majority of analogous purpose criteria uses single-objective optimization by criteria listed in Section 1.2 (Misra, Saha 2010).

2.2. Introduction to Implementation Technique

A direct implementation of LLN with its cost-effective $\mathfrak{t}_3$ type training circuit in a hardware will require: $9M + 1$ multiplications for the forward LLF and its regressor calculations. $4M + 2$ multiplications are required for weight updates in (1.5) and (1.6), and 2 multiplications for δ calculation in (1.7). Thus, a hardware implementation of the LLN with N inputs and Mth order synapses with its training algorithm in total requires $N(13M + 3) + 2$ multiplication operations. Having $N = 5$ inputs and $M = 3$ order synapses LLN implementation will utilize, e.g., $96\,\%$ of DSP slices available in Artix-7 XC7A100T FPGA. Therefore, willing to increase the size of LLN the DSP slice resources must be shared. Moreover, the LLF with the regressor lattice cannot be pipelined to receive and process data simultaneously with filter coefficients update at each rising edge of clock, because the value of updated weight for next input sample is dependent on the present value of neuron output. Therefore, the FPGA resources must be shared in time over several operations. Such approach eliminates long idle period for single DSP slice and tends to compact all necessary operations for LLMLP also optimizing the total latency.

All recent FPGA have dedicated DSP slices for fast arithmetic implementation. Such a DSP slice can be shared to process data from different sources and also it can be dynamically reconfigured to implement different processing orders. The DSP block is purposely designed for pipelined data processing with maximum performance. This is achieved inserting flip-flops and minimizing critical path in a circuit, as shown in Fig. 1.2. Sequentially connected flip-flops introduce latency of 4 clock cycles in the data flow processing independently on the selected instantaneous DSP configuration and subgraph constellation (Xilinx 2014a).

The flexible structure of DSP slice (Fig. 1.2) can be considered as a configurable directed subgraph, where each vertex express certain arithmetic operator. The source vertices are interconnected with destination vertices by edges. Each edge has a delay element consisting of D-type flip-flops, that fragments a signal-flow path of the subgraph in smallest possible pieces (operators isolated by flip-flops) and enables the shortest propagation time of the signal between adjacent vertices.

The LLMLP with its training circuit contains addition, subtraction and multiplication operations. All these operations are supported by DSP. Therefore, the LLMLP can be also successfully represented as a data flow graph (DFG), which is composed form a set of subgraphs that are supported by the DSP block. The aim is to partition whole LLMLP graph into subgraphs and then cover the graph with largest detected subgraphs minimizing resource utilization. The DSP slice subgraph has maximum four input terminal vertices: `A`, `B`, `C`, `D` and a single output terminal vertex `P` in Fig. 1.2. The source vertex can be connected with a destination vertex to create a subgraph only if the source vertex has no branches to other vertices except the unique branch to destination vertex. Such a constraint follows from physical DSP slice structure. It is reasonable to cover largest subgraphs first, as such approach decreases total number of subgraphs resulting in minimal latency and reduces FPGA resource usage, since the internal signals in DSP do not need to be stored in registers outside DSP.

In general, each LLMLP can be described as a set (2.12) of equations. The LLMLP implementation on FPGA is based on the translation of LLMLP equations from human-readable high-level of abstraction to the low level machine code. Such tool, which translates the high-level language into a lower-level code, is called a compiler. In our case the compiler takes a file with LLMLP equations and transforms it to HDL file understandable for machine in register-transfer level. The HDL file later is used in FPGA vendor tool for implementation of bit file for FPGA programming.

The stages of the technique for efficient LLMLP implementation on FPGA is shown in Fig. 2.1. It is based on the conversion of LLMLP equation to the

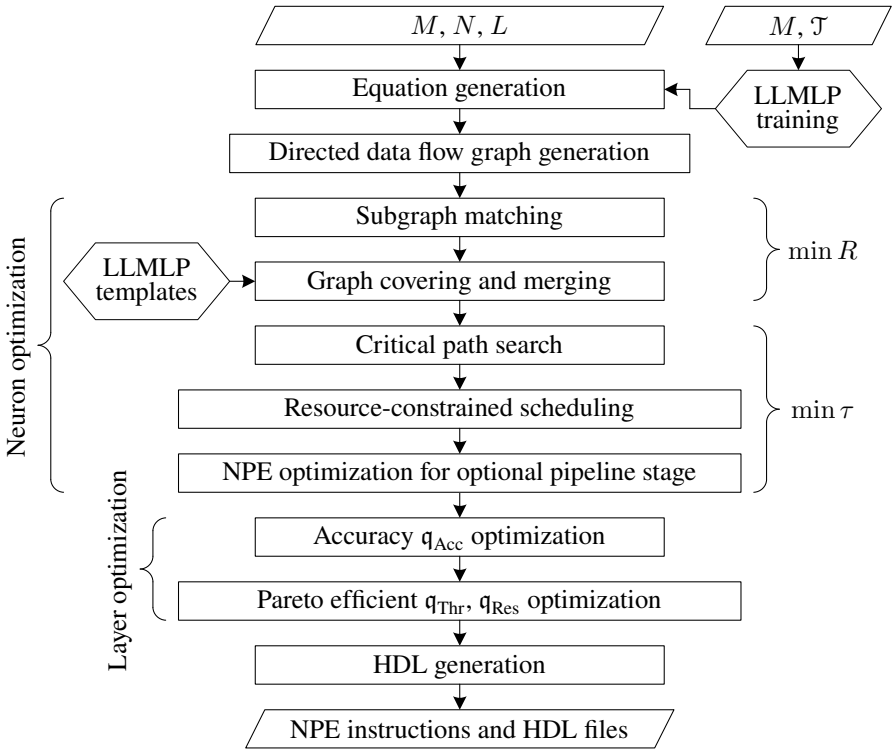


Fig. 2.1. The proposed technique for lattice-ladder multilayer perceptron implementation in field programmable gate array

DFG and splitting it to the DSP supportable subgraphs and further scheduling of DSP operations. At the first stage the LLMLP is expressed as set of equations:

$$S_{\text{eq}} = \{\text{eq}_1, \text{eq}_2, \ldots, \text{eq}_K\}. \tag{2.12}$$

Each equation must have two operands and single operator (e.g., $\text{eq}_1 = f_1 + b_1$). Such only two operand restriction comes from physical property for implementation of arithmetic operation in FPGA keeping shortest possible delay in critical path for triggered circuits. The generation of these equations is automated for the LLMLP with various filter order M, number of neurons N, layers L and training types $\mathfrak{T}$. The list of arbitrary operators necessary to describe LLMLP is summarized in Table 2.1.

Input equations for first order direct and regressor (type $\mathfrak{t}_3$) lattices are shown in Example 1. The equations describes a flow graph presented in Fig. 2.4.

Table 2.1. The list of operators required for lattice-ladder multilayer perceptron implementation

Operator	Operator meaning
$+$	Addition of two signals
$-$	Subtraction of two signals
$*$	Multiplication of two signals
$=$	Signal delay for single clock cycle
$\#$	Read the value of activation function from LUT
$\sim$	Read the value of trigonometric function from LUT

Example 2.1 (Equations for first order direct and regressor lattices)

```
 1) * f1c1 x c1 ;          12) + AD1 Svb0 b1v1 ;
 2) * zb0s1 zb0 s1 ;       13) + N1 b1v1 LCs1 ;
 3) * f1s1 x s1 ;          14) - CB1 f1v1 B1 ;
 4) * zb0c1 zb0 c1 ;       15) * LCs1 LC1 s1 ;
 5) - b0 f1c1 zb0s1 ;      16) - LCsAD1 AD1 LCs1 ;
 6) + b1 f1s1 zb0c1 ;      17) * LCsADs1 LCsAD1 s1 ;
 7) * b1v1 b1 v1 ;         18) + B1 LCsADs1 L1 ;
 8) * b0v0 b0 v0 ;         19) + D0 N1 b0v0 ;
 9) + Svb0 b1v1 b0v0 ;     20) * Nabla1 CB1 ncos1 ;
10) * f1v1 x v1 ;          21) = zb0 b0 b0 ;
11) + LC1 L1 f1v1 ;        22) = L1 D0 D0 ;
```

As it follows from analytical review of ANN implementation on FPGA the LLMLP was never before implemented on FPGA. Proposed method shares the arithmetic, memory and logic resources of FPGA in order to minimize resource usage. The main differences between proposed and known methods for ANN implementations on FPGA is that they do not take advantage of dynamic reconfigurability of DSP and dedicate single DSP for synapse of neuron (Misra, Saha 2010). Even if the reconfigurability feature of DSP is employed, but not shared, it restricts the size of synthesized circuit (Ronak, Fahmy 2014) and therefore the desired high complexity of LLMLP is limited.

2.3. Neuron Processing Element Optimization

2.3.1. The Data Flow Graphs Generation

At the DFG generation stage the operators in the set S_{eq} are converted to a set of vertices $\mathcal{V}(\mathcal{G})$ connected by set of edges $\mathcal{E}(\mathcal{G})$ in a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ together with initial and terminal edges to vertex $(\mathcal{E} \rightarrow \mathcal{V})$ maps assigning

to every edge E an initial vertex $\mathrm{init}(E)$ and a terminal vertex $\mathrm{ter}(E)$. The vertex $V \in \mathcal{V}$ represents an operator or an input port. Each operational vertex V has two inputs and attributes specifying its operator type. An edge $E = \big(\mathrm{init}(E), \mathrm{ter}(E), P\big) \in \mathcal{E}$ indicates a data transfer from $\mathrm{init}(E)$ to input port P of $\mathrm{ter}(E)$.

The DFG of LLMLP is stored as an adjacency list, since list is a compact and memory efficient way for formal description of DFG instead of sparse adjacency or incidence matrices. Each row in adjacency list is related with certain destination vertex and store a set of source vertices.

2.3.2. Subgraph Matching

On this step all the DSP supportable subgraphs $\mathcal{G}_{\text{DSP}}$ described by (1.1) and illustrated in Fig. 2.2 must be found in a graph $\mathcal{G}$. We consider that the mark $+/-$ denotes two different operators, then 15 types of subgraphs $\mathcal{G}^{(x)}_{\text{DSP}} \subseteq \mathcal{G}_{\text{DSP}} \mid x \in [1, 15]$ must be enumerated. The first subgraph $\mathcal{G}^{(x)}_{\text{DSP}}$ in Fig. 2.2 has indices $x \in [1, 4]$, the 2nd–4th subgraphs have $x \in [5, 12]$ and the 5th–6th subgraphs have $x \in [13, 15]$.

Through subgraphs $\mathcal{G}^{(x)}_{\text{DSP}}$ search in a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, all $\mathcal{G}^{(x)}_{\text{DSP}}$ are matched with a subgraphs $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$, where $\mathcal{V}' \subseteq \mathcal{V}$ and $\mathcal{E}' \subseteq \mathcal{E}$. The $\mathcal{G}^{(x)}_{\text{DSP}}$ is detected in $\mathcal{G}$ if there exist one-to-one correspondence between vertices of subgraphs $\mathcal{G}^{(x)}_{\text{DSP}} = \big(\mathcal{V}^{(x)}_{\text{DSP}}, \mathcal{E}^{(x)}_{\text{DSP}}\big)$ and $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$, when $\mathcal{G}^{(x)}_{\text{DSP}}$ contains all the edges $E \in \mathcal{E}'$ with initial vertex $\mathrm{init}(E) \in \mathcal{V}^{(x)}_{\text{DSP}}$ and terminal vertex $\mathrm{ter}(E) \in \mathcal{V}^{(x)}_{\text{DSP}}$, then two subgraphs $\mathcal{G}^{(x)}_{\text{DSP}}$ and $\mathcal{G}'$ are equal $\mathcal{G}^{(x)}_{\text{DSP}} = \mathcal{G}'$ and isomorphic. During the subgraph matching the DSP constraints must be satisfied. Therefore, the subgraph is registered and indexed if it has only single branch between any two internal nodes. The depth-first search (DFS) strategy with respect to DSP constraints is implemented in proposed subgraph matching Algorithm 2.1.

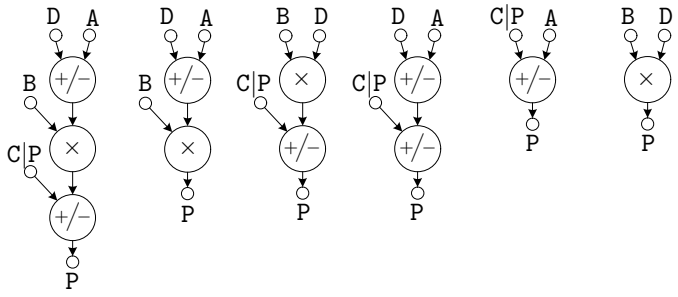


Fig. 2.2. The subgraphs compatible with configurations of digital signal processing slice

Algorithm 2.1 (The DFS and constraints based subgraph matching)

Require: Graphs: $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, $\mathcal{G}_{\text{DSP}}^{(x)} \mid x \in \{1, \ldots, 15\}$.

for i in 1 to $\mid \mathcal{G} \mid$ **do**
  if $\exists! x \in \{13, 14, 15\} : V_i = V_{\text{DSP},1}^{x}$ **then**
    $\text{reg}(\mathcal{V}_{\text{Sg}}^{(x,l(x))} = \{V_i\}, \mathcal{E}_{\text{Sg}}^{(x,l(x))} = 0); l(x)++.$
    if $\mid \mathcal{E}(V_i) \mid = 3$ **and** $\exists! m \in \{1, 2, 3\} : V_i = \text{init}(E_m)$ **then**
      $V_j = \text{ter}(E_m).$
      if $\exists x \in \{5, \ldots, 12\} : V_j = V_{\text{DSP},2}^{x}$ **then**
        $\text{reg}(\mathcal{V}_{\text{Sg}}^{(x,l(x))} = \{V_i, V_j\}, \mathcal{E}_{\text{Sg}}^{(x,l(x))} = \{E_m\}); l(x)++.$
        if $\mid \mathcal{E}(V_j) \mid = 3$ **and** $\exists! n \in \{1, 2, 3\} : V_j = \text{init}(E_n)$ **then**
          $V_k = \text{ter}(E_n).$
          if $\exists x \in \{1, \ldots, 4\} : V_j = V_{\text{DSP},3}^{(x)}$ **then**
            $\text{reg}(\mathcal{V}_{\text{Sg}}^{(x,l(x))} = \{V_i, V_j, V_k\}, \mathcal{E}_{\text{Sg}}^{(x,l(x))} = \{E_m, E_n\}); l(x)++.$
          end if
        end if
      end if
    end if
  end if
end for
return $\mathcal{V}_{\text{Sg}}^{(x,l(x))} \in \mathcal{V}, \mathcal{E}_{\text{Sg}}^{(x,l(x))} \in \mathcal{E}, l(x) \mid x \in \{1, \ldots, 15\}.$

The proposed subgraph matching algorithm return subsets of vertex $\mathcal{V}_{\text{Sg}}^{(x,l(x))}$ and edges $\mathcal{E}_{\text{Sg}}^{(x,l(x))}$ for detected subgraphs $\mathcal{G}_{\text{DSP}}^{(x,l(x))}$ in $\mathcal{G}$ and the number $l(x)$ of registered subgraphs for each type of subgraph x.

2.3.3. Graph Covering and Merging

The graph $\mathcal{G}$ covering is a $\mathcal{G}_{\text{DSP}}^{(x)}$ selection process, that attempts to find an appropriate set of subgraphs which minimizes resources usage and latency. The largest subgraphs $\mathcal{G}_{\text{DSP}}^{(x)}$ detected in graph $\mathcal{G}$ are covered first using greedy algorithm (Cong, Jiang 2008; Ronak, Fahmy 2014). The greedy covering algorithm adapted to our set of subgraphs is shown in Algorithm 2.2. It checks presence of largest subgraphs $\mathcal{G}_{\text{Sg}}^{(x,i)}$ first in $\mathcal{G}$ scanning through all subgraph types $x \in \{1, \ldots, 15\}$ and through all ith subgraphs belonging to particular type x. The covered subgraphs are registered in $\mathcal{G}_{\text{C}}^{(k)}$. Afterwards, all subgraphs $\mathcal{G}_{\text{Sg}}^{(y,j)}$ with overlapping vertices are eliminated from a set of candidates for further covering.

Algorithm 2.2 (Graph covering, subgraph elimination and merging)

Require: Graphs: $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, $\mathcal{G}_{\text{Sg}}^{(x)}, l(x) \mid x \in \{1, \ldots, 15\}$.

for x in 1 to 15 **do**
  for i in 1 to $l(x)$ **do**
    if $\mathcal{G}_{\text{Sg}}^{(x,i)}$ **is** valid **then**
      Cover $\mathcal{G}_{\text{Sg}}^{(x,i)}$ on $\mathcal{G}$: $\mathcal{G}_{\text{C}}^{(k)} = \mathcal{G}_{\text{Sg}}^{(x,i)}$, $k{+}{+}$.
      if $\exists\, \mathcal{V}_{\text{Sg}}^{(y,j)} \wedge \mathcal{V}_{\text{Sg}}^{(x,i)} \mid y \in \{1, \ldots, 15\}, j \in \{1, \ldots, l(y)\}$ **then**
        Eliminate candidate graph $\mathcal{G}_{\text{Sg}}^{(y,j)}$.
      end if
    end if
  end for
end for
for i in 1 to k **do**
  if $\mid \mathcal{V}_{\text{C}}^{(i)} \mid \geq 2$ **and** operator of $\mathcal{V}_{\text{C}}^{(i)}(\mid \mathcal{V}_{\text{C}}^{(i)} \mid)$ **is** $+/-$ **then**
    if $\exists! V^{(j)} \in \mathcal{V}_{\text{C}}^{(j)} : V^{j} = \text{init}\left(\mathcal{E}_{\text{C}}^{(i)}\left(\mathcal{V}_{\text{C}}^{(i)}(\mid \mathcal{V}_{\text{C}}^{(i)} \mid)\right)\right) \neq \mathcal{V}_{\text{C}}^{(i)}(\mid \mathcal{V}_{\text{C}}^{(i)} \mid -1)$ **then**
      if $\exists! \,\text{ter}\left(\mathcal{E}(V^{(j)})\right) = \mathcal{V}_{\text{C}}^{(i)}(\mid \mathcal{V}_{\text{C}}^{(i)} \mid)$ **then**
        Merge $\mathcal{G}_{\text{C}}^{(i)}$ with $\mathcal{G}_{\text{C}}^{(j)}$.
        Modify instruction and latency for $\mathcal{G}_{\text{C}}^{(i)}$.
      end if
    end if
  end if
end for
return $\mathcal{G}_{\text{C}}^{(x)} \mid x \in \{1, \ldots, k\}$.

The DSP has a property to add/subtract output value P of previous clock cycle through the multiplexed output feedback to last vertex of DSP subgraph, as shown in Fig. 1.2. Therefore, DSP must be reconfigured to access output P instead of input C (1.1). Such DSP configuration allows to process a subgraph in single clock cycle due to the pipelined subgraph processing. In order to apply the pipelining for two or more adjacent subgraphs, the pairs of subgraphs must be merged, when a common edge exists between two terminal vertices. We distinguish few important circuits inherent for LLMLP, where subgraph merging can be applied, as shown in Fig. 2.3. The graph for first order lattice consists of four subgraphs as shown in Fig. 2.3a. The $\mathcal{G}_{\text{C}}^{(1)}$ is merged with $\mathcal{G}_{\text{C}}^{(2)}$ and $\mathcal{G}_{\text{C}}^{(3)}$ with $\mathcal{G}_{\text{C}}^{(4)}$. The DSP instructions for subgraphs $\mathcal{G}_{\text{C}}^{(2)}$ and $\mathcal{G}_{\text{C}}^{(4)}$ are modified to $\texttt{P} = \texttt{P} + \texttt{D} \times \texttt{B}$. The graph of ladder filter (Fig. 2.3b) and sum of synapses (Fig. 2.3c) consists of merged subgraphs $\mathcal{G}_{\text{C}}^{(1)}$, $\mathcal{G}_{\text{C}}^{(2)}$ and $\mathcal{G}_{\text{C}}^{(3)}$. In a

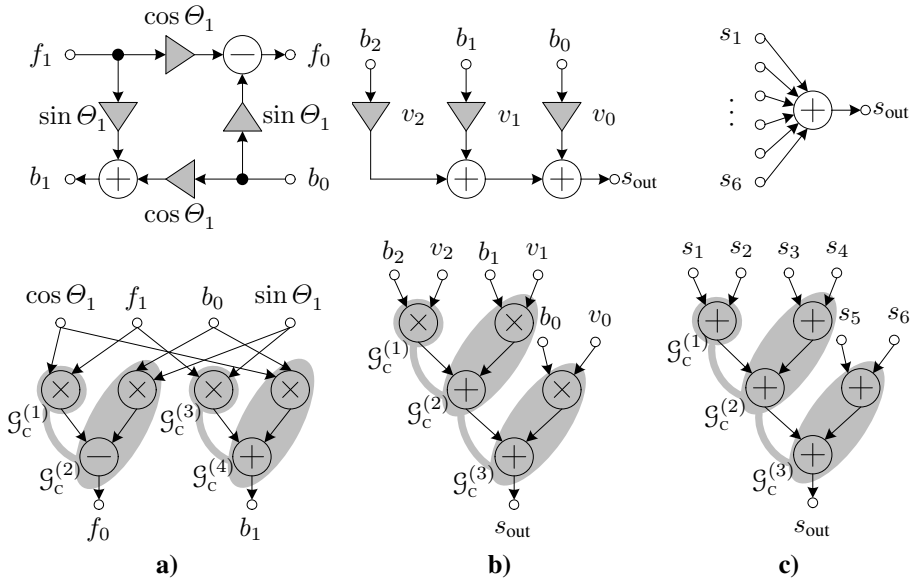


Fig. 2.3. The covered and merged subgraphs (in grey) for a) lattice filter, b) ladder filter, c) sum of synapses

ladder graph the instructions for $\mathcal{G}_{\mathrm{C}}^{(2)}$ and $\mathcal{G}_{\mathrm{C}}^{(3)}$ are modified to $\mathtt{P} = \mathtt{P} + \mathtt{D} \times \mathtt{B}$. The $\mathtt{P} = \mathtt{P} + \mathtt{D} + \mathtt{A}$ instruction is assigned to $\mathcal{G}_{\mathrm{C}}^{(2)}$, $\mathcal{G}_{\mathrm{C}}^{(3)}$ for sum of synapses graph. All modified instructions will be executed in single cycle. The covered and merged subgraphs $\mathcal{G}_{\mathrm{C}}^{(k)}$ are returned for further scheduling.

The example of first order synapse with t_3 type regressor DFG and its partitioning in subgraphs (in grey) is shown in Fig. 2.4. The merged subgraphs are connected by grey curves.

2.3.4. Critical Path Search

The scheduling is resources constrained due to limited FPGA resources for LLMLP implementation. Only single neuron with 5 inputs and 3rd order synapses is suitable to fit into FPGA without DSP sharing (Sledevič, Navakauskas 2015). Therefore, the arithmetic resources must be shared through other operations simultaneously exploiting the pipeline property of DSP. Moreover, each instruction must be exactly placed in time maintaining shortest latency for whole covered graph $\mathcal{G}_{\mathrm{C}}^{(x)}$ execution. It is known, that for resources unconstrained scheduling the critical path method (CPM) gives a shortest possible delay limited only by data dependence in critical path. For the reason that

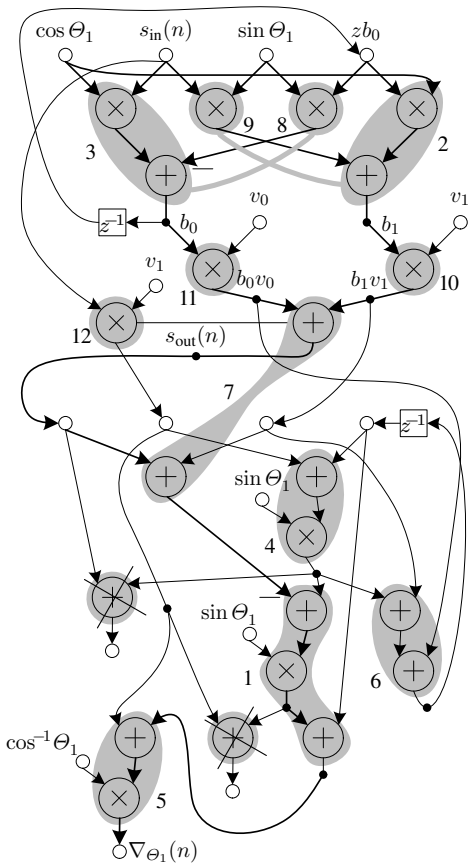


Fig. 2.4. A graph of the first order direct and $\mathfrak{t}_3$ regressor lattices and its partitioning into subgraphs (in grey) for digital signal processing slice

the results of first subgraph are needed by another before the first has finished executing, the pipeline hazard occurs, because the second subgraph must wait for the first to complete.

Giving a set of graphs $\mathcal{G}_{\mathrm{C}}^{(x)} \mid x \in \{1, \ldots, k\}$ for CPM it returns: the minimal latency needed for k subgraphs computation, the set of subgraphs $\mathcal{G}_{\mathrm{CP}}^{(x)}$ located on critical path and the vector t_{ES}^{x} with early start time values. The subgraphs, which are on the critical path (bold arrows in Fig. 2.4), must be executed with highest priority, since the latency is directly related with delay in critical path. The priority of all other subgraphs $\mathcal{G}_{\mathrm{C}}^{(x)} \notin \mathcal{G}_{\mathrm{CP}}^{(x)}$, which lie not on critical path, depends on the early start time. The earlier the $\mathcal{G}_{\mathrm{C}}^{(x)}$ starts, the higher is its priority. Assuming, that we have unconstrained resources with

unlimited interconnections, the schedule for first order synapse and its training circuit is shown in Fig. 2.5. The instructions connected by arrows lie on critical path. The dashed lines mark slack for instructions, which not influence the total latency.

2.3.5. Resource Constrained Scheduling

In a Gantt chart (Fig. 2.5) there are 12 instructions and each one requires a DSP slice to be executed. To make it implementable on single shared and pipelined DSP, the instructions must be scheduled according to hardware constraints, maintaining shortest possible delay in critical path. The pipelined resources can be shared, even if the corresponding operations overlap. This necessarily requires that the operations do not start in the same time step and no data dependency exists between the operations. Moreover, the buffers are required on DSP inputs to load signals synchronously in particular time step. As a consequence, each instruction is lengthen by 2 clock cycles. One cycle is needed to write result in buffer and second one to read signal from buffer.

The Gantt chart for resources constrained scheduling is shown in Fig. 2.6. The latency of $\mathcal{G}_{\mathrm{C}}^{(x)}$ calculation is shortest possible. It can be seen from the sequence of the scheduled instructions located exactly one after another in critical path and connected by directed arrows. The 9th instruction starts not at 0 time step, however 2 cycles later increasing total latency by 2 cycles. It is affected by resource constraints, since only single instruction can start at each discrete time step. The two sets of data dependent subgraphs $\{\mathcal{G}_{\mathrm{C}}^{(9)}, \mathcal{G}_{\mathrm{C}}^{(2)}, \mathcal{G}_{\mathrm{C}}^{(10)}\}$ and $\{\mathcal{G}_{\mathrm{C}}^{(8)}, \mathcal{G}_{\mathrm{C}}^{(3)}, \mathcal{G}_{\mathrm{C}}^{(11)}\}$ have same 15 clock cycles latency and both belong to

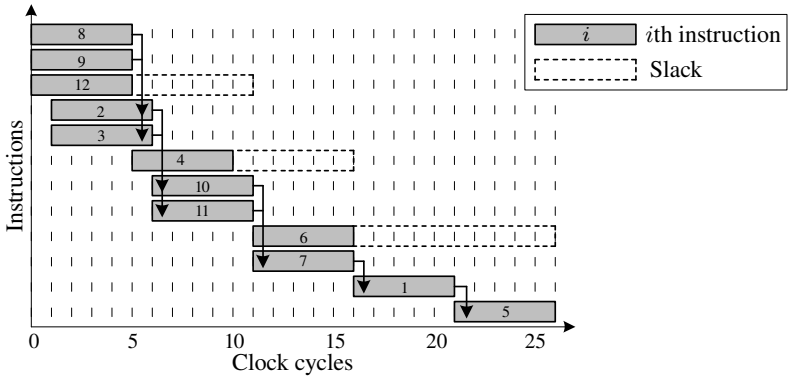


Fig. 2.5. The resources unconstrained instruction schedule for first order lattice-ladder and regressive lattice after critical path search

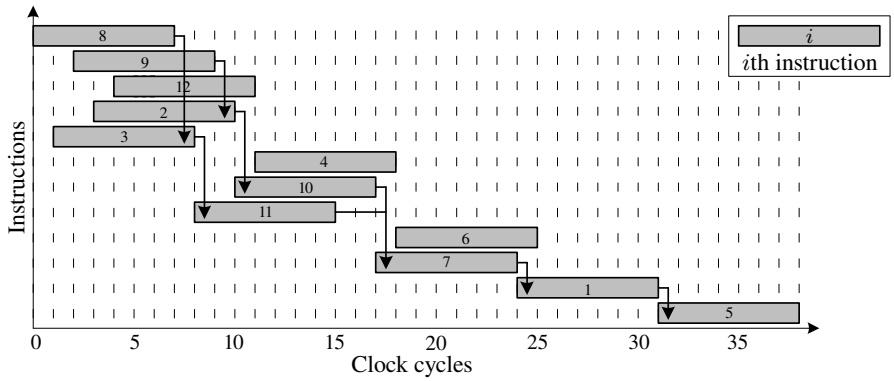


Fig. 2.6. The resource constrained schedule for first order lattice-ladder and regressive lattice graph

critical path, as shown in Fig. 2.5. Therefore, it is not important which one set $\{\mathcal{G}_{\mathrm{C}}^{(9)}, \mathcal{G}_{\mathrm{C}}^{(2)}, \mathcal{G}_{\mathrm{C}}^{(10)}\}$ or $\{\mathcal{G}_{\mathrm{C}}^{(8)}, \mathcal{G}_{\mathrm{C}}^{(3)}, \mathcal{G}_{\mathrm{C}}^{(11)}\}$ starts first, since $\mathcal{G}_{\mathrm{C}}^{(10)}$ and $\mathcal{G}_{\mathrm{C}}^{(11)}$ have same priority and both are source subgraphs for the $\mathcal{G}_{\mathrm{C}}^{(7)}$.

The proposed subgraph scheduling algorithm handles pipelined DSP resources by allowing the scheduling of overlapping operations with no data dependency and different start times. If two or more operations have same start time, then they are delayed until DSP will be released. The stages in Algorithm 2.3 lines 8–14 ensure the minimal latency for instruction execution. The algorithm includes the candidate subgraph $\mathcal{G}_{\mathrm{C}}^{(x)}$ to set of subgraphs $\mathcal{G}_{\mathrm{L}}^{(x)}$ on schedule list only if all predecessor vertices for $\mathcal{G}_{\mathrm{C}}^{(x)}$ are computed. Then it schedules exactly one subgraph from set $\mathcal{G}_{\mathrm{L}}^{(x)}$ with highest priority. Thus the subgraphs $\mathcal{G}_{\mathrm{CP}}^{(x)}$ located on critical path are scheduled first before all other subgraphs. The subgraphs $\mathcal{G}_{\mathrm{C}}^{(x)} \notin \mathcal{G}_{\mathrm{CP}}^{(x)}$ have a slack of free time steps and are scheduled with as soon as possible (ASAP) strategy. The algorithm returns instruction start vector of time steps $t_{\mathrm{S}}^{(x)}, x \in \{1, \ldots, k\}$, which is used next in HDL generation placing DSP instructions in right order.

Algorithm 2.3 (Resource constrained scheduling)

Require: Graphs: $\mathcal{G}_{\mathrm{C}}^{(x)} \mid x \in \{1, \ldots, k\}$, k – number of covered $\mathcal{G}_{\mathrm{C}}^{(x)}$.

1: $[\mathcal{G}_{\mathrm{CP}}^{(x)}, t_{\mathrm{ES}}^{(x)}, \tau^{\mathrm{CP}}] = \mathrm{CPM}(\mathcal{G}_{\mathrm{C}}^{(x)} \mid x \in \{1, \ldots, k\})$.
2: **if** $\exists\, \mathcal{G}_{\mathrm{CP}}^{(x)} \wedge \mathcal{G}_{\mathrm{C}}^{(x)} \mid x \in \{1, \ldots, k\}$ **then**
3: Set $\mathrm{pri}(\mathcal{G}_{\mathrm{C}}^{(x)}) = \max_x(\tau^{\mathrm{CP}} - t_{\mathrm{ES}}^{(x)})$ for $\mathcal{G}_{\mathrm{C}}^{(x)}$ on critical path.
4: **else**
5: Set $\mathrm{pri}(\mathcal{G}_{\mathrm{C}}^{(x)}) = \tau^{\mathrm{CP}} - t_{\mathrm{ES}}^{(x)}$ for all $\mathcal{G}_{\mathrm{C}}^{(x)} \notin \mathcal{G}_{\mathrm{CP}}^{(x)}$.

end if
while $\exists\, \mathcal{G}_{\mathrm{C}}^{(x)} \neq \mathcal{G}_{\mathrm{S}}^{(x)} \mid x \in \{1, \ldots, k\}$ **do**
  Update predecessors $\forall\ \mathcal{V}_{\mathrm{C}}^{(x)} : \mathcal{G}_{\mathrm{C}}^{(x)} = \mathcal{G}_{\mathrm{S}}^{(x)}$.
  if $\forall$ init$(\mathcal{E}_{\mathrm{C}}^{(x)})$ predecessors are ready for $\mathcal{E}_{\mathrm{C}}^{(x)} \in \mathcal{G}_{\mathrm{C}}^{(x)}$ **then**
    Add $\mathcal{G}_{\mathrm{C}}^{(x)}$ candidates to schedule list $\mathcal{G}_{\mathrm{L}}^{(x)} = \mathcal{G}_{\mathrm{C}}^{(x)}$.
  end if
  if $\exists!\, \mathcal{G}_{\mathrm{L}}^{(x)} : \mathrm{pri}(\mathcal{G}_{\mathrm{L}}^{(x)}) \geq \mathrm{pri}(\mathcal{G}_{\mathrm{L}}^{(y)}), y \in \{1, \ldots, k\}$ **and** $y \neq x$ **then**
    Schedule subgraph $\mathcal{G}_{\mathrm{S}}^{(x)} = \mathcal{G}_{\mathrm{L}}^{(x)}$ and remove $\mathcal{G}_{\mathrm{L}}^{(x)}$ form schedule list.
    Capture instruction start time $t_{\mathrm{S}}^{(x)} = t$.
    if $\mathcal{G}_{\mathrm{S}}^{(x)}$ merged with child subgraph **then**
      Reserve DSP for next cycle.
    else
      Release DSP resource.
    end if
  end if
  $t++$.
end while
return $t_{\mathrm{S}}^{(x)}$.

2.3.6. Design Description

The final architecture of processing element described in HDL is shown in Fig. 2.7. It contains: single DSP as a configurable subgraph execution unit, four buffers connected to DSP inputs and the memory dedicated for nonlinear functions. The index generator consists of binary counter. With this counter a proper instruction is selected at each clock cycle from a set of instructions. The size of memory buffer depends only on the instruction width. The instruction set format consists of: four write/read addresses, DSP operation select bits, multiplexer select bits and nonlinear functions select bits in address modification block. The data buffers as well as nonlinear functions are implemented on dual (left and right) port memory. The left port of buffer is a write port, through which multiplexed data from: NPE input, result of nonlinear function or DSP output is stored in buffers. Data are loaded to DSP through right port of buffers. The DSP output `P` is: returned back to buffers, connected to NPE output, used as argument in nonlinear activation function accessible through left port of dual port ROM. The $\sin\Theta_j$ $\cos\Theta_j$, $\cos^{-1}\Theta_j$ tables are accessed through the right port of ROM. Due to the additional multiplexer requirement and buffer limitation to write only single value on clock cycle, the four nonlinear functions are implemented on the same ROM. The access of $\tanh$ `P` do not need any address modification, since the highest two bits are zeros. To access the $\sin\Theta_j$ $\cos\Theta_j$, and $\cos^{-1}\Theta_j$ the address of right port is modified changing

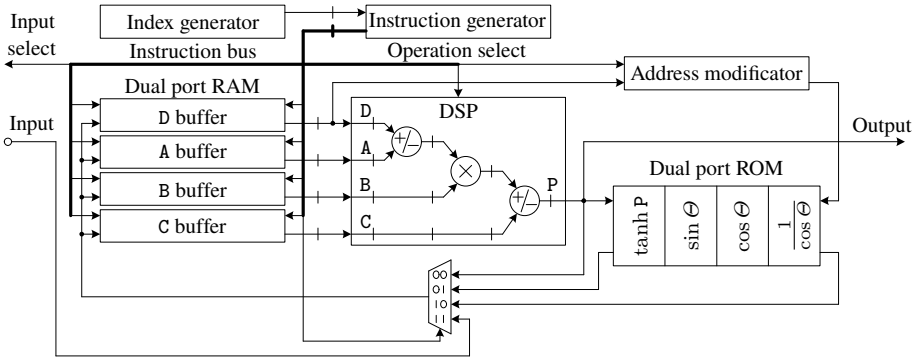


Fig. 2.7. The processing element based on single digital signal processing (DSP) slice

the highest two bits to "01", "10", "11" respectively in address modification core.

Due to the simplification the delay elements in Fig. 2.7 are marked by "|" crossing data lines. Through the read/write operation memories brings additional single clock cycle latency, therefore the minimal latency of single instruction execution is equal to seven clock cycles: one cycle to read data from buffer, five cycles to process a certain subgraph and last one to write result back to buffer. The two cycle latency of RAM and four cycle latency of DSP are determined by hardware constraints. The additional delay elements on each data line between buffers and DSP was inserted during the experimental investigation of the NPE performance. The experiments were done changing the number of pipeline stages (delay elements) also replacing the training circuits of LLMLP and generating the HDL file of NPE (Fig. 2.7). The timing analysis after PAR gives the maximum clock frequency $f^{\top}$ at which the implemented design can run (see Table 2.2). The latency and execution time are averaged over LLF orders $M = \{1, \ldots, 10\}$ and four different training circuits. An increased number of pipeline stages results in increased $f^{\top}$, but also contributes to an increase of latency taken to complete the processing of an instruction. Therefore, the time optimal instruction execution was achieved with one additional flip-flop (delay element) between flip-flops located in buffers output and DSP inputs, as shown in Table 2.2 and in Fig. 2.8.

Only the forward pass synapse and its regressor lattice is used in the investigation of optional pipeline stages in the NPE, since all other particular circuits which form LLMLP are independent on training type and are same for all the configurations of LLMLP. To identify an optimal number of optional pipeline stages in NPE, the execution time of forward pass synapses $\tau^{s\hookrightarrow}_{M,\mathfrak{T},f^{\top}}$

Table 2.2. Timing results of neuron processing element

Optional pipeline stages	0	1	2	3	4
Average latency, cycles	613	668	728	798	869
$f^{\top}$, MHz	207	320	339	339	339
Average execution time, μs	2.96	2.09	2.15	2.35	2.56

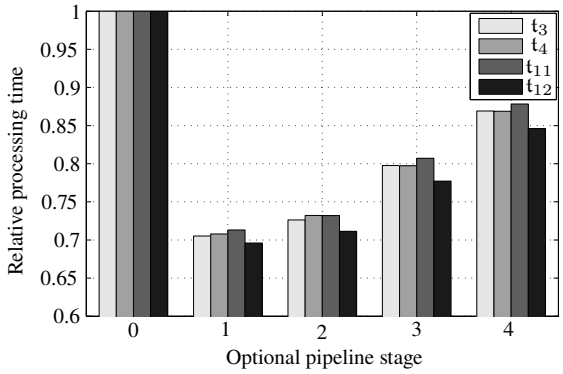


Fig. 2.8. The relative processing time dependence on optional pipeline stage and training type

must be compared between order M, train type $\mathcal{T}$ and maximal clock frequency $f^{\top}$ of the design. For the comparison purposes, the execution time $\bar{R}^{\text{time}}_{M,\mathcal{T},f^{\top}}$ is averaged over synapse order M in (2.13) and normalized to the highest value over four types of investigated regressors in (2.14):

$$\bar{R}^{\text{time}}_{M,\mathcal{T},f^{\top}} = \frac{1}{M}\sum_{m=1}^{M} \tau^{\text{s}\hookrightarrow}_{m,\mathcal{T},f^{\top}} \Big/ f^{\top}, \tag{2.13}$$

$$\widehat{R}^{\text{time}}_{M,\mathcal{T},f^{\top}} \equiv \bar{R}^{\text{time}}_{M,\mathcal{T},f^{\top}} \Big/ \max_{\mathcal{T}} \bar{R}^{\text{time}}_{M,\mathcal{T},f^{\top}}, \tag{2.14}$$

here $\mathcal{T}$ – regressor type, $\mathcal{T} \in \{\text{t}_3, \text{t}_4, \text{t}_{11}, \text{t}_{12}\}$; $f^{\top}$ – maximum clock frequency, which is investigated changing the number of pipeline stage; $\widehat{R}^{\text{time}}_{M,\mathcal{T},f^{\top}}$ – execution time of NPE relative to highest average execution time $\bar{R}^{\text{time}}_{M,\mathcal{T},f^{\top}}$ over different regressor. For all the training types $\mathcal{T}$ the fastest execution is achieved with single optional pipeline stage, as shown in Fig. 2.8.

Maintaining synapse parallelism the LLN with N inputs consists of $N+1$ NPE, as shown in Fig. 2.9. The weight updating circuit, forward and regressor lattice of the certain nth input is implemented on nth NPE. The additional NPE is used for the summation of synapse outputs and instantaneous error cal-

culation. To form a layer with N neuron parallelism the single neuron design (Fig. 2.9) must be implemented N times.

The scheduled list of instructions for DSP generated by our technique can also be used in NPE created in Xilinx System Generator (XSG) tool. In this case, the NPE must be assembled manually in XSG by adding input buffers, memories for instructions and nonlinear functions, multiplexers and delay elements to the DSP slice. After the creation of NPE (Fig. 2.7) it remains only to load four initial coefficient files for instruction indices, instructions, nonlinear functions and buffers.

2.4. Neuron Layers Optimization

Two strategies are proposed for neuron layers optimization. Throughput optimized strategy, when NPE is dedicated for each forward and backward synapse and node. The layer parallelism is reduced untill LLMLP structure fits in the constrained FPGA resources. The resources optimized implementation strategy begins from LLMLP implementation on single NPE and iteratively adds new NPE to the design untill meets processing time requirements.

2.4.1. Accuracy Optimization

A. Bit Width Selection

The workflow for the constrained lattice-ladder neuron implementation analysis is shown (Fig. 2.10). Program Matlab is used to create a reference design working on floating-point data. The essential for the investigation parameters are two K length vectors of the lower $\boldsymbol{f}_{\mathrm{l}}$ and the upper $\boldsymbol{f}_{\mathrm{h}}$ cut-off frequencies that determine various band-pass Mth order lattice-ladder synapses. Other investigation common parameters are N length input and desired output sig-

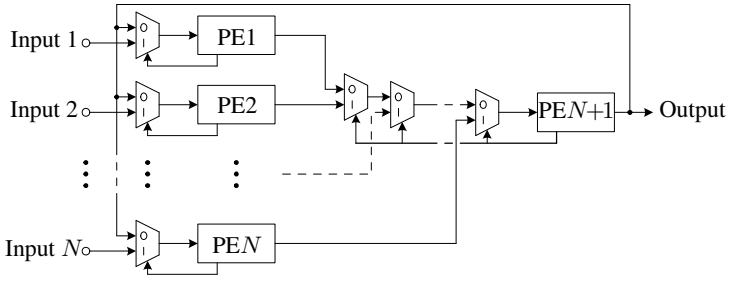


Fig. 2.9. The architecture of neuron with synapse parallelism

nals ($\boldsymbol{s}_{\text{in}}$, $\boldsymbol{s}^{\star}_{\text{out}}$) to be used for training and validation. Program Vivado HLS from Xilinx is used to create, simulate and analyse constrained design. Constraint parameters: precision of fixed-point arithmetic in W bits and BRAM size. After the simulations of floating-point and fixed-point designs the LLN parameters Θ_j and v_j are compared to determine the bandwidth and band-pass central frequency discrepancies.

The latency of $\sin\Theta_j$ and $\cos\Theta_j$ values generation on parallel CORDIC core is directly proportional to the word length used for Θ_j variables. Therefore, CORDIC is expensive for high accuracy. Alternative approach could be to store in memory only quarter of sine period and to generate the rest values by simple manipulations with values and memory addresses, i.e., interpret Θ_j as an address in the LUT. During the investigation $T = 16\,\text{k}$ addresses are allocated in BRAM to form quarter of sine period (similarly for inverse cosine) and that was not changed even when word length of data was varied.

The Vivado HLS lets to create a new $W = W_{\text{i}} + W_{\text{f}}$ length fixed-point types with the desired number of bits for integer W_{i} and fractional W_{f} parts. The overflow of integer bits is avoided by enabling saturation, thus value of the signal never exceeds the margins, e.g., $[-4, 4)$ for $W_{\text{i}} = 3$.

The neuron training step needs to be constrained from both sides. The lower bound of μ needs to be restricted because during weights Θ_j and v_j updates in (1.5) and (1.6) the product of three small parameters must be considered. Therefore the precision of $\mu\delta\nabla_{\Theta_j}(n)$ and $\mu\delta\nabla_{b_j}(n)$ is doubled inten-

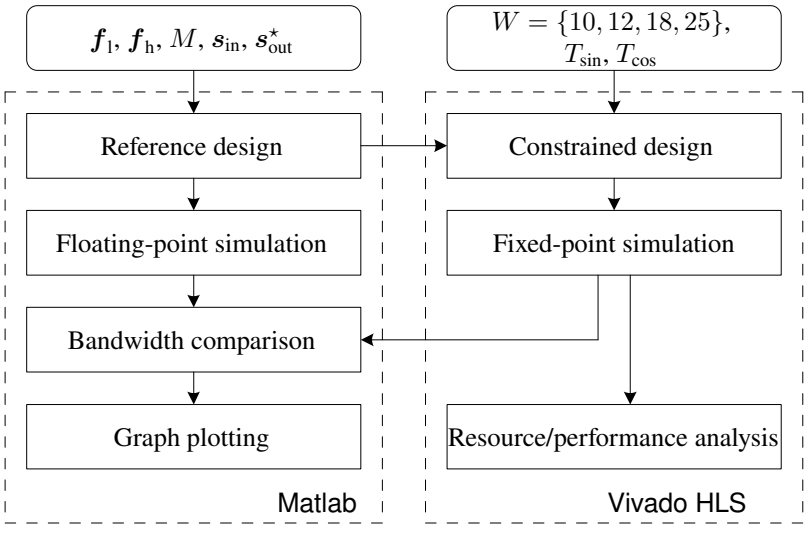


Fig. 2.10. The workflow for investigation of a fixed-point implementation of lattice-ladder linear neuron and its training circuit

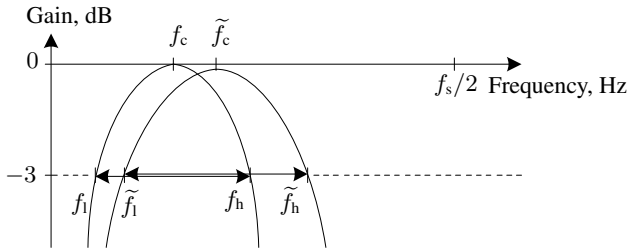


Fig. 2.11. Illustration of bandpass discrepancies measurements

tionally only in the weights update loops. The upper bound of μ is restricted by Θ_j and v_j parameter convergence to the reference. Increasing μ will generally accelerate adaptation process and it is advised to increase μ to the largest value for which convergence is observed Regalia (1995). The largest value of μ beyond which convergence no longer occurs also depends on precision. The lower precision is, the larger μ must be used to ensure adjustment of Θ_j and v_j parameter.

The band-pass frequencies (measured at gain of -3 dB, see Fig. 2.11) for floating-point and fixed-point designs can be expressed by $f_b \triangleq f_h - f_l$ and $\widetilde{f}_b \triangleq \widetilde{f}_h - \widetilde{f}_l$, accordingly, when by a tilde symbol constrained design results are outlined. Because f_l and f_h frequencies shifts independently and does not yield desired central frequency $f_c^\star$, central frequencies of both design types $f_c \triangleq f_l + f_b/2$ and $\widetilde{f}_c \triangleq \widetilde{f}_l + \widetilde{f}_b/2$ must be measured, too.

In order to investigate the constrained implementation quality independent on the reference bandwidth f_b to be varied, we fix desired central frequency $f_c^\star \equiv f_s/4$ and use relative errors of bandwidth ϵ_b and central frequency ϵ_c expressed by:

$$\epsilon_b\left(f_b\right) \triangleq \frac{\left|f_b - \widetilde{f}_b\right|}{f_b}, \qquad \epsilon_c\left(f_b\right) \triangleq \frac{\left|f_c - \widetilde{f}_c\right|}{f_b}. \tag{2.15}$$

Additionally output signal accuracy is determined by l_1-norm construct – *Normalized Mean Absolute Error* (normalized MAE) of reference and constrained designs output signals:

$$\mathscr{E}_{\text{MA}} \triangleq \left(\frac{1}{N}\sum_{n=1}^{N}\left|s_{\text{out}}(n) - \widetilde{s}_{\text{out}}(n)\right|\right) \Big/ \left(\frac{1}{N}\sum_{n=1}^{N}\left|s_{\text{out}}(n)\right|\right). \tag{2.16}$$

See Chapter 4 for the use of defined error indicators in the evaluation of experimentation results.

B. Nonlinearity Implementation

The proposed neuron activation function is divided in three regions: pass, processing and saturation (see Fig. 2.12). In a pass region $\Phi_{\text{tanh}} \equiv \Phi_{\text{ident}} \equiv s$, the signal $s(n)$ is directly transferred to the output $s_{\text{out}}(n)$. The processing region of the function $\Phi_{\text{tanh}}(s)$ is implemented in table and stored in BRAM. In a saturation region, the LLN output is always $\Phi_{\text{tanh}} \equiv 1$.

The hyperbolic tangent function is defined as follows:

$$\widetilde{\Phi}_{\text{tanh}}\big(s(n)\big) \triangleq \frac{e^{2.03g\bar{s}(n)} - 1}{e^{2.03g\bar{s}(n)} + 1}, \tag{2.17}$$

here $s(n) = g\bar{s}(n)$ is the input of activation function.

To make the smoothed junction between the pass and processing regions, the exponent value is set to 2.03 instead of original 2. The approximated LLN output can be expressed:

$$s_{\text{out}} = \begin{cases} 1, & s(n) \geq 16; \\ \widetilde{\Phi}_{\text{tanh}}\big(s(n)\big), & 16 > s(n) > 1; \\ s(n), & -1 \leq s(n) \leq 1; \\ -\widetilde{\Phi}_{\text{tanh}}\big(s(n)\big), & -16 < s(n) < -1; \\ -1, & s(n) \leq 16. \end{cases} \tag{2.18}$$

The activation function output of negative value is obtained using asymmetry principle $\Phi_{\text{tanh}}\big(-s(n)\big) = -\Phi_{\text{tanh}}\big(s(n)\big)$. The amplification coefficient g controls the output range of the hyperbolic tangent. If $g = 1$, then activation function is linear. For $g > 1$, the nonlinearity in the LLN output grows. The higher gain is the wider range of LUT is accessible. If $g = 16$, then $s_{\text{out}}(n)$ varies in range $[1, 1]$. The amplification of $\bar{s}(n)$ more than 16 times is equivalent to the squeeze of activation function or similarly to increase of the slope of hyperbolic tangent for $s(n)$ in range $[1, 1]$.

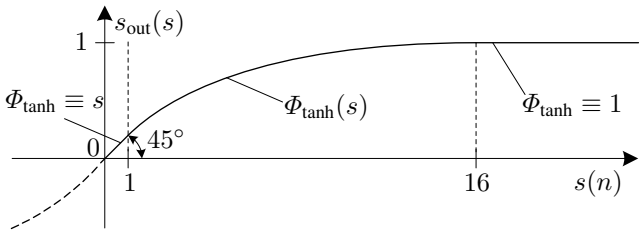


Fig. 2.12. The plot of the proposed activation function

Program Matlab is used to create a reference design working on floating-point data. Program Vivado HLS is used to create, simulate and analyse constrained design with fixed-point precision. After the simulations of floating-point and fixed-point designs, the corresponding output signals $s_{\text{out}}(n)$ and $\widetilde{s}_{\text{out}}(n)$ are compared to determine the synapse transfer function and the output signal discrepancies.

The accuracy of the hyperbolic tangent approximation is evaluated using l_1-norm derived and commonly used indicator – *Mean Absolute Error* (MAE), that will depend on memory size R_{BRAM} and gain g (Armato *et al.* 2011):

$$\mathscr{E}_{\text{MA}}\big(g, R_{\text{BRAM}}\big) \triangleq \frac{1}{n}\sum_{n=0}^{N-1}\Big|s_{\text{out}}\big(n\big) - \widetilde{s}_{\text{out}}\big(n; g, R_{\text{BRAM}}\big)\Big|. \tag{2.19}$$

Moreover, fixing gain to its maximum value ($g = 16$), the accuracy of the hyperbolic tangent approximation is also evaluated using l_∞-norm – *Maximum Magnitude Error* (MME), when memory size R_{BRAM} and gain g of $s(n)$ signal varies (Armato *et al.* 2011):

$$\mathscr{E}_{\text{MM}}\big(R_{\text{BRAM}}\big) \triangleq \max_{\substack{n \\ g=16}}\Big|s_{\text{out}}\big(n\big) - \widetilde{s}_{\text{out}}\big(n; g, R_{\text{BRAM}}\big)\Big|. \tag{2.20}$$

The lattice-ladder can be set to work as nonlinear low, high, band-pass or band-stop filter with additional gain control. Such a system has the transfer function, which will have distortions dependent on the limited LUT size dedicated to the hyperbolic tangent function. To check the accuracy of the LLN transfer function, the LLN must be scanned by $\sin(n)$ signal, that contains all the frequency components in range $[0, f_{\text{s}}/2]$, where f_{s} is signal sampling frequency. The frequency of $\sin(n)$ must linearly change in time. This property has the chirp signal $s_{\text{chirp}}(n)$:

$$s_{\text{chirp}}(n) = \cos\big(2\pi f_i(n) + \phi_0\big), \tag{2.21}$$

with instantaneous frequency sweep function expressed by:

$$f_i(n) = f_0 + \frac{f_1 - f_0}{n_1}n, \tag{2.22}$$

here f_0 and f_1 are desired starting and breakpoint frequencies at time $n = 0$ and $n = n_1$; $\phi_0 = 0$ – signal initial phase.

To check the worst case of LLN transfer function implementation, the synapse parameters $\Theta_m = v_m \;\forall\; m \in [1, M]$ except $v_0 = 1$ are set to pass through all frequency components to the input of activation function.

The transfer function of LLN activation function is obtained by the estimator (Broersen 1994):

$$T(f) = \frac{\mathcal{P}_{\text{xy}}(f)}{\mathcal{P}_{\text{xx}}(f)}, \tag{2.23}$$

here $\mathcal{P}_{\text{xy}}(f)$ – cross power spectral density of input $s_{\text{in}}(n)$ and output $s_{\text{out}}(n)$ signals; $\mathcal{P}_{\text{xx}}(f)$ – auto power spectral density of $s_{\text{in}}(n)$, when $s_{\text{in}}(n) \equiv s_{\text{chirp}}(n)$ $\forall\, n \in [1, N]$.

The distortions between reference and proposed LLN transfer functions are evaluated through corresponding amplitude responses by use of from l_2-norm derived and commonly used indicator – *Root Mean Square Error* (RMSE):

$$\mathscr{E}^{\text{AR}}_{\text{RMS}}\big(g, R_{\text{BRAM}}\big) \triangleq \sqrt{\frac{2}{f_{\text{s}}} \sum_{f=0}^{f_{\text{s}}/2} \Big(\big|T_{\text{LNN}}(f; g)\big| - \big|\widetilde{T}_{\text{LNN}}(f; g, R_{\text{BRAM}})\big| \Big)^2}, \tag{2.24}$$

here $\big|T_{\text{LNN}}(f; g)\big|$, $\big|\widetilde{T}_{\text{LNN}}(f; g, R_{\text{BRAM}})\big|$ – floating-point and fixed-point implementation LLN amplitude response with pre-selected gain g and BRAM size R_{BRAM} calculated by (2.23).

2.4.2. Throughput Optimized Implementation Strategy

The total latency of the lattice-ladder multilayer perceptron training consists of accumulated delay of the signal forward pass and error backpropagation. The latency in a forward and backward LL synapse is defined as $\tau^{\hookrightarrow}_{\mathcal{T},l,M^{(l)}}$ and $\tau^{\hookleftarrow}_{l,M^{(l)}}$ respectively. The latency in a forward and backward node is defined as $\tau^{\hookrightarrow}_{l,N^{(l-1)}}$ and $\tau^{\hookleftarrow}_{l,N^{(l+1)}}$ respectively. The total latency depends on the structure of the desired network. Before the optimization by latency there are known only the LLMLP settings: L, $N^{(l)}$, $M^{(l)}$ and the available resources $\overline{R} = \{\overline{R_{\text{DSP}}}, \overline{R_{\text{LUT}}}, \overline{R_{\text{RAM}}}\}$ on selected FPGA. Depending on above parameters the LLMLP is implemented through using variable degree of parallelism (Savich *et al.* 2012). The fully parallel design is possible only when it is enough FPGA resources for processing signals in all synapses and nodes in parallel. The best performance is achieved with pipelined design and layers parallel weight update rule when next condition is satisfied for each resource type $x = \{\text{DSP}, \text{LUT}, \text{RAM}\}$:

$$2 \sum_{l=2}^{L} \Big(N^{(l-1)} + 1 \Big) N^{(l)} < w_x \overline{R_x}. \tag{2.25}$$

The latency is defined by the maximal signal delay in one of the four pipelined stages (forward synapse, forward node, backward synapse, backward node):

$$\tau = \max_{2\le l\le N^{(L)}} \left(\min_{\tau}(\tau^{\mathrm{s}\hookrightarrow}_{\mathcal{T},l,M^{(l)}}), \tau^{\mathrm{s}\hookleftarrow}_{l,M^{(l)}}, \tau^{\mathrm{n}\hookrightarrow}_{l,N^{(l-1)}}, \tau^{\mathrm{n}\hookleftarrow}_{l,N^{(l+1)}} \right). \tag{2.26}$$

When the condition (2.25) is not meet the LLMLP is still implemented with synapse and node parallelism, however only for single layer with maximum number of synapses:

$$2 \max_{2\le l\le N^{(L)}} \left((N^{(l-1)}+1)N^{(l)} \right) < w_x \overline{R_x}. \tag{2.27}$$

Such LLMLP design is shared through all layers, therefore latency is a sum of maximum delays in forward pass or backward pass circuits over all layers:

$$\tau = \sum_{l=2}^{L} \left(\max \left(\min_{\tau}(\tau^{\mathrm{s}\hookrightarrow}_{\mathcal{T},l,M^{(l)}}) + \tau^{\mathrm{n}\hookrightarrow}_{l,N^{(l-1)}}, \tau^{\mathrm{s}\hookleftarrow}_{l,M^{(l)}} + \tau^{\mathrm{n}\hookleftarrow}_{l,N^{(l+1)}} \right) \right). \tag{2.28}$$

When it is not enough resources $w_x \overline{R_x}$ to fit single layer of LLMLP on FPGA, the same resources are shared through forward and backward pass circuits, if the next term is satisfied beginning from $j = 1$:

$$\max_{2\le l\le N^{(L)}} \left((N^{(l-1)}+1)N^{(l)} \right) \Big/ j < w_x \overline{R_x}, \tag{2.29}$$

here $1 \le j \le N^{(l^\star)}$, when

$$l^\star = \arg\max_{l} \left(N^{(l-1)} N^{(l)} \right), \tag{2.30}$$

here $l^\star$ is a layer with maximal number of synapses.

If the design does not meet (2.29) requirement, the j will be increased until single neuron fits constrained FPGA resources. On this step the synapse parallelism is still maintained, however the node parallelism decreases sequentially with rising j. For the above condition latency can be described by:

$$\tau = \sum_{l=2}^{L} \left(\min_{\tau}(\tau^{\mathrm{s}\hookrightarrow}_{\mathcal{T},l,M^{(l)}}), \tau^{\mathrm{n}\hookrightarrow}_{l,N^{(l-1)}}, \tau^{\mathrm{s}\hookleftarrow}_{l,M^{(l)}}, \tau^{\mathrm{n}\hookleftarrow}_{l,N^{(l+1)}} \right) j. \tag{2.31}$$

And finally, if in $N^{(l-1)}$ layer the number of synapses exceeds the available resources of field programmable gate array, then the synapse parallelism will

be decreased with rising synapse parallelism reduction index i. The $N^{(l-1)}/i$ synapses will be shared through all other synapses in $N^{(l)}$ layer if the term is satisfied:

$$\max_{2\leq l\leq N^{(L)}} \left(N^{(l-1)}/i+1\right) < w_x\overline{R_x}, \tag{2.32}$$

here i increases in range $1 \leq i \leq N^{(\arg\max_l N^{(l-1)})}$. The latency of such LLMLP can be estimated by:

$$\tau = \sum_{l=2}^{L} \left(i \min_{\tau}(\tau^{\text{s}\hookrightarrow}_{\mathcal{T},l,M^{(l)}}), i\tau^{\text{n}\hookrightarrow}_{l,N^{(l-1)}}, \tau^{\text{s}\hookleftarrow}_{l,M^{(l)}}, \tau^{\text{n}\hookleftarrow}_{l,N^{(l+1)}} \right) N^{(L)}. \tag{2.33}$$

The algorithm for throughput optimal LLMLP implementation is summarized in Algorithm 2.4.

Algorithm 2.4 (Throughput optimal LLMLP implementation)

Require: Load settings: L, N, M and FPGA resources $\overline{R_x} = \{\overline{R_{\text{DSP}}}, \overline{R_{\text{DSP}}}, \overline{R_{\text{DSP}}}\}$.

Compute latency matrices: $\tau^{\text{s}\hookrightarrow}_{\mathcal{T},l,M^{(l)}}, \tau^{\text{s}\hookleftarrow}_{l,M^{(l)}}, \tau^{\text{n}\hookrightarrow}_{l,N^{(l-1)}}, \tau^{\text{n}\hookleftarrow}_{l,N^{(l+1)}}$.
Compute the resources utilization $R_x(\mathcal{T},L,N,M)$ for fully parallel LLMLP.
if $R_x(\mathcal{T},L,N,M) < w_x\overline{R_x}$ is **true for** all x **then**
  Implement LLMLP with maximal performance.
  return latency τ computed by (2.26).
end if
Change topology of LLMLP to single layer and compute $\rho(\mathcal{T},L,N,M)$.
if $R_x(\mathcal{T},L,N,M) < w_x\overline{R_x}$ is **true for** all x **then**
  Implement single layer of LLMLP and share through whole net.
  return latency τ computed by (2.28).
end if
Compute $R_x(\mathcal{T},L,N,M)$ for single layer with shared forward/backward circuit.
if $R_x(\mathcal{T},L,N,M) < w_x\overline{R_x}$ is **true for** all x **then**
  Implement single layer and share it through forward/backward circuits.
  return latency τ computed by (2.31) when $j = 1$.
end if
for j in 2 to $N^{l^\star}$ **do**
  Decrease node parallelism by j for single layer and compute $R_x(\mathcal{T},L,N,M)$.
  if $R_x(\mathcal{T},L,N,M) < w_x\overline{R_x}$ is **true for** all x **then**
    Implement single layer LLMLP with j reduced node parallelism.
    return latency τ computed by (2.31).
  end if
end for
for i in 1 to $N^{(\arg\max_l(N^{(l-1)}))}$ **do**
  Compute $R_x(\mathcal{T},L,N,M)$ for a neuron with i reduced synapse parallelism.
  if $R_x(\mathcal{T},L,N,M) < w_x\overline{R_x}$ is **true for** all x **then**

27: Implement single neuron with i reduced synapse parallelism.
28: **return** latency τ computed by (2.33).
29: **end if**
30: **end for**
31: **return** insufficient resources.

If the FPGA resources do not meet all the conditions: (2.25), (2.27), (2.29), (2.32), then the desired complexity LLMLP can not be implemented on the verified FPGA. This problem can be solved with the simplification of LLMLP structure or switching to resource-rich FPGA.

2.4.3. Resource Optimized Implementation Strategy

In this case, the required latency τ^{req} is given together with LLMLP settings: $L, N^{(l)}, M^{(l)}$. According to above parameters the LLMLP is implemented with minimal resources. The new processing element NPE is sequentially added to the design, until the latency condition will be satisfied. The whole LLMLP fits to single NPE if the next requirement is meet for $i = j = k = 1$:

$$\sum_{l=2}^{L}\Bigg(\Big(\big(\min_{\tau}(\tau^{\text{s}\hookrightarrow}_{\Im,l,M^{(l)}}) + \tau^{\text{s}\hookleftarrow}_{l,M^{(l)}}\big)N^{(l-1)}\Big/i + \\ + \big(\tau^{\text{n}\hookrightarrow}_{l,N^{(l-1)}} + \tau^{\text{n}\hookleftarrow}_{l,N^{(l+1)}}\big)\Big)N^{(l)}\Big/j\Bigg) < \tau^{\text{req}}, \tag{2.34}$$

here $1 \leq i \leq N^{l^\star-1}$ is the number of synapses in layer $l^\star - 1$ covered by NPEs; $1 \leq j \leq N^{l^\star}$ is the number of neurons in layer $l^\star$ covered by NPEs. The synapse and node parallelism grows with the increasing i and j accordingly. The number of NPEs is:

$$R_{\text{PE}} = (i+1)jk, \tag{2.35}$$

here k shows how the forward and backward (fw/bf) pass circuits are implemented: on the same NPE ($k = 1$) or on different NPE ($k = 2$). When on different NPE, then latency is defined by maximal delay in forward or backward pass and it must meet the conditions:

$$\sum_{l=2}^{L}\left(\max\left(\min_{\tau}(\tau^{\text{s}\hookrightarrow}_{\Im,l,M^{(l)}}) + \tau^{\text{n}\hookrightarrow}_{l,N^{(l-1)}}, \tau^{\text{s}\hookleftarrow}_{l,M^{(l)}} + \tau^{\text{n}\hookleftarrow}_{l,N^{(l+1)}}\right)\right) < \tau^{\text{req}}. \tag{2.36}$$

If the above requirements are still not satisfied, then all the layers of lattice-ladder multilayer perceptron are implemented in parallel. Therefore, the la-

tency, defined by the maximal signal delay in one of the four pipelined stages, must be less than required latency:

$$\max_{2\leq l\leq N^{(L)}} \left(\min_{\tau}(\tau^{\text{s}\hookrightarrow}_{\mathcal{T},l,M^{(l)}}), \tau^{\text{s}\hookleftarrow}_{l,M^{(l)}}, \tau^{\text{n}\hookrightarrow}_{l,N^{(l-1)}}, \tau^{\text{n}\hookleftarrow}_{l,N^{(l+1)}} \right) < \tau^{\text{req}}. \tag{2.37}$$

The number of NPE for pipelined LLMLP implementation is:

$$R_{\text{PE}} = 2\sum_{l=2}^{L} \Big(N^{(l-1)} + 1 \Big) N^{(l)}. \tag{2.38}$$

The algorithm for resource optimal LLMLP implementation is summarized in Algorithm 2.5.

Algorithm 2.5 (Resource optimal LLMLP implementation)

Require: Load settings: L, N, M and required latency τ^{req}.
Compute latency matrices: $\tau^{\text{s}\hookrightarrow}_{\mathcal{T},l,M^{(l)}}$,$\tau^{\text{s}\hookleftarrow}_{l,M^{(l)}}$,$\tau^{\text{n}\hookrightarrow}_{l,N^{(l-1)}}$,$\tau^{\text{n}\hookleftarrow}_{l,N^{(l+1)}}$.
Check single layer with i synapse, j node parallelism and shared fw/bw circuits:
for j in 1 to $N^{l^\star}$ **do**
  for i in 1 to $N^{l^\star-1}$ **do**
    Compute $R(\mathcal{T}, L, N, M)$ for a neuron with i reduced synapse parallelism.
    if (2.34) is **true then**
      Implement the LLMLP on $j(i+1)$ NPEs.
      Calculate resources $R(\mathcal{T}, L, N, M)$.
      return $R(\mathcal{T}, L, N, M)$.
    end if
  end for
end for
Check fully parallel layer with separate circuits for forward and backward pass:
if (2.36) is **true then**
  Implement shared layer on $2N^{l^\star}(N^{l^\star}+1)$ NPEs.
  Calculate resources $R(\mathcal{T}, L, N, M)$.
  return $R(\mathcal{T}, L, N, M)$.
end if
Check fully parallel LLMLP:
if (2.37) is **true then**
  Implement LLMLP on $2\sum_{l=2}^{L}\Big(N^{(l-1)}+1\Big)N^{(l)}$ NPEs.
  Calculate resources $R(\mathcal{T}, L, N, M)$.
  return $R(\mathcal{T}, L, N, M)$.
end if
return τ^{req} is too low.

The desired complexity LLMLP can not be implemented on FPGA if the required latency τ^{req} do not meet all (2.34), (2.36), (2.37) conditions.

2.5. Conclusions of the 2nd Chapter

1. The technique for lattice-ladder multilayer perceptron (LLMLP) efficient implementation in FPGA is created and experimentally affirmed that structure of proposed neuron processing element is speed optimal with only 7 clock cycle latency.
2. The application of proposed lattice-ladder specific subgraph matching, covering, merging and scheduling algorithms ensures latency efficient schedule of the generated instructions for DSP, keeping the maximal clock frequency of neuron processing element IP core at 320 MHz.
3. Pareto frontiers estimation for the LLMLP specialized criteria of circuitry synthesis is proposed. Two implementation strategies: throughput or resources optimization, are developed and enable us to make an optimal choice of FPGA chip according to given requirements for LLMLP structure, sampling frequency and FPGA resources.
4. The work-flow for investigation of the constrained design precision based on band-pass discrepancies measurement is developed, ensuring a balance between normalized bandwidth of the synapse and they central frequency, bandwidth and output signal mean error.
5. The nonlinearity implementation criteria based on the evaluation of distortions in transfer function of lattice-ladder neuron enables us to make and efficient choice of memory size as a trade off between the error of proposed activation function, gain and block RAM size.

3

Implementation of Lithuanian Speech Recognizer in Field Programmable Gate Array

Fulfilling the Task 3 of the Thesis the results of a novel Lithuanian speech recognizer (LSR) for disabled persons development and implementation on FPGA are presented in this chapter. A general block diagram of LSR for disabled persons and its final assemblage are commented in Section 3.1. Then developed FPGA IP cores for speech feature extraction (Section 3.2) and word recognition (Section 3.3) are analyzed. Previously introduced original technique for efficient implementation of LLMLP (Section 2.2), here is used for speech filtering and noise reduction FPGA IP core development. Moreover a new accelerated pattern matching technique to speed up the word recognition process using constrained dynamic time warping (DTW) is proposed. An original the double random seed matching Algorithm 3.1 is developed and programmed in FPGA IP core. Lastly in Section 3.4 the original iterative voice response interface enabling computer guided dialog for communication with disabled persons is briefly described.

The research results are published in author publications (Tamulevičius *et al.* 2015, Serackis *et al.* 2014, Tamulevičius *et al.* 2014, Sledevič *et al.* 2013, Stašionis, Sledevič 2013, Serackis *et al.* 2013, Sledevič, Navakauskas 2013). The main results are announced in international: “Electronics” (Palanga, 2014, 2013), DAMSS (Druskininkai, 2014), ICISVC (Paris, 2014), EMS (Manchester, 2013), EUROCON (Zagreb, 2013); and national: “Science – Future of Lithuania” (Vilnius, 2013), “Multidisciplinary Research in Natural and Technology Sciences" (Vilnius, 2014) scientific conferences.

3.1. Speech Recognition System Overview

Whole recognition system is implemented in single FPGA chip. The word recognition algorithm consists of several steps, as shown in Fig. 1.17. The speech preprocessing involves filtering, framing and windowing. The extracted features are stored in the dictionary and the spoken word is compared with a part of context dictionary. The developed IP cores are used for all features extraction and comparison algorithms. The hardware part of the proposed system (Fig. 3.1) is clocked at 50 MHz while software part – at 100 MHz.

Fig. 3.1. Illustration of six prototypes of speech recognizers (primary versions are on the top, final version is on the bottom)

The proposed system can work in non-real-time and real-time word recognition modes. In the first mode, speech signal is read from external SD card memory for the record dataset recognition accuracy verification. In the second mode, the speech signal is constantly captured from microphone for device control. Each implemented IP core must meet the 11.61 ms delay condition in order to work in real-time. Such requirement is defined by the 23.22 ms duration of signal frame. Due to the half overlapping frames the time between incoming frames is twice shorter.

The block diagram in Fig. 3.2 shows the connectivity between IP cores with bus width information. Each IP core in the proposed system works independently and communicates with others via synchronization signals. The synchronization, clock and reset signals are not shown in Fig. 3.2. The audio chip is configured to sample data at 44.100 kHz. The filtering IP core involves

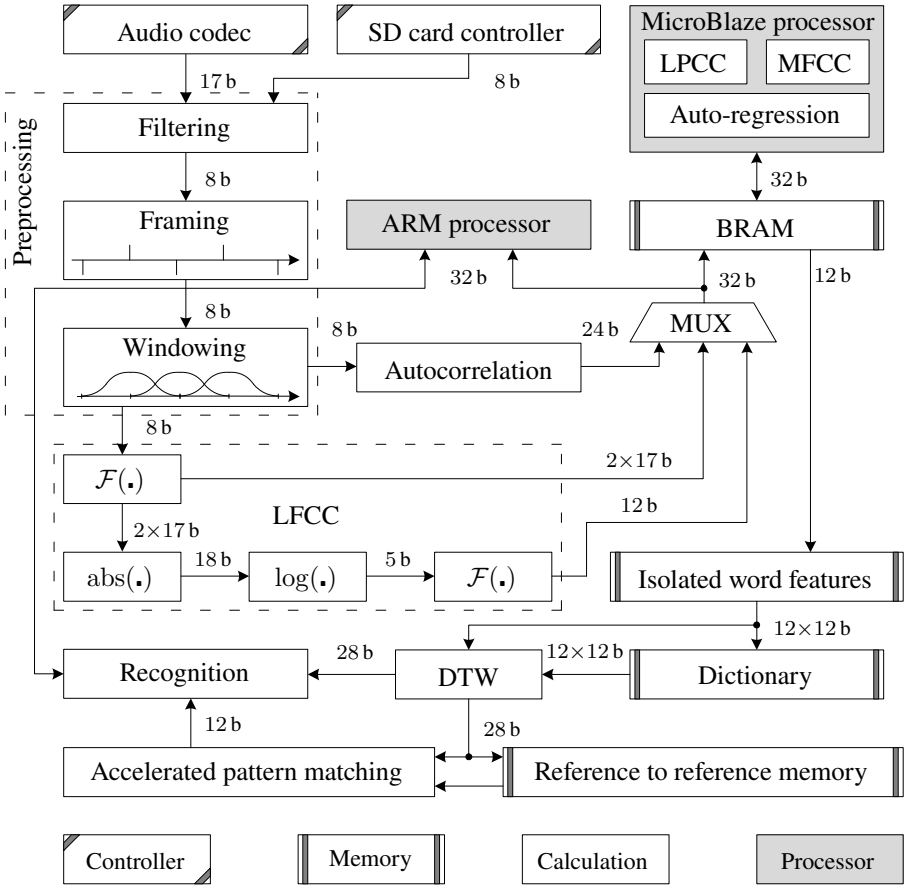


Fig. 3.2. The block diagram of the isolated word recognition system

the LLMLP compiled by using technique proposed in Section 2.2. The filtered speech signal is stored temporarily in internal FPGA four memory blocks. These blocks are alternatively used to write and read 256 samples of partially overlapped and framed speech signal. For the windowing operation the Hanning function $K_{\text{Han}}(n)$ is implemented as look-up table and is described by equation:

$$K_{\text{Han}}(n) = \sin^2\left(\frac{\pi n}{N_{\text{w}} - 1}\right), \quad \forall\, n \in [0, N_{\text{w}} - 1], \tag{3.1}$$

here n is the index of the sample in a window; $N_{\text{w}} = 256$ is the total number of samples in a frame.

The soft-core processor (grey block in Fig. 3.2) is used for the IP cores that require precision and operation on floating-point numbers. The decision maker is a linear function that sequentially compares two time series and matches errors given from DTW IP core. In the real-time mode system firstly looks for activation word. If this word is recognized, then second spoken command is compared with dictionary in a features space. Up to 8 copies of utterance can be assigned to a single command to improve recognition rate. There are reserved 2^{14} samples for each isolated word regardless of its real length. Each word has 128 features vectors. Therefore, the size of DTW matrix is always permanent and equal to 128×128. The control signal is transmitted to device, when associated command is recognized correctly.

The isolated word recognition process consists of two main parts. First one is the features extraction in speech signal. Second one is the comparison of feature vectors of the just spoken word with the set of command stored in dictionary. Therefore, in the next sections the implementation of speech analysis and comparison methods are presented.

3.2. Features Extraction Implementations

Four features extraction methods (LFCC, MFCC, LPC, LPCC) (cf. Section 1.3) are selected for speech signal analysis with the aim of future evaluation and selection of the best one in final application. The LFCC analysis is implemented in FPGA logic. The other analysis methods are partially implemented in FPGA and in general purpose soft-core processor, because of the requirements for float-point arithmetic.

3.2.1. Linear Frequency Cepstral Analysis Intellectual Property Core

The LFCC is performed by (1.11). The hardware implementation of LFCC consists of two fast Fourier transforms, single logarithm and $\mathrm{abs}(.)$ IP cores. After the windowing 8 b and 256 sample length speech signal is passed to the $\mathcal{F}(.)$ core. The $\mathcal{F}(.)$ core utilizes 3 BRAMs and 3 DSP slices. The length of $\mathcal{F}(.)$ is equal to the framed signal length. $\mathcal{F}(.)$ is implemented using the Radix-2 butterfly core from the Xilinx IP core generator. The latency of $\mathcal{F}(.)$ computation is 1662 clock cycles. The $\mathcal{F}(.)$ core outputs 17 b width real and imaginary spectrum series. In the abs core the absolute 18 b value is calculated and sent to the $\log(.)$ core. Because the radix of a logarithm is equal to 2, this operation can be implemented in the hardware as the search for the highest bit in data (see Fig. 3.3). The experimental verification confirms that

the maximum spectrum value is never higher than 16 b. Therefore, the result of a logarithm is in the range from 0 to 15 b and the output is described only by 4 b. The result is rounded to the lowest integer number.

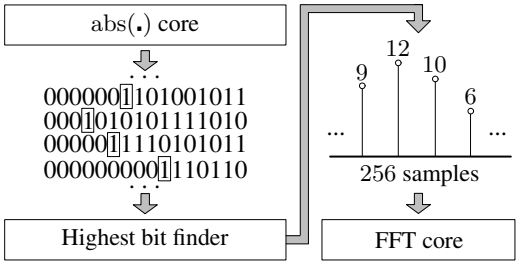


Fig. 3.3. Illustration of the fast $\log_2$ operator implementation

The 4 b width logarithm of the spectrum is forwarded to the next $\mathcal{F}(.)$ core that calculates 12 b width LFCC features. Only real part of LFCC is saved in memory dedicated for one word features that utilizes one BRAM. In order to collect one word features the LFCC calculation is repeated 128 times. That ends the LFCC feature extraction.

The LFCC calculation algorithm is sequential and there is no need to parallelise it because one feature extraction module can be successfully shared between overlapped frames and used at the end of the frame only. The $\mathrm{abs}(.)$ and $\log(.)$ operators are applied immediately for the complex spectrum that comes from the $\mathcal{F}(.)$ core. The second $\mathcal{F}(.)$ operation gives 256 cepstrum coefficients for the one frame speech signal. Only first 12 coefficients are used in further DTW calculation.

3.2.2. Mel-Frequency Cepstral Analysis Intellectual Property Core

The signal-flow graph of MFCC extraction was presented in Fig. 1.19 and described by (1.12)–(1.13). The same $\mathcal{F}(.)$ core, previously used in LFCC, is shared with MFCC implementation, as is shown in Fig. 3.2. The $\mathrm{power}(.)$ operation of the spectrum utilises single DSP and the result of $\mathrm{power}\left(\mathcal{F}(.)\right)$ is transfered to processor through shared BRAM. The multiplication with transfer function of the filter bank (1.13), $\log(.)$ and discrete cosine transform $\mathcal{C}(.)$ cores are implemented on the soft-core processor due to the precision requirements for trigonometric, logarithmic and division operations (1.12a). The calculated MFCC are returned back through the shared BRAM to the single isolated word features memory.

3.2.3. Linear Predictive Cepstral Analysis Intellectual Property Core

The signal-flow graph of LPC and LPCC extraction was presented in Fig. 1.20 and described by equations (1.14)–(1.17). The 12th order autocorrelation (1.14) algorithm is implemented using 13 shift registers with analysis length $p = 12$, as shown in Fig. 1.21 and Fig. 3.4. Each element of this register is accessible

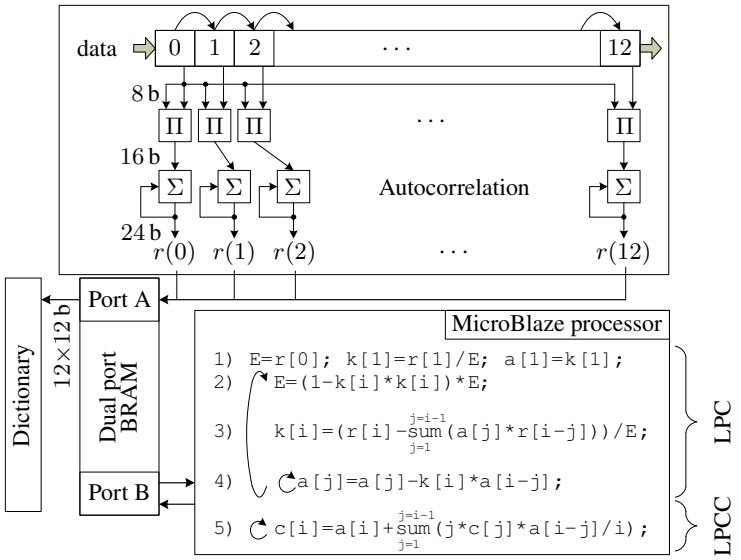


Fig. 3.4. Implementation of autocorrelation, Levinson-Durbin and linear predictive cepstral coefficients calculation algorithms

in parallel by 13 DSPs slices. The multiplication result is accumulated until the last byte No. 256 reaches the shift register. Autocorrelation coefficients $r(i)$ are stored in dual port BRAM buffer via port `A`. This memory is used to exchange the data between hardware and software parts of the system. For the soft-core processor the autocorrelation coefficients are accessible via port `B`. At the first iteration of Levinson-Durbin algorithm the values of energy coefficient E, first reflection coefficient $k(1)$ and first LPC are initialized. For each ith iteration energy coefficient E is updated and used as divisor for calculation of reflection coefficient $k(i)$. The steps 2–4 are repeated 12 times giving the 12th order LPC. At step 5 the LPCC are calculated from LPC features. The processor calculates both types of coefficients and returns them to BRAM buffer to store in the dictionary and further word recognition using DTW IP core.

3.3. Word Recognition Implementations

In order to recognize the spoken words in a real-time the IP cores for speech features comparison are implemented. A new accelerated pattern matching technique to speed up the word recognition process using constrained dynamic time warping is proposed. An original the double random seed matching algorithm is developed and programmed in FPGA IP core.

3.3.1. Dynamic Time Warping Intellectual Property Core

A dynamic time warping algorithm is used to find the correspondence in two data series by matching two features vectors. The actual output is the estimate of dissimilarity between two signals. The DTW is based on filling the error matrix with differences between two signals. The DTW algorithm is sequential by nature (see Fig. 3.5).

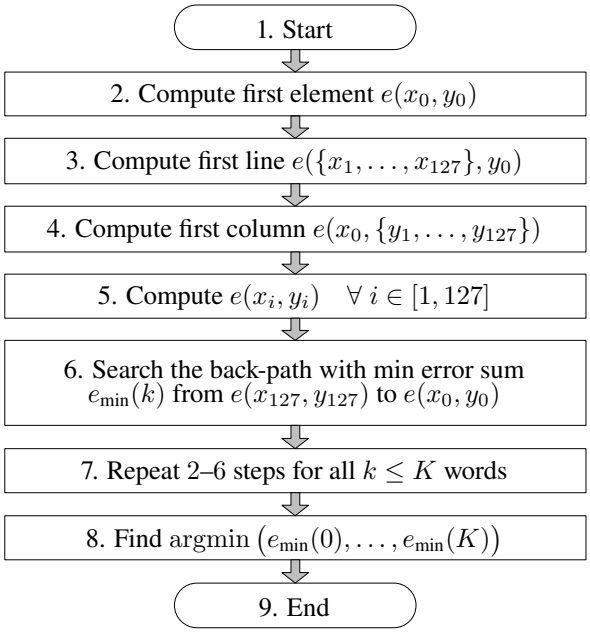


Fig. 3.5. Implemented dynamic time warping algorithm

Step 2 in the DTW algorithm is to compute the first matrix element $e(0, 0)$ that is equal to the Euclidean distance between first cepstrum values of both series. In steps 3, 4 the errors in the first row $e(0, [1, 127])$ and first column

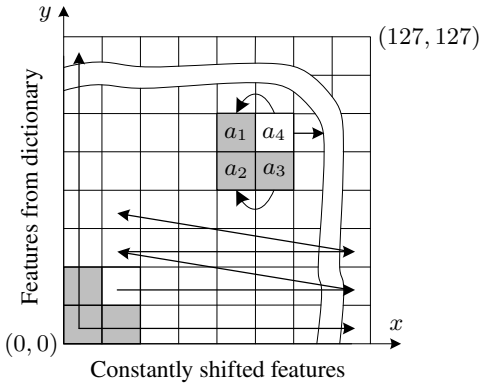


Fig. 3.6. Illustration of 128×128 element size error matrix filling

$e([1, 127], 0)$ are computed. The rest of the area is filled moving from left to right and from bottom to top. Backward search for the path with the minimum error starts when the error matrix is filled. Steps 2–6 are repeated for all K words in the dictionary. Every second matrix element includes the error value of the previous element, thus the error matrix cannot be filled by several processes in parallel. The chosen way to accelerate the matrix calculation is a pipelined preparation of values a_1–a_4 (Fig. 3.6).

In step 5 the error matrix is filled using a sliding 2×2 size window. Element a_4 is calculated as the sum of the current Euclidean distance $e(x_i, y_i)$ with the minimal distance at neighbouring points:

$$a_4 = e(x_i, y_i) + \min(a_1, a_2, a_3). \tag{3.2}$$

Here x_i, y_i are the row and column indices at i-th iteration. Initially $e(x_i, y_i) = e(1, 1)$, while $a_1 = e(0, 1)$, $a_2 = e(0, 0)$ and $a_3 = e(1, 0)$. In the next iteration the window is shifted to left by one element, thus:

$$a_1^{\text{next}} = a_4, \qquad a_2^{\text{next}} = a_3. \tag{3.3}$$

Only element a_3 must be constantly read from the memory at address $\text{addr}(a_3^{\text{next}})$. As the BRAM is one-dimensional, the address mapping from 2D into 1D is implemented by the index decoder described by:

$$\text{addr}(a_3^{\text{next}}) = x_i + 128(y_i - 1), \tag{3.4}$$

here addr is the addressing function.

The Euclidean distance is calculated in parallel with the calculation of the minimum neighbour value, as shown in Fig. 3.7. There is no need to separately

compute and fill the Euclidean distance matrix, because of pipelined implementation. The error matrix memory is implemented on the dual port BRAM memory. In one clock cycle this memory is accessed twice: while reading new a_3 and writing calculated a_4 value. One DTW calculation (including a backward search) needs 16 640 clock cycles to make one comparison.

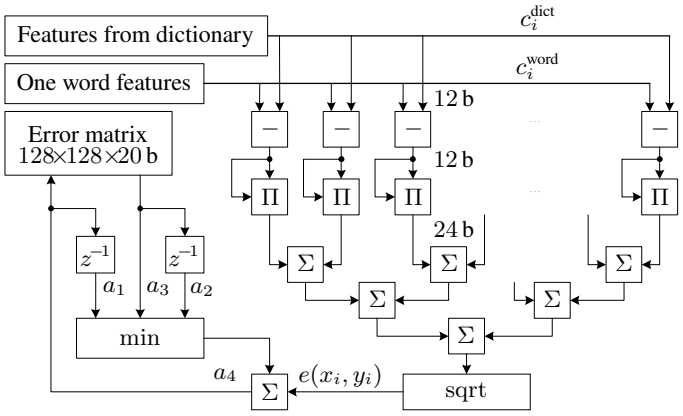


Fig. 3.7. Implementation of Euclidean distance and error matrix filling

The block diagram in Fig. 3.7 presents fast Euclidean distance estimation using 11 additions, 12 subtractions, 12 multipliers and one square root core. The square root core has pipelined implementation. The calculation of Euclidean distance $e(x_i, y_i)$ is done per one clock cycle. The difference between vectors is estimated by:

$$e(x_i, y_i) = \sqrt{\sum_{j=1}^{12} \left(c_i^{\text{word}}(j) - c_i^{\text{dict}}(j) \right)^2}, \tag{3.5}$$

here $e(x_i, y_i)$ – Euclidean distance between two vectors c_i^{dict} and c_i^{word}; $c_i^{\text{word}}(j)$ – jth feature index of ith feature vector in one word feature memory; $c_i^{\text{dict}}(j)$ – jth feature index of ith feature vector stored in dictionary.

To ensure synchronous filling of error matrix the data from dictionary, one word buffer and error matrix are read out in parallel. Because the BRAM memory has one clock latency for data read, therefore the valid address must be prepared before accessing the data. Therefore the address counter is implemented as separate process independent of the error calculation process. These two address and error estimation processes run synchronous. The advantage of hardware implementation is the pipelined calculation of error matrix. At each

rising edge of clock new error $e(x_i, y_i)$ value is calculated and stored in error matrix memory immediately.

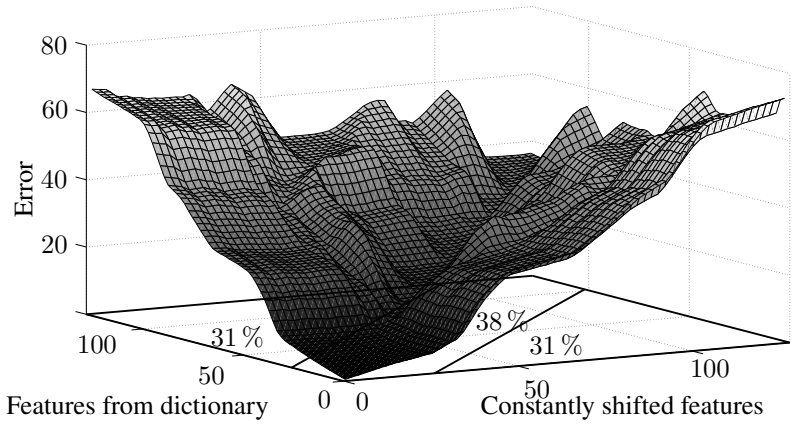


Fig. 3.8. The surface of the dynamic time warping error matrix with constrained area outlined

In step 6 the use of border constraints additionally accelerates the $\mathrm{DTW_c}$ algorithm up to 2.6 times without a negative influence on the recognition accuracy. Only 38 % of the total matrix area is used (see Fig. 3.8). The error values near the corners are always higher than in the central part of the matrix. Therefore, the back path never comes upwards and 62 % of the matrix area can be ignored. The minimal value of the error path is iteratively calculated using the same 2×2 size sliding window by:

$$e_{\text{min}}^{\text{next}} = e_{\text{min}} + \min(a_1, a_2, a_3). \tag{3.6}$$

Initially $e_{\text{min}} = e(127, 127)$, $a_1 = e(126, 127)$, $a_2 = e(126, 126)$ and $a_3 = e(127, 126)$. At the next rising edge of clock the sliding window is shifted to the element with the lowest error at address described by:

$$(x_i, y_i) = \text{addr}\left(\min(a_1, a_2, a_3)\right). \tag{3.7}$$

That corresponds to the following updates:

$$\begin{aligned} a_1^{\text{next}} &= e(x_i - 1, y_i), \\ a_2^{\text{next}} &= e(x_i - 1, y_i - 1), \\ a_3^{\text{next}} &= e(x_i, y_i - 1). \end{aligned} \tag{3.8}$$

The minimal error path $e_{\min}$ is computed until (x_i, y_i) arrives at $(0, 0)$ location. The steps 2–6 are repeated K times, where K is the number of words in context dictionary. And finally, in step 7 the calculated argument of the minimal error $e_{\min}(k)$ shows the index of recognized word.

A single DTW_c core is able to compare maximum 90 word features in 11.61 ms (at 7800 words/s) satisfying real-time requirements. To check a part of context dictionary larger than 90 words, there is proposed accelerated pattern matching below.

3.3.2. Accelerated Pattern Matching Intellectual Property Core

The idea of pattern match search acceleration is based on the assumption that the difference between pattern to be recognised and the reference, given in dictionary is close to the difference of reference, selected for matching and reference that should be found. In case this pattern matching technique is used for isolated Lithuanian word recognition, the matching error between the pattern extracted for word “traukti" and randomly selected reference “augti" should be close to the matching error between reference word “augti" and reference word “traukti", stored in the dictionary.

In order to apply given idea, the differences between references inside dictionary should be estimated. The differences are estimated using DTW based

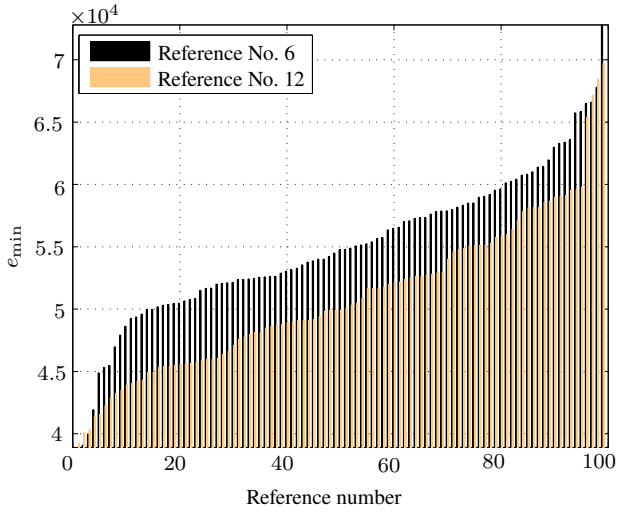


Fig. 3.9. Reference-to-reference error distribution, calculated for two references

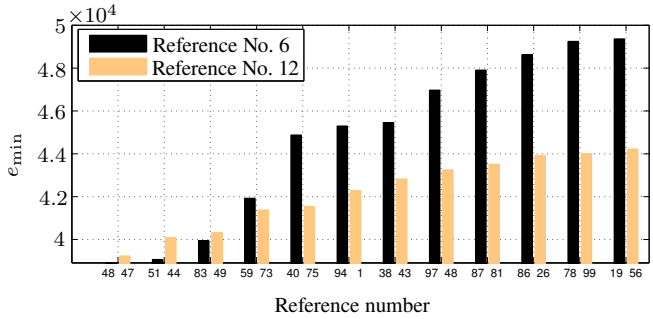


Fig. 3.10. Comparison of the 12 neighbouring reference number distribution for two references

algorithm. The estimation results are added to each word as a $(N-1)\times 2$ size matrix $\mathbf{M}_i$, where N is the number of references stored in dictionary. This is a connection of two unique vectors: one with calculated reference-to-reference matching errors for each r_i, where $i = 1, 2, \ldots, N$ and another – constructed from reference indices, corresponding to calculated matching errors.

The sorting of calculated reference-to-reference matching errors in ascending order indicates references, which has similar distance (calculated using DTW based algorithm) to analysed reference r_i. Fig. 3.9 shows the comparison of reference-to-reference distance estimation results for two references.

As it is seen in Fig. 3.9 for 6^{th} reference we may find several references in database that are more similar to this reference than others – the error increases with noticed nonlinearity (Fig. 3.10). These references are potential causes of pattern recognition failures. In order to reduce the risk of pattern recognition failure, the DTW matching is calculated between pattern p and a set of references, situated in the reference-to-reference matching vector neighborhood.

An example of reference-to-reference error matrix for 100 references is shown in Fig. 3.11. As it is seen in the figure, the reference-to-reference error matrix is symmetrical. Taking into account this feature we may reduce the amount of additional data to store in the dictionary. However, in that case the sorting of the values will require additional algorithm to be applied.

The pattern recognition algorithm is presented in Algorithm 3.1.

Algorithm 3.1 (The double random seed matching algorithm)

Require: Load feature vectors.

1: A. Random selection of initial reference r_i, here i is the index of reference in non-sorted dictionary set C.
2: B. Compute the DTW matching error e_{p,r_i} for pattern p and initial reference r_i.
3: **while** Indicator of match $M = 0$ **do**
4: 1. Select n references $\mathbf{r}_{\mathrm{h}} = \{r_{j+1}, r_{j+2}, \ldots, r_{j+n}\}$, for $j = i$, having reference-

to-reference matching error $e_{r_i,r_{\in C_i}}$ higher than estimated e_{p,r_i}.
for $j = 1 : n$ **do**
Compute the DTW matching error $e_{p,r_{j+indx}}$.
end for
2. Select m references $\mathbf{r}_\mathrm{l} = \{r_{j-1}, r_{j-2}, \ldots, r_{j-m}\}$, for $j = i$, having intra-dictionary matching error $e_{r_a,r_{\in C_i}}$ lower than estimated $e_{\mathrm{p},r_\mathrm{i}}$.
for $indx = 1 : m$ **do**
Compute the DTW matching error $e_{p,r_{j-indx}}$.
end for
3. Find the minimum matching error:
$e_{\min} = \min\{e_{p,r_{j-m}}, e_{p,r_{j-m+1}}, \ldots, e_{p,r_{j+n}}\}$;
if $e_{\min} = e_{p,r_i}$ **then**
r_i is set as a matched reference to pattern p. $M = 1$.
else
A new initial reference is selected r_i at point with minimal error $e_{\min}$.
end if
end while
C. Selection of the second reference r_a.
D. Compute the DTW matching error $e_{\mathrm{p},a}$.
while Indicator of match $M = 0$ **do**
1. Select n references $\mathbf{r}_\mathrm{h} = \{r_{j+1}, r_{j+2}, \ldots, r_{j+n}\}$, for $j = a$, having intra-dictionary matching error $e_{r_a,r_{\in C_i}}$ higher than estimated e_{p,r_a}.
for $j = 1 : n$ **do**
Compute the DTW matching error $e_{p,r_{j+indx}}$.
end for
2. Select m references $\mathbf{r}_\mathrm{l} = \{r_{j-1}, r_{j-2}, \ldots, r_{j-m}\}$, for $j = a$, having intra-dictionary matching error $e_{r_a,r_{\in C_i}}$ lower than estimated e_{p,r_a}.
for $j = 1 : m$ **do**
Compute the DTW matching error $e_{p,r_{j-indx}}$.
end for
3. Find the minimum matching error:
$e_{\min} = \min\{e_{p,r_{j-m}}, e_{p,r_{j-m+1}}, \ldots, e_{p,r_{j+n}}\}$.
if $e_{\min} = e_{p,r_a}$ **then**
r_a is set as a matched reference to pattern p. $M = 1$.
else
A new initial reference is selected r_a at point with minimal error $e_{\min}$.
end if
end while
E. Select pattern match with lower error: $\min\{r_i, r_a\}$.

Taking into account that reference stored in the dictionary has $(W/2) \times O$ coefficients (here W is the width of the signal analysis window used for parameter estimation; O is the analysis order), adding additional matrix $\mathbf{M}_i$ to each reference will increase the dictionary by $(N-1) \times 2$ values. It can be estimated by using the following equation:

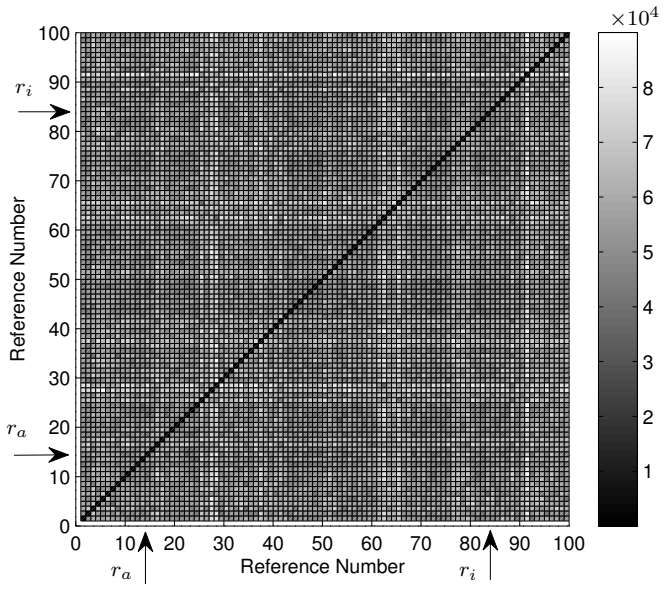


Fig. 3.11. Illustration of reference-to-reference error matrix

$$L = \frac{2 \times (N-1)^2 \times 100}{N \times (W/2) \times O}, \tag{3.9}$$

here L is the additional percentage amount of data to be loaded to memory; N is the number of references. For example, the dictionary consisting of 100 words, near $13\,\%$ of the data amount will be additionally load together with references in dictionary.

3.4. Iterative Voice Response Interface

Along with the implemented features extraction and comparison IP cores on FPGA, we are developed iterative voice response algorithm on the integrated ARM processor for human communication with the LSR. Through the iterative voice response interface (Fig. 3.12) the person is able to control devices hands-free and by voice only. It contains three operational states selected by voice or push-button. In a training state users can choose device type, give the name for commands and select device control type. To improve the recognition accuracy the recognizer asks to pronounce each new name of the command twice. During the training recognizer saves pronounced word features and a voice record.

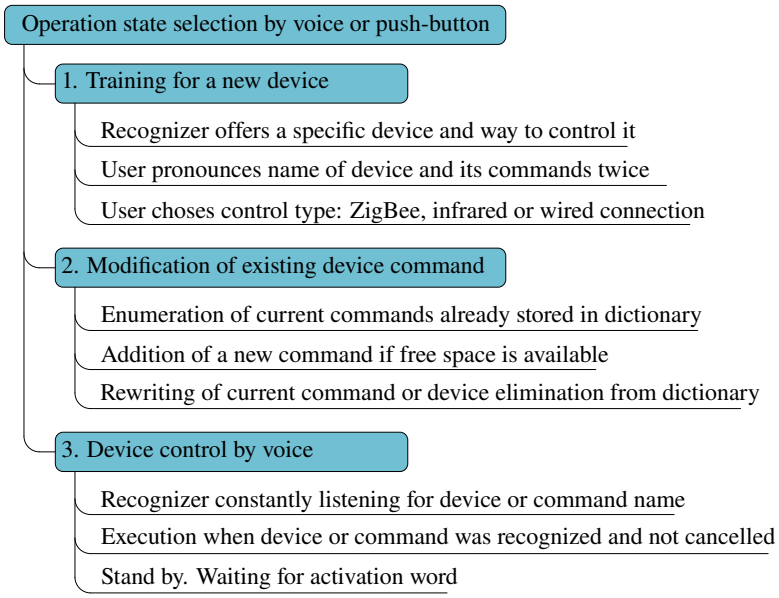


Fig. 3.12. The behavior tree of the iterative voice response algorithm

The command elimination or editing is performed under the modification state. User can check the list of already saved commands by asking the recognizer for enumeration of word stored in dictionary. In the third state a specific command is transmitted to the destination device, when the pronounced words are recognized as the names of device and its control command. The recognizer goes to the stand by mode if the words are not pronounced longer than 30 s. And it wakes up by pronouncing an activation command.

The information about current state and proper command recognition is guided by RGB light diode in conjunction with sound response. For user convenience the state selection is also available by button on the front panel of the recognizer. Next two buttons are dedicated for volume control and switching between two voice sources – speaker or headphone.

The algorithm (Fig. 3.12) implemented on ARM communicates with the feature extraction and comparison IP cores through the control lines and shared memory. The speech signal, word features and new infrared commands are transfered from FPGA to ARM during the training state with the aim to store it on the flash memory. The written software on the ARM controls which part of the context dictionary must be load to a dictionary memory on FPGA depending on the actually controlled device. A pronounced and correctly recognized command can be cancelled by voice in 2 s by pronouncing word “reject”

("atšaukti" in Lithuanian) if user decides to reject the further execution. After rejection identification the previuosly recognized command is not executed and recognizer is listening for next command. The results of isolated word recognition speed and accuracy are presented in the next chapter.

The technical details of created isolated word LSR are following:

- ZynQ-7000 chip with Artix-7 FPGA and Dual Core ARM Cortex A9.
- FPGA clock frequency 50 MHz, ARM – 667 MHz.
- Programmatically selectable 12th order speech analysis methods: LPC, LPCC, LFCC and MFCC.
- Signal quantization: 8–16 b.
- Signal sampling rate up to 44.1 kHz.
- Word matching speed 7800 word/s.
- Speech recognition rate up to 97 %.
- Device control by: IR, ZigBee, UART, USB.
- FPGA slice logic utilization 27 %.
- Power consumption 3.6 W.
- 1 GB of DDR3 SDRAM.

3.5. Conclusions of the 3rd Chapter

1. A first working prototype of Lithuanian speech recognizer in FPGA for disabled persons is created. A new accelerated pattern matching algorithm uses constrained dynamic time warping and enables to recognize words 2.6 times quicker without the loose of recognition accuracy.
2. The optimized FPGA intellectual property (IP) cores are developed and experimentally verified in Lithuanian speech recognizer. Hardware optimized single isolated word matching (DTW_c) is executed in 128 µs achieving 7800 word/s comparison speed.
3. The developed accelerated pattern matching algorithm requires additional up to 13 % of dictionary storage space in order to save reference-to-reference matching values for 100 word dictionary, computed using DTW algorithm.
4. The implemented feature extraction and comparison IP cores, used the speech recognition system, are accelerated for device control in a real-time.

4

Experimental Verification of Developed Intellectual Property Cores

In the following results on the performance experiments of numerous LLMLP IP core implementations are presented. In Section 4.1 the influence of constrained bit width on the accuracy of LLN with its activation function is experimentally verified changing synapse bandwidth, number of bits dedicated for each signal, memory size and gain of the activation function. The selection of optimal training scheme for LLN is investigated with respect to the synapse order, number of inputs and analysing logic resource utilisation and latency. The efficiency of proposed technique for LLMLP generation (cf. Section 2.2) is compared with a commercial tool considering on improvement of sampling frequency and equivalent resources utilization. The qualitative LLMLP implementation evaluation is investigated through Pareto optimal frontiers (cf. Section 2.1).

In Section 4.2 the experiments of isolated word recognition accuracy and speed for different speakers, signal to noise ratios, features extraction and accelerated comparison methods are presented. The results of word recognition speed and accuracy are compared with initial soft-core based recogniser implementation. The performance of developed IP cores for Lithuanian speech recognizer (cf. Chapter 3) is experimentally verified taking into account execution speed and FPGA resource utilization.

The research results are published in author publications (Sledevič, Navakauskas 2016, Sledevič *et al.* 2013, Sledevič *et al.* 2013, Sledevič, Stašionis 2013, Sledevič, Navakauskas 2015, Sledevič, Navakauskas 2014). The main

results are announced in international: “Electronics” (Palanga, 2015, 2013), AIEEE (Riga, 2015), AIEEE (Vilnius, 2014), EMS (Manchester, 2013), EUROCON (Zagreb, 2013), ICCCCE (Istanbul, 2013); and national “Science – Future of Lithuania” (Vilnius, 2016–2014) scientific conferences.

4.1. Investigation of Lattice-Ladder Multilayer Perceptron and its Implementation Technique

The experimental verification begins from accuracy investigation of basic building block of LLMLP. The aim of this investigation was to get insights on the selected neuron model fixed-point architecture (necessary to use word length) and its complexity (required number of LUT and DSP slices and BRAM size) by the evaluation of the reproduced by lattice-ladder neuron accuracy of bandwidth and central frequency as also as output signal normalized mean error. Next, the experiments on neuron activation function was performed. The aim of this investigation was to get insights on the distortions of the selected neuron model output by the evaluation of transfer function RMS error and neuron output signal mean and maximum errors while changing the gain and memory size of the activation function.

Afterwards, optimization of the latency and DSP slice usage for the LLN and its simple gradient training algorithm implementation on FPGA is investigated. Four alternative regressor lattices to be used in LLN training were considered and experimentally evaluated. The experiments of LLN implementation technique were performed by varying the number of synapses and the order of LLF. Using the resource and latency criteria the optimized LLN IP core implemented by our technique is compared with the same LLN created in commercial tool Xilinx Vivado HLS. The LLMLP is investigated through Pareto optimal frontiers changing the number of layers, neurons and synapse order.

4.1.1. Word Length Selection

We are interested in comparatively narrow band single ($L \equiv 1$, $N^{(1)} \equiv 1$, $\Phi(\cdot) \equiv \Phi_{\text{ident}}(\cdot)$) lattice-ladder neuron (LLN) implementation. Thus during the investigation the bandwidth of synapse was changed in a preselected range. The reference bandwidth f_{b} can be normalized taking into account the sampling frequency f_{s} and then expressed in percentages by $f_{\text{b}} \triangleq \|f_{\text{b}}\| = (2f_{\text{b}}/f_{\text{s}}) \cdot 100\,\%$. For the experiments the frequencies $f_{\text{s}} = 11,025\,\text{Hz}$ and range of $f_{\text{b}} \in [2, 1380]\,\text{Hz}$ is selected, that yields normalized bandwidth range $f_{\text{b}} \in [0.04, 25]\,\%$.

The normalized white noise signal is used as an input $s_{\text{in}}(n) \equiv \omega(n)$. The neuron training circuit is considered to be correct if all Θ_j and v_j weights represented in fixed-point arithmetics converge to the corresponding truncated parameters from the reference design. Therefore, the magnitude responses for the LLN with and without regressor lattice are the same.

The 18 b and 25 b precision is investigated because of FPGA constrains – one and two DSP slices are required to multiply corresponding word length signals. The 10 b and 12 b precision is taken as the possible alternative implementation for low accuracy LLN.

Bandwidth and central frequency relative errors ϵ_{b}, ϵ_{c} by (2.15) as well as output signal normalized MAE $\mathscr{E}_{\text{MA}}$ by (2.16) were estimated for each specific reference bandwidth f_{b} LL neuron implementation and afterwards corresponding normalized relative errors $\epsilon_{\text{b}}(f_{\text{b}})$, $\epsilon_{\text{c}}(f_{\text{b}})$ additionally were calculated. Accuracy results for different word length implementations are summarized in Fig. 4.1. From Fig. 4.1a follows that the narrow band filter requires denser grid on zero-pole plane near the unit circle. The narrower band is, the more bits are necessary to preserve low normalized relative error ϵ_{b} of the bandwidth.

While enlarging the bandwidth the ϵ_{b} decreases exponentially. The LLF with wider bandwidth reach the same ϵ_{b} as LLF with higher precision, i.e., 12 b LLF with band of $f_{\text{b}} = 7\,\%$ has same $0.1\,\%$ mean error as 18 b LLF with band of $f_{\text{b}} = 0.1\,\%$ (Fig. 4.1a). Increase of the LLF precision not evenly improves the ϵ_{b} for all bandwidths. The reason is in constrained address width of $\sin\Theta_j$ and $\cos^{-1}\Theta_j$ implemented in LUTs.

For the narrow band filter until $f_{\text{b}} = 0.5\,\%$ the central frequency mean errors ϵ_{c} don't depend on implementation precision and exponentially decrease while expanding the bandwidth (Fig. 4.1b). The use of one or two DSP slices gives the same ϵ_{c} below $f_{\text{b}} = 20\,\%$. The output signal mean error (see Fig. 4.1c) decreases exponentially for wider normalized reference bandwidth. The replacement of DSP slice by the LUT-based multiplier can be considered when $f_{\text{b}} \geq 10\,\%$.

4.1.2. Neuron Activation Function Implementation

We are interested in compact and enough precise hyperbolic tangent activation function $\Phi_{\text{tanh}}(\centerdot)$ for lattice-ladder neuron implementation. Therefore, the transfer function and output signal discrepancies are measured under different BRAM size. During experiments BRAM size is increased 2 times in each step from 64 B to 64 kB. The gain is increased linearly from 2 to 16. Other LLN parameters are kept unchanged. Independent on the BRAM size two bytes precision is used for each sample of the approximated hyperbolic tangent. The

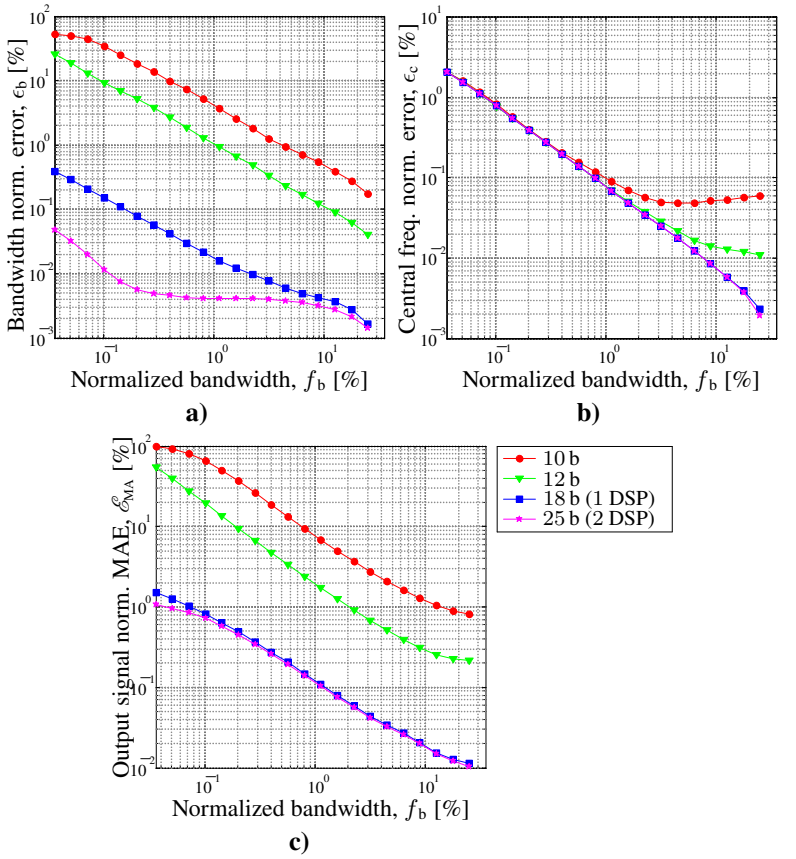


Fig. 4.1. Accuracy results for the second order lattice-ladder linear neuron and its training circuit implementations on Artix-7 FPGA using different word length: bandwidth a), central frequency b) and output signal c) normalized errors dependencies on a normalized reference bandwidth

average and maximum errors ($\mathscr{E}_{MA}$ and $\mathscr{E}_{MM}$) are obtained by uniformly sampling $\sin(x)$ on 10^6 equally spaced points $x \in [-1, 1]$.

The average and maximum errors of the activation function output depends almost linearly on the LUT size as shown in Table 4.1. In comparison with the (Armato *et al.* 2011), the $\mathscr{E}_{MM}$ less than $1.5\,\%$ can be achieved using $512\,\mathrm{B}$ LUT size or only $0.09\,\%$ of total BRAM in xc7z020 FPGA chip (Tommiska 2003) used here for experimental investigation. Maximum error less than $0.4\,\%$ (Xilinx 2013c) can be achieved using $2\,\mathrm{kB}$ of BRAM (one of 280 available BRAMs in xc7z020) utilizing only $0.36\,\%$ of total block memory in FPGA chip. The large LUT ensures small error, e.g., $64\,\mathrm{kB}$ memory yields

$\mathscr{E}_{\text{MM}} = 0.01\,\%$ and $\mathscr{E}_{\text{MM}} = 0.002\,\%$ (see Fig. 4.2), however utilizes $11.4\,\%$ of available memory. Such high precision is very expensive as for single LLN, however is useful for big neural networks with shared activation function.

Table 4.1. The hyperbolic tangent function implementation errors

BRAM size [B]	128	256	512	1k	2k	4k
$\mathscr{E}_{\text{MA}}(R_{\text{BRAM}})$ [%]	0.54	0.27	0.13	0.06	0.03	0.02
$\mathscr{E}_{\text{MM}}(R_{\text{BRAM}})$ [%]	5.48	2.74	1.39	0.70	0.35	0.16

The maximum $\mathscr{E}_{\text{MA}}$ is observed at $g = 3$ for all tested memory sizes (Fig. 4.2). The rising gain allows to access wider range of LUT addresses, therefore decreases the quantization error and $\mathscr{E}_{\text{MA}}$ falls slowly.

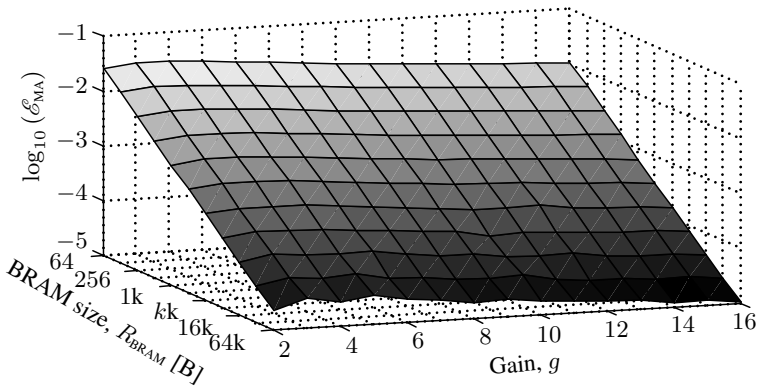


Fig. 4.2. The normalized mean absolute error of the lattice-ladder neuron output, while changing gain and block random access memory size for approximated hyperbolic tangent function

The chirp signal with sampling frequency $f_{\text{s}} = 11,025\,\text{Hz}$ and duration $N_1 = 10\,\text{s}$ is used for the transfer function error estimation experiments. The results of transfer function RMSE $\mathscr{E}^{\text{AR}}_{\text{RMS}}(g, R_{\text{BRAM}})$ under two marginal gains and three different filter bandwidths $f_{\text{b1}} = 5512\,\text{Hz}$, $f_{\text{b2}} = 2756\,\text{Hz}$, $f_{\text{b3}} = 200\,\text{Hz}$, are presented in Fig. 4.3, as a dependence on BRAM size. Decrease of the bandwidth will reduce the gap between $\mathscr{E}^{\text{AR}}_{\text{RMS}}$ limits. The $\mathscr{E}^{\text{AR}}_{\text{RMS}}$ for LLN with f_{b1} decreases almost linearly when memory size grows. The lower $\mathscr{E}^{\text{AR}}_{\text{RMS}} = 1.2{\times}10^{-4}$ and upper $\mathscr{E}^{\text{AR}}_{\text{RMS}} = 1.8{\times}10^{-3}$ limits belong to the LLN with widest bandwidth using 2 kB BRAM for activation function.

When LLN is set to pass through all frequencies $f_{\text{b1}} = 5512\,\text{Hz}$ (Fig. 4.4), then the error $\mathscr{E}^{\text{AR}}_{\text{RMS}}$ decreases almost linearly increasing both gain and memory size. The gain set to 2 yields highest error for a given memory size, because at

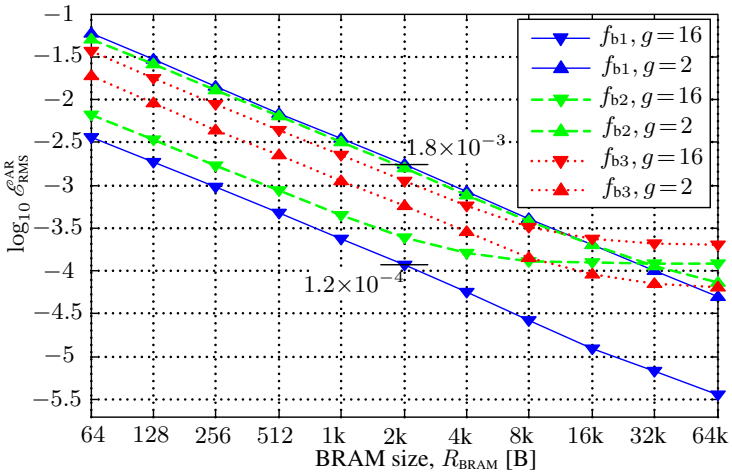


Fig. 4.3. The root mean square error of the lattice-ladder neuron transfer function using different size look-up tables

$g = 2$ with the $s(x)$ signal only $1/16$ of the approximated hyperbolic tangent can be accessed. Setting the LLN to work in narrower frequency band f_{b3} increases the error $\mathscr{E}^{\mathrm{AR}}_{\mathrm{RMS}}$ for activation function with higher gain and vice versa, smaller gain (less nonlinearity in the activation function) decreases the $\mathscr{E}^{\mathrm{AR}}_{\mathrm{RMS}}$. The increase of BRAM size more than 2 kB nonlinearly improves the error of transfer function. Therefore, 2 kB BRAM is identified as sufficient memory size for activation function implementation as a trade-off between resources and precision.

The LLN implemented in HLS tool was translated to the low-level model described in VHDL for further synthesis and uploading to the FPGA. The timing analysis after circuit placement and routing shows that, in the generated LLN circuit the hyperbolic tangent samples can be accessed with frequency $f^{\top} = 312$ MHz, while $f^{\top} > 300$ MHz is considered as a high maximum frequency achievable in a practical FPGA systems (Dessouky *et al.* 2014; Ronak, Fahmy 2014).

4.1.3. Lattice-Ladder Neuron Implementation

The single LLN (Fig. 4.5) is a basic building block of the LLMLP ($L \equiv 1$, $N^{(1)} \equiv 1$). In the following novel results on the investigation of a such neuron with $N \equiv N^{(0)}$ inputs and the $M \equiv M^{(1)}$th order LLF synapses and together with its training algorithm implementation in FPGA are presented. The main aim of this investigation is to optimize the latency and DSP slice usage for the normalized LLN and its gradient training algorithm. Four alternative re-

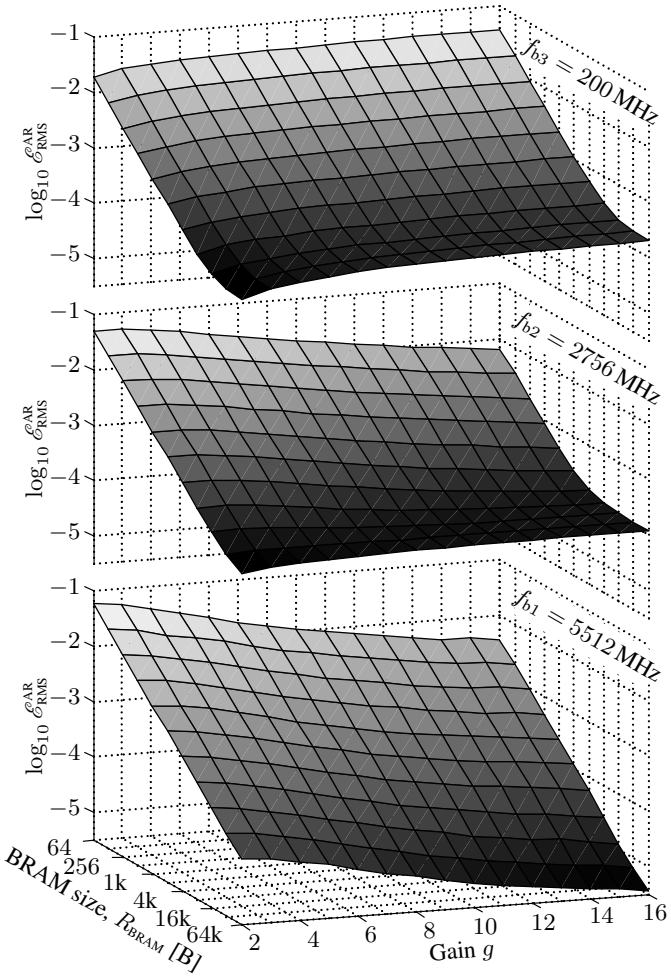


Fig. 4.4. The root mean square error of the lattice-ladder neuron transfer function, while changing gain and memory size for approximated hyperbolic tangent function

gressor lattices to be used in LLN training are considered and experimentally evaluated. The optimal resource sharing is approached by the LLN data flow graph partitioning into DSP slice subgraphs employing previously proposed technique. The experiments are performed by varying the number of synapses and the order of LLF.

The output of the LLN (Fig. 4.5) with N inputs and Mth order LLF as its synapses is expressed by (1.3), when local flow of information in the lattice part of filters for all i inputs and all j sections is defined by (1.2).

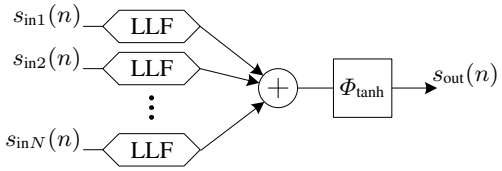


Fig. 4.5. The lattice-ladder neuron

A single DSP slice is assigned for each LLN input and additional one for the δ estimation, thus the use of DSP blocks $R^{\text{DSP}}_{N,M,\mathcal{T}}$ depends only on the number of synapses, N:

$$R^{\text{DSP}}_{N,M,\mathcal{T}} \equiv N + 1. \tag{4.1}$$

The use of BRAM $R^{\text{BRAM}}_{N,M,\mathcal{T}}$ depends only on N, too. 8 units of BRAM are dedicated for non-linear functions and a single unit is used as a buffer for each of 4 DSP slice inputs, thus:

$$R^{\text{BRAM}}_{N,M,\mathcal{T}} \equiv 4(N + 1) + 8. \tag{4.2}$$

For the comparison purposes, the use of logic resources $R^{\text{LUT}}_{N,M,\mathcal{T}}$ and achieved latency $R^{\text{Lat}}_{N,M,\mathcal{T}}$ will be normalized to the maximal value over 4 types of investigated regressors:

$$R^{\text{LUT}}_{N,M,\mathcal{T}} \equiv R^{\text{LUT}}_{N,M,\mathcal{T}} \Big/ \max_{\mathcal{T}} R^{\text{LUT}}_{N,M,\mathcal{T}}; \tag{4.3}$$

$$R^{\text{Lat}}_{N,M,\mathcal{T}} \equiv R^{\text{Lat}}_{N,M,\mathcal{T}} \Big/ \max_{\mathcal{T}} R^{\text{Lat}}_{N,M,\mathcal{T}}, \tag{4.4}$$

here $\mathcal{T}$ – set of considered LLMLP training types, $\mathcal{T} \in \{\mathfrak{t}_3, \mathfrak{t}_4, \mathfrak{t}_{11}, \mathfrak{t}_{12}\}$.

The LUT relative utilization $R^{\text{LUT}}_{N,M,\mathcal{T}}$ and the relative latency $R^{\text{Lat}}_{N,M,\mathcal{T}}$ are independent on the number of inputs N, but depend on the LLF order M and regressor type $\mathcal{T}$ (see results of experimental study in Fig. 4.6 and Fig. 4.7). If $R^{\text{LUT}}_{N,M,\mathcal{T}} = 1$ (or $R^{\text{Lat}}_{N,M,\mathcal{T}} = 1$), then the implementation of LLN with $\mathfrak{t}_x$ regressor is worst in a sense of LUT use (or achieved latency) over all tested types of regressors at a certain order M of synapses. It is evident that in majority of cases LLN implementations based on $\mathfrak{t}_{11}$ regressor lattice were the worst. Any $R^{\text{LUT}}_{N,M,\mathcal{T}}$ or $R^{\text{Lat}}_{N,M,\mathcal{T}}$ value that is less than one shows how much the implementation of LLN with $\mathcal{T}$ regressor lattice outperforms the worst case. At $M = 1$ the $R^{\text{LUT}}_{N,M,\mathcal{T}}$ and $R^{\text{Lat}}_{N,M,\mathcal{T}}$ values are arranged in a same order beginning from the worst case with $\mathfrak{t}_{12}$ regressor to the best case with $\mathfrak{t}_3$ regressor. The LLN with $\mathfrak{t}_3$ regressor saves 16 % of FPGA logic resources and has about 10 %

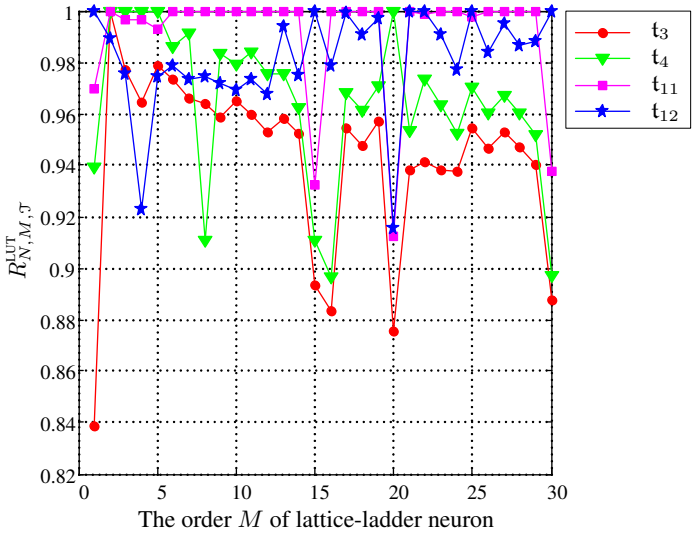


Fig. 4.6. The relative look-up table usage dependency on the order of synapses of lattice-ladder neuron that were implemented in FPGA together with the training scheme based on four different types of regressor lattices

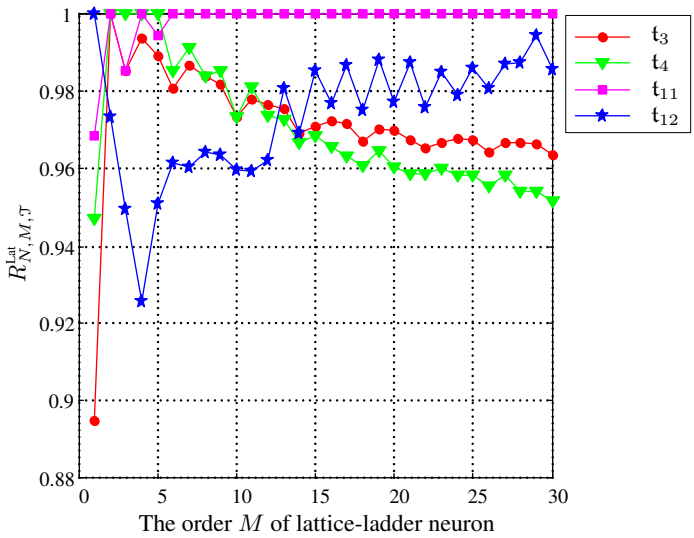


Fig. 4.7. The relative latency dependency on the order of synapses of lattice-ladder neuron that were implemented in FPGA together with the training scheme based on four different types of regressor lattices

less latency in comparison with t_{12}. When $M \in [2, 12]$ the LLN with t_{12} regressor yields the shortest processing time. However for the t_{12} regressor case the advantage of LUT use is observer only when $M \in [2, 5]$. Beginning from $M = 13$ the t_3 and t_4 are the two best regressors for LLN implementation.

In order to better outline found tendencies in Fig. 4.8 three cases are presented: resources q_{Res}, throughput q_{Thr} and both criteria optimization. Types of regressor lattice to be used with specific order of synapses that guarantee the best LLN and its training implementation in FPGA are shown in color. Becomes clear that in major cases t_3 regressor based implementation uses minimum of LUT, while t_4 regressor – guarantees the lowest latency. In the case of LUT and latency optimization, t_{12} regressor based implementation dominates in a range $M \in [2, 12]$ while the t_3 – becomes superior in a range $M \in [13, 28]$.

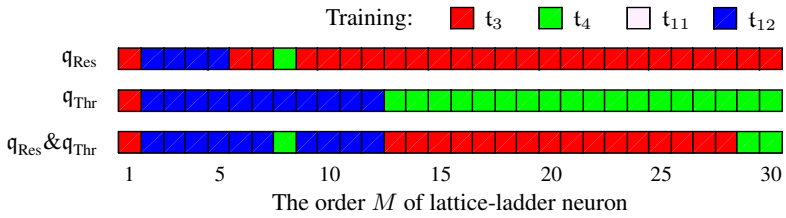


Fig. 4.8. The best lattice-ladder neuron implementations in field programmable gate array, when look-up tables, latency or both resources are minimized: types of regressor lattice to be used when specific synapse order M is needed

Fig. 4.9 shows the relative use of LUT in Artix-7 FPGA when LLN with its training is implemented using the best (the minimum LUT usage) regressor lattice. The LUT usage grows with the number of inputs and order of synapses as expected. The simplest LLN ($N = 1$, $M = 1$) needs less than 1% of available LUT, while $\sim 35\,\%$ of LUT are necessary when moderate size LLN ($N = 10$, $M = 10$) has to be implemented.

Similarly, Fig. 4.10 shows the latency in Artix-7 FPGA when LLN with its training is implemented using the best (the minimum latency) regressor lattice. The simplest LLN ($N = 1$, $M = 1$) implementation execution takes 90 clock cycles, while moderate size LLN ($N = 10$, $M = 10$) implementation executes in 300 clock cycles. Implementation latency dependence on the number of inputs and the order of synapses can be roughly approximated by the plane equation:

$$R_{Lat} \approx 22.5M + 1.7N + 57.4 \ \ (\pm 4.2) \text{ [clock cycles]}. \tag{4.5}$$

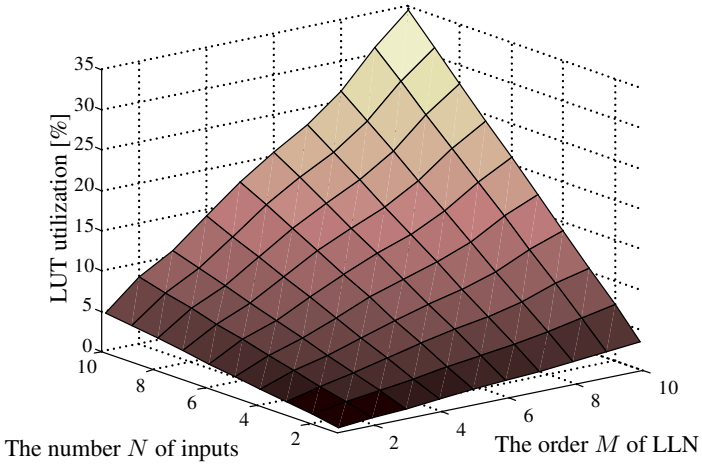


Fig. 4.9. The relative use of look-up tables in Artix-7 FPGA when lattice-ladder neuron with its training is implemented using the best regressor lattice

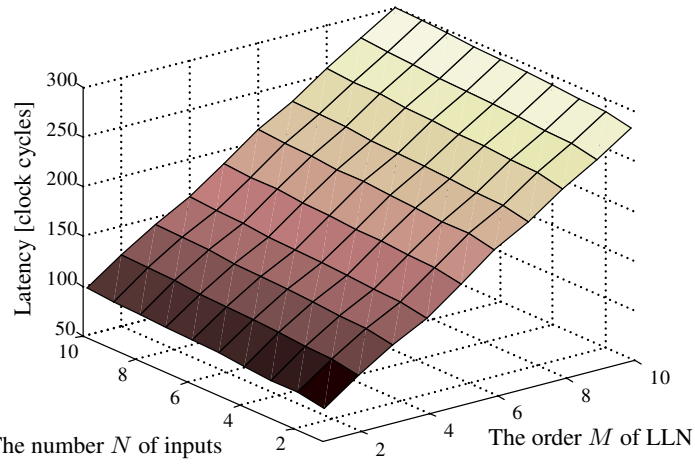


Fig. 4.10. The latency in Artix-7 FPGA when lattice-ladder neuron with its training is implemented using the best regressor lattice

The timing analysis of the LLN and its training circuit after placement and routing stage in ISE Design Suite 14.7 shows that the FPGA implemented neuron can be clocked at 300 MHz. Therefore, the LLN with 10 inputs and 10th order synapses can process input data and learn at a million samples per second speed.

4.1.4. Single Layer of Lattice-Ladder Multilayer Perceptron Implementation

The LLN IP core generated by our technique is compared with the same LLN core created in Xilinx Vivado HLS tool. For each NPE it is selected the identical 18 b precision for signal width. The neurons in both designs are implemented in the way shown in Fig. 2.9 dedicating NPE for each synapse and one additional for summation of synapse outputs and instantaneous error calculation. The both generated HDL designs of the IP cores are synthesized, mapped, placed and routed in Xilinx ISE Design Suite 14.7. The resource and timing reports are used to evaluate the performance of the implemented designs under the maximal sampling frequency f_{s}, LUT equivalent resources and recalculate the quality criteria: $\mathrm{q}^{\star}_{\mathrm{Thr}}$ and $\mathrm{q}^{\star}_{\mathrm{Res}}$.

The ratio of throughput criteria $\mathrm{q}^{\star}_{\mathrm{Thr}} \,/\, \mathrm{q}^{\mathrm{HLS}}_{\mathrm{Thr}}$ shows how much faster the LLN can sample and learn the signals exploiting our design in comparison with Vivado HLS. The ratio of the sampling frequency depends on the synapse order M and the type of regressor lattice, as shown in Fig. 4.11. It will not depend on number of neurons and layers, since their parallel implementation. The ratio of $\mathrm{q}_{\mathrm{Thr}}$ grows with the increasing of M and is in a range 3–11. The high ratio is affected by the relatively low maximal clock frequency $f_{\mathrm{s}}^{\mathrm{HLS}}$ of the Vivado HLS generated cores, e.g., the synapse cores with a training type t_3, t_4, t_{11}, t_{12} can be clocked by maximal frequencies 174 MHz, 179 MHz, 179 MHz, and 158 MHz respectively. However, synapse IP core generated by our technique

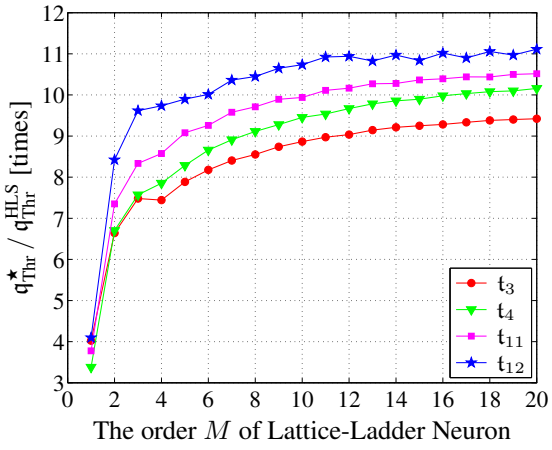


Fig. 4.11. The $\mathrm{q}_{\mathrm{Thr}}$ criteria rate dependency on the order of synapses of lattice-ladder neuron that were implemented in FPGA together with the training scheme based on four different types of regressor lattices

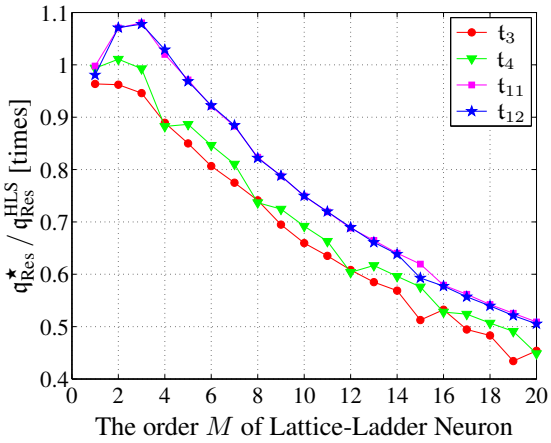


Fig. 4.12. The ratio of resource criteria $\mathfrak{q}_{\mathrm{Res}}$ dependent on the order of synapses of lattice-ladder neuron that were implemented in FPGA together with the training scheme based on four different types of regressor lattices

can be clocked at 320 MHz. The Vivado HLS cores also requires more clock cycles to process signals inducing largest latency. The highest clock frequency and lowest processing latency of our cores yields the $\mathfrak{q}_{\mathrm{Thr}}$ criteria improvement up to 11 times.

The ratio of the resource criteria $\mathfrak{q}^{\star}_{\mathrm{Res}} / \mathfrak{q}^{\mathrm{HLS}}_{\mathrm{Res}}$ shows how many times our LLN core outperforms the Vivado HLS generated one in terms of LUT equivalent resource utilization. In range of $M \in [2, 4]$ our core is advantageous, however in other M cases the Vivado HLS design uses less resources and the utilization curve decreases with rising M, as is shown in Fig. 4.12.

The evaluation of the LLN performance through both $\mathfrak{q}_{\mathrm{Thr}}$ and $\mathfrak{q}_{\mathrm{Res}}$ multiplied ratios shows, that our compiler generates at least 3 times more efficient core in comparison with Vivado HLS. Due to the Vivado HLS tool can not properly distinguish the DSP supportable subgraphs of LLN, our core compiler outperforms it by 3–10 times in investigated $M \in [1, 20]$ range.

4.1.5. Qualitative Lattice-Ladder Multilayer Perceptron Implementation

The investigation of LLMLP is performed changing the number of neurons N in each layer, synapse order M and total number of layers L. The LUT and latency optimal type of regressor lattice was selected according to specific synapse order and $\mathfrak{q}_{\mathrm{Thr}}$, $\mathfrak{q}_{\mathrm{Res}}$ criteria, as it was shown in Fig. 4.8. The results of different LLMLP configurations are described by Pareto efficient frontiers in

Fig. 4.13, Fig. 4.14, Fig. 4.15. Each connected lines of points represent Pareto optimal choices of the possible solutions for certain LLMLP configuration. All other points, which not belongs to Pareto frontier, are not shown here. The bold dashed lines show resources upper bounds of the existing FPGA families. These lines helps to select proper FPGA chip for final implementation of LLMLP core.

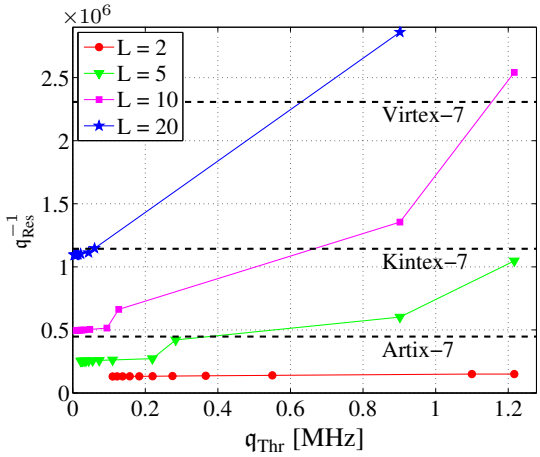


Fig. 4.13. Pareto frontiers of lattice-ladder multilayer perceptron implementation with different number of layers L when $M = 10$, $N = 10$

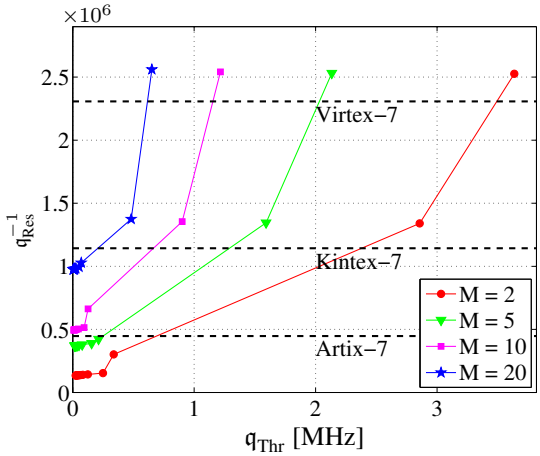


Fig. 4.14. Pareto frontiers of lattice-ladder multilayer perceptron implementation with different orders M when $L = 10$, $N = 10$

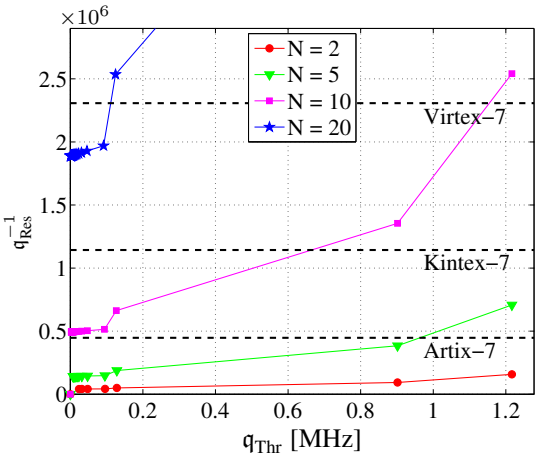


Fig. 4.15. Pareto frontiers of lattice-ladder multilayer perceptron implementation with different number of neurons N when $M = 10$, $L = 10$

All the investigated LLMLP configurations have same tendency reflected on the Pareto frontiers, where the increase of the throughput criteria $\mathfrak{q}_{\text{Thr}}$ gives the increment of LUT equivalent resources $R_{\equiv\text{LUT}}$ and vice versa. Thereby, it confirms the state of Pareto efficiency, when it is impossible to make one individual quantity better without making second one worse. The density of the points on a front is defined by the optimization process. On the left side of Pareto front it is observed that the optimizer tends to add NPEs to the LLMLP design with a small step forming a single layer first. Going to the right side of the frontiers the distribution of point is sparse due to larger resource increasing step, since the optimizer reproduces the layers and parallelizes forward and backward pass circuits. The right side points on the frontiers define the maximal sampling frequency for the particular LLMLP. The further increase of the maximal frequency is limited not by the signal latency in the LLMLP circuit, however because of the constrained FPGA resources.

The points in Pareto frontiers, which are above the Virtex-7 bound, show that there is impossible to implement concrete configuration LLMLP with maximal sampling frequency on the real 7th series FPGA chips.

4.2. Investigation of Lithuanian Speech Recognizer

The experimental verification of our created speech recognizer begins with the comparison to the initial Lithuanian speech recognizer (LSR) devepoled by

other researchers (Tamulevičius *et al.* 2010). Afterwards, the isolated word recognition accuracy is comparative evaluated for records of different speakers and four features extraction algorithms. The robustness of features extraction algorithms was tested recognizing the speech records at different signal to noise ratios and w/o signal preprocessing using LLN. Finally, the execution speed of developed IP cores for LSR is experimantally verified.

4.2.1. Comparison with Initial Developments

A previous FPGA implementation (Tamulevičius *et al.* 2010) was used as a benchmark for the evaluation of the current implementation. Both recognizers were tested in recognition of the Lithuanian isolated words (loaded to FPGA sequentially as **.wav* files) using the same LFCC features extraction.

The used dictionary in both implementations contains 100 unique words. All these words are often used in daily speech: 51 nouns, 23 verbs, 8 adjectives, 6 conjunctions, 5 adverbs, 4 pronouns and 3 prepositions. The dictionary contains: 1 word with 1 syllable, 57 words with 2 syllables, 31 words with 3 syllables, 7 words with 4 syllables, and 1 word with 5 syllables.

Recognizers were tested in a speaker dependent mode. All utterances were recorded in the office environment. 10 speakers (5 females and 5 males) participated to record the utterances. 10 sessions of 100 words were recorded for each speaker. Records of first 6 sessions were used in creation of the dictionary. Recognizers were tested using records of remaining 4 sessions. The average 97.7 % recognition rate over 40 experiments is achieved for the current implementation against 92.8 % for the previous implementation (see 1 and 4 columns in Table 4.2).

The important difference between previous and current recognizers is in the speech sampling frequency. The current fully hard-core implementation of the Lithuanian isolated word recognition uses a higher sampling frequency ($f_s = 11.025$ kHz), thus gives the 4.9 % better recognition accuracy in comparison with the FPGA based soft-core implementation ($f_s = 6$ kHz). It is worth mentioning that in the hard-core implementation only integer types of data are used, while in the soft-core implementation – float type of data was used. Moreover, the signal quantized in 16 b was used in the previous soft-core based implementation. In the current implementation an incoming speech signal is quantized in 8 b. Despite the principal algorithm changes and simplifications (particularly in FFT data depth and logarithm computation) it does not reduces the speaker dependent recognition rate.

Recognizers under investigation use the same dictionary, thus the duration of their recognition sub-processes can be directly compared. Only difference between clock frequencies needs to be taken into account. The sub-process

Table 4.2. Lithuanian isolated word recognition results

Recognition Stage	Time [ms]	Clock [MHz]	Recognition Rate [%]
Previous Implementation ($f_s = 6$ kHz)			
Feature Extraction	1370.000	100	92.8
Comparison	22.400		
Current Implementation ($f_s = 11.025$ kHz)			
Feature Extraction	8.520	50	97.7
Comparison	0.333		
Comparison (with constr.)	0.128		
Improvement [times]			
Feature Extraction	160	2	+5 %
Comparison	67		
Comparison (with constr.)	174		

of feature extraction (from 1.5 s duration signal) takes 1370 ms and 8.52 ms in previous initial and current implementations correspondingly (see columns 1–3 of Table 4.2). The hard-core implemented LFCC features extraction is accelerated 160 times, despite the used 1.8 times higher signal sampling frequency and twice lower main clock frequency. That actually recalculates to 320 times (160×2) speed up of feature extraction process.

The right way to evaluate the comparison sub-process is to measure the time required to calculate one DTW in both implementations. It appears that 22.407 ms is required in the previous implementation and 0.333 ms in the current implementation (without constraints) to calculate one DTW. The additional use of border constraints reduced the DTW calculation duration to 0.128 ms, thus 348 times higher acceleration was achieved.

It is important to emphasize that the current recognizer is able to compute approximately 7800 DTW/s at a relatively low (compared with CPU or GPU) 50 MHz clock frequency. On the other hand, only 1 s is needed to find the best match of up to 1.5 s duration signal features in the dictionary of records with the total length of 11700 s (3.2 h). In order to take this advantage fully the connection speed of approximately 17 MB/s with the external dictionary has to be ensured.

A hard-core implementation of the Lithuanian isolated word recognition utilizes in total 38 % of FPGA slice memory, as is shown in Fig. 4.16. It is 1.75 times less than the soft-core based algorithm implementation. There is enough logic memory space to duplicate a few DTW units and run them in parallel to reduce the comparison time. The amount of DSP slices is reduced from 35 to 21 units. The total number of BRAM units is increased by 46 %,

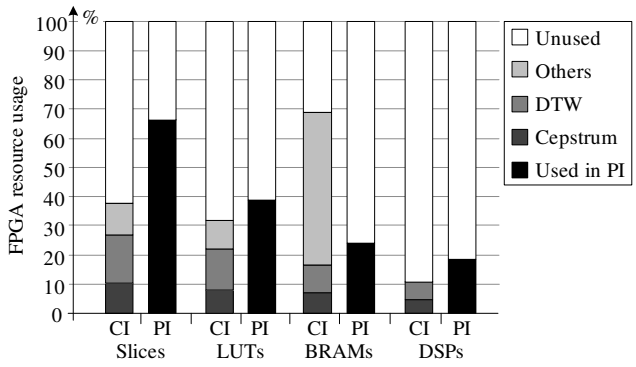


Fig. 4.16. Comparative FPGA utilization in: previous implementation (PI) and current implementation (CI) versions

because the context dictionary is stored in the internal FPGA memory for fast features access. The number of LUTs is reduced by 7 % in comparison with the previous initial LSR implementation, as shown in Fig. 4.16.

4.2.2. Recognition Accuracy Tune-Up

The dictionary contains the isolated Lithuanian words pronounced by 5 male and 5 female speakers. 4 sessions of 100 words were recorded for each speaker in the office environment. The first session was used for the training and the rest 3 sessions were used for the testing. The recognition rate was tested using original records and using the same words with 30 dB and 15 dB signal to noise ratios (SNR). The aim is to determine most suitable features extraction algorithm among MFCC, LFCC, LPCC and LPC for speaker dependent recognizer. The averaged results over male and female speakers without filtering IP core are given in Table 4.3. The best average recognition rate was achieved for original records using MFCC (93.2 %) and LFCC (93.0 %) features. The LFCC (90.8 %) and little less LPCC (89.8 %) features are more robust to noise when SNR is 30 dB. The LPCC (83.7 %) features give better recognition accuracy when SNR is 15 dB. The LPC features give the worst recognition rates in all cases, comparing with others features extraction algorithms.

The recognition accuracy highly depends on speaker gender, tone of voice, articulation and speed of pronunciation. It is important to diagnose the reasons of differences in rates while using same features extraction method. Each entry in the Table 4.4 and Table 4.5 evaluates the recognition accuracy over three times repeated experiments. In the male ($\mathcal{M}_i$) speaker case there are two speakers ($\mathcal{M}_1$ and $\mathcal{M}_2$) with high recognition accuracy over all features ex-

Table 4.3. Averaged isolated word recognition results

Features	Original records	Records with SNR = 30 dB	Records with SNR = 15 dB
MFCC	93.2	86.7	79.3
LFCC	93.0	90.8	72.4
LPCC	91.5	89.8	83.7
LPC	82.5	78.6	67.1

Table 4.4. Isolated word recognition rates for male speakers [%]

Speaker	$\mathcal{M}_1$	$\mathcal{M}_2$	$\mathcal{M}_3$	$\mathcal{M}_4$	$\mathcal{M}_5$	Average	With LLN
MFCC							
Original	100	98	91	83	86	91.6	0
30 dB	90	95	83	76	78	84.4	+2.2
15 dB	87	92	77	62	66	76.8	+5.0
LFCC							
Original	99	97	91	84	89	92.0	0
30 dB	96	94	79	83	91	88.6	+1.8
15 dB	91	72	65	69	75	74.4	+5.1
LPCC							
Original	99	95	90	77	79	88.0	0
30 dB	94	91	77	84	88	86.8	+0.8
15 dB	93	85	72	79	77	81.2	+4.5
LPC							
Original	91	85	73	68	57	74.8	0
30 dB	79	82	57	66	64	69.6	+1.6
15 dB	82	64	48	57	56	61.4	+2.8

traction methods and different SNRs. There are speakers ($\mathcal{M}_4$ and $\mathcal{M}_5$) with relatively lower rates comparing with others. These two speakers have low tone of voice, therefore the spectral information is distributed in low frequency domain. Some errors are caused by different pronunciation of the same word in the following sessions.

In the case of female speakers ($\mathcal{F}_i$) the $\mathcal{F}_2$ has lowest rate. In the $\mathcal{F}_2$ records the low-frequency periodic noise is observed. The speaker $\mathcal{F}_5$ pronounces the words faster than $\mathcal{F}_1$, $\mathcal{F}_3$ and $\mathcal{F}_4$. It influences the accuracy, as is shown in Table 4.5. The average recognition rate of the female records outperforms the rate of male records in all features extraction cases (Fig. 4.17). The confidence interval of recognition accuracy with 0.95 probability was calculated from the Beta distribution law approximated by χ^2 distribution (Kruopis 1993). Other researchers declare similar 91–96 % recognition rates in the speech recognition systems based on software (Lileikytė, Telksnys 2011; Maskeliūnas, Esposito

Table 4.5. Isolated word recognition rates for female speakers [%]

Speaker	$\mathcal{F}_1$	$\mathcal{F}_2$	$\mathcal{F}_3$	$\mathcal{F}_4$	$\mathcal{F}_5$	Average	With LLN
MFCC							
Original	98	87	98	99	92	94.8	0
30 dB	94	82	91	97	81	89.0	+1.8
15 dB	92	76	83	93	65	81.8	+4.7
LFCC							
Original	95	86	95	99	95	94.0	0
30 dB	95	85	93	98	94	93.0	+0.2
15 dB	67	65	72	81	67	70.4	+4.2
LPCC							
Original	97	89	98	99	92	95.0	0
30 dB	97	85	93	99	90	92.8	+0.6
15 dB	87	74	83	98	89	86.2	+4.9
LPC							
Original	93	87	92	97	82	90.2	0
30 dB	89	81	92	95	81	87.6	+0.8
15 dB	72	66	73	87	66	72.8	+3.3

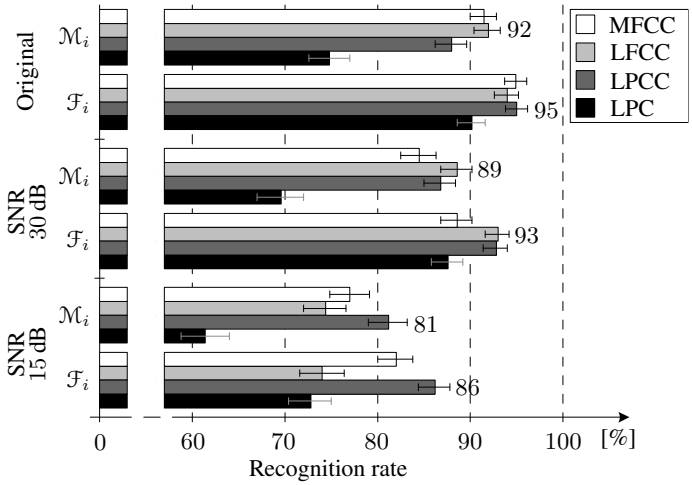


Fig. 4.17. The average recognition rate of isolated word pronounced by male and female speaker using different features extraction algorithms

2012; Wijoyo 2011; Zhou *et al.* 2011) and hardware (Cheng *et al.* 2011; Pan, Li 2012; Zhang *et al.* 2011) implementations.

The last column in Table 4.4 and Table 4.5 shows the recognition rate improvement using the LLN (Fig. 4.5) in filtering IP core. The experiments

with misrecognised words are repeated with the aim to verify the influence of noise filtering on recognition rate improvement. The original isolated word records are used to train the LLN. Therefore, no rate improvement is noticed under "Original" data row. During the training stage the synapses adapt their pass-band to voice band of certain speaker. The experiment result of 15 dB SNR records recognition shows that the rate can be improved up to 5 % using filtering IP core. The rate is improved up to 2 % for records with 30 dB SNR.

An experimental investigation was performed in order to evaluate the performance of the proposed double random seed matching algorithm. Taking into account that the initial reference is selected randomly the number of experiments were selected equal to 20 for each algorithm parameter set used during experimental investigation. The $\mathcal{M}_1$ and $\mathcal{F}_4$ sets of speaker dependent original records of isolated words are used for the dictionary formation. A 100 Lithuanian speech words were selected for experimental investigation. Each word is recorded during four different record sessions.

The success of pattern matching depends on:

- the differences between signals;
- features selected for matching;
- pattern matching algorithm.

The performance of DTW based matching using experimental data was performed using different feature metrics for the speech signal, such as LFCC, LPC, LPCC, MFCC to find the best features for speaker dependent isolated word recognition. The Mel-scale based cepstrum coefficients MFCC were selected as the best feature vectors by comparing the pattern matching results between different speech recording sessions. The use of MFCC and a full search based on DTW let us receive 99–100 % precision on isolated word recognition for the best case $\mathcal{M}_1$ and $\mathcal{F}_4$ sets.

The experimental investigation of double random seed matching algorithm should find the answer to the following questions:

- Do the selection of initial reference r_i has the influence to the matching performance?
- How the number of neighbouring reference-to-reference matches selected in Algorithm 3.1 affects the number of pattern matching failures?
- Do the use of additional initial reference r_a may reduce the number of pattern matching failures not increasing the number or DTW computations twice?

The mean number of computations performed using one initial reference (r_i) and two (r_i and r_a) is shown in Fig. 4.18. The lower curve shows the

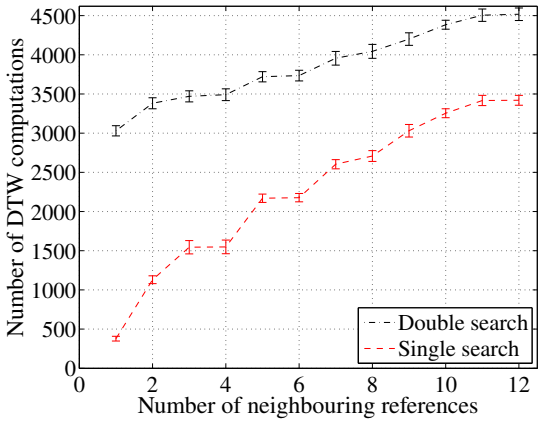


Fig. 4.18. Illustration of the computational load dependence on the number of selected neighbouring references

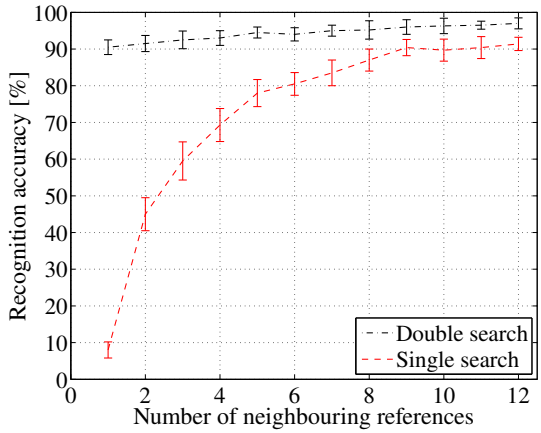


Fig. 4.19. Pattern match rate dependence on the number of selected neighbouring references

amount of DTW computations performed to make a decision accordingly to the rule of the Algorithm 3.1, proposed in Section 3.3. The upper curve shows the total number of DTW computations that should be performed if additional reference r_a is used for matching. It can be noticed, that the total amount of computations rises with lower speed comparing to the situation with only one reference (r_i) selected. However we should notice also, that at least 30 % of references should be compared to make a decision when both r_i and r_a are used.

The use of additional reference r_a increases the precision of pattern matching significantly (Fig. 4.19). Comparing the number of correctly matched references (Fig. 4.19) and the number of computations performed (Fig. 4.18) we see that the double increase of the number of neighbouring references, used for matching, has a little impact to the number of correctly matched patterns. Additional 3–4 % of matching precision may cost the increase of number of DTW computations from 30 % to 38 %.

The matching rate should be always put in front of the computational load of the algorithm. However for the real-time application of the algorithm, e.g., implementation of the algorithm in speech recognition based portable controller, the high computational load may limit the size of the dictionary, that could be applied and may consume too much power from the battery, making the controller not portable anymore.

4.2.3. Execution Speed Determination

The time-line diagram of feature extraction and comparison stages is shown in Fig. 4.20. The gray and white rows mark the amount of time needed for FPGA-based and CPU-based computations respectively (Matlab was used as software running on personal computer (PC) with 50 % usage of 3 GHz CPU). CPU-based MFCC, LPCC and LPC extraction is more than 10 and 5 times faster respectively. The higher calculation speed is strongly influenced by 30 times higher CPU clock frequency. The advantage of FPGA against CPU is observed in LFCC and DTW calculations. FPGA-based LFCC runs 1.5 times faster than CPU-based because of simplifications in logarithm and FFT core employment. FPGA-based DTW and DTW_c calculation is speed-up more than 280 times in comparison with same algorithm implemented in Matlab.

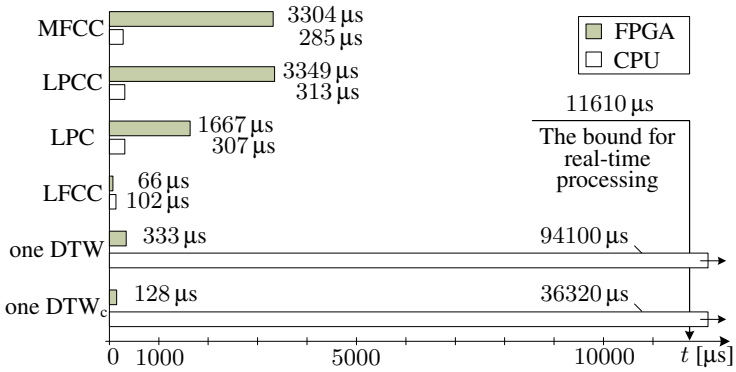


Fig. 4.20. The time-line diagram of feature extraction and comparison stages based on FPGA and CPU

In order to run recognition in real-time the feature extraction delay must be less than 11.61 ms. All feature extraction cores satisfies the requirement for operation in a real-time. Duration of LPCC or MFCC calculation is more than 50 times longer in comparison with LFCC calculation time. The reason is the linear prediction implementation on soft-core processor. The autocorrelation, LFCC and DTW modules run at 50 MHz, while the soft-core processor runs at 100 MHz clock frequencies. The delay of signal processing is very convenient to present in a time-line, as is shown in Fig. 4.21.

Calculation of LFCC features for one word takes 66 µs. This is 174.8 times faster than real-time recognition requires. LPCC feature extraction takes 3349 µs and it is 3.46 times faster that real-time operation requirement. Similarly, the MFCC are calculated 3.51 times faster than real-time operation requirement. Therefore, one soft-core processor can be used for LPCC or MFCC analysis of speech signals from 3 different data channels simultaneously. The delays for one DTW and DTW_c process are 333 µs and 128 µs respectively. In non real-time mode the whole set of reference words is sequentially analyzed by DTW (or DTW_c) algorithm and it takes 33.30 ms (12.81 ms in DTW_c case). This timespan does not enables real-time recognition because only 34 DTW (or 90 DTW_c) comparisons are performed in 11.61 ms. In order to make system more applicable and usable with larger dictionaries it is proposed to use activation word to initiate the recognition process. Recognition of this activation word will require only one DTW comparison in real-time. After the successful recognition of activation word the system captures pronounced utterance and recognizes it in real-time if number of comparable words is less than 90, otherwise the word will be recognized in non-real-time mode with 7800 DTW/s speed.

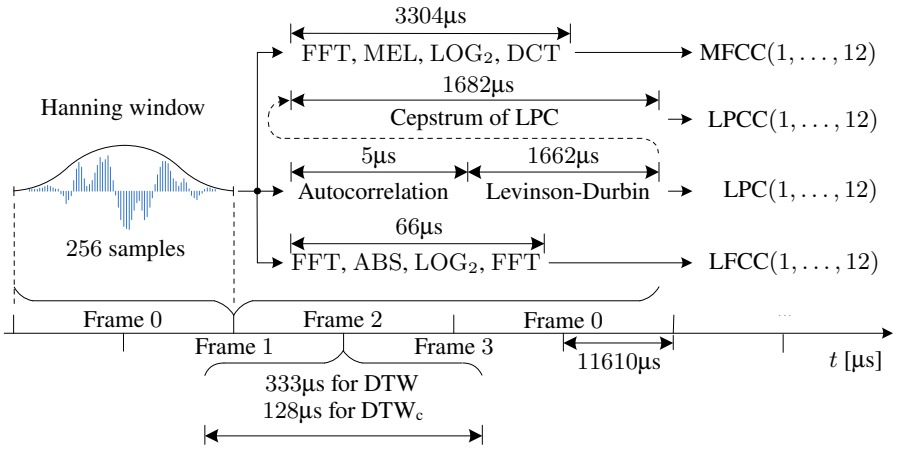


Fig. 4.21. Time-line of features extraction and comparison

The fastest version of software-based comparison process (Tamulevičius *et al.* 2010) is speeded-up 134 times using DTW algorithm and 348 times using DTW_c algorithm. LFCC feature extraction is speeded-up 160 times. The average recognition rate is improved by 1 % (up to 94 %) using LFCC and by 2 % (up to 95 %) using LPCC and MFCC features. Other researchers declare similar recognition rates of 90–93 % on hardware-based speech recognizers (Cheng *et al.* 2011; Choi *et al.* 2010; Pan *et al.* 2011; Veitch *et al.* 2010; Zhang *et al.* 2011).

4.3. Conclusions of the 4th Chapter

1. The application of MFCC and LFCC is more suitable for recognition of clean records. The LFCC and LPCC features are appropriate for 30 dB environment noise. The LPCC features gives the best accuracy at 15 dB SNR comparing to other algorithms at same noise level.
2. Using filtering intellectual property (IP) core the recognition rate of 15 dB and 30 dB SNR records can be improved up to 5 % and 2 % respectively.
3. The double random seed matching algorithm reduces the computational load of the classical dynamic time warping (DTW) based speech recognition algorithm by 62–70 % keeping the 90–97 % rate of correctly recognised words.
4. Comparing with a previous soft-core recognizer the current hardware based implementation accelerates features extraction up to 320 times. The DTW with border constraints is speeded up 348 times.
5. For the investigated neuron with the bandwidth wider than 10 % of normalized reference bandwidth even 10 b word length representation is appropriate as its output signal mean error is less than 1 %.
6. The use of t_{11} regressor lattice must be avoided. For look-up table (LUT) critical lattice-ladder neuron (LLN) implementations regressor lattice t_3 has to be used when $M \geq 6$. For latency critical LLN implementations two regressor lattices are superior: t_{12} ($M \in [2, 12]$), t_3 ($M \leq 13$). For LUT and latency critical LLN implementations (with a few exceptions $M = \{1, 8\}$) the same regressor lattices are superior: t_{12} ($M \in [2, 12]$) and t_3 ($M \in [13, 28]$).
7. Block RAM size of 2 kB is sufficient to achieve tolerable less than 0.4 % maximum error of the approximated activation function output with small RMS error $\mathscr{E}_{\text{RMS}}^{\text{AR}} = 1.8{\times}10^{-3}$ of the lattice-ladder neuron transfer function.

8. The corresponding lattice-ladder multilayer perceptron (LLMLP) compiler generates at least 3 times more efficient LLMLP core in comparison with Vivado HLS tool.
9. The Pareto efficient frontiers allows to properly select concrete FPGA chip according to required amount of resources, maximal data sampling frequency and LLMLP structure.

General Conclusions

The problem of efficient and straightforward implementation of operating in a real-time electronic intelligent systems on FPGA was solved. The following significant results for scientific field of Electrical and Electronic engineering are obtained:

1. The technique for lattice-ladder multilayer perceptron (LLMLP) efficient implementation in FPGA is created and experimentally affirmed:
   1.1. The corresponding LLMLP compiler generates at least 3 times more efficient core in comparison with Vivado HLS tool.
   1.2. The developed neuron processing element based on FPGA DSP slice structure guarantees less than $1\,\%$ output signal mean absolute error, $\epsilon_{\mathrm{T}} = 1.8{\times}10^{-3}$ root mean square error of the lattice-ladder neuron transfer function and is speed optimal with only 7 clock cycle latency.
   1.3. The elucidated experimental evidence grounded knowledge on the quickest training algorithms of LLMLP and the best lattice regressors $\mathcal{T} \in \{\mathrm{t}_3, \mathrm{t}_4, \mathrm{t}_{11}, \mathrm{t}_{12}\}$ use for look-up table (LUT) and latency critical implementations is incorporated in the technique.
2. Pareto frontiers estimation for the LLMLP specialized criteria of circuitry synthesis is proposed:
   2.1. An optimal choice of FPGA chip according to given requirements for LLMLP structure, sampling frequency and other resources is done.

2.2. Two implementation strategies: throughput or resource optimization, are developed.

3. The optimized FPGA intellectual property (IP) cores are developed and experimentally verified in Lithuanian speech recognizer:

3.1. The use of the lattice-ladder neuron LLN IP core improves the recognition rate at least by $4\,\%$ for 15 dB SNR records.

3.2. The application of MFCC and LFCC IP cores is suitable for recognition of clean records. Moreover the LFCC and LPCC IP cores are appropriate for use at 30 dB SNR, while only the LPCC delivers the best accuracy at 15 dB SNR.

3.3. Hardware optimized single isolated word matching (DTW_c) IP core executes in $128\,\mu$s achieving 7800 word/s comparison speed.

4. A first working prototype of Lithuanian speech recognizer in FPGA for disabled persons is created and tested:

4.1. A new accelerated pattern matching algorithm uses constrained dynamic time warping and enables to recognize words 2.6 times quicker without the loose of recognition accuracy at $97\,\%$.

4.2. The use of a new double random seed matching algorithm additionally can speed up the recognition up to 2.6 times with a disadvantage of recognition accuracy reduction till $90\,\%$.

References

Ai, O. C.; Hariharan, M.; Yaacob, S.; Chee, L. S. 2012. Classification of Speech Dysfluencies with MFCC and LPCC Features, *Expert Systems with Applications* 39(2): 2157–2165. [see 37 p.]

Alberto de Albuquerque Silva, C.; Duarte Doria Neto, A.; Alberto Nicolau Oliveira, J.; Dantas Melo, J.; Simonetti Barbalho, D.; Medeiros Avelino, A. 2015. Definition of an architecture to configure artificial neural networks topologies using partial reconfiguraton in FPGA, *Latin America Transactions, IEEE (Revista IEEE America Latina)* 13(7): 2094–2100. [see 29 p.]

Alecsa, B.; Cirstea, M.; Onea, A. 2012. Simulink modeling and design of an efficient hardware-constrained FPGA-based PMSM speed controller, *IEEE Transactions on Industrial Informatics* 8: 554–562. ISSN 1551-3203. [see 10 p.]

Altera 2013. *Chip Editor* [interactive] [14 March 2014]. Prieiga per internetą: <http://www.altera.com/literature/hb/qts/qts_qii52006.pdf>. [see 32 p.]

Altera 2014. *FFT IP Core User Guide* [interactive] [25 August 2015]. Prieiga per internetą: <https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/ug/ug_fft.pdf>. [see 38 p.]

Amin, H.; Curtis, K.; Hayes-Gill, B. 1997. Piecewise linear approximation applied to nonlinear function of a neural network, *Circuits, Devices and Systems, IEE Proceedings* - 144(6): 313–317. ISSN 1350-2409. [see 33 p.]

Amin, H.; Curtis, K.; Hayes Gill, B. 1999. Two-ring systolic array network for artificial

neural networks, *Circuits, Devices and Systems, IEE Proceedings* - 146(5): 225–230. ISSN 1350-2409. [see 31 p.]

Amudha, V.; Venkataramani, B.; Manikandan, J. 2008. FPGA implementation of isolated digit recognition system using modified back propagation algorithm, in *Proc. ICED'08*, 1–6. [see 36, 41 p.]

Anguita, D.; Carlino, L.; Ghio, A.; Ridella, S. 2011. A FPGA core generator for embedded classification systems, *Journal of Circuits, Systems, and Computers* 20(2): 263–282. ISSN 0218-1266. [see 28 p.]

Anguita, D.; Ghio, A.; Pischiutta, S.; Ridella, S. 2008. A support vector machine with integer parameters, *Neurocomputing* 72(1?3): 480 – 489. ISSN 0925-2312. [see 28 p.]

Arias-Garcia, J.; Braga, A.; Llanos, C. H.; Ayala-Rincon, M.; Pezzuol Jacobi, R.; Foltran, A. 2013. FPGA HIL simulation of a linear system block for strongly coupled system applications, in *Industrial Technology (ICIT), 2013 IEEE International Conference on*, 1017–1022. ISBN 978-1-4673-4567-5. [see 10 p.]

Armato, A.; Fanucci, L.; Scilingo, E. P.; De Rossi, D. 2011. Low-error digital hardware implementation of artificial neuron activation functions and their derivative, *Microprocessors and Microsystems* 35(6): 557–567. [see 29, 30, 67, 94 p.]

Arminas, V.; Tamulevicius, G.; Navakauskas, D.; Ivanovas, E. 2010. Acceleration of feature extraction for FPGA-based speech recognition, in *Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments 2010*, International Society for Optics and Photonics, 511–516. [see 35 p.]

Atri, M.; Sayadi, F.; Elhamzi, W.; Tourki, R. 2012. Efficient Hardware/Software Implementation of LPC Algorithm in Speech Coding Applications, *Journal of Signal and Information Processing* 3(9): 122–129. [see 37, 40 p.]

Back, A. D.; Tsoi, A. C. 1993. A simplified gradient algorithm for iir synapse multilayer perceptrons, *Neural Computation* 5(3): 456–462. [see 16 p.]

Bahoura, M. 2014. FPGA implementation of high-speed neural network for power amplifier behavioral modeling, *Analog Integrated Circuits and Signal Processing* 79(3): 507–527. [see 29, 30 p.]

Bahoura, M.; Ezzaidi, H. 2013. Hardware implementation of mfcc feature extraction for respiratory sounds analysis, in *8th Workshop on Systems, Signal Processing and their Applications*, 226–229. [see 38, 39 p.]

Beiu, V. 1998. How to build vlsi-efficient neural chips, in *Proceedings of the International ICSC Symposium on Engineering of Intelligent Systems*, 66–75. [see 33 p.]

Bosque, G.; del Campo, I.; Echanobe, J. 2014. Fuzzy systems, neural networks and neuro-fuzzy systems: A vision on their hardware implementation and platforms over two decades, *Engineering Applications of Artificial Intelligence* 32(1): 283–331. ISSN 0952-1976. [see 16 p.]

Broersen, P. M. 1994. A comparison of transfer function estimators, in *Instrumentation*

and Measurement Technology Conference, 1994. IMTC/94. Conference Proceedings. 10th Anniversary. Advanced Technologies in I & M., 1994 IEEE, IEEE, 1377–1380. [see 68 p.]

Buyukkurt, B.; Najjar, W. A. 2008. Compiler generated systolic arrays for wavefront algorithm acceleration on FPGAs, in *Proc. FPL'08*, 655–658. [see 42 p.]

Cadence 2013. *C-to-Silicon Compiler High-Level Synthesis* [interactive] [10 December 2013]. Prieiga per internetą: <http://www.cadence.com/rl/Resources/datasheets/C2Silicon_ds.pdf>. [see 14 p.]

del Campo, I.; Finker, R.; Echanobe, J.; Basterretxea, K. 2013. Controlled accuracy approximation of sigmoid function for efficient FPGA-based implementation of artificial neurons, *Electronics Letters* 49(25): 1598–1600. [see 29, 30 p.]

Carrillo, S.; Harkin, J.; Mcdaid, L.; Pande, S.; Cawley, S.; Mcginley, B.; Morgan, F. 2012. Advancing interconnect density for spiking neural network hardware implementations using traffic-aware adaptive network-on-chip routers, *Neural Netw.* 33: 42–57. ISSN 0893-6080. [see 20 p.]

A look at trends from Consumer Electronics Show [interactive]. 2016. Green Tech Media [14 January 2016]. Prieiga per internetą: <http://www.greentechmedia.com/>. [see 3 p.]

Chakravarty, A. 2014. *Speech recognition toolkit for the Arduino* [interactive] [20 April 2014]. Prieiga per internetą: <http://http://arjo129.github.io/uSpeech/>. [see 35 p.]

Chan, K. Y.; Nordholm, S.; Yiu, K. F. C.; Togneri, R. 2013. Speech enhancement strategy for speech recognition microcontroller under noisy environments, *Neurocomputing* 118(1): 279–288. ISSN 0925-2312. [see 15 p.]

Chen, D.; Cong, J.; Pan, P. 2006. FPGA design automation: A survey, *Foundations and Trends in Electronic Design Automation* 1(3): 195–330. [see 10 p.]

Cheng, O.; Abdulla, W.; Salcic, Z. 2011. Hardware–software codesign of automatic speech recognition system for embedded real-time applications, *Industrial Electronics, IEEE Transactions on* 58(3): 850–859. [see 39, 110, 115 p.]

Choi, J.; You, K.; Sun, W. 2010. An FPGA implementation of speech recognition with weighted finite state transducers, in *Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on*, IEEE, 1602–1605. [see 34, 35, 115 p.]

Chung, J.-G.; Parhi, K. 1996. *Pipelined Lattice and Wave Digital Recursive Filters*: The International Series in Engineering and Computer Science. Springer. [see 18 p.]

Chung, J.-G.; Parhi, K. 2012. *Pipelined lattice and wave digital recursive filters*. Springer Science & Business Media. [see 22 p.]

Chung, J.-H.; Yoon, H.; Maeng, S. R. 1992. A systolic array exploiting the inherent parallelisms of artificial neural networks, *Microprocess. Microprogram.* 33(3):

145–159. ISSN 0165-6074. Prieiga per internetą: <http://dx.doi.org/10.1016/0165-6074(92)90017-2>. [see 31 p.]

Clayton, C. 2008. Digital duct tape with FPGA editor, *XCell* 66: 54–57. [see 32 p.]

Cong, J.; Jiang, W. 2008. Pattern-based behavior synthesis for FPGA resource reduction, in *Proceedings of the 16th international ACM/SIGDA symposium on Field programmable gate arrays*, 107–116. [see 15, 54 p.]

Cong, J.; Liu, B.; Neuendorffer, S.; Noguera, J.; Vissers, K.; Zhang, Z. 2011. High-level synthesis for fpgas: From prototyping to deployment, *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* 30(4): 473–491. ISSN 0278-0070. [see 15 p.]

Cornu, T.; Ienne, P. 1994. Performance of digital neuro-computers, in *Microelectronics for Neural Networks and Fuzzy Systems, 1994., Proceedings of the Fourth International Conference on*, 87–93. [see 33 p.]

Czajkowski, T. S. 2008. *Physical synthesis toolkit for area and power optimization on FPGAs*: Dissertation. University of Toronto. University of Toronto. 134 p. ISBN 978-0-494-57976-3. [see 10 p.]

Darabkh, K. A.; Khalifeh, A. F.; Bathech, B. A.; Sabah, S. W. 2012. Efficient DTW-Based Speech Recognition System for Isolated Words of Arabic Language, *World Academy of Science, Engineering and Technology* 77: 85–88. [see 37 p.]

Dessouky, G.; Klaiber, M. J.; Bailey, D. G.; Simon, S. 2014. Adaptive dynamic on-chip memory management for FPGA-based reconfigurable architectures, in *Field Programmable Logic and Applications (FPL), 2014 24th International Conference on*, IEEE, 1–8. [see 96 p.]

Ding, H.; Trajcevski, G.; Scheuermann, P.; Wang, X.; Keogh, E. 2008. Querying and mining of time series data: Experimental comparison of representations and distance measures, in *Proc. VLDB'08*, 1542–1552. [see 41 p.]

Dinu, A.; Cirstea, M.; Cirstea, S. 2010. Direct neural-network hardware-implementation algorithm, *Industrial Electronics, IEEE Transactions on* 57(5): 1845–1848. ISSN 0278-0046. [see 28 p.]

Er, M. J.; Li, Z.; Cai, H.; Chen, Q. 2005. Adaptive noise cancellation using enhanced dynamic fuzzy neural networks, *Fuzzy Systems, IEEE Transactions on* 13(3): 331–342. [see 16 p.]

Farjo, J.; Aoun, M.; Kassem, A.; Hamouche, M.; *et al.* 2012. Speaker identification on compactrio, in *Electrotechnical Conference (MELECON), 2012 16th IEEE Mediterranean*, IEEE, 399–403. [see 39 p.]

Ferreira, P.; Ribeiro, P.; Antunes, A.; Dias, F. M. 2007. A high bit resolution FPGA implementation of a fnn with a new algorithm for the activation function, *Neurocomputing* 71(1): 71–77. [see 19, 29, 30 p.]

Finker, R.; del Campo, I.; Echanobe, J.; Doctor, F. 2013. Multilevel adaptive neural

network architecture for implementing single-chip intelligent agents on FPGAs, in *Neural Networks (IJCNN), The 2013 International Joint Conference on*, 1–9. ISSN 2161-4393. [see 18 p.]

Firdauzi, A.; Wirianto, K.; Arijal, M.; Adiono, T. 2013. Design and implementation of real time noise cancellation system based on spectral subtraction method, *Procedia Technology* 11(1): 1003–1010. ISSN 2212-0173. [see 16 p.]

Fook, C. Y.; Hariharan, M.; Yaacob, S.; Ah, A. 2012. Malay speech recognition in normal and noise condition, in *IEEE 8th International Colloquium on Signal Processing and its Applications (CSPA 2012)*, 409–412. [see 37 p.]

Gavat, I.; Militaru, D. M.; Dumitru, C. O. 2008. Knowledge resources in automatic speech recognition and understanding for Romanian language, *InTech Speech Recognition Technologies and Applications* 1: 241–260. [see 41 p.]

Gomperts, A.; Ukil, A.; Zurfluh, F. 2011. Development and implementation of parameterized FPGA-based general purpose neural networks for online applications, *Industrial Informatics, IEEE Transactions on* 7(1): 78–89. [see 29, 30 p.]

Gray, A. H.; Markel, J. D. 1975. A Normalized Digital Filter Structure, *IEEE Transaction on Acoustics, Speech, and Signal Processing* 23(3): 268–277. [see 22 p.]

Hadei, S. A.; Lotfizad, M. 2011. A family of adaptive filter algorithms in noise cancellation for speech enhancement, *International Journal of Computer and Electrical Engineering* 2(2). [see 16 p.]

Hammerstrom, D. 1990. A vlsi architecture for high-performance, low-cost, on-chip learning, in *Neural Networks, 1990., 1990 IJCNN International Joint Conference on*, vol. 2, 537–544. [see 31, 33 p.]

Harkin, J.; Morgan, F.; McDaid, L.; Hall, S.; McGinley, B.; Cawley, S. 2009. A reconfigurable and biologically inspired paradigm for computation using network-on-chip and spiking neural networks, *Int. J. Reconfig. Comput.* 2009: 1–13. ISSN 1687-7195. [see 20 p.]

Himavathi, S.; Anitha, D.; Muthuramalingam, A. 2007. Feedforward neural network implementation in FPGA using layer multiplexing for effective resource utilization, *Neural Networks, IEEE Transactions on* 18(3): 880–888. [see 29 p.]

Hirsimaki, T.; Kurimo, M. 2004. Decoder issues in unlimited Finnish speech recognition, in *Proc. NORSIG'04*, 320–323. [see 34, 41 p.]

Holt, J.; Hwang, J.-N. 1993. Finite precision error analysis of neural network hardware implementations, *Computers, IEEE Transactions on* 42(3): 281–290. ISSN 0018-9340. [see 17, 33 p.]

Hussain, S. M. A.; Rashid, A. B. M. 2012. Optimization of VLSI architectures for DTW, in *Proc. ICECE'12*, 737–740. [see 41 p.]

Ivanovas, E. 2012. *Development and implementation of means for word duration signal processing*: Doctoral Dissertation. Vilnius Gediminas Technical University. Vil-

nius: Technika. [see 35 p.]

Johnston, S.; Prasad, G.; Maguire, L.; McGinnity, M. 2005. Comparative investigation into classical and spiking neuron implementations on FPGAs, in *Proceedings of the 15th International Conference on Artificial Neural Networks: Biological Inspirations - Volume Part I*: ICANN'05, 269–274. [see 20 p.]

Kazanavičius, E.; Venteris, R. 2000. Architectures for ultrasonic delay time estimation tasks, *Ultragarsas" Ultrasound"* 34(1): 23–27. [see 15 p.]

van Keulen, E.; Colak, S.; Withagen, H.; Hegt, H. 1994. Neural network hardware performance criteria, in *Neural Networks, 1994. IEEE World Congress on Computational Intelligence., 1994 IEEE International Conference on*, vol. 3, 1955–1958. [see 33 p.]

Kim, C.-M.; Park, H.-M.; Kim, T.; Choi, Y.-K.; Lee, S.-Y. 2003. FPGA implementation of ica algorithm for blind signal separation and adaptive noise canceling, *Neural Networks, IEEE Transactions on* 14(5): 1038–1046. [see 16 p.]

Kim, D.; Kim, H.; Kim, H.; Han, G.; Chung, D. 2005. A simd neural network processor for image processing, in *Proceedings of the Second International Conference on Advances in Neural Networks - Volume Part II*: ISNN'05, Berlin, Heidelberg: Springer-Verlag, 665–672. ISBN 3-540-25913-9. [see 31 p.]

Kruopis, J. 1993. *Matematinė statistika*: vadovėlis. Vilnius: Mokslas. 416 p. [see 109 p.]

Kuon, I.; Rose, J. 2007. Measuring the gap between FPGAs and ASICs, *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* 26(2): 203–215. ISSN 0278-0070. [see 10 p.]

Laird, J.; Szymanski, R.; Ryan, C.; Gonzalez-Alvarez, I. 2013. A Labview based FPGA data acquisition with integrated stage and beam transport control, *Nuclear Instruments and Methods in Physics Research Section B: Beam Interactions with Materials and Atoms* 7(2): 1–5. ISSN 0168-583X. [see 10 p.]

Lavin, C.; Nelson, B.; Hutchings, B. 2013. Impact of hard macro size on FPGA clock rate and place/route time, in *Field Programmable Logic and Applications (FPL), 2013 23rd International Conference on*, IEEE, 1–6. [see 11 p.]

Lileikytė, R.; Telksnys, L. 2011. Quality Estimation Methodology of Speech Recognition Features, *Electronics and Electrical Engineering* 110: 113–116. [see 34, 36, 37, 109 p.]

Liu, H.; Bergmann, N. W. 2010. An FPGA softcore based implementation of a bird call recognition system for sensor networks, in *Design and Architectures for Signal and Image Processing (DASIP), 2010 Conference on*, IEEE, 1–6. [see 39 p.]

Lotrič, U.; Bulić, P. 2012. Applicability of approximate multipliers in hardware neural networks, *Neurocomputing* 96: 57–65. [see 29, 30 p.]

Lotrič, U.; Bulić, P. 2011. Logarithmic multiplier in hardware implementation of neural networks, in *Proceedings of the 10th international conference on Adaptive and*

natural computing algorithms - Volume Part I, 158–168. ISBN 978-3-642-20281-0. [see 19 p.]

MA16 – Programmable VLSI Array Processor for Neural Networks and Matrix-Based Signal Processing [interactive]. 1993. Siemens AG [October]. Prieiga per internetą: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.52.8192&rep=rep1&type=pdf>. [see 31 p.]

Manikandan, J.; Venkataramani, B. 2011. Design of a real time automatic speech recognition system using modified one against all svm classifier, *Microprocessors and Microsystems* 35(6): 568–578. [see 39 p.]

Martinčić-Ipšić, S.; Pobar, M.; Ipšić, I. 2011. Croatian large vocabulary automatic speech recognition, *AUTOMATIKA* 52(2): 147–157. [see 34, 37 p.]

Maskeliūnas, R.; Esposito, A. 2012. Multilingual Italian-Lithuanian Small Vocabulary Speech Recognition via Selection of Phonetic Transcriptions, *Electronics and Electrical Engineering* 121: 85–88. [see 34, 36, 110 p.]

MathWorks 2013. *HDL Coder* [interactive] [1 December 2013]. Prieiga per internetą: <http://www.mathworks.se/products/hdl-coder/>. [see 14 p.]

Mauduit, N.; Duranton, M.; Gobert, J.; Sirat, J.-A. 1992. Lneuro 1.0: a piece of hardware lego for building neural network systems, *Neural Networks, IEEE Transactions on* 3(3): 414–422. ISSN 1045-9227. [see 31 p.]

MD1220 Neuro Bit Slice [interactive]. 1990. Micro Devices [March]. Prieiga per internetą: <http://www.datasheetarchive.com/MD-1220-datasheet.html>. [see 31 p.]

Means, R.; Lisenbee, L. 1991. Extensible linear floating point simd neurocomputer array processor, in *Neural Networks, 1991., IJCNN-91-Seattle International Joint Conference on*, vol. i, 587–592. [see 31 p.]

Meher, P. K. 2010. An optimized lookup-table for the evaluation of sigmoid function for artificial neural networks, in *VLSI System on Chip Conference (VLSI-SoC), 2010 18th IEEE/IFIP*, IEEE, 91–95. [see 30 p.]

Misra, J.; Saha, I. 2010. Artificial neural networks in hardware: A survey of two decades of progress, *Neurocomput.* 74(1-3): 239–255. ISSN 0925-2312. [see 11, 16, 18, 31, 32, 33, 49, 52 p.]

Moussa, M.; Areibi, S.; Nichols, K. 2006. *On the Arithmetic Precision for Implementing Back-Propagation Networks on FPGA: A Case Study*: FPGA Implementations of Neural Networks. Springer. [see 17 p.]

Muthuramalingam, A.; Himavathi, S.; Srinivasan, E. 2008. Neural network implementation using FPGA: issues and application, *International journal of information technology* 4(2): 86–92. [see 17, 19 p.]

Nambiar, V. P.; Khalil-Hani, M.; Sahnoun, R.; Marsono, M. 2014. Hardware implementation of evolvable block-based neural networks utilizing a cost efficient sigmoid-

like activation function, *Neurocomputing* 140: 228–241. [see 29, 30 p.]

Navakauskas, D. 1999. *Artificial Neural Networks for the Restoration of Noise Distorted Songs Audio Records*: Doctoral Dissertation. Vilnius Gediminas Technical University. Vilnius: Technika. 158 p. [see 16 p.]

Navakauskas, D. 2003. Quick training algorithm for extra reduced size lattice-ladder multilayer perceptrons, *Informatica, Lith. Acad. Sci.* 14(2): 223–236. [see 16, 23 p.]

Disability level assessment [interactive]. 2016. Disability and Working Capacity Assessment Office [22 March 2016]. Prieiga per internetą: <http://www.ndnt.lt>. [see 2 p.]

Nedjah, N.; da Silva, R.; Mourelle, L.; da Silva, M. 2009. Dynamic mac-based architecture of artificial neural networks suitable for hardware implementation on {FPGAs}, *Neurocomputing* 72(10–12): 2171–2179. ISSN 0925-2312. Prieiga per internetą: <http://www.sciencedirect.com/science/article/pii/S0925231209000411>. [see 32 p.]

NLX-420 Datasheet [interactive]. 1992. NeuraLogix [June]. Prieiga per internetą: <http://www.datasheetarchive.com/NLX-420-datasheet.html>. [see 31 p.]

Noory, B.; Groza, V. 2003. A reconfigurable approach to hardware implementation of neural networks, in *Electrical and Computer Engineering, 2003. IEEE CCECE 2003. Canadian Conference on*, vol. 3, 1861–1864. ISSN 0840-7789. [see 33 p.]

Omondi, A. R.; Rajapakse, J. C.; Bajger, M. 2006. *FPGA Neurocomputers*: FPGA Implementations of Neural Networks. Springer. [see 19 p.]

Oniga, S.; Tisan, A.; Mic, D.; Buchman, A.; Vida-Ratiu, A. 2008. Optimizing FPGA implementation of feed-forward neural networks, in *Optimization of Electrical and Electronic Equipment, 2008. OPTIM 2008. 11th International Conference on*, 31–36. [see 28 p.]

Onoo, A.; Hikawa, H.; Miyoshi, S.; Maeda, Y. 2009. On automatic generation of vhdl code for self-organizing map, in *Neural Networks, 2009. IJCNN 2009. International Joint Conference on*, 2366–2373. ISSN 1098-7576. [see 28 p.]

Pan, S.-T.; Lai, C.-C.; Tsai, B.-Y. 2011. The implementation of speech recognition systems on FPGA-based embedded systems with soc architecture, *Int. Journal of Innovative Computing, Information and Control* 7(11): 6161–6175. [see 35, 115 p.]

Pan, S.-T.; Li, X.-Y. 2012. An FPGA-based embedded robust speech recognition system designed by combining empirical mode decomposition and a genetic algorithm, *Instrumentation and Measurement, IEEE Transactions on* 61(9): 2560–2572. [see 39, 42, 110 p.]

Parhi, K. 2013. Hierarchical folding and synthesis of iterative data flow graphs, *Circuits and Systems II: Express Briefs, IEEE Transactions on* 60(9): 597–601. ISSN 1549-7747. [see 22 p.]

Prochazka, V.; Pollak, P.; Zdansky, J.; Nouza, J. 2011. Performance of Czech speech recognition with language models created from public resources, *Radioengineering* 20: 1002–1008. [see 34, 41 p.]

Pyz, G.; Simonyte, V.; Slivinskas, V. 2012. Lithuanian speech synthesis by computer using additive synthesis, *Elektronika ir Elektrotechnika* 18(8): 77–80. [see 34 p.]

Regalia, P. A. 1995. *Adaptive IIR Filtering in Signal Processing and Control*. New York: Marcel Dekker. [see 21, 23, 65 p.]

Ronak, B.; Fahmy, S. A. 2012. Evaluating the efficiency of dsp block synthesis inference from flow graphs, in *Field Programmable Logic and Applications (FPL), 2012 22nd International Conference on*, IEEE, 727–730. [see 15 p.]

Ronak, B.; Fahmy, S. A. 2014. Efficient mapping of mathematical expressions into dsp blocks, in *Field Programmable Logic and Applications (FPL), 2014 24th International Conference on*, IEEE, 1–4. [see 11, 15, 52, 54, 96 p.]

Rosado-Munoz, A.; Gomez-Chova, L.; Gomez-Chova, L.; Francés, J. V. 2008. An ip core and gui for implementing multilayer perceptron with a fuzzy activation function on configurable logic devices, *Journal of Universal Computer Science* 14(10): 1678–1694. [see 28 p.]

Sarkar, G.; Saha, G. 2010. Real time implementation of speaker identification system with frame picking algorithm, *Procedia Computer Science* 2: 173–180. [see 38, 39 p.]

Sart, D.; Mueen, A.; Najjar, W.; Niennattrakul, V. 2010. Accelerating dynamic time warping subsequence search with GPUs and FPGAs, in *Proc. ICDM'10*, 1001–1006. [see 35, 36, 41, 42 p.]

Savich, A.; Moussa, M.; Areibi, S. 2012. A scalable pipelined architecture for real-time computation of mlp-bp neural networks, *Microprocessors and Microsystems* 36(2): 138–150. [see 68 p.]

Savich, A. W.; Moussa, M.; Areibi, S. 2007. The impact of arithmetic representation on implementing mlp-bp on fpgas: A study, *Neural Networks, IEEE Transactions on* 18(1): 240–252. [see 29 p.]

Schmadecke, I.; Blume, H. 2013. Hardware-accelerator design for energy-efficient acoustic feature extraction, in *Consumer Electronics (GCCE), 2013 IEEE 2nd Global Conference on*, IEEE, 135–139. [see 39 p.]

Sensory 2012. *Natural Language Processor* [interactive] [3 May 2012]. Prieiga per internetą: <http://www.sensoryinc.com/products/NLP-5x.html>. [see 35 p.]

Social reports [interactive]. 2015. Republic of Lithuania Ministry of Social Security and Labour [22 March 2016]. Prieiga per internetą: <http://www.socmin.lt>. [see 2 p.]

Sojka, P.; Kopeček, I.; Pala, K. (eds.) 2004. *Large Vocabulary Continuous Speech Recognition for Estonian Using Morphemes and Classes*: Lecture Notes in Computer Science. Berlin, Germany: Springer. [see 34, 41 p.]

Spectrum, I. 2014. *Technology, Engineering, and Science News* [interactive] [20 January 2014]. Prieiga per internetą: <http://spectrum.ieee.org>. [see 1 p.]

Stašionis, L.; Serackis, A. 2011. Selection of an optimal adaptive filter for speech signal noise cancellation using c6455 dsp, *Elektronika ir Elektrotechnika* 115(9): 101–104. [see 35 p.]

Staworko, M.; Rawski, M. 2010. FPGA implementation of feature extraction algorithm for speaker verification, in *Mixed Design of Integrated Circuits and Systems (MIXDES), 2010 Proceedings of the 17th International Conference*, IEEE, 557–561. [see 37, 39 p.]

Tamulevičius, G. 2008a. Vilnius Gedinimas Technical University. Prieiga per internetą: <http://www.mii.lt/files/mii_dis_08_tamulevicius.pdf>. [see 36 p.]

Tamulevičius, G. 2008b. *Isolated word recognition systems implementation*: Doctoral Dissertation. Vilnius Gediminas Technical University. Vilnius: Technika. [see 40, 41 p.]

Tamulevičius, G.; Arminas, V.; Ivanovas, E.; Navakauskas, D. 2010. Hardware accelerated FPGA implementation of Lithuanian isolated word recognition system, *Electronics and Electrical Engineering* 99: 57–62. [see 35, 37, 41, 106, 115 p.]

Thilagam, S.; Karthigaikumar, P. 2015. Implementation of adaptive noise canceller using FPGA for real-time applications, in *Electronics and Communication Systems (ICECS), 2015 2nd International Conference on*, 1711–1714. [see 16 p.]

Tisan, A.; Cirstea, M. 2012. {SOM} neural network design: A new simulink library based approach targeting FPGA implementation, *Mathematics and Computers in Simulation* 91(1): 134–149. ISSN 0378-4754. [see 28 p.]

Tommiska, M. 2003. Efficient digital implementation of the sigmoid function for reprogrammable logic, in *Computers and Digital Techniques, IEE Proceedings-*, vol. 150, IET, 403–411. [see 29, 30, 94 p.]

Ursutiu, D.; Samoila, C.; Dabacan, M. 2013. Cross platform methods in digital electronics engineering education, in *Remote Engineering and Virtual Instrumentation (REV), 2013 10th International Conference on*, 1–4. [see 10 p.]

Vaitkus, V.; Zylius, G.; Maskeliunas, R. 2014. Electrical spare parts demand forecasting, *Elektronika ir Elektrotechnika* 20(10): 7–10. [see 29 p.]

Van Beeck, K.; Heylen, F.; Meel, J.; Goedemé, T. 2010. Comparative study of model-based hardware design tools, *Campus De Nayer, Association KU Leuven, Jan De Nayerlaan* 5: 2860. [see 14 p.]

Vandenbout, D. 2013. *FPGAs!? Now What?*: Learning FPGA Design with the XuLA Board. XESS Corporation. [see 11 p.]

Čeidaitė, G.; Telksnys, L. 2010. Analysis of Factors Influencing Accuracy of Speech Recognition, *Electronics and Electrical Engineering* 105(9): 69–72. [see 36, 41 p.]

Veitch, R.; Aubert, L.-M.; Woods, R.; Fischaber, S. 2010. Acceleration of hmm-based speech recognition system by parallel FPGA gaussian calculation, in *Programmable Logic Conference (SPL), 2010 VI Southern*, IEEE, 197–200. [see 34, 35, 39, 115 p.]

Veitch, R.; Aubert, L. M.; Woods, R.; Fischaber, S. 2011. FPGA implementation of a pipelined Gaussian calculation for HMM-based large vocabulary speech recognition, *International Journal of Reconfigurable Computing* 2011: 1–10. [see 36 p.]

Vu, N.-V.; Whittington, J.; Ye, H.; Devlin, J. 2010. Implementation of the mfcc front-end for low-cost speech recognition systems, in *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, IEEE, 2334–2337. [see 37, 39 p.]

Wang, J.-C.; Wang, J.-F.; Weng, Y.-S. 2002. Chip design of mfcc extraction for speech recognition, *INTEGRATION, the VLSI journal* 32(1): 111–131. [see 38 p.]

Wijoyo, T. S. 2011. Speech Recognition Using Linear Predictive Coding and Artificial Neural Network for Controlling Movement of Mobile Robot, in *International Conference on Information and Electronics Engineering IPCSIT*, vol. 6, 179–183. [see 37, 110 p.]

Wu, F.; Chen, S.; Leung, H. 2006. Data hiding for speech bandwidth extension and its hardware implementation, in *Multimedia and Expo, 2006 IEEE International Conference on*, IEEE, 1277–1280. [see 39 p.]

Xilinx 2007. *StateCAD Manual Reference* [interactive] [1 December 2013]. Prieiga per internetą: <http://www.xilinx.com/itp/xilinx10/help/iseguide/mergedProjects/state/whnjs.htm>. [see 14 p.]

Xilinx 2012a. *7 Series FPGAs Configurable Logic Block User Guide* [interactive] [20 April 2013]. Prieiga per internetą: <http://www.xilinx.com/support/documentation/user_guides/ug474_7Series_CLB.pdf>. [see 12 p.]

Xilinx 2012b. *AccelDSP Synthesis Tool* [interactive] [20 January 2014]. Prieiga per internetą: <http://www.xilinx.com/tools/acceldsp.htm>. [see 14 p.]

Xilinx 2013a. *FPGA Editor* [interactive] [14 March 2014]. Prieiga per internetą: <http://www.xilinx.com/support/documentation/sw_manuals/help/iseguide/mergedProjects/fpga_editor/fpga_editor.htm>. [see 32 p.]

Xilinx 2013b. *System Generator for DSP* [interactive] [1 December 2013]. Prieiga per internetą: <http://www.xilinx.com/tools/sysgen.htm>. [see 14 p.]

Xilinx 2013c. *Zynq-7000 All Programmable Software on Chip* [interactive] [2 November 2013]. Prieiga per internetą: <http://www.xilinx.com/products/silicon-devices/soc/zynq-7000/>. [see 30, 94 p.]

Xilinx 2013d. *Zynq-7000 Technical Reference Manual All Programmable SoC* [interactive] [20 April 2013]. Prieiga per internetą: <http://www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf>. [see 13 p.]

Xilinx 2014a. *7 Series DSP48E1 Slice User Guide* [interactive] [10 November 2014].

Prieiga per internetą: <http://www.xilinx.com/support/documentation/user_guides/ug479_7Series_DSP48E1.pdf>. [see 26, 29, 50 p.]

Xilinx 2014b. *Vivado Design Suite Partial Reconfiguration* [interactive] [30 August 2015]. Prieiga per internetą: <http://www.xilinx.com/support/documentation/sw_manuals/xilinx2014_4/ug909-vivado-partial-reconfiguration.pdf>. [see 18 p.]

Xilinx 2015a. *Fast Fourier Transform v9.0* [interactive] [25 August 2015]. Prieiga per internetą: <http://www.xilinx.com/support/documentation/ip_documentation/xfft/v9_0/pg109-xfft.pdf>. [see 38 p.]

Xilinx 2015b. *Vivado Design Suite High-Level Synthesis* [interactive] [1 April 2015]. Prieiga per internetą: <http://www.xilinx.com/support/documentation/sw_manuals/xilinx2015_1/ug902-vivado-high-level-synthesis.pdf>. [see 14 p.]

Xu, J.; *et al.* 2005. Migrate levinson-durbin based linear predictive coding algorithm into fpgas, in *2005 12th IEEE International Conference on Electronics, Circuits and Systems*, 1–4. [see 37, 41 p.]

Yamamoto, K.; Oba, Y.; Rikuhashi, Z.; Hikawa, H. 2011. Automatic generation of hardware self-organizing map for FPGA implementation, in *Intelligent Signal Processing and Communications Systems (ISPACS), 2011 International Symposium on*, 1–6. [see 28 p.]

YANO 2015. *Research institute, market solution provider* [interactive] [14 March 2015]. Prieiga per internetą: <http://yanoresearch.com>. [see 1 p.]

Yiu, K. F. C.; Li, Z.; Low, S. Y.; Nordholm, S. 2014. FPGA multi-filter system for speech enhancement via multi-criteria optimization, *Applied Soft Computing* 21(1): 533–541. ISSN 1568-4946. [see 15 p.]

Yujin, Y.; Peihua, Z.; Qun, Z. 2010. Research of speaker recognition based on combination of LPCC and MFCC, in *IEEE International Conference on Intelligent Computing and Intelligent Systems (ICIS 2010)*, vol. 3, 765–767. [see 37 p.]

Zamanlooy, B.; Mirhassani, M. 2014. Efficient vlsi implementation of neural networks with hyperbolic tangent activation function, *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* 22(1): 39–48. [see 29, 30 p.]

Zhang, G.; Yin, J.; Liu, Q.; Yang, C. 2011. A real-time speech recognition system based on the implementation of FPGA, in *Cross Strait Quad-Regional Radio Science and Wireless Technology Conference (CSQRWC), 2011*, vol. 2, IEEE, 1375–1378. [see 35, 39, 42, 110, 115 p.]

Zhang, Y.; Adl, K.; Glass, J. 2012. Fast spoken query detection using lower-bound dynamic time warping on graphical processing units, in *Proc. ICASSP'12*, 5173–5176. [see 35, 36, 41, 42 p.]

Zhou, X.; Garcia-Romero, D.; Duraiswami, R.; Espy-Wilson, C.; Shamma, S. 2011.

Linear versus Mel Frequency Cepstral Coefficients for Speaker Recognition, in *Automatic Speech Recognition and Understanding (ASRU), 2011 IEEE Workshop on*, 559–564. [see 37, 110 p.]

Žvironas, A.; Kazanavičius, E. 2006. Implementation of correlation analysis task in the multichannel structure, *Information Technology And Control* 35(3). [see 15 p.]

List of Scientific Publications by the Author on the Topic of the Dissertation

Papers in the Reviewed Scientific Journals

Sledevič, T.; Navakauskas, D. 2016. FPGA Implementation of Range Addressable Activation Function for Lattice-Ladder Neuron, *Elektronika ir elektrotechnika* 22(2): 92–95. ISSN 1392-1215. DOI:10.5755/j01.eie.22.2.14598. IF = 0.561 (2014).

Tamulevičius, G.; Serackis, A.; Sledevič, T.; Navakauskas, D. 2015. Vocabulary Distance Matrix Analysis – based Reference Template Update Technique, *Proceedings of the Romanian academy, Series A* 16(1): 103–109. ISSN 1454-9069. IF = 1.115 (2014). Available from Internet: <http://www.acad.ro/sectii2002/proceedings/doc2015-1/14-Tamulevicius.pdf>.

Serackis, A.; Tamulevičius, G.; Sledevič, T.; Stasionis, L.; Navakauskas, D. 2014. Self-Organizing Feature Map Preprocessed Vocabulary Renewal Algorithm for the Isolated Word Recognition System, *Elektronika ir elektrotechnika* 20(6): 114–117. ISSN 1392-1215. DOI:10.5755/j01.eee.20.6.7280. IF = 0.561.

Tamulevičius, G.; Serackis, A.; Sledevič, T.; Navakauskas, D. 2014. Bidirectional Dynamic Time Warping Algorithm for the Recognition of Isolated Words Impacted by Transient Noise Pulses, *International Journal of Computer, Electrical, Automation, Control and Information Engineering* 8(4): 710–714. ISSN 2010-376X. Available from Internet: <http://scholar.waset.org/1999.4/9998049>.

Sledevič, T.; Navakauskas, D.; Tamulevičius, G.; 2013. Upgrading FPGA Implementation of Isolated Word Recognition System for a Real-Time Operation, *Elektronika ir elektrotechnika* 19(10): 123–128. ISSN 1392-1215. DOI: 10.5755/j01.eee.19.10.5907. IF = 0.445.

Sledevič, T.; Serackis, A.; Tamulevičius, G.; Navakauskas, D. 2013. Evaluation of Features Extraction Algorithms for a Real-Time Isolated Word Recognition System, *International Journal of Electrical, Computer, Energetic, Electronic and Communication Engineering* 7(12): 302–307. ISSN 2010-376X. Available from Internet: <http://scholar.waset.org/1999.5/9996651>.

Sledevič, T.; Stasionis, L. 2013. FPGA-Based Implementation of Lithuanian Isolated Word Recognition Algorithm, *Mokslas – Lietuvos ateitis* 5(2): 101–104. ISSN 2029-2252. DOI: 10.3846/mla.2013.18.

Stasionis, L.; Sledevič, T. 2013. Energy Detector Implementaton in FPGA for Estimation of Word Boundaries, *Mokslas – Lietuvos ateitis* 5(2): 105–108. ISSN 2029-2252. DOI: 10.3846/mla.2013.19.

Other Papers

Sledevič, T; Navakauskas, D. 2015. Towards Optimal FPGA Implementation of Lattice-Ladder Neuron and Its Training Circuit, in *Proceedings of IEEE 3rd Workshop on advances in information, electronic and electrical engineering*, 1–4. ISBN: 978-1-5090-1201-5. DOI: 10.1109/AIEEE.2015.7367311.

Sledevič, T; Navakauskas, D. 2014. The Lattice-Ladder Neuron and its Training Circuit Implementation in FPGA, in *Proceedings of IEEE 2nd Workshop on advances in information, electronic and electrical engineering*, 1–4. ISBN 978-1-4799-7122-0. DOI: 10.1109/AIEEE.2014.7020327.

Serackis, A.; Sledevič, T.; Tamulevičius, G.; Navakauskas, D. 2013. Word Recognition Acceleration by Double Random Seed Matching in Perceptual Cepstrum Error Space, in *Proceedings of IEEE European Modelling Symposium*, 274–279. ISBN 978-1-4799-2577-3. DOI: 10.1109/EMS.2013.50.

Sledevič, T; Navakauskas, D. 2013. FPGA based Fast Lithuanian Isolated Word Recognition System, in *Proceedings of IEEE EUROCON Conference*, 1630–1636. ISBN 978-1-4673-2230-0. DOI: 10.1109/EUROCON.2013.6625195.

Santrauka lietuvių kalba

Įvadas

Problemos formulavimas

Spartus mobiliųjų, multimedijos, 3D ir virtualizavimo technologijų vystymasis yra šiuolaikinės elektronikos pasaulinės raidos pamatas. Šiais 2016 m. prognozuojamas išmaniųjų telefonų pardavimų augimas 16,8 % ir tikimasi, kad planšetinių kompiuterių pardavimai pasieks 458 mln. vienetų apimtis, o tai, pagal YANO tyrimų centro duomenis, lems išmaniųjų technologijų pasaulinį įsivyravimą.

Plintant išmaniosioms technologijoms bei mobiliųjų įrenginių ir jų paslaugų apimtims vis aktualesnė tampa elektronikos įrenginių efektyvaus įgyvendinimo problema. Sistemos įgyvendinimo efektyvumas yra prieštaringų reikalavimų derinys. Augančios skaičiavimų apimtys, spartėjantys duomenų mainai, o tuo pačiu didėjantis energijos suvartojimas verčia ne tik optimizuoti duomenų apdorojimo algoritmus, tačiau taip pat svarbu tampa juos greitai įgyvendinti specializuotoje aparatūroje. Lauku programuojamų loginių matricų (LPLM) technologijos taikymas leidžia sumažinti elektroninės sistemos kūrimo ir derinimo trukmę, o matricos struktūros dėka – galimas smarkus skaičiavimų pagreitinimas taikant lygiagretinimo ir konvejerizavimo principus. Būtent todėl, disertacijoje sprendžiama realiuoju laiku veikiančių elektroninių intelektualiųjų sistemų efektyvaus įgyvendinimo lauku programuojama logine matrica problema. Darbe iškeliama ir įrodoma pagrindinė hipotezė – pasirinktos elektroninės intelektualiosios sistemos klasės būdingųjų struktūrinių požymių pagrindu galima suformuluoti specializuotą sistemos grandynų sintezės kriterijų ir šiai sistemai įgyvendinti skirtą metodą, kurių taikymas optimizuotų visos sistemos įgyvendinimą lauku programuojama logine matrica.

Darbo aktualumas

Per paskutinius porą metų Google, Nuance paskelbė apie balsu valdomo televizoriaus kūrimą, pasiūlyti balsu valdomi *GPS* navigatoriai, o balso technologijos, pagal *Consumer Electronics Show 2016*, vėl vertinamos kaip vienos perspektyviausių sąsajos formų. Atsiradusi galimybė pagaminti valdymo balsu įrenginius leidžia sukurti naujas paslaugas ne tik pramogai, bet ir socialinėms užduotims spręsti. Viena iš potencialių kalbos technologijų taikymo sričių yra žmonės su fizine negalia. Neįgalumo ir darbingumo nustatymo tarnybos duomenimis 2015 m. Lietuvoje nuolatinės slaugos poreikis pripažintas 17731 asmeniui, o 16694 asmenims nustatytas nuolatinės pagalbos poreikis. Galimybė veiksmus inicijuoti balsu padidintų žmogaus su fizine negalia savarankiškumą, pagerintų jo gyvenimo kokybę bei leistų sumažinti slaugos personalo poreikį.

Didėjant apdorojamų duomenų skaičiui ir tam, kad šnekos atpažinimas vyktų realiuoju laiku, yra būtina didinti šnekos atpažintuve įgyvendintų algoritmų greitaveiką. Šiai užduočiai išspręsti lauku programuojama logine matrica įgyvendinami skaičiavimus spartinantys intelektinės nuosavybės (IN) moduliai. Vis populiarėjantis lauku programuojamų loginių matricų technologijos taikymas glūdi algoritmų išlygiagretinimo galimybėje, todėl įrenginiai, grįsti lauku programuojamomis loginėmis matricomis, dirba efektyviau (vertinant pagal vartojamą energiją ir spartą) lyginant su šiuolaikiniais procesoriais, net jei LPLM taktinis dažnis yra 10–300 MHz diapazone.

Tyrimų objektas

Tyrimo objektas yra specializuoti lauku programuojamos loginės matricos intelektinės nuosavybės (LPLM IN) moduliai veikiantys realiuoju laiku. Disertacijoje tiriami šie, su tiriamuoju objektu susiję, dalykai: įgyvendinimo kriterijai ir metodas.

Darbo tikslas

Darbo tikslas – optimizuoti pynučių-kopėtėlių daugiasluoksnio perceptrono įgyvendinimo lauku programuojama logine matrica procesą ir jo rezultatus pritaikyti neįgaliesiems skirtame lietuviškos šnekos atpažintuve.

Darbo uždaviniai

Siekiant išspręsti nurodytą problemą ir pasiekti tikslą suformuluojami šie pagrindiniai disertacijoje sprendžiami uždaviniai:

1. Pagrįsti įgyvendinimui ir tyrimams atrinktų pynučių-kopėtėlių daugiasluoksnio perceptrono (PKDP) klasės ir jos elektroninės intelektualiosios testavimo aplinkos – neįgaliesiems skirto priklausomo nuo kalbėtojo lietuviškos šnekos atpažintuvo, pasirinkimą.
2. Sukurti specializuotais grandynų sintezės kriterijais paremtą pynučių-kopėtėlių daugiasluoksnio perceptrono klasės įgyvendinimo lauku programuojama logine matrica metodą.
3. Sukurti optimizuotus lauku programuojamos loginės matricos intelektinės nuosavybės modulius ir taikant lietuvių šnekos atpažintuve eksperimentiškai patvirtinti jų efektyvumą.

Tyrimų metodika

Darbe taikomos skaitmeninio signalo apdorojimo, spektrinės ir kepstrinės analizės, priklausomo nuo kalbėtojo žodžių atpažinimo, dirbtinių neuronų tinklų, optimizavimo, stasistinės analizės bei grafų teorijos. Pritaikyti ir įgyvendinti tiesinės prognozės, melų dažnių skalės, dinaminio laiko skalės kraipymo, pynučių-kopėtėlių daugiasluoksnio perceptrono skaičiavimo ir mokymo, grafų padengimo, pografių paieškos ir instrukcijų planavimo metodai.

Eksperimentams taikomi lietuvių šnekos pavienių žodžių įrašų garsynai. Kompiuterinis modeliavimas atliekamas Matlab ir ModelSim programiniais paketais. Pavienių žodžių atpažinimo sistemos tikslumo ir greitaveikos testams ir lyginimams su ankstesniąja versija taikomas specializuotas bandymų modulis ML402 su Virtex-4 šeimos LPLM lustu. Galutinis šnekos atpažintuvas įgyvendinamas ZynQ-7000 luste su integruotu Artix-7 šeimos LPLM ir dviejų branduolių *ARM* Cortex A9 procesoriumi. IN moduliai įgyvendinti taikant sukurtą originalų kompiliatorių ir Xilinx ISE Design Suite 14.7 bei Vivado HLS 2015.4 programinę įrangą.

Darbo mokslinis naujumas

1. Sukurtas naujas pynučių-kopėtėlių daugiasluoksnių perceptronų įgyvendinimo metodas, įvertinantis lauku programuojamos loginės matricos specifiką ir generuojantis efektyvesnį, lyginant su bendrosios paskirties komerciniu įrankiu, intelektinės nuosavybės modulį.

2. Pasiūlytas pynučių-kopėtėlių daugiasluoksnių perceptronų grandynų sintezės specializuotų kriterijų Pareto frontų skaičiavimo būdas leidžiantis optimaliai parinkti lauku programuojamos loginės matricos tipą pagal užsiduotus pynučių-kopėtėlių daugiasluoksnių perceptronų struktūros, diskretizavimo dažnio ir lauku programuojamos loginės matricos resursų reikalavimus.

3. Sukurti nauji algoritmai: spartesnis žodžių sutapdinimo ir dvigubo atsitiktinio iniciavimo rezultatų sutapdinimo, spartinantys žodžių atpažinimo procesą lauku programuojamoje loginėje matricoje įgyvendintame lietuvių šnekos atpažintuvo prototipe.

4. Pasiūlyta triukšmo šalinimui taikyti pynučių-kopėtėlių neuroną, kuris pagerina žodžių atpažinimo tikslumą.

Darbo rezultatų praktinė reikšmė

Pasiūlyto metodo pagrindu sukurtas kompiliatorius efektyviam PKDP IN modulių įgyvendinimui LPLM. Siūloma specializuotų PKDP grandynų sintezės kriterijus vertinti Pareto frontais, kurie leidžia optimaliai parinkti LPLM lustą pagal PKDP struktūros, diskretizavimo dažnio ir LPLM resursų reikalavimus.

Šnekos atpažintuvo prototipas įgyvendintas taikant naujos kartos ZynQ-7000 lustą geba realiu laiku atpažinti lietuviška šneka duodamus nurodymus, formuoti specifikuotus kontrolinius signalus ir gali būti naudojamas žmonių su fizine negalia specializuotų funkcinių įrenginių valdymui balsu. Kiekvienam valdomam įrenginiui galima formuoti individualų komandų sąrašą, taip padidinant valdymo balsu efektyvumą. Maketą, be didesnių aparatinių pakeitimų, galima pritaikyti ir kitokių signalų atpažinimui.

Šiuo požiūriu maketas neturi analogų Lietuvoje ir yra vienas iš nedaugelio pasaulyje. Žodžių atpažinimo sistema buvo tiriama ir taikoma vykdant:

- Lietuvos mokslo tarybos remiamą mokslininkų grupių technologinės plėtros projektą „Neįgaliesiems skirto valdymo lietuviška šneka įrenginio maketo kūrimas ir patikra“ (Nr. MIP-092/2012, 2012–2014);
- kvalifikacinį mokslo darbą „Skaitmeninio signalų apdorojimo realaus laiko sistemoms tyrimas“ (Nr. TMT 335, 2013–2017).

Ginamieji teiginiai

1. Taikant siūlomą pynučių-kopėtėlių daugiasluoksnių perceptronų įgyvendinimo lauku programuojama logine matrica metodą sukurti intelektinės nuosavybės moduliai yra ne mažiau kaip 3 kartus efektyvesni už modulius generuojamus komerciniu įrankiu Vivado HLS.
2. Specializuotų pynučių-kopėtėlių daugiasluoksnių perceptronų grandynų sintezės kriterijų vertinimas Pareto frontais leidžia optimaliai parinkti lauku programuojamos loginės matricos lustą pagal pynučių-kopėtėlių daugiasluoksnių perceptronų struktūros, diskretizavimo dažnio ir lauku programuojamos loginės matricos resursų reikalavimus.
3. Optimizuoti intelektinės nuosavybės moduliai taikomi lietuvių šnekos atpažintuve yra tinkami pavieniams žodžiams atpažinti realiuoju laiku, pasiekiant 7800 žodžių/s sutapdinimo greitį.
4. Pynučių-kopėtėlių neurono intelektinės nuosavybės modulio taikymas pirminiame kalbos apdorojime 4 % padidina žodžių atpažinimo tikslumą esant 15 dB signalas triukšmas santykiui.

Darbo rezultatų aprobavimas

Darbo rezultatai paskelbti 12-oje mokslinių straipsnių:

- keturios publikacijos atspausdintos žurnaluose, įtrauktuose į Thomson Reuters Web of Science sąrašą ir turinčiuose citavimo indeksą (Sledevič, Navakauskas 2016, Tamulevičius *et al.* 2015, Serackis *et al.* 2014, Sledevič *et al.* 2013);
- dvi publikacijos atspausdintos recenzuojamame mokslo žurnale, cituojamame Index Copernicus duomenų bazėje (Sledevič, Stašionis 2013, Stašionis, Sledevič 2013);
- dvi publikacijos atspausdintos recenzuojamuose mokslo žurnaluose, įtrauktuose į SCImago duomenų bazę (Tamulevičius *et al.* 2014, Sledevič *et al.* 2013);
- keturios publikacijos atspausdintos kituose mokslo leidiniuose: dvi – tarptautinių konferencijų straipsnių rinkiniuose, cituojamuose ISI Proceedings duomenų bazėje (Serackis *et al.* 2013, Sledevič, Navakauskas 2013) ir dvi – tarptautinių konferencijų straipsnių rinkiniuose, cituojamuose IEEEXPLORE (INSPEC) duomenų bazėje (Sledevič, Navakauskas 2015, Sledevič, Navakauskas 2014).

Pagrindiniai disertacijos rezultatai paskelbti 17-oje mokslinių konferencijų:

- 2012 tarptautinėje konferencijoje „13th Biennial Baltic Electronics Conference“ Estijoje, Taline;
- 2013 tarptautinėje konferencijoje „International Conference on Communication, Control and Computer Engineering“ Turkijoje, Stambule;
- 2013 tarptautinėje konferencijoje „7th European Modelling Symposium“ Didžiojoje Britanijoje, Mančesteryje;
- 2013 tarptautinėje konferencijoje „International Conference on Computer as a Tool (EUROCON)“ Kroatijoje, Zagrebe;
- 2014 tarptautinėje konferencijoje „16th International Conference on Image, Signal and Vision Computing“ Prancūzijoje, Paryžiuje;
- 2014 respublikinėje konferencijoje „Multidisciplinary Research in Natural and Technology Sciences“ Lietuvoje, Vilniuje;
- 2014 tarptautinėje konferencijoje „3rd Workshop on Bio-Inspired Signal and Image Processing“ Lietuvoje, Vilniuje;
- 2014 tarptautinėje konferencijoje „6th International Workshop on Data Analysis Methods for Software Systems“ Lietuvoje, Druskininkuose;
- 2014 tarptautinėje konferencijoje „2nd Workshop on Advances in Information, Electronic and Electrical Engineering“ Lietuvoje, Vilniuje;
- 2013–2015 tarptautinėse konferencijose „Electronics“ Lietuvoje, Palangoje;
- 2015 tarptautinėje konferencijoje „3rd Workshop on Advances in Information, Electronic and Electrical Engineering“ Latvijoje, Rigoje;
- 2013–2016 respublikinėse konferencijose „Science – Future of Lithuania“ Lietuvoje, Vilniuje.

Lietuvių šnekos pavienių žodžių atpažintuvo pristatymas konferencijoje „Multidisciplinary Research in Natural and Technology Sciences“ buvo pripažintas vienu iš geriausių ir įvertintas Lietuvos mokslų akademijos diplomu bei Infobalt skatinamąja stipendija.

Disertacijos struktūra

Disertaciją sudaro įvadas, keturi skyriai, bendrosios išvados, 3 priedai, literatūros sąrašas su atskirai pateiktomis autoriaus publikacijomis. Pateikiami sąrašai simbolių, santrumpų, raktažodžių ir terminų sarašas. Darbo apimtis yra 156 puslapiai, kuriuose yra pateikta: 87 formulės, 73 paveikslai, 12 lentelių, 6 algoritmai ir 1 pavyzdys. Disertacijoje remtasi 152 kitų autorių literatūros šaltiniais.

1. Elektroninių sistemų įgyvendinimo lauku programuojamomis loginėmis matricomis apžvalga

Per pastaruosius du dešimtmečius LPLM taikymas vis populiarėja inžinierių ir mokslininkų bendruomenėje dėl galimybės spartinti tradicinius centriniu procesoriniu įrenginiu (angl. *Central Processing Unit – CPU*) grįstus algoritmus. Grandynų įgyvendinimui būdingi bendri visoms LPLM etapai: grandyno aprašymas taikant aukšto lygio sintezės (angl. *High Level Synthesis – HLS*) kodą, kodo pakeitimas į registrų perdavimo lygmenį (angl. *Register Transfer Level – RTL*), *RTL* grandyno sintezė, grandyno atvaizdavimas LPLM palaikomais struktūriniais blokais, blokų tarpusavyje sujungimas, sudaryto grandyno laikinė analizė, bylos generavimas įkėlimui į LPLM. Algoritmų greitam įgyvendinimui taikomi *HLS* įrankiai: Simulink, LabView, Vivado HLS. Pastarieji naudoja daugiau LPLM resursų grandyno sudarymui bei mažina jos greitaveiką lyginant su įgyvendinimu *RTL* lygmenyje, kuriame įmanoma valdyti kiekvieną specifinį LPLM bloką bei nustatynėti sujungimus tarp jų, taikant aparatūrą aprašančias kalbas (angl. *Hardware Description Language – HDL*), tokias kaip VHDL ar Verilog. Išskiriami trys pagrindiniai konfigūruojami LPLM blokai: skaitmeninis signalų apdorojimas (angl. *Digital Signal Processing – DSP*), blokinė adresuojama atmintis (angl. *Block RAM – BRAM*), peržvalgos lentelė (angl. *Look-Up Table – LUT*). Duomenų mainams tarp išvardintų blokų yra taikomi programuojami sujungimai.

Dėl periodinės struktūros ir paskirstytų aritmetinių resursų LPLM struktūra tinka dirbtinių neuronų tinklams (DNT) įgyvendinti. Konfigūruojami LPLM blokai leidžia išlygiagretinti DNT struktūrą pagal sluoksnius ar neuronus ir jiems taikyti grandininį signalo apdorojimą. DNT efektyviam įgyvendinimui LPLM yra sprendžiamas daugiakriterinis optimizavimo uždavinys, kai konkrečiai LPLM struktūrai turi būti pasirinkti: LPLM aprašymo lygmuo, perkonfigūravimo galimybė, testavimo būdas, tikslumas, skaičiavimų greitaveika ir LPLM tipas su tam tikru resursų skaičiumi. Todėl būtina išgryninti LPLM efektyvaus įgyvendinimo kriterijus. Pagrindinės DNT sudaromosios dalys yra sinapsės, kurios yra pritaikomos tinklo mokymo metu. Sinapses pakeitus begalinės impulsinės reakcijos filtrais gaunami dinaminiai DNT, kuriems priskiriami pynučių-kopėtėlių daugiasluoksnių perceptronų (PKDP) tinklai. Pynučių-kopėtėlių filtrai yra stabilūs iš prigimties ir PKDP yra tinkami kalbos signalo apdorojimui su išvestomis mokymo struktūromis, tačiau iki šiol nėra tirtas jų įgyvendinimas LPLM, greitaveika ir reikalingi LPLM resursai.

Šioje disertacijoje nagrinėjamos keturios perspektyviausios rekursinės pynutės taikytinos pynučių gradientų skaičiavime ir PKDP mokyme. Yra pareikta reikalingų aritmetinių operatorių skaičiaus suvestinė PKDP įgyvendinimui LPLM. PKDP struktūrai sudėtingėjant didėja skaičiavimų apimtys, todėl lygiagrečiam tinklo įgyvendinimui nepakanka LPLM resursų. Todėl priklausomai nuo tinklo struktūros yra būtinas LPLM resursų dalijimasis tarp sluoksnių, neuronų ar sinapsių. Šiuos apribojimus būtina įvertinti kuriant PKDP efektyvaus įgyvendinimo kriterijus ir metodą. LPLM įgyvendintas grandynas per vieną taktą įvykdo aritmetines operacijas su fiksuoto kablelio skaičiais. Todėl skaičių pločiui dažniausiai skiriama nuo 12 b iki 32 b, o netiesinei aktyvavimo funkcijai nuo 6 b iki 32 b, priklausomai nuo norimo tikslumo. Didesnis

tikslumas veda prie spartesnio ribotų resursų sunaudojimo ir mažesnio tinklo sukūrimo konkrečiame LPLM. Todėl turi būti ieškomas kompromisas tarp greitaveikos, tinklo dydžio ir LPLM tipo. Egzistuojantys specializuoti DNT integriniai grandynai turi ribotą tikslumą, įėjimų/išėjimų bei neuronų skaičių, todėl vis dažniau DNT įgyvendinimui pagal pageidaujamus efektyvumo kriterijus yra taikomi LPLM lustai, kurie turi nuo kelių šimtų iki kelių tūkstančių *DSP* blokų su konfigūruojamomis instrukcijomis, paskirstytus *BRAM* ir *LUT* resursus.

Sudėtingų konfigūracijų DNT įgyvendinimui LPLM yra siūlomi neuronų modeliai, kurių įgyvendinimų kopijos yra nuo kelių iki kelių šimtų kartų atkartojamos LPLM priklausomai nuo siektino išlygiagretumo lygio. Dėl ribotų LPLM išteklių, mokslininkų siūlomi metodai DNT įgyvendinimui taip pat naudoja aritmetinių, loginių bei atminties resursų dalijimasį tarp sluoksnių, neuronų ar sinapsių. Tačiau pagrindinis skirtumas tarp šioje disertacijoje siūlomo ir žinomų metodų DNT įgyvendinimui LPLM yra tas, kad egzistuojantys metodai nenaudoja dinaminio *DSP* persikonfigūravimo savybės ir skiria visą *DSP* bloką sinapsei ar neuronui. Net jeigu persikonfigūravimo savybė yra pritaikoma, bet nėra bendrinama visam kuriamam tinklui, tai riboja sintezuojamos grandinės dydį ir yra suvaržomos galimybės įgyvendinti siektinos sudėtingos konfigūracijos PKDP.

Disertacijoje tiriami PKDP – tai dirbtinių neuronų tinklai, kuriais galima modeliuoti dinaminį procesą atpažįstant nestacionarius signalus laike. PKDP tyrimams šnekos atpažintuvas yra tinkama testavimo aplinka dėl galimybės optimizuotus PKDP IN modulius taikyti: kalbos filtravime, požymių išskyrime ar sprendimo priėmime. Požymiams kalbos signale išskirti taikomi tiesinės prognozės koeficientai (angl. *Linear Predictive Coefficients – LPC*), tiesinės prognozės kepstriniai koeficientai (angl. *Linear Predictive Cepstral Coefficients – LPCC*), tiesinio spektro kepstriniai koeficientai (angl. *Linear Frequency Cepstral Coefficients – LFCC*) ir melų dažnių skalės kepstriniai koeficientai (angl. *Mel Frequency Cepstral Coefficients – MFCC*). Pavieniams žodžiams klasifikuoti naudojamas dinaminio laiko skalės kraipymo (DLSK) metodas dėl greito skaičiavimo taikant grandininio apdorojimo principą ir fiksuoto kablelio aritmetiką LPLM luste. Sukurtų šnekos atpažintuvo ir PKDP IN modulių efektyvumas turi būti eksperimentiškai patvirtintas.

2. Efektyvaus įgyvendinimo kriterijai ir metodas

Apžvalga atskleidė, kad dideliam PKDP įgyvendinimui nepakanka LPLM resursų, todėl siūloma kurti konfigūruojamus neurono apdorojimo elementus (angl. *Neuron Processing Element – NPE*) su mažiausiu galimu skaičiavimo vėlinimu ir didžiausiu taktiniu dažniu bei dalintis *NPE* resursais, kai tinklo, sluoksnio, neurono ar sinapsės neįmanoma dubliuoti siekiant lygiagretaus veikimo. PKDP optimizavimas yra daugiakriterinis uždavinys. Įgyvendinto PKDP kokybę apsprendžia tinklo efektyvumo kriterijai: signalo apdorojimo greitaveika $\mathfrak{q}_{\text{Thr}}$, tinklo sudėtingumas $\mathfrak{q}_{\text{Com}}$. LPLM efektyvumui vertinti priskiriami šie kriterijai: resursai $\mathfrak{q}_{\text{Res}}$, tikslumas $\mathfrak{q}_{\text{Acc}}$, naudojama energija $\mathfrak{q}_{\text{Pow}}$ ir kaina $\mathfrak{q}_{\text{Price}}$. PKDP kokybė išreiškiama per aibę minėtų kriterijų. Kadangi PKDP pagal išvardintus kriterijus negali būti optimizuojamas vienu metu dėl

prieštaravimų tarp jų, todėl siūloma atsižvelgti į svarbius kriterijus iš vartotojų keičiamų parametrų perspektyvos. Laikoma, kad svarbu yra modifikuoti PKDP struktūrą, pasirinkti norimą LPLM lustą, užsiduoti tinklo greitaveiką ir išreikšti kriterijus ($\mathfrak{q}_{\text{Thr}}$, $\mathfrak{q}_{\text{Res}}$) per Pareto kokybę:

$$\mathcal{Q}_{\text{Pareto}} \triangleq \max(\mathfrak{q}_{\text{Thr}}, \mathfrak{q}_{\text{Res}}), \tag{S1}$$

$$\mathfrak{q}_{\text{Thr}} = f^{\top}/\tau_{\text{NPE}}, \tag{S2}$$

$$\mathfrak{q}_{\text{Res}} = \frac{1}{R_{\text{LUT}} + 196 \times R_{\text{DSP}} + 576 \times R_{\text{BRAM}}}, \tag{S3}$$

čia $f^{\top}$ – didžiausias galimas susintezuoto PKDP grandyno taktinis dažnis, τ_{NPE} – signalo apdorojimo trukmė matuojama taktiniais impulsais, R_{LUT} – peržvalgos lentelių skaičius, R_{DSP} – *DSP* skaičius, R_{BRAM} – *BRAM* skaičius. Koeficientai 196 ir 576 taikomi R_{DSP} ir R_{BRAM} išreikšti ekvivalenčiais *LUT* resursais.

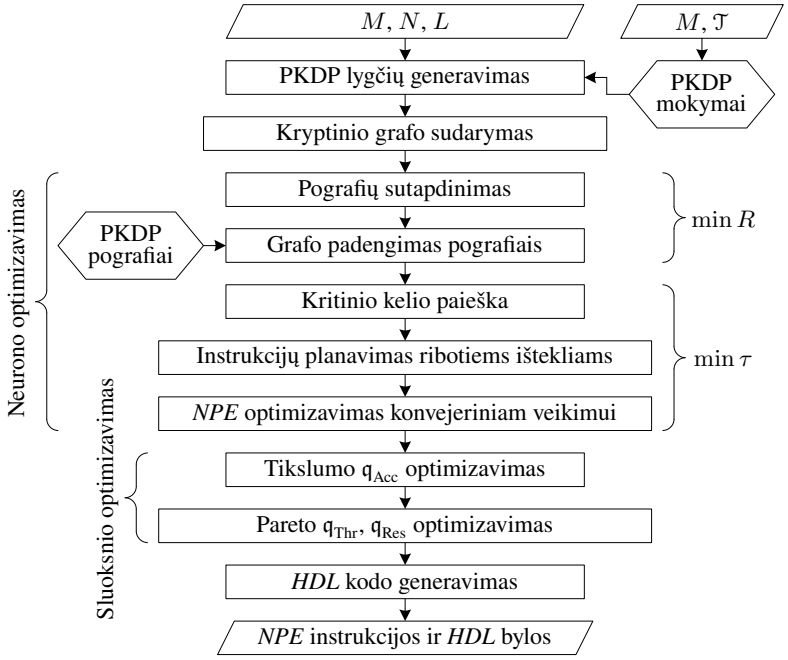
Pareto frontas parodys konkrečiam sudėtingumo tinklui būtiną LPLM resursų skaičių (ir LPLM lustą) prie užsiduodamos greitaveikos. Optimizavimas pagal mokymo tipo ir tikslumo $\mathfrak{q}_{\text{Acc}}$ kriterijus vykdomas prieš optimizavimą pagal $\mathcal{Q}_{\text{Pareto}}$ (S1 pav.). Optimalus mokymo tipas priklauso nuo sinapsės eilės, o $\mathfrak{q}_{\text{Acc}}$ yra nustatomas lyginant fiksuoto ir slankaus kablelio PKDP įgyvendinimus. $\mathfrak{q}_{\text{Power}}$ ir $\mathfrak{q}_{\text{Price}}$ laikomi antros svarbos kriterijais, kurie glaudžiai susieti su $\mathfrak{q}_{\text{Res}}$, nes didėjant LPLM resursams didėja kaina ir energijos vartojimas.

Pagrindinis skirtumas tarp siūlomų efektyvaus PKDP įgyvendinimo kriterijų ir analogiškų kriterijų DNT įgyvendinimo efektyvumui vertinti yra tas, kad siūlomas kokybės kriterijus kartu įvertina prieštaringus greitaveikos ir naudojamų išteklių reikalavimus. Kai tuo tarpu įgyvendinimas pagal daugumą analogiškų kriterijų taiko vieno kriterijaus optimizavimą. Siūlomas metodas PKDP įgyvendinimui LPLM yra išskirtinis tuo, kad jis PKDP tinklą apibūdinančiame grafe ieško ir optimizuoja pynutėms-kopėtėlėms bei jų mokymams būdingas struktūras.

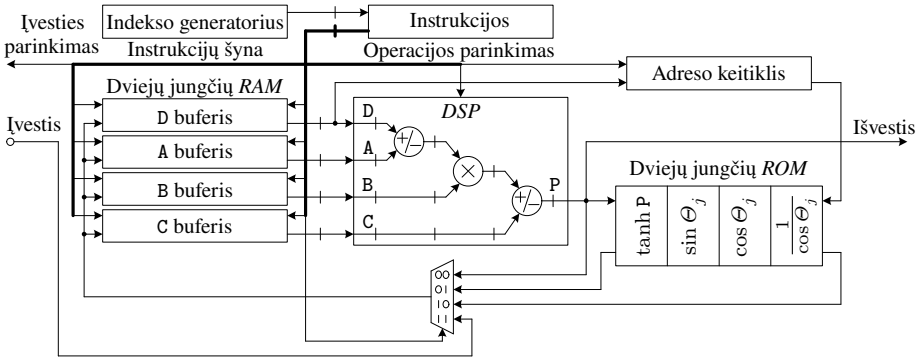
Pasiūlytos dvi efektyvios pagal Pareto frontą optimizavimo strategijos taikytinos pilnam PKDP įgyvendinimui LPLM naudojant ribotą *NPE* skaičių. Pirmuoju atveju užsiduodami resursai ir PKDP konfigūracija. Pradedama nuo didžiausio resursų sunaudojimo siekiant didžiausios greitaveikos (mažiausio vėlinimo tarp duomenų padavimo į įėjimo sluoksnį ir jų gavimo paskutiniojo sluoksnio išėjime). Nuosekliai redukuojant sluoksnių, neuronų ir vėliausiai sinapsių lygiagretumą yra mažinamas būtinų resursų skaičius. Tai vykdoma tol, kol PKDP netelpa į LPLM lustą kaskart perskaičiuojant PKDP duomenų apdorojimo vėlinimą. Optimizavimas kartojamas vis kitai PKDP konfigūracijai ir naujai užsiduotiems resursams ar LPLM tipui. Antruoju atveju užsiduodamas maksimalus signalo apdorojimo vėlinimas bei PKDP konfigūracija. Optimizatorius nuosekliai prideda naujus *NPE* įvedant sinapsių, neuronų ir vėliausiai sluoksnių lygiagretumą, kol vėlinimo sąlyga nebus tenkinama. Optimizavimas kartojamas vis kitai PKDP konfigūracijai ir naujai užsiduotam vėlinimo apribojimui.

PKDP įgyvendinimui taikomas grafų teorija grįstas metodas, kurio pagrindiniai etapai pateikti S1 paveiksle. PKDP generuoti taikomi įėjimo parametrai: mokymo tipas $\mathcal{T}$, sinapsių eilė M, neuronų skaičius N ir sluoksnių skaičius L. PKDP apibūdinamas per aibę lygčių, kur kiekviena lygtis turi vieną operatorių ir du operandus. Iš lygčių sudaromas kryptinis grafas, kuriame kiekvieną viršūnę atitinkantis operatorius

turi dvi įeinančias briaunas žyminčias operandus. Grafe ieškoma visų įmanomų pografių išsidėstymų (su 1–3 viršūnėmis), kurias palaiko perkonfigūruojamas *DSP* blokas. Grafas nuosekliai padengiamas pografiais, pografių dydžio mažėjimo tvarka iki kol nebus pilnai padengtas. Tikslas yra kuo mažesniu pografių skaičiumi ir kuo didesniais pografiai padengti visą grafą minimizuojant vėlinimą, nes *DSP* pografio apdorojimo greitis vienodas visiems pografių dydžiams. Yra panaudota *DSP* savybė apdoroti sekantį pografį per vieną takto ciklą, kai esamo pografio išėjimas yra sekančio pografio įėjimu. Norint nustatyti kokiu laiko momentu koks pografis turi būti pradėtas vykdyti, padengtam grafui taikomas kritinio kelio metodas, kuris taip pat parodo mažiausią galimą vėlinimą, su kuriuo grafas gali būti apdorotas. Pografių pradžios ir pabaigos laikai išdėstomi Ganto diagramoje. Jei keli pografiai turi vienodą pradžios laiką, tai vieno iš jų vykdymo pradžia nukeliama į laisvą laiko žymę, kad *DSP* galėtų pradėti vykdyti vieną pografį per ciklą taikant grandininį apdorojimą. Pasiūlyta *NPE* struktūra (S2 pav.) optimizuojama greitam grandininiam apdorojimui, įvedant trigerius tarp atminčių ir *DSP* blokų ir eksperimentiškai matuojant apdorojimo trukmę. Nustatyta, kad *NPE* veikia greičiausiai su vienu papildomu trigeriu tarp įėjimo buferių ir *DSP*, kuris didina LPLM taktinį dažnį bei mažina signalo sklidimo vėlinimą tarp dviejų trigerių kritiniame kelyje. Įskaitant 4 trigerius *DSP* bloke, 2 integruotus trigerius įėjimo buferių įvestyje ir išvestyje gautas optimalus 7 taktų vėlinimas vienai instrukcijai apdoroti. Paskutiniame kompiliatoriaus etape generuojamos instrukcijos apdorojimo elementui bei *HDL* byla registrų perdavimo lygmenyje apibūdinanti pasiūlytą *NPE*



S1 pav. Siūlomas pynučių-kopėtėlių daugiasluoksnio perceptronų tinklo įgyvendinimo metodas



S2 pav. Neurono apdorojimo elementas grįstas vienu skaitmeniniu signalų apdorojimo bloku

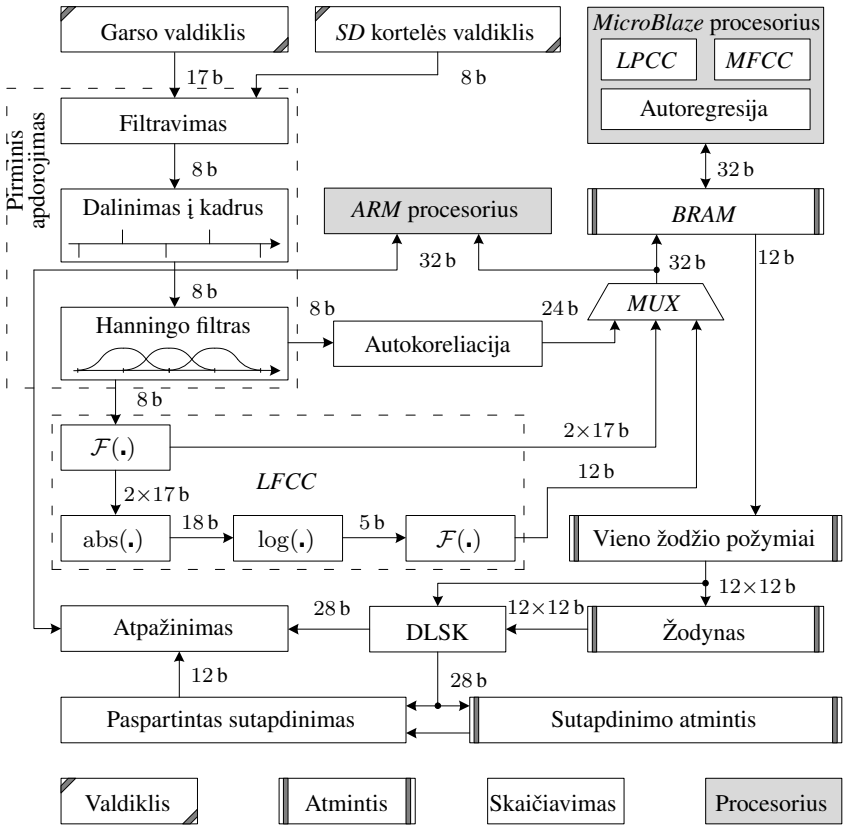
struktūrą. Sudarant N įėjimų neuroną sinapsėms skiriama N *NPE*, jos veikia lygiagrečiai, papildomas $(N+1)$-asis *NPE* naudojamas sinapsių sumos ir klaidos skaičiavimui neuronui besimokant.

Bitų skaičius duomenims nustatomas matuojant klaidą tarp fiksuoto ir slankaus kablelio neuronų įgyvendinimų taikant atitinkamai Vivado HLS ir Matlab įrankius. Keičiama sinapsių pralaidumo juostą ir matuojamos sinapsių pralaidumo juostos, centrinio dažnio ir neurono išėjimo klaidos. Netiesinei aktyvavimo funkcijai įgyvendinti taikoma blokinė atmintis. Aproksimuotos hiperbolinio tangento funkcijos išėjimo tikslumui vertinti taikomi vidutinės, maksimalios bei vidutinės kvadratinės klaidos įverčiai keičiant stiprinimo koeficientą ir atminties dydį.

3. Lietuvių šnekos atpažintuvo įgyvendinimas lauku programuojama logine matrica

Eksperimentiniam PKDP efektyvumui patvirtinti yra įgyvendinti šnekos atpažintuvo optimizuoti IN moduliai. Atpažintuvas, įgyvendintas viename LPLM luste, geba realiu laiku atpažinti į mikrofoną ištartas komandas. Greitam šnekos įrašų atpažinimo tikslumui patikrinti atpažintuvas gali nuskaitinėti įrašų duomenų bazę iš *SD* kortelės. Dauguma pagrindinių IN modulių, kurie reikalauja greitaveikos, yra įgyvendinti aparatūrą aprašančia kalba VHDL (balti blokai S3 pav.). Tikslumo reikalaujantys IN moduliai perkelti į programinį procesorių MicroBlaze. Atpažinimo eigos valdymui ir bendravimui su vartotoju taikomas *ARM* procesorius. IN modulių įgyvendintų LPLM, MicroBlaze ir *ARM* procesoriuose taktinis dažnis atitinkamai 50 MHz, 100 MHz ir 667 MHz. Kalbos signalas diskretizuojamas 11,025 kHz dažniu, kvantuojamas 256 lygiais.

Pirminio apdorojimo IN modulyje triukšmo filtravimui taikomi optimizuoti PKDP IN moduliai. Nufiltruotas šnekos signalas dalinamas į 23,22 ms trukmės kadrus po 256 imtis, kurioms pritaikomas Hanningo filtro IN modulis, kraštų efektui kadruose šalinti. Tam, kad atpažintuvas veiktų realiu laiku ir dėl pusiau persidengiančių kadrų, požymiai turi būti išskiriami ir palyginami per 11,61 ms. *LFCC* požymių išskyrimo



S3 pav. Pavienių žodžių atpažinimo sistemos blokinė diagrama

IN modulį sudaro du greitosios Furjė transformacijos IN moduliai (iš Xilinx intelektinių modulių bibliotekos), absoliutinės vertės ir logaritmo IN moduliai. *MFCC*, *LPC* ir *LPCC* metodų įgyvendinimas išskirstytas tarp LPLM ir MicroBlaze procesoriaus. Šnekos signalo autokoreliacijos koeficientų ir spektro skaičiavimo IN moduliai įgyvendinti LPLM dėl greitaveikos reikalavimo ir jų rezultatas perduodamas į MicroBlaze duomenų apdorojimui su slankaus kablelio skaičiaus tikslumu. MicroBlaze yra įgyvendinti: rekursinis Levinsono-Durbino, melų dažnių skalės filtravimo ir diskrečiosios kosinusinės transformacijos algoritmai. Keturi įgyvendinti kalbos analizės metodai skaičiuoja 12 požymių kiekvienam kadrui.

Požymiams palyginti taikomas dinaminio laiko skalės kraipymo (DLSK) metodas. Žodyno požymiai saugomi vidinėje LPLM *BRAM* atmintyje. Pavienio žodžio trukmė – iki $1{,}5\,\mathrm{s}$. Optimizuotas DLSK IN modulis palygina vieną žodį per 16640 taktinių impulsų. DLSK yra papildomai paspartintas $2{,}6$ karto taikant apribojimus sutapdinimo klaidų matricos skaičiavime (toliau vadinamas $\mathrm{DLSK_c}$). Naudojant vieną $\mathrm{DLSK_c}$ IN modulį vienas žodis palyginamas per $128\,\mu\mathrm{s}$ pasiekiant 7800 žodžių/s lyginimo greitį. Žodžių požymių palyginimui pagreitinti pasiūlytas dvigubo atsitiktinio

inicijavimo rezultatų sutapdinimo algoritmas, kuris kartu su DLSK naudoja papildomą atmintį saugoti surūšiuotus žodžius tarpusavio klaidos mažėjimo tvarka. Pagreitinimas grįstas tuo, kad ištarto žodžio požymiai lyginami ne su visu žodynu, bet su tais žodžio požymiais, tarp kurių klaida yra panaši. Šis būdas sumažina paieškos trukmę 62–70 %, tačiau reikalauja papildomai 13 % atminties bei sumažina iki 90 % žodžių atpažinimo tikslumą.

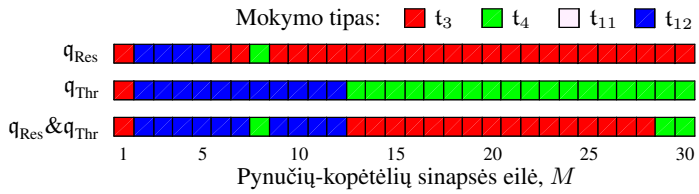
ARM procesorius taikomas žodžių atpažinimo procesui valdyti ir vartotojo sąsajai įgyvendinti. Atpažintuvas turi tris pagrindines būsenas: mokymo, modifikavimo ir valdymo. Mokymo metu atpažintuvas balsu siūlo vartotojui pasirinkti valdomą įrenginį ir suteikti pavadinimus komandoms, kurių požymiai ir garsiai įrašai išsaugomi. Komandų šalinimas, redagavimas, papildymas nauja, viso sąrašo perklausimas bei atpažintuvo atstatymas įvykdomi modifikavimo būsenoje. Valdymo būsenoje laukiama įrenginio pavadinimo bei komandos pavadinimo. Teisingai atpažintos komandos atveju išsiunčiamas signalas valdomam įrenginiui per infraraudonųjų spindulių, belaidį (*ZigBee*) ar laidinį ryšį. Atpažintuvas pereina į laukimo būseną, jei 30 s nebuvo ištartas joks žodis ir atsibunda, kai detektuojamas aktyvavimo žodis. 2 s po komandos ištarimo galima ją atšaukti pasakant atšaukimui priskirtą komandą. Signalizacija apie esamą būvį ir komandos atpažinimo teisingumą vartotojui pateikiama garso ir šviesos pavidalu. Tiek komandų mokymo, tiek šnekos atpažinimo metu atpažintuvo bendravimas su vartotoju vyksta balsu. Komandų atpažinimo tikslumui pagerinti yra taikomas kontekstinis žodynas – atskiras komandų rinkinys kiekvienam valdomam specializuotam funkciniam įrenginiui (neįgaliųjų lova, keltuvus ar ryšio su slauga sistema) bei kitiems buities prietaisams.

4. Įgyvendintų intelektinių modulių eksperimentinė patikra

Atlikti pynučių-kopėtėlių neurono IN modulių skaičiavimo tikslumo eksperimentai. Fiksuoto ir slankaus kablelio tikslumo neuronų įgyvendinimai lyginami tarpusavyje. Yra keičiama sinapsių pralaidumo juosta, duomenų bitų skaičius bei skaičiuojmos sinapsių pralaidumo juostos, centrinio dažnio ir neurono išėjimo klaidos. Nustatyta, kad 18 b tikslumas yra pakankamas norint pasiekti $0{,}1$ % pralaidumo juostos santykinę klaidą, kai juostos plotis sudaro $0{,}1$ % nuo testuojamo spektro.

Netiesinė aktyvavimo funkcija įgyvendinta *BRAM*. Jos tikslumui patikrinti keičiamas atminties dydis, stiprinimo koeficientas ir neurono pralaidumo juostos plotis matuojant vidutinę $\mathscr{E}_{\text{MA}}$, maksimalią $\mathscr{E}_{\text{MM}}$ klaidą neurono išėjime bei perdavimo funkcijos vidutinę kvadratinę klaidą $\mathscr{E}^{\text{AR}}_{\text{RMS}}$. 2 kB atminties dydis laikomas pakankamu $\mathscr{E}_{\text{MM}}$ esant $0{,}35$ % bei $\mathscr{E}^{\text{AR}}_{\text{RMS}}$ mažiau už $1{,}8\times10^{-3}$.

Pagal resursų $\mathfrak{q}_{\text{Res}}$ ir greitaveikos $\mathfrak{q}_{\text{Thr}}$ kriterijus tiriamas pynučių-kopėtėlių neuronas keičiant jo mokymo tipą (regresinę pynutę) $\mathcal{T} \in \{\mathfrak{t}_3, \mathfrak{t}_4, \mathfrak{t}_{11}, \mathfrak{t}_{12}\}$, sinapsių eilę M ir įėjimų skaičių. *NPE*, *DSP* ir *BRAM* skaičius tiesiškai priklauso nuo įėjimų skaičiaus (Sledevič, Navakauskas 2015). Pagal $\mathfrak{q}_{\text{Res}}$, $\mathfrak{q}_{\text{Thr}}$ ir abu kriterijus kiekvienai pynučių-kopėtėlių sinapsės eilei pateikiamas geriausias iš keturių mokymo tipų (S4 pav.). Neurono su vienu įėjimu ($N = 1$) ir pirmos eilės sinapse ($M = 1$) įgy-



S4 pav. Geriausi pynučių-kopėtėlių neurono įgyvendinimai pagal resursų, greitaveikos arba abu kriterijus: regresinių pynučių tipai, kurie turi būti taikomi esant konkrečiai sinapsės eilei M

vendinimas Artix-7 LPLM sunaudoja mažiau negu 1 % loginių resursų. Toks neuronas apsimokina per 90 taktinių impulsų. Atitinkamai 35 % loginių resursų skiriama neuronui su $N = 10$, $M = 10$, kurio mokymo vėlinimas 300 taktinių impulsų. Tokio neurono mokymo trukmę galima aproksimuoti funkcija:

$$t \approx 22{,}5M + 1{,}7N + 57{,}4 \ \ (\pm 4{,}2) \text{ [taktiniai impulsai]}. \tag{S4}$$

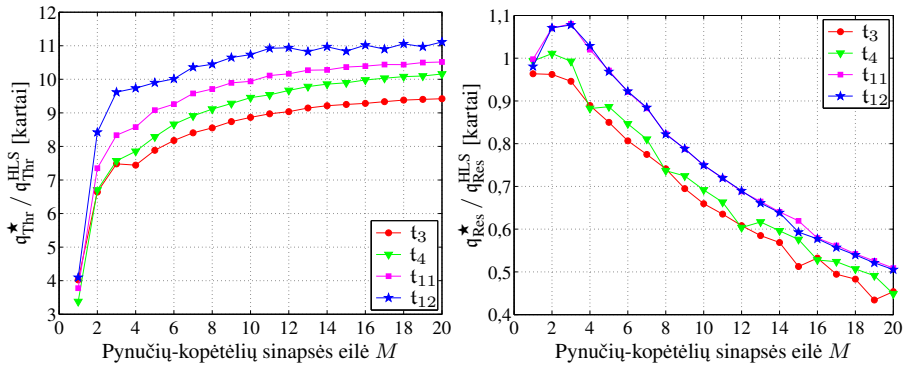
Taikant siūlomą metodą pasiektas 300 MHz LPLM įgyvendinto neurono grandyno maksimalus taktinis dažnis. Greitaveikos ir resursų kriterijų palyginimui neuronai su mokymo tipais $\mathcal{T}$ ir keičiama sinapsių eile M yra įgyvendinti taikant komercinį įrankį Vivado HLS. Įgyvendintiems sinapsių grandynams su mokymo tipais $\mathfrak{t}_3$, $\mathfrak{t}_4$, $\mathfrak{t}_{11}$, $\mathfrak{t}_{12}$ yra gauti atitinkamai 174 MHz, 179 MHz, 179 MHz ir 158 MHz didžiausi taktiniai dažniai. Lyginant su siūlomu metodu Vivado HLS įgyvendinti neuronai ilgiau apsimokina, nes reikalaujama daugiau taktinių impulsų. Dėl didesnio schemos taktinio dažnio ir mažesnio vėlinimo siūlomas metodas pagal greitaveikos kriterijų pranoksta Vivado HLS iki 11 kartų, kai yra taikomas santykinai sparčiausias mokymo tipas $\mathfrak{t}_{12}$ (S5 pav., a). Lyginant resursų panaudojimo kriterijus siūlomas metodas yra geresnis tik sinapsės eilės rėžiuose $M \in [2, 4]$ (S5 pav., b). Vertinant neuronų įgyvendinimo pranašumą pagal abu kriterijus $\mathfrak{q}_{\text{Thr}}$ ir $\mathfrak{q}_{\text{Res}}$ gautas mažiausiai 3 kartų pynučių-kopėtėlių neurono PKN kokybės pagerinimas sinapsių eilės tyrimo rėžiuose $M \in [1, 20]$.

PKDP įgyvendinimas LPLM tiriamas keičiant sluoksnių L, neuronų N skaičių ir sinapsių eilę M. Tyrime jau naudojamas optimalus mokytojo tipas pagal $\mathfrak{q}_{\text{Thr}}$ ir $\mathfrak{q}_{\text{Res}}$ kriterijus. Pareto efektyvioji kokybė pagal $\mathfrak{q}_{\text{Thr}}$ ir $\mathfrak{q}_{\text{Res}}$ skirtingų konfigūracijų PKDP pateikta Pareto frontų pavidalu (S6 pav.). Kiekviena kreivė jungia taškus žyminčius Pareto optimalų konkrečios konfigūracijos PKDP įgyvendinimą LPLM. Visi kiti galimi sprendimai, kurie nepriklauso Pareto frontui, nėra pateikiami. Abscisių ašyje išreikštas greitaveikos kriterijus $\mathfrak{q}_{\text{Thr}}$ parodantis didžiausią PKDP mokymosi greitį. Brūkšninė linija parodo egzistuojančių LPLM šeimų išteklių ribas, pagal kurias pasirenkamas tinkamas LPLM lustas konkrečiam PKDP IN modulio įgyvendinimui. Sprendimai, kurie yra už Virtex-7 ribos, parodo, kad didžiausios greitaveikos PKDP įgyvendinimas viršija septintos serijos LPLM galimybes.

Tiriami IN modulių taikomų pavienių žodžių atpažintuve tikslumas bei greitaveika. Požymių šnekos signale išskyrimo IN modulių tikslumo tyrimui taikomi 10-ties kalbėtojų (5 vyrai, 5 moterys) 100 žodžių įrašai po keturias sesijas. Pirmoji sesija taikoma atpažintuvo apmokymui, likusios sesijos – testavimui. 30 dB ir 15 dB signalas triukšmas santykio įrašai yra gauti originalius įrašus paveikus baltu triukšmu. Didžiau-

sias 93,2 % vidutinis originalių įrašų atpažinimo tikslumas gautas taikant *MFCC* IN modulį, 30 dB įrašams – 90,8 % taikant *LFCC* IN modulį ir 15 dB įrašams – 83,7 % taikant *LPCC*. Skliaustuose (S1 lentelėje) pateiktas žodžių atpažinimo tikslumo pagerinimas, kai pirminiame šnekos apdorojimo etape taikomas pynučių-kopėtėlių neuronas triukšmui šalinti klaidingai atpažintuose įrašuose. PKN mokymui taikomi originalūs įrašai be triukšmo. Pasiektas geriausias vidutinis 4,9 % atpažinimo tikslumo pagerinimas 15 dB signalas triukšmas santykio įrašams ir 2,0 % pagerinimas – 30 dB signalas triukšmas santykio įrašams, kai požymiams išskirti taikomas *MFCC* IN.

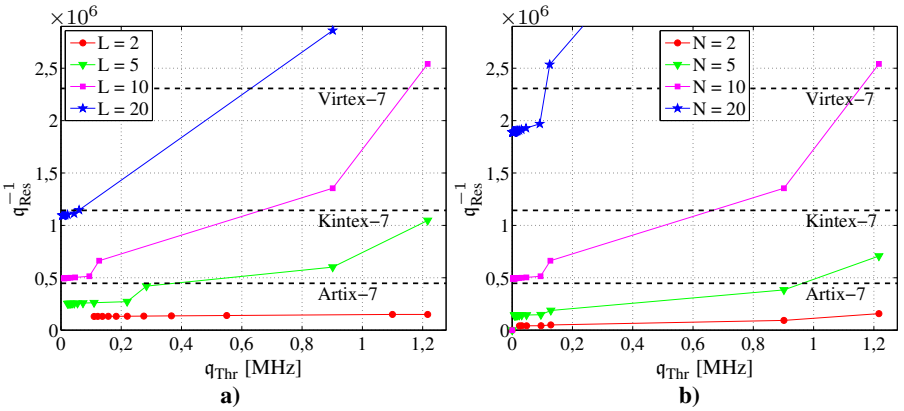
Taikant pasiūlytą dvigubo atsitiktinio inicijavimo rezultatų sutapdinimo algoritmą, *MFCC* IN modulį požymiams išskirti bei $\mathcal{M}_1$, $\mathcal{F}_4$ kalbėtojų įrašus, kurių atpažinimo tikslumas be pagreitinimo yra 99–100 % ribose, paieškos trukmę galima sumažinti 62–70 % aukojant kelis žodžių atpažinimo tikslumo procentus, tačiau testuojamų kalbėtojų žodžių atpažinimo tikslumas vis tiek išlieka 90–97 % ribose. LPLM įgyvendintų požymių išskyrimo ir palyginimo IN modulių greitaveika palyginta su Matlab terpėje įgyvendintomis atpažintuvo funkcijomis. Taikomas 3 GHz *CPU* su 50 % naudojamu procesoriumi. Iš laikinės diagramos (S7 pav.) matoma, kad visi LPLM įgyvendinti IN moduliai tenkina realaus laiko sąlygą ir paskaičiuojami greičiau negu per 11610 µs, kai šnekos signalo imtys yra surenkamos į kadrą. *LFCC* skaičiuojami 174 kartus, o *LPCC* ir *MFCC* – 3,5 karto greičiau negu to reikalauja veikimo rea-



S5 pav. Greitaveikos a) ir resursų b) kriterijų santykis, kai keičiamos M-tosios eilės pynučių-kopėtėlių neuronas įgyvendintas taikant siūlomą metodą ir Vivado HLS įrankį esant keturiems skirtingiems mokymo būdams

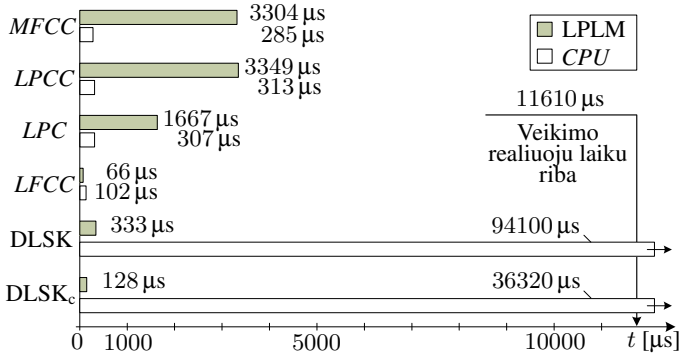
S1 lentelė. Vidutinis pavienių žodžių atpažinimo tikslumas

Požymiai	Originalūs įrašai	Įrašai su SNR = 30 dB	Įrašai su SNR = 15 dB
MFCC	93,2	86,7 (+2,0)	79,3 (+4,9)
LFCC	93,0	90,8 (+1,0)	72,4 (+4,7)
LPCC	91,5	89,8 (+0,7)	83,7 (+4,7)
LPC	82,5	78,6 (+1,2)	67,1 (+3,1)



S6 pav. Pareto fronto priklausomybė nuo resursų ir spartos kriterijų konkrečiam PKDP įgyvendinimui, kai a) $M = 10$, $N = 10$ ir keičiamas sluoksnių skaičius L arba kai b) $L = 10$, $M = 10$ ir keičiamas neuronų skaičius N

liuoju laiku sąlyga. DLSK_c per 11610 µs spėja palyginti ištartą žodį su 90 įrašų saugomų žodyne. Dėl santykinai 30 kartų didesnio *CPU* taktinio dažnio *MFCC*, *LPCC* ir *LPC* požymiai paskaičiuojami atitinkamai 10 ir 5 kartus greičiau negu LPLM įgyvendintame 100 MHz MicroBlaze procesoriaus IN modulyje. LPLM pranašumas pastebimas *LFCC* ir DLSK IN modulių įgyvendinimuose. Lyginant su Matlab įgyvendinimu *LFCC* yra paspartintas 1,5 kartus, o DLSK ir DLSK_c – daugiau negu 280 kartų.



S7 pav. LPLM ir *CPU* įgyvendintų intelektinės nuosavybės modulių požymiams išskirti ir palyginti laikinė diagrama

Atpažintuvo greitaveika ir tikslumas palyginti su kitų autorių pilnai programiniu įgyvendinimu grįstu LPLM programiniu procesoriumi MicroBlaze. Eksperimentai atlikti su vienodais žodynais: (5 vyrai, 5 moterys) 10 sesijų po 100 žodžių, 4 sesijos – apmokymui, 6 sesijos – testavimui. Atsižvelgiant į skirtingus diskretizavimo ir taktinį dažnius *LFCC* IN modulio greitaveika paspartinta 320 kartų, DLSK_c IN modulio – 348 kartus. Dėl didesnio šnekos signalo diskretizavimo dažnio žodžių atpažinimo tikslumas padidėjo 4,9 % iki 97,7 %.

Bendrosios išvados

Disertacijoje pasiūlyti sprendimai pynučių-kopėtėlių daugiasluoksnio perceptronų intelektinės nuosavybės (PKDP IN) moduliams įgyvendinti lauku programuojama logine matrica (LPLM) ir ištirtas modulių efektyvumas juos taikant neįgaliesiems skirtame lietuvių šnekos atpažintuve. Gauti šie elektros ir elektronikos inžinerijos mokslo krypčiai svarbūs rezultatai:

1. Sukurtas ir eksperimentiškai patvirtintas efektyvus metodas PKDP įgyvendinti LPLM:
   1.1. Sukurtas PKDP kompiliatorius generuoja daugiau nei 3 kartus efektyvesnį IN modulį lyginant su komerciniu Vivado HLS įrankiu.
   1.2. Sukurtas neurono apdorojimo elementas, grįstas LPLM *DSP* bloko struktūra, yra optimizuotas greitaveikai, jis vėlina tik 7 taktiniais impulsais, garantuojant mažesnę nei $1\,\%$ vidutinę absoliučią išėjimo signalo klaidą ir $1{,}8\times10^{-3}$ šaknies vidutinė kvadratinę pynučių-kopėtėlių neurono perdavimo funkcijos klaidą.
   1.3. Eksperimentiniais tyrimais išgautos žinios apie greičiausius PKDP mokymo algoritmus ir jų geriausias regresines pynutes $\mathcal{T} \in \{\mathrm{t}_3, \mathrm{t}_4, \mathrm{t}_{11}, \mathrm{t}_{12}\}$ pritaikytos minimizuojant peržvalgos lentelių skaičių ir vėlinimą.
2. Įgyvendinimams vertinti pasiūlytas PKDP grandynų sintezės specializuotų kriterijų Pareto frontų skaičiavimo būdas:
   2.1. Atrinktas LPLM lustas yra optimalus pagal reikalavimus PKDP struktūrai, diskretizavimo dažnį ir kitus resursus.
   2.2. Įgyvendintos dvi optimizavimo strategijos: greitaveikai ir resursams.
3. Neįgaliesiems skirtame lietuvių šnekos atpažintuve įgyvendinti ir eksperimentiškai patikrinti optimizuoti LPLM IN moduliai:
   3.1. Pynučių-kopėtėlių neurono IN modulio taikymas $4\,\%$ padidina atpažinimo tikslumą esant $15\,\mathrm{dB}$ signalas triukšmas santykiui.
   3.2. Žodžiams įrašuose be triukšmo atpažinti tinka *MFCC* ir *LFCC* IN moduliai.
   3.3. *LFCC* ir *LPCC* IN moduliai taikytini esant $30\,\mathrm{dB}$ signalo ir triukšmo santykiui, o esant $15\,\mathrm{dB}$ – *LPCC* IN modulis užtikrina didžiausią atpažinimo tikslumą.
4. Sukurtas ir patikrintas pirmasis neįgaliesiems skirto valdymo lietuviška šneka įrenginio, grįsto LPLM, prototipas:
   4.1. Naujas spartesnis žodžių sutapdinimo algoritmas naudoja apribotą dinaminį laiko skalės kraipymą ir atpažįsta žodžius $2{,}6$ karto greičiau, neprarasdamas $97\,\%$ siekiančio atpažinimo tikslumo.
   4.2. Pasiūlytas dvigubo atsitiktinio iniciavimo rezultatų sutapdinimo algoritmas papildomai iki $2{,}6$ karto paspartina atpažinimo procesą, tačiau sumažina iki $90\,\%$ žodžių atpažinimo tikslumą.

Subject Index

A

algorithm
  depth first search 53
  gradient descent training 23, 92, 96
  greedy 54
  Levinson-Durbin 40, 80, 114
  Viterbi 41
  word recognition . 35, 76, 77, 105, 106, 111
analysis
  cepstral 37–39, 78–80
  spectral 37, 78
ANN . . viii, 3, 9, 16–21, 27–29, 31–33, 41, 43, 49, 52
approximation 30
ARM 5, 88–90
ASAP 59
ASIC 10, 35
autocorrelation 39, 80, 113

B

backpropagation through time 23
BRAM . . ix, 13, 30, 31, 47, 64, 66, 68, 78–80, 82, 83, 92–96, 98, 107

C

CAD 9, 10
chip familly
  Artix-7 5, 18, 49, 90, 94, 100, 101, 137, 147
  Cortex A9 . . . . . 5, 90, 137
  MicroBlaze 149
  ML402 137
  Virtex-4 . . . . . 4, 35, 137
  Virtex-7 147
  ZynQ-7000 . . . . . 4, 5, 18, 90, 137
circuit . . 10, 22, 24, 31, 32, 39, 60, 63, 69, 92, 96, 101, 104
company
  Google 3, 136
  Nuance 3, 136
  Xilinx . 4, 13, 14, 17, 28, 29, 47, 63, 64, 78, 92, 102, 137, 145
condition 15, 21, 24, 68, 69, 71, 76
confidence interval 109
constrained scheduling 51, 56, 58, 59
CPM 56, 57
critical path . 12, 22, 29, 33, 46, 49, 56, 58, 59

D

DFG 50–53, 56
DFS 53
dictionary . . 35, 41, 76, 77, 80–83, 85–87, 89, 106, 107, 111, 113
directed graph 50–52, 58
DSP . . . . . viii, ix, 11–15, 18–20, 26–30, 40, 42, 47, 49–53, 55, 56, 58–61, 63, 73, 78–80, 92, 93, 96–98, 103, 107, 117

DTW 41, 42, 75, 78–81, 83–86, 90, 107, 111–115, 118

E

efficient implementation . . . 32, 46, 48–50, 73, 103

F

features
  classification 41, 42, 81, 106, 108, 109
  extraction 34, 78, 105, 108, 110, 113
  selection 78, 85, 86
  vector 77, 83, 85, 86, 111
FFT 106, 113
filter 14, 16, 17, 21, 22, 39
  band-pass 21, 63
  band-stop 67
  stability 21–23
FIR 14, 16
FPGA . . . . . v, ix, 1, 3–6, 8–20, 26–33, 35, 36, 38, 39, 41–43, 46–52, 56, 63, 68–71, 73, 75–78, 81, 88–94, 96, 98–108, 113, 116–118
  clock frequency . 22, 29, 33, 46, 48, 61, 73, 90, 102, 106, 107, 113
FPGA resource
  BRAM . . . 11, 12, 30, 47, 63, 66, 77, 78, 80, 82, 93, 95, 98, 107
  DSP . . 11, 12, 29, 47, 49, 53, 58, 60, 78, 80, 93, 96, 97, 107
  LUT . . . 11, 12, 30, 47, 51, 93, 98, 100, 107
  LUT equivalent 104
  Slice 11, 12
function
  activation 29, 51, 60, 66, 93, 95
  Hanning 76, 114
  trigonometric 51, 60, 64
  window 39, 76, 78, 82, 84, 87, 114

G

graph 50, 51, 53, 55
  covering 51, 54, 56, 59
  isomorphism 53
  matching 51, 53, 54
  merging 51, 54, 56

H

HDL 10, 14, 17, 18, 32, 50, 59–61, 102
hidden Markov model 34, 41
HLS 10, 11, 14, 15, 18, 42

I

IIR 16
implementation
  criteria 33, 46, 48, 100, 102–104
  parallel . . 16, 19, 25, 33, 35, 41, 42, 62, 68, 71, 80–82, 102
  pipeline . . 35, 49, 51, 55, 58, 61, 62, 71, 81, 83
  quality 33, 46, 48
  strategy . . 45, 46, 48, 53, 59, 63, 68, 71, 73, 117
instruction
  modification 55, 60, 61
  scheduling 13, 50, 56, 58, 59, 63, 73
investigation 61, 63, 92, 94, 96, 103, 104, 106, 111
IP . . . . . v, 3–6, 14, 17, 28, 38, 43, 73, 75, 76, 78, 80, 81, 88–92, 102, 106, 108, 110, 111, 115, 118
IP module
  autocorrelation 39, 80, 114
  autoregression 39, 80, 114
  DTW . . 41, 77, 78, 81, 83, 85, 86, 107, 111, 113–115
  FFT 78, 79, 113, 114
  framing 76, 79
  infrared 89, 90
  iterative voice response 88
  LFCC . . . 37, 77, 78, 90, 106, 108, 113–115
  logarithm 78, 79, 113, 114
  LPC 39, 78, 80, 90, 108, 113–115
  LPCC 39, 78, 80, 90, 108, 113–115
  MFCC . . . 38, 77–79, 90, 108, 109, 113–115
  MicroBlaze 35, 77, 80, 144, 145
  neuron processing element . . 46, 51, 60–63, 71, 102, 104
  synchronization 58, 76, 83
  UART 90
  voice activity detector 89
  windowing 39, 76–78, 114
  Zigbee 89, 90

L

latency . . 12, 14, 19, 28–30, 32, 45–47, 49, 50, 54, 56, 58, 61, 64, 68–73, 78, 83, 98, 100, 102, 103, 115
LFCC . . viii, 36–38, 78, 79, 90, 106–108, 111, 113–115, 118
LL 16, 68
LLF . . 17, 18, 21–24, 26, 49, 61, 92, 93, 96, 97

LLMLP . v, ix, 3–6, 11, 12, 16, 22, 24–27, 42, 43, 45–53, 55, 56, 61, 63, 68–73, 75, 77, 91, 92, 96, 98, 103–105, 116, 117
LLN . ix, 21, 49, 62, 64, 66–68, 91–93, 95–98, 100–103, 106, 110, 111, 115, 118
LPC . viii, 36, 37, 39–41, 78, 80, 90, 108, 111, 113
LPCC viii, 36, 37, 39–41, 78, 80, 90, 108, 111, 113–115, 118
LSR 75, 88, 90, 105, 106, 108
LUT . 11, 13, 18, 29–31, 33, 38, 39, 47, 52, 64, 66, 67, 92–95, 98, 100, 102, 103, 105, 108, 115, 117

M

MAC 28, 29, 39
MAE . . . . . viii, 65, 67, 93
MFCC . . . . . viii, 36–39, 78, 79, 90, 108, 111, 113–115, 118
MLP 16, 19, 22, 28, 29
MME . . . . . viii, 67
model 19, 34, 39, 41
multilayer perceptron . . 16, 23, 25, 63, 68, 71, 104

N

noise 15, 35, 75, 92, 108–110
NPE 45, 46, 60–63, 71, 72, 102, 105

O

optimization strategy 63, 68, 71
order recursion 23

P

PAR 11, 28, 32, 33, 46, 61
parameters . . 21, 23, 25, 28, 63, 64, 67, 71, 87, 92, 93, 111
Pareto frontier . . . 45, 48, 73, 92, 103, 104, 115
pattern matching 81, 85, 86, 111, 112
power spectral density 67
precision 19
  fixed-point 13, 14, 16, 28, 39, 41, 47, 49, 63, 65, 66, 92
  floating-point 14, 16, 41, 47, 49, 63, 66, 77, 78, 106
programming language
  C 10, 15, 28, 42
  C++ 15, 28
  C# 28
  Matlab 28
  RTL 28
  SystemC 10, 15
  Verilog 10, 18
  VHDL . . . . . vii, 10, 18, 28, 42, 96, 144
PWL 29, 30

Q

quality criteria 32, 45, 46, 102

R

RAM 11–13, 30, 33, 47, 61, 73, 115
real-time . . 35–39, 42, 76, 77, 81, 85, 113, 114
reconfigurable block 18, 29, 31, 49, 52, 55
regressor lattice . 24, 25, 49, 51, 56, 61, 62, 98, 100, 102, 103, 115
RMSE . . . . . viii, 95
ROM 13, 18, 60
RTL 10–12, 14, 29

S

Schur recursion 21
software
  AccelDSP 14, 15
  C-to-Silicon 14, 15
  Catapult C 14, 15
  HDL Coder 14, 15
  HLS 96
  ISE Design Suite 4, 14, 101, 102, 137
  LabView 10, 15, 39, 140
  Matlab . . 4, 10, 14, 28, 63, 64, 67, 113, 137, 144, 148, 149
  ModelSim 4, 137
  RapidSmith 11
  Simulink 10, 14, 19, 28, 39, 140
  StateCAD 14, 15
  Synphony 14, 15
  System Generator 14, 63
  Verilog 140
  VHDL 140
  Vivado HLS . . 4, 5, 14, 15, 64, 67, 92, 102, 103, 116, 117, 137, 138, 140, 144, 147, 148, 150
  XSG 14, 15
speech recognition 34, 41, 76, 81, 105, 113
speech records 76, 88, 105, 107–109, 111
structure . 10, 12, 16–19, 21, 28, 32, 48, 50, 63, 68, 71
subgraph 50, 53–55, 59, 60, 96, 103
  elimination 54
support vector machine 28

synapse . 13, 19, 21, 22, 29, 31, 32, 46, 47, 49, 55, 56, 61–63, 66, 68, 71, 92, 96, 98, 100–103, 110
systolic array 31, 40, 41

T

threshold 19
transfer function 67, 95, 96

U

user interface 88

V

vector35, 37–39, 41, 42, 57, 63, 77, 81, 83, 85, 86, 111
VLSI 31

Z

zero-pole plane 93

Annexes[4]

Annex A. Created Intellectual Property Cores

Annex B. The Co-authors' Agreement to Present Publications Material in the Dissertation

Annex C. The Copies of Scientific Publications by the Author on the Topic of the Dissertation

[4]The annexes are supplied in the enclosed compact disc

Tomyslav SLEDEVIČ

AN EFFICIENT IMPLEMENTATION OF LATTICE-LADDER MULTILAYER PERCEPTRONS IN FIELD PROGRAMMABLE GATE ARRAYS

Doctoral Dissertation

Technological Sciences,
Electrical and Electronic Engineering (01T)

PYNUČIŲ-KOPĖTĖLIŲ DAUGIASLUOKSNIŲ PERCEPTRONŲ EFEKTYVUS ĮGYVENDINIMAS LAUKU PROGRAMUOJAMOMIS LOGINĖMIS MATRICOMIS

Daktaro disertacija

Technologijos mokslai,
elektros ir elektronikos inžinerija (01T)

2016 05 06. 14,25 sp. l. Tiražas 20 egz.
Vilniaus Gedimino technikos universiteto leidykla „Technika“,
Saulėtekio al. 11, LT-10223 Vilnius,
http://leidykla.vgtu.lt
Spausdino BĮ UAB „Baltijos kopija“,
Kareivių g. 13B, LT-09109 Vilnius