Declaration:

Hereby I declare that I have prepared my doctoral thesis independently based on the sources mentioned in bibliography.

I have no relevant reason against using this work according to Copyright Act (§ 60 Zákona č. 121/2000 Sb.).

In Prague _____ _____

Ing. Raissa Likhonina

Acknowledgement

I would like to acknowledge and to express my gratitude to my supervisor doc. Ing. Evženie Uglickich, CSc. and Ing. Jiří Kadlec, CSc., the head of the signal processing department of ÚTIA AV ČR, v.v.i., for their help, useful comments, valuable advices and engagement through the research process.

I would like to thank doc. Ing. Ivan Nagy, CSc. and my colleagues for their support and useful comments during my work on the thesis.

I would like to greatly thank my parents and my sister for their support and patience during my studies.

The thesis has been supported by the following projects:

- ECSEL project SILENSE "(Ultra)Sound Interfaces and low Energy iNtegrated Sensors", Project No.: ECSEL 737487, MSMT 8A17006.
- ECSEL project StorAIge "Embedded storage elements on next MCU generation ready for AI on the edge", Project No.: ECSEL 101007321, MSMT 8A21009

TITLE: Fast Bayesian Algorithms for FPGA Platforms

AUTHOR: Ing. Raissa Likhonina

DEPARTMENT: Department of Applied Mathematics, K611

held in ÚTIA AV ČR, v.v.i., Department of Signal Processing, under supervision of Ing. Jiří Kadlec, CSc.

BRANCH OF STUDY: Engineering Informatics in Transportation and Telecommunications

SUPERVISOR: doc. Ing. Evženie Uglickich, CSc., Department of Applied Mathematics, Faculty of Transportation Sciences, Czech Technical University in Prague

ABSTRACT:

The thesis is devoted to fast Bayesian algorithms, more precisely to the QRD RLS Lattice algorithm combined with hypothesis testing and applied to hand detection problem solution based on ultrasound technology. Due to the proposed structure of regression models and the offered approach to hypothesis testing in the work, the algorithm under consideration is able to solve the problem of noise cancellation and additionally to compute the distance between the hand and the device; thus, potentially enabling to identify simple gestures. Further, the algorithm was implemented in parallel on the HW platform of Xilinx Zynq Ultrascale+ device with a quad-core ARM Cortex A53 processor and FPGA programmable logic and proved to function reliably and accurately in real time using real data from an ultrasound microphone.

The work contains an investigation of the state of the art in the corresponding field and gives the theoretical background necessary for the development and modification of the algorithm to fulfill the goals of the thesis.

The thesis also includes thorough description of experiments and an analysis of the results including those from simulation and from computation using real ultrasound data both in the MATLAB R2019b environment and on the HW platform of Xilinx Zynq Ultrascale+.

KEYWORDS: RLS algorithms, FIR filters, hypothesis testing, Bayesian approach, FPGA, parallel processing, pipelining, ultrasound, hand detection

NÁZEV: Rychlé algoritmy Bayesovského rozhodování pro FPGA platformy

AUTOR: Ing. Raissa Likhonina

ÚSTAV: Ústav aplikované matematiky, K611

řešená v ÚTIA AV ČR, v.v.i., oddělení zpracování signálů, pod vedením Ing. Jiří Kadlec, CSc.

OBOR: Inženýrská informatika v dopravě a spojích

VEDOUCÍ PRÁCE: doc. Ing. Evženie Uglickich, CSc., Ústav aplikované matematiky, Fakulta dopravní, České vysoké učení technické v Praze

ABSTRAKT:

Disertační práce je věnována rychlým algoritmům Bayesovského rozhodování, přesněji řečeno QRD RLS Lattice algoritmu s testováním hypotéz, který byl aplikován na řešení problému detekce ruky na základě ultrazvukové technologie. Během výzkumu se ukázalo, že je potřeba navrhnout strukturu regresních modelů a přistupovat k testování hypotéz určitým způsobem pro zvolený případ, tj. aby algoritmus byl schopen potlačit šum a navíc vypočítat vzdálenost ruky od zařízení, což by potenciálně umožnilo identifikovat jednoduchá gesta. Dalším cílem bylo implementovat algoritmus na HW platformě za použitím reálných dat z ultrazvukových mikrofonů. Algoritmus byl implementován na zařízení Xilinx Zynq Ultrascale+ s programovatelnou logikou FPGA a běží paralelně na jeho čtyřjádrovém procesoru ARM Cortex A53. Algoritmus byl zatím implementován na FPGA programovatelné logice. Během experimentů se ukázalo, že algoritmus funguje spolehlivě a přesně v reálném čase s využitím reálných dat z ultrazvukového mikrofonu.

Práce obsahuje zkoumání současného stavu problematiky v příslušném oboru a poskytuje teoretické podklady nezbytná pro vývoj a modifikaci algoritmu pro splnění cílů práce.

Součástí práce je také důkladný popis experimentů a analýza výsledků ze simulace a z výpočtu za využitím skutečných ultrazvukových dat jak v prostředí MATLAB R2019b, tak i na HW platformě Xilinx Zynq Ultrascale+.

KLÍČOVÁ SLOVA: RLS algoritmy, FIR filtry, testování hypotéz, Bayesovský přístup, FPGA, paralelní zpracování, pipelining, ultrazvuk, detekce ruky

CONTENT

GLOSSARY

Abbreviation	Definition
ASIC	**A**pplication **S**pecific **I**ntegrated **C**ircuits
BRAM	**B**lock **R**andom **A**ccess **M**emory
CLB	**C**onfigurable **L**ogic **B**lock
CMOS	**C**omplementary **M**etal-**O**xide **S**emiconductor
CPU	**C**entral **P**rocessing **U**nit
DAS	**D**elay-**A**nd-**S**um
DDR	**D**ouble **D**ata **R**ate
DF	**D**irectional **F**orgetting
DMA	**D**irect **M**emory **A**ccess
DP	**D**ouble **P**recision
DPU	**D**ata **P**rocessing **U**nit
DSP	**D**igital **S**ignal **P**rocessing
EF	**E**xponential **F**orgetting
FIFO	**F**irst **I**n **F**irst **O**ut
FIR	**F**inite **I**mpulse **R**esponse
FP	**F**loating **P**oint
FPGA	**F**ield **P**rogrammable **G**ate **A**rray
FSM	**F**inite **S**tate **M**achine
FTF	**F**ast **T**ransversal **F**ilter
GUI	**G**raphic **U**ser **I**nterface

HDL	**H**ardware **D**escription **L**anguage
HLS	**H**igh **L**evel **S**ynthesis
HMI	**H**uman-**M**achine **I**nterface
HT	**H**ouseholder **T**ransformation
HW	**H**ard**W**are
IC	**I**ntegrated **C**ircuits
I/O	**I**nput/**O**utput
IP cores	**I**ntellectual **P**roperty cores
LAB	**L**ogic **A**rray **B**lock
LS	**L**east **S**quares
LMS	**L**east **M**ean **S**quares
LUT	**L**ook**U**p **T**able
MIMO	**M**ultiple-**I**nput-**M**ultiple-**O**utput
MMC	**M**ulti**M**edia**C**ard
MPSoC	**M**ulti**P**rocessor **S**ystem **o**n **C**hip
NLMS	**N**ormalized **L**east **M**ean **S**quares
PC	**P**ersonal **C**omputer
PCI	**P**eripheral **C**omponent **I**nterconnect
PCM	**P**ulse-**C**ode **M**odulation
PLL	**P**hase-**L**ocked **L**oop
QRD	**QR D**ecomposition
QRD-LSL	**QRD L**east **S**quares **L**attice
RAM	**R**andom **A**ccess **M**emory
RLS	**R**ecursive **L**east **S**quares
RTL	**R**egister-**T**ransfer **L**evel
RMS	**R**oute **M**ean **S**quare

SDK	**S**oftware **D**evelopment **K**IT
SDSoC	**S**oftware **D**evelopment **S**ystem **o**n **C**hip
SFG	**S**ignal-**F**low **G**raph
SILENSE	(ultra)**S**ound **I**nterfaces and **L**ow **E**nergy i**N**tegrated **S**ensors
SIMD	**S**ingle-**I**nstruction-**M**ultiple-**D**ata
SISD	**S**ingle-**I**nstruction-**S**ingle-**D**ata
SISO	**S**ingle-**I**nput-**S**ingle-**O**utput
SoC	**S**ystem **o**n **C**hip
SoM	**S**ystem **o**n **M**odule
SP	**S**ingle **P**recision
SRAM	**S**tatic **R**andom **A**ccess **M**emory
SRC	**S**quare-**R**oot **C**ovariance
SRI RLS	**S**quare-**R**oot **I**nformation **R**ecursive **L**east **S**quares
SVM	**S**upport **V**ector **M**achine
SW	**S**oft**W**are
TD	**T**ime **D**elay
TTL	**T**ime-**T**o-**L**ive
URAM	**U**ltra **R**andom **A**ccess **M**emory
VHDL	**V**ery High Speed Integrated Circuit **H**ardware **D**escription **L**anguage
VLIW	**V**ery **L**arge **I**nstruction **W**ord
VLSI	**V**ery-**L**arge-**S**cale-**I**ntegration

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

The present work is devoted to Fast Recursive Bayesian Algorithms and their implementation on hardware platforms with Field Programmable Gate Array (FPGA) programmable logic. The area of algorithm application is very wide, especially in digital signal processing applications, and includes, but is not limited to, parameter estimation, echo suppression, beam-forming, radar applications, equalization, etc. However, there are certain difficulties in implementing the algorithms under consideration on hardware platforms due to their high computational complexity and problems with numerical stability.

In turn, FPGA platforms are also well known and commonly used in aerospace and automotive industries, bioinformatics, high performance computing, medical and industrial applications. One of the reasons for its popularity is in the fact that they are faster for some applications, which is due to their parallel capabilities and optimality in terms of the number of gates used for a certain process. The FPGA platforms are frequently used during development of pre-defined applications, before implementing them on Application Specific Integrated Circuits (ASIC) [100].

The work under consideration is motivated and supported by the European project called "SILENSE" standing for (ultra)Sound Interfaces and Low Energy iNtegrated Sensors. The project started from the 1st of May, 2017 and lasted for 36 months. Its main field was acoustic technologies used for activation and control of devices by gesture as well as for data communication and indoor positioning. The domains, which were targeted by the project, were wearables, automotive and smart home applications. Within the project and in respect to these domains, it was planned to achieve such important goals as to create intuitive user interfaces in mobile and wearable devices, to improve hygiene due to touchless control, to increase safety by developing gesture recognition applications for in-car system control and for machinery control in industrial applications, to increase security by gestural authentication and to improve the quality of life for disabled and old people. The project comprised development of acoustic technology on all levels, i.e. hardware, software and the system. In terms of hardware, it was supposed to develop new micro-acoustic transducers to decrease the cost and energy consumption as well as to improve performance of the end devices. Besides, it was planned to develop more specifically heterogeneously and monolithically integrated arrays of micro-acoustic transducers with their supporting electronics and to provide a dedicated low-power Integrated Circuits (IC) design. As far as software is concerned, the project aimed at developing smart algorithms for acoustic data communication, sensing and gesture recognition [26, 95].

The present work is connected with that part of the project, which is targeted to the development of new algorithms for gesture recognition applications. Examples

of such applications can be in-car applications for controlling audio and video systems, dashboards or applications used for navigation and entertainment purposes or air-conditioning control settings. In this case it is supposed that gesture-based applications will improve safety as the time, when the driver's attention is distracted from the road, will be significantly reduced. The passengers can also benefit from such kind of a system by easily interacting control elements, which are far or even unreachable for them [26].

The system is supposed to be based on a network of ultrasound transducers with an integrated pre-processing unit (see Fig. 1). These transducer arrays will be integrated behind the screen or on the flexible foil, which will be placed at an optimal position in the interior of the car (headliners, door panels, seats, etc.) [26].

The ultrasound impulses are supposed to be transmitted by the system, reflected from a hand and returned back to the system. On the basis of responses and their characteristics, the device should be able to detect the presence, position and distance of the hand. Because the hand will not be necessarily perpendicular to the surface of the display, it is supposed that microphones will be turning to the direction of hand movement. Therefore, in a final application an adaptive beamforming technique may be used for directional signal reception [105].

However, it is obvious that there will be also reflections from the objects in the environment other than the hand (see Fig. 2). These undesired responses can present a great challenge for a recognition process and, therefore, should be removed from the target signal.

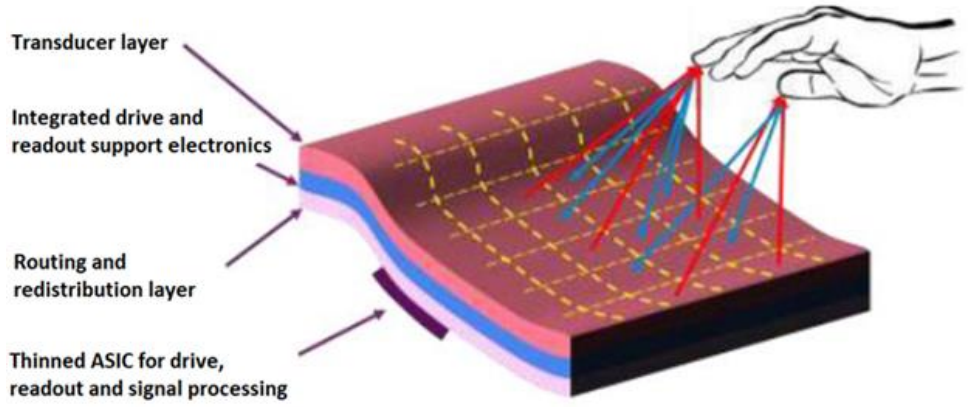

Figure 1: Thin flexible foil with a matrix of ultrasound transceivers for gesture recognition [26]

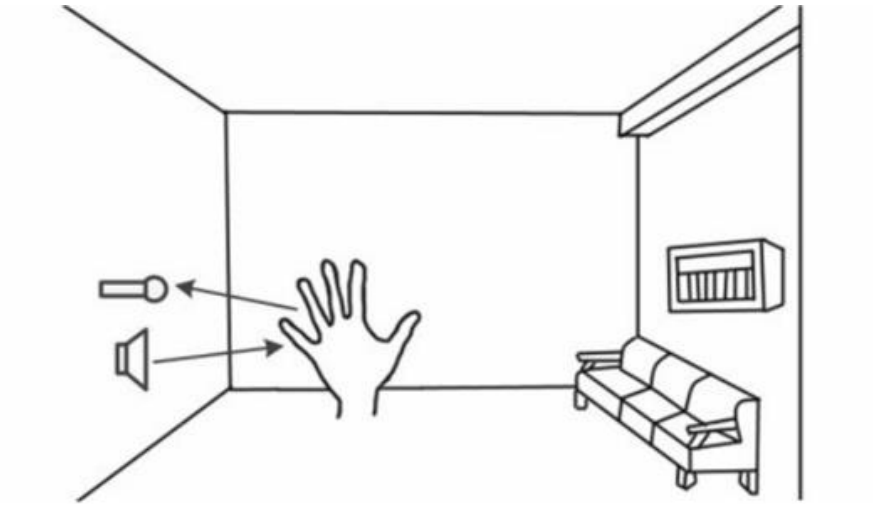

Figure 2: Example of a hand detection application [60-61]

It follows that for pre-processing of incoming data in a way of reducing acoustic noise, it is necessary to develop the corresponding algorithms based on the noise cancellation technique. These algorithms have to be numerically robust and fast, being capable to provide user interaction in low-power, low-cost situations and to process incoming data in real-time. Therefore, the main goal of the thesis is to develop appropriate algorithms for such kind of applications.

The work is also supported by the ECSEL project "StorAIge", which stands for "Embedded storage elements on next MCU generation ready for AI on the edge". It focuses on increasing high performance of new platforms and decreasing the energy consumption. It targets automotive, industrial and security markets [97].

State of the Art

The main interest of the thesis is the adaptive Recursive Least Squares (RLS) algorithms used for system identification [40-41, 76]. These algorithms can be derived from Bayesian theory for adaptive system identification in real time and will be extended with hypothesis testing to identify the probability of an identification model best suited for a particular situation (hand presence/absence) [40-41, 81].

It is worth noting that the adaptive RLS filters are already widely used in many real applications including speech analysis, video compression, noise and echo cancellation, equalization, mobile and multimedia systems, beam-formers, system identification and radar applications [21]. It is clear that the research made in this field is enormous and very profound.

However, while attempting to implement the algorithms on hardware platforms, it appeared that there were certain problems due to their high computational complexity and numerical stability issues [76]. To deal with the computational complexity, the fast versions of the RLS algorithms were developed. To name a few investigations in this area, the following works should be mentioned [15, 29, 76]. To solve the issue with the numerical stability, a so-called QR decomposition of recurring updated matrices was proposed [11, 76, 90, 96].

Hereby, let us give a short insight into studies of the RLS algorithms and their development to understand what is already done and what requires further investigations.

One of the first practically applicable algorithms from this group was so-called Levinson-Durbin recursion, which represented an effective method for parameter estimation of one-step predictor of a stationary random process monitored on a finite time interval. This algorithm was well described in a work by Markel [40].

Itakura F. and et al continued to work with this algorithm and replaced the gradual calculation of a predictor of an increasing order with corresponding relations between prediction errors. The relations were represented in the form of a lattice structure and the parameters of the algorithms ranged from -1 to 1. This property was very important and contributed to further development of the algorithms with the same structure [40].

A work by Lee D., Morf M. and Fridlander B. had a great importance for the RLS algorithm development in identification of an autoregression model [52]. The authors developed the lattice algorithms equivalent with the least squares methods, a so-called LS lattice, and for the first time they used normalization of the variables in the range from -1 to 1. So the normalized algorithms had several benefits as compared with the non-normalized: mainly in a fewer storage and lower computational requirements. Besides, they could be easily implemented in a fixed-point arithmetic [52]. This work was very popular and a number of scientists continued to work in this direction. Among them are Lev-Ari [54], Ljung [67-68], Porat [86-87].

Porat B. and co-authors at this time explored the square root normalized ladder algorithms, where they developed the growing memory and sliding memory covariance ladder algorithms and used the estimated reflection coefficients for computing the model parameters [86]. The other work from 1983 was devoted to the least squares identification of the finite impulse response (FIR) models and to the development of the square-root normalized lattice algorithms both for the time-invariant models and for tracking the time-varying parameters [87].

Lev-Ari H. and et al described the least squares adaptive lattice and transversal filters using a unified geometric theory [54]. The filters described in these works were applicable for the nonstationary processes. The authors also described the windowed fast transversal filters adaptive algorithms with normalization and discussed the trade-off between the growth rate of numerical errors and the computational requirements for the fixed-order algorithms.

At the same time the other algorithms of a recursive parameter identification of an autoregression model, which were equivalent to the least squares method, were developed. The algorithms were called the fast Kalman and the fast lattice and some of them also allowed the normalization of the variables by their time varying ranges [40].

At this time Ljung S. and Ljung L. focused on the analysis of the recursive algorithms and the error propagation of the RLS adaptation algorithms [67-68]. In their work the authors proved the exponential stability of the conventional LS algorithms and the fast lattice algorithms in terms to such errors and that the base of the decay

was equal to the forgetting factor. However, the fast least squares algorithms or the fast Kalman algorithms were shown to be numerically unstable.

Cioffi J. and et al made a great contribution to investigation in this field. Their works were dedicated to the fast RLS transversal filters for adaptive filtering, where the authors offered substantial reductions in the computational requirements relative to the fast RLS algorithms such as the fast Kalman algorithms of Morf M., Ljung L. and Falconer D. and the fast lattice algorithms of Morf M. and Lee D. [15-16]. Besides, Cioffi J. focused on the limited-precision effects in adaptive filtering and discussed the problem of the overflow due to the accumulating errors [18]. In the work [17] the author proposed to replace the Givens rotations used for the fast QR algorithms with the Householder transformation to significantly reduce the computation.

The work by Samson L., Ardalan S. H. and Bottomley G. E. were devoted to the analysis of the algorithm errors. Assuming rounding arithmetic, Samson made the analysis of fixed point errors of the normalized lattice algorithm used for autoregressive system identification [92].

Bottomley also focused on the round-off errors of the fixed-point RLS and stated that they caused the instability [11]. The solution proposed by the author was to bias these round-off errors.

Ardalan S. H. investigated both the floating point errors of the RLS and Least Mean Squares (LMS) adaptive filters and the fixed-point round-off errors of the exponentially windowed RLS algorithms used for time-varying systems [4-5]. In both cases the author concluded that the forgetting factor lambda played a very important role and influenced the resulting noise. The researcher stated that to reduce the algorithm sensitivity to the additive noise, it was necessary to set lambda close to one. But, on the other hand, the round-off error would increase as lambda -> 1.

Farbre P. and et al in their work offered to use normalization to improve the fast RLS algorithms [29]. The results were shown on the example of the fast Kalman algorithm.

The stability problem was also discussed in works [38, 53, 96].

Leung H. and Haykin S. analyzed the stability of the recursive QRD-LS algorithms in regards to the finite precision systolic array implementation [53].

Slock D. T. M. developed the numerically stable fast transversal filters with exponential weighting for the RLS adaptive filtering [96]. The stability was achieved due to the feedback gains, which became possible with introducing redundancy into the algorithms.

Horita E. and et al offered a new RLS criterion to solve a numerical stability problem resulted from the finite precision errors [38]. This criterion included a strong parameter energy factor, which contributed to the algorithm stability.

The QRD-based RLS algorithms are proved to be stable, but due to the square-root computations for the Givens rotations they can cause a problem of a so-called computational bottleneck. This was investigated by Hsieh S. F. and et al in their work,

where the authors tried to develop a unified approach for the QRD-based RLS estimation without computing the square roots [37].

An interesting work from this point is one by Sakai H. and et al, where the RLS algorithm of a modified Gram-Schmidt type of the parallel extraction is presented [91]. These algorithms are the counterpart of the algorithms using an inverse QR decomposition based on the Givens rotations and do not contain the square root operations. Thus, the problem of a bottleneck is also solved.

The probability approach to identification of stochastic systems was formulated in works by Peterka V., Kárný M. and Kulhavý R. [47-48, 50, 81]. Unknown parameters of a model in these works were supposed to be random variables. After data being measured, it was possible to calculate the posterior probability distributions conditioned by the data from the prior probability distributions. The technique used for the recursive parameters identification and described in these works was based on actualization of a root matrix decomposition of the positive definitive symmetric matrices. This resulted in the excellent numerical properties, so there was no risk of instability due to the loss of the positive definiteness. The Bayesian approach to the algorithm development enabled to solve the problem of a selection of the initial conditions and to use forgetting factors. Kárný M. implemented an exponential forgetting in the algorithms [47] and Kulhavý R. described and showed the advantages of a directional forgetting [50]. Moreover, Kulhavý R. formulated identification of time varying systems in independence to a model of the parameter development. The weak point of these algorithms was the difficulty to implement them in a fixed-point arithmetic; therefore, the calculations were to be held in a floating-point arithmetic [47-48, 50, 81].

Kadlec J. tried to solve the above mentioned problem in the work [40], where the author developed the algorithm of probability identification for a model of the vocal tract. Both the model parameters and the model order could be time varying. The algorithm had a lattice structure and, therefore, could benefit from the parallel implementation. Considering a recursive actualization of the order probability distribution, it was possible to decide about the number of parameters describing the vocal tract. Moreover, the variable normalization was proposed, which allowed implementing the algorithm in a fixed-point arithmetic and using then fast microprocessors [40].

In work [41] Kadlec J. continued the investigation and tried to find a method of the recursive probability identification of a regression model. This method had to allow the variable normalization using the time varying ranges from -1 to 1 in such a way that the algorithms could be easily implemented in a fixed-point arithmetic. The method had similar numerical properties as the square root algorithms of the probability identification implemented in floating–point arithmetic [41].

Besides, Kadlec J., McWhirter J. G. and Walke R. L. in 1995 proposed the normalized Givens rotation algorithm for the RLS processing and showed an important consequence of the normalization as for the algorithms being implemented in fixed point arithmetic [73]. This fact allows performing a design of a simpler application on the specific integrated circuits for the adaptive filtering and beamforming.

Zhu Li and Chao Li perform a comparative study of the LMS and RLS algorithms [56] in terms of a convergence rate in the system identification applications. Another comparative analysis is presented in [89].

There is also an interesting work by Gaensler T. and Bensty J. giving insights into the RLS algorithm and discussing the fast versions of the RLS algorithm [7].

The fast versions of the RLS algorithms, their modifications, the methods for increasing the throughput, the precision analysis are also described in works [3, 22-23, 55, 75, 79].

Due to the fact that the algorithms allow the parallel pipelined implementation, there are a lot of works dedicated to this topic.

Shanbhag N. R. and et al developed the pipelined adaptive digital filters, which were suitable for the low-power, low-area and higher-speed applications [94]. In their work the authors described the pipelined adaptive lattice filter architecture, the relaxed look-ahead pipelined LMS adaptive filters and quantizers, the pipelined adaptive differential vector quantizer architecture, the pipelined Kalman filter architecture and different applications [94].

The finite-precision error analysis of the QRD-RLS and STAR-RLS adaptive filters was made by Raghunath K. J. [88]. The author supposed that the QRD RLS adaptive filtering algorithm was suitable for the Very-Large-Scale-Integration (VLSI) implementation due to its numerical properties. Thus, the researcher developed a new fine-grain pipelinable STAR-RLS algorithm suitable for the high-speed applications. It was claimed that the algorithm could be implemented with as few as 8 bits for the fractional part, depending on the filter size and the forgetting factor used [88].

Matsubara K. and et al in the same year developed the pipelined LMS adaptive digital filter based on the look-ahead delayed LMS algorithm and proposed an efficient architecture for the hardware implementations [72].

Another works devoted to the pipelined adaptive filters are a paper by Douglas S. C. and et al discussing a pipelined architecture for the LMS adaptive FIR filter architecture without the adaptation delay [25], and a work by Nishikawa K. and et al describing the pipeline implementation of the gradient-type adaptive filters [78].

A new approach to the householder transformation (HT) for the RLS filters was described by Liu K. J. R. and et al. This approach made the HT suitable for the VLSI implementation and applicable to the real-time signal processing applications [65]. In their further work the authors modified the HT algorithms in a way that it became possible to perform a two-level pipelined implementation of the systolic block householder transformations at both the vector and the word levels [66].

Djigan V. I. describes a family of the sliding window RLS adaptive filtering algorithms with the regularization of the adaptive filter correlation matrix fitted for the parallel computations. The author claims that this approach can be used in all traditional applications of the adaptive filters [24].

There are also a number of books with a comprehensive description of the current situation in the adaptive filtering and with different examples of the algorithm applications. Among them it is worth mentioning [21, 30, 36]. In [36] the author examines both the mathematical theory behind various linear adaptive filters with the FIR and the elements of the supervised neural networks.

Due to the fact that the algorithms proposed in the thesis will be implemented on the HW platform with the FPGA programmable logic, it is worth mentioning several publications devoted to this field and areas connected.

Bondalapati K. and Prasanna V. in [9] discuss the advantages of the reconfigurable computing systems and the methodologies used for developing the configurable computing models.

Kung S. Y. provides a general overview of the VLSI array processors and a unified treatment for the algorithm, architecture and their application [51].

Pirsch P. also investigates this field and provides a very detailed description of basic architectures for the VLSI implementations of the Digital Signal Processing (DSP) tasks including a description of the parallel processing and pipelining, the applications of the specific array processors and the programmable digital signal processors [83].

Lightbody G. has several works devoted to the VLSI and Intellectual Property (IP) cores development. In the early work the author describes the VLSI architectures in connection to the RLS adaptive filtering algorithms [57]. In the work from 2003 in cooperation with Woods R. and Walke R. the researcher develops a parameterizable generic architecture for the RLS filtering in the form of a hardware description language (HDL) [58].

The synthesis and optimization of the DSP algorithms are covered in a work by Constantinides G. and et al, where the authors focused on the digital design and architectural synthesis, the signal scaling, the methodologies of the DSP design, the precision optimization, the importance of the scheduling, the allocation and binding problems [19]. The authors also described the trade-off between the numerical accuracy for the area and power-consumption advantages.

A very detailed description and analysis of the FPGAs can be found in works by Goslin G., Meyer-Baese U., Wolf W. and Woods R. and et al. [31, 74, 105-106].

Other works devoted to the FPGA architecture for the RLS algorithms can be found in [2, 10, 46, 101].

It should be noted as well that to be able to use the discussed algorithms for the applications running on the small platforms, the problem of the power consumption should be solved.

A general description of a low power digital design can be found in work [13] by Chandrakasan A. and Brodersen R. and in work [14] by Chen C. S. and et al. Besides, there are several works devoted to a floating-point design for the low-power

signal processing applications. Among them are works by Pillai R. V. K. and et al and Fang F. and et al. [28, 82].

As it was stated before, the RLS algorithms are very often used for the noise/echo cancellation applications. The most popular applications are those in the area of telecommunication and mobile speech recognition application. In this respect the works [8, 49, 77, 93] are worth of attention.

Other works in this field solve the problem of a variable forgetting factor [33, 99], the simplified versions of the RLS algorithms for the acoustic echo cancellation [110], and provide a comparison of the performance of the LMS, NLMS, RLS and QR-RLS in the noise suppression [69].

A very interesting work in the field of using the RLS algorithms for the noise cancellation is the one by Iglesias M. E. [39], who describes a noise reduction technique based on the QRD RLS algorithm and the ways of its implementation on the FPGA-based platform. The researcher performs a simulation in MATLAB and in the FPGA and discusses the obtained results [39]. However, in this case there is no parallel architecture being used.

As far as the ultrasound technology and RLS algorithms concern, only one more or less related article was found [1]. In their work the authors try to use a new method based on the RLS adaptive filtering to eliminate the effect of the blurring of the tissue reflectivity, which deteriorates the biomedical ultrasound image quality. The experiments proved that due to the RLS algorithms it is possible to improve the contrast and resolution of the image and the algorithm itself can be considered reliable. The authors also managed to reduce the dimensionality, which led to the computational complexity decrease [1].

From the above description it follows that there is a lot of publications devoted to different areas of the field under consideration. However, the major interest for the thesis is presented by the following works:

- Works [40-41] by Kadlec J. – the work about the probability identification of an autoregression model with an unknown order with the help of lattice structures [40] and the work about the probability identification of a regression model in a fixed point arithmetic [41].
- Work [76] by Moonen M. about the adaptive signal processing, where the author performs different adaptive algorithms based on the RLS and LMS and considers their complexity, convergence and stability [76]. The author shows how these algorithms can be implemented for the parallel processing with the help of Signal-Flow Graph (SFG) diagrams and how the complexity or a number of operations per iteration can be decreased. This is very important for the fast identification and decrease of the power consumption. Among the algorithms described in the book are LMS, RLS NLMS, QRD-RLS, square-root free QRD, RLS with the inverse updating, fast transversal filters, lattice algorithms, QRD least squares lattice, fast QR algorithms. All algorithms are implemented on the example of FIR filters, where the exponential forgetting is considered [76].

- Work [81] by Peterka V., where a Bayesian approach to system identification and hypothesis testing are presented.
- Work [43] by Kadlec J. and Likhonina R. about the adaptive RLS algorithm implementations with a custom arithmetic and work [60] by Kadlec J. and Likhonina R. about the noise cancellation using the QRD RLS algorithms. Both works are related to the project SILENSE. The work [43] describes the algorithms created by Kadlec J. and discusses the results obtained from the MATLAB simulation, while the work [60] describes the simulation results using the noise cancellation technique.

Challenges, Goals and Contributions

It is worth mentioning several challenges, which exist in the research area, and underlining the contributions of the thesis.

Firstly, though literature analysis clearly shows that the field of investigation is well studied and profoundly described in many works, and that the existing algorithms function very efficiently on large computers; however, there is still a problem to implement them on small area chips. The microprocessors have usually a small memory footprint. Processing a large amount of data, which is often the case in acoustic signal processing, can cause slow performance.

Secondly, this particular work is performed within the project focused on ultrasound technology. It can be noticed that there is a gap in the research area. Only one more or less related work, which is dealing with ultrasonic diagnostics and improvement of diagnostics with the help of the RLS algorithm, was found [1]. Still no work was found, which would describe how to use the RLS algorithms for hand detection applications based on ultrasound.

Thirdly, there is a large amount of scientific investigations focusing on noise cancellation techniques using the RLS algorithms, e.g. [8, 49, 77, 93, 110]. However, they mainly concentrate on telecommunication and mobile speech recognition applications, which are not the case for the present research, where the algorithms are supposed to pre-process incoming data in a way to remove undesired ultrasound responses from the target signal, subject to use for hand detection.

Last, but not least is the fact that the algorithms, which constitute the basis of the thesis and will serve as a reference model for further development, were already proposed in [40-41, 47-48, 50, 81]. They were supplemented with estimation of the order of a regression model, which is based on recursive Bayesian hypothesis testing. The algorithms were successfully tested for RLS Lattice in an application for speech coding [40-41]. However, hand detection applications based on ultrasound technology have their specific features. In this context hypothesis testing is supposed to be applied in a different way. As far as the signals can come to microphones at different angles (not necessarily perpendicular) and with different delay, it seems to be more appropriate and important to identify the structure of a regression model and to choose a particular identification model, which corresponds better to a real-time situation, rather than to estimate only the order. Such kind of a solution in the field under consideration was not found in literature.

To summarize the last four points, it would mean that there is a strong need within the project to propose such kind of algorithms, which will deal with ultrasound signals, efficiently remove undesired responses from the target signal (noise reduction), ensure fast execution and processing of a large amount of data in real time, and guarantee high reliability and low power consumption on small platforms.

This and all previously mentioned challenges and considerations specify **the main goals** of the thesis, which can be formulated as follows:

1. to develop a numerically robust adaptive signal processing algorithm of recursive identification of regression models for ultrasound signals, performing noise cancellation and measuring hand distance from the device in real time.

It is supposed that there will be several models, which will correspond to different situations, e.g. whether a hand is present or not in front of the device. Using recursive hypothesis testing and calculated probabilities, it will be possible to decide, which model suits better for incoming data. The final goal is to use recursive Bayesian hypothesis testing to improve the functionality of an ultrasound hand detection application by reducing undesired responses (noise cancellation) and measuring hand distance from the device. The latter will be possible due to the special nature of the input signal (chirps). Though such kind of a signal is challenging as far as it refers to weakly exciting types and, thus, it requires numerically robust computation; however, it enables to compute the distance between the hand and the device as soon as the response from the hand, i.e. the exact moment the hand appears, is known.

After being supplemented with recursive probability estimation, the algorithm will come through verification process. For these purposes data corresponding to several regression models of different orders will be modelled and fed to the algorithm. At first the algorithm will be tested and verified in MATLAB R2019b [70] in double precision arithmetic. When the results are satisfactory, i.e. the algorithm is proved to identify correctly the most appropriate regression model and to estimate its parameters, then, it will be tested in floating-point representation.

After the algorithm pass the verification successfully both in double precision in MATLAB R2019b and in C code with single precision in MATLAB R2019b, real data from an ultrasound microphone are supposed to be used to verify the validity of the proposed models and correct performance of the algorithm.

It is also worth comparing the performance of different algorithms in terms of computational time, memory usage and other metrics, and, thus, to prove that a chosen algorithm for the present work is more suitable for the implementation on the HW platform.

Overall, the innovation shall lead to a newly improved, optimized algorithm with good numerical properties, capabilities of identification of regression models and distance computation based on incoming data. The algorithm, moreover, shall ensure low power consumption as well as sufficiently accurate performance in real time.

2. to implement the algorithm on an embedded hardware platform, potentially used for applications for hand detection with data processing from a microphone.

It should not be forgotten that the existing algorithm [40-41] serving as a reference model for testing the correct performance of the C coded implementation was already implemented in MATLAB R2019b and the C simulation environment. However, there is still a need to implement it for systolic, pipelined System on Chip (SoC) IP core (28 nm, 20 nm, possibly 16 nm); to be more specific, on the integrated circuit Xilinx Zynq Ultrascale+ with the multi-core processor ARM Cortex A53 and the FPGA programmable logic part [34-35].

After the new algorithm is developed and specified and its functionality is verified using MATLAB, the next step is to convert and to map it on the Xilinx Zynq Ultrascale+ device. It will be necessary to test how the algorithm performs on the processor; therefore, there will be experiments performed in a way that data corresponding this or that regression model will come to Xilinx Zynq Ultrascale+ from a flash memory card or from a computer. On the basis of the mapped algorithm, Xilinx Zynq Ultrascale+ will identify the structure of the regression models and choose the most appropriate one. In the end the information about power consumption, time of calculations and other characteristics can be obtained and a conclusion about the algorithm performance on small platforms can be made**.**

As far as the algorithm has high computational complexity, it is very probable that such kind of SW implementation will not be fast enough for real-time processing. Therefore, the next step will be to parallelize the computation process in a way that each core of the quad-core ARM Cortex A53 processor will be busy with computation of a certain part of the algorithm at each time step. In this case it will be possible to reach efficiency of the computation process and to decrease the computational time.

The parallel version of the HW implementation will be tested and verified with the golden model received from MATLAB R2019b. If verification tests are successfully fulfilled and if the computational time corresponds to the real-time processing, then the SW implementation will be considered successful. The prototype device in this case will represent an FPGA-based HW platform, where FPGA programmable logic will be responsible for providing data from an ultrasound microphone and the ARM part of the FPGA-based HW platform will compute the algorithm on four cores of ARM Cortex A53 and provide identification results: the presence or the absence of the hand in front of the device.

However, if the computation of the algorithm is slow and does not correspond to real-time processing, further modification will be needed. In this case the accelerators in the FPGA part of the device will be used and the algorithm will be mapped on the accelerators.

The SW implementation using four cores of ARM Cortex A53 processor is supposed to be done both in double and single precision arithmetic, while the HW implementation using FPGA accelerators is supposed to be fulfilled only in single precision arithmetic due to limited computational resources, which can be mapped to the programmable logic. The FPGA implementation of the algorithm is supposed to be performed using 8xSIMD FP03x8 accelerators designed in ÚTIA [42].

This new solution area promises to have a great potential for touchless device control using an operator's hand detection based on the digital processing of responses of ultrasound signals.

3. to apply the developed algorithm for tracking of hand movement-based gestures (possibly).

The final goal within the project SILENSE, which supports the present research, is to create a functional prototype of a gesture recognition device, which can be used as a Human Machine Interface (HMI) in the automotive industry or smart home/building domain. This goal is out of the scope of the thesis, because the algorithm presented in this work are only a part of the hand detection application and aim at eliminating undesired ultrasound responses and cleaning the target signal for further processing. However, if succeeded it can be good evidence and a spectacular example of a practical value of the algorithm.

To summarize **the main steps for achieving the above mentioned goals**, the following steps should be mentioned:

1. to achieve goal 1:
   a. implementation of a hypothesis testing algorithm for identification of regression models (within considered context) based on a Bayesian approach with recursive estimation,
   b. incorporation of hypothesis testing into existing algorithms,
   c. pipelining and parallelizing the chosen algorithm,
   d. validation experiments both with modelled and real data from an ultrasound microphone:
      i. verification in MATLAB R2019b in double precision,
      ii. verification in MATLAB R2019b using C code in single precision,
      iii. verification of a parallelized version of the algorithm in MATLAB R2019b using MATLAB Parallel Computing Toolbox.
   e. comparing the performance of different algorithms.
2. to achieve goal 2:
   a. converting the algorithm to the Xilinx Zynq Ultrascale+ device (computation is performed on a single core of the multi-core ARM Cortex A53 processor),
   b. pipelining and parallelizing the algorithm for computing on four cores of the ARM Cortex A53 processor,
   c. verification of the algorithm mapped on Zynq Ultrascale+ on the basis of the golden model from MATLAB R2019b to prove its functionality and to test it in terms of power consumption, time of computation, and other metrics,
   d. implementation of the algorithm on the FPGA programmable logic part of Xilinx Zynq Ultrascale+,
   e. verification of the algorithm mapped to FPGA accelerators on the basis of the golden model from MATLAB R2019b.
3. to achieve goal 3:
   a. application of the developed algorithm for hand tracking and identification of simple gestures possible due to the measurement of the hand distance from the device, i.e. creating a functional prototype, verification and final version (possibly).

To summarize the **expected contributions**, the following should be mentioned:

1. A newly designed structure of regressions models, which is chosen in a way enabling to compare the models and to obtain information for the predetermined goals of the specified application, i.e. a) noise cancellation, b) distance computation between the hand and the device.
2. Application of recursive Bayesian identification to the defined problem, i.e. incorporation of hypothesis testing to the algorithm so that on the basis of computed probabilities it is possible to compare the structures of regression models and to make a decision between two use-cases ("there is a hand in front of the device" vs "there is no hand in front of the device") at a certain computation step. It results in determining the exact moment when there is a response from the hand and due to the specific form of the input signal (chirps) to measure the distance between the hand and the device.
3. Implementation of the algorithm: a) in the MATLAB R2019b environment, b) in parallel in the MATLAB R2019b environment using the Parallel Computing Toolbox, c) in C code, d) in parallel in C code on four threads.
4. Simulation close to the real situation, i.e. the models used for simulation in MATLAB R2019b will be retrospectively recalibrated in a way that the parameters from identification with real data will be used to create regression models for simulation purposes.
5. Algorithm implementation on an embedded HW platform, on the ARM Cortex A53 processor with four cores using real data from an ultrasound microphone and performing computation in real time.
6. Algorithm implementation in the FPGA programmable logic part of the Xilinx Zynq Ultrascale+ device using 8xSIMD FP03x8 accelerators to accelerate the computation process.

Structure of the Work

To conclude this introductory description, the structure of the present work is described. The first chapter is devoted to mathematical methods including description of RLS algorithms and FIR filters, the Bayesian approach to system identification problems, modification of existing algorithms and their optimization. It also presents techniques and SW tools needed for the implementation of the algorithm on the HW platform.

In the second chapter the algorithm implementation on a PC, in MATLAB R2019b is provided. Simulation results as well as experiments with real data are discussed and a comparison of two algorithms is performed.

The third chapter describes steps of algorithm optimization on a PC. It includes the pipelining and parallel processing technique as well as a discussion of parallelization of the algorithms using the Parallel Computing Toolbox in the MATLAB R2019b environment.

The algorithm implementation on the Trenz Electronic board is presented in Chapter 4, while the Conclusion section of the work provides a short summary of the work and analyzes the obtained results.

The remaining two parts of the work shortly describe potential applications of the algorithm and future prospects.

There are also appendices to Chapter 1, which describe the applied algorithm in more details.

CHAPTER 1

Mathematical Methods, Tools and Techniques

This part describes the RLS algorithms, the Bayesian approach to system identification and the incorporation of hypothesis testing in the chosen algorithm. The mathematical tools and methods presented in this chapter serve as a basis for further implementation of the chosen algorithm first in the MATLAB R2019b environment [70] and then on the HW platform. More detailed information together with the derivation of equations presented in this chapter can be found in Appendices.

Moreover, some basic information about the FPGA-based platforms including the FPGA technology, tools and techniques used for programming on microprocessors and on the FPGA chips are provided in the following sections of the chapter.

Finally, in the last section the main outputs of this stage of the research and the related publications are mentioned.

1.1. Bayesian Approach to System Identification: Estimation of the Model Parameters

To continue the discussion about the development of the chosen RLS algorithm with probability estimation of a corresponding model, it is worth giving a brief insight into the Bayesian approach to system identification and the recursive least square estimation from the viewpoint of the probabilistic theory [81].

In contrast to the classical statistics, the Bayesian statistics observes the probability as the subjective experience of the uncertainty and unknown parameters being uncertain and random, described by a probability distribution [20, 81].

T. Bayes underlines three basic components, which constitute the Bayesian statistics: the prior distribution, the information in the data themselves and the posterior inference [20].

The prior distribution is represented by the previous knowledge available before testing. It can be, for example, in the form of a normal distribution. Its variance defines the level of uncertainty about the value of the parameter under consideration. It is obvious that the larger variance the more uncertainty the value of the parameter has. The prior variance in this case is defined as the precision, which is the inverse of the variance.

The smaller the prior variance, the more confidence there is that the prior mean reflects the mean [20].

The second component, the information in the data, presents what is observed. It is defined by the likelihood function of the data given the parameters [20].

The posterior inference is based on combining the prior distribution and the observed evidence via the Bayes' theorem, which is a fundamental theorem in the Bayesian statistics. As the result, a so-called posterior distribution is obtained, which reflects the knowledge updated on the basis of new data [20].

The Bayes' theorem can be written as follows [20]:

$$\boldsymbol{posterior} = \frac{\boldsymbol{prior\ x\ likelikhood}}{\boldsymbol{marginal\ likelihood}},$$

or in the form of the equation for parameter estimation [81]:

$$\boldsymbol{p}(\boldsymbol{\theta}|\boldsymbol{y}) = \frac{\boldsymbol{p}(\boldsymbol{\theta}) \cdot \boldsymbol{p}(\boldsymbol{y}|\boldsymbol{\theta})}{\boldsymbol{p}(\boldsymbol{y})}, \qquad (\mathbf{1.1})$$

where $\boldsymbol{p}(\cdot)$ denotes the probability, $\boldsymbol{y}$ is the output, $\boldsymbol{\theta}$ are parameters.

The equation can be simplified by dropping $\boldsymbol{p}(\boldsymbol{y})$, which represents a normalizing constant for $\boldsymbol{p}(\boldsymbol{\theta}|\boldsymbol{y})$ sums to 1. In this case it is obtained [81]:

$$\boldsymbol{p}(\boldsymbol{\theta}|\boldsymbol{y}) \sim \boldsymbol{p}(\boldsymbol{\theta}) \cdot \boldsymbol{p}(\boldsymbol{y}|\boldsymbol{\theta}), \qquad (\mathbf{1.2})$$

where $\sim$ means proportionally.

Summarizing the points mentioned above, the Bayes' theorem states that the updated knowledge about the parameters of the interest given the current data depends on the prior knowledge about the parameters weighted by the current information given those parameters [20].

The probabilistic approach gives a path between the probabilistic theory and the least square error estimation. It allows extending the estimation task by the hypotheses probability estimation [84].

The Bayesian approach provides a different view of hypothesis testing as far as it uses the background knowledge for the analyses [20]. It is emphasized in [81] that using the Bayesian approach for hypothesis testing the uncertainty of the hypotheses should be described by a probability distribution on the set of hypotheses. These hypotheses are priori considered as possibly true. Then, the solution is presented in the form of the posterior probability distribution. This probability distribution is defined on the set of hypotheses conditional on the input-output data observed on the system [81].

Obviously, the probability of the hypotheses $\boldsymbol{H_n}$ will be equal to the probability that a true system model $\boldsymbol{M_{true}}$ belongs to the class $\boldsymbol{C_n}$, which represents the subset of the models.

In this case the probability distribution is determined as follows [81]:

$$\boldsymbol{p(H_n|D(t)) = Pr[M_{true} \in C_n|D(t)], \qquad n = 1, 2, \dots, N} \tag{1.3}$$

where $\boldsymbol{D}$ is a set of data.

Note also that the notation $\boldsymbol{D(t)}$ means all data up to time $\boldsymbol{t}$.

Usually, when there is no need explicitly to choose the model structure, the calculation can be made simultaneously with all model structures, setting weights for each structure. The weights determine the probability of the individual models [81].

To determine the posterior probability distribution, the prior probability distribution on the entire set of all models has to be defined. The prior probability is usually assigned to each of the hypotheses $\boldsymbol{p(H_n)}$ (for $\boldsymbol{n = 1, 2, \dots, N}$) and the prior probability distribution is determined on the set of the possible parameter values within each of the hypotheses $\boldsymbol{p(\theta_n|H_n)}$ (for $\boldsymbol{n = 1, 2, \dots, N}$). Note that $\boldsymbol{p(H_n)}$ represents the prior uncertainty about the validity of the hypotheses before the knowledge is updated with the new data, while $\boldsymbol{p(\theta_n|H_n)}$ is the prior uncertainty about the values of an unknown parameter $\boldsymbol{\theta_n}$ given the hypothesis $\boldsymbol{H_n}$ was true. Thus, the product of these two probabilities $\boldsymbol{p(\theta_n|H_n) \cdot p(H_n)}$ determines the prior probability for every subset of the models within the corresponding class [81].

According to [81], the assumption is

$$\boldsymbol{\mathbf{Pr}[M \in C_n \cap C_m] = 0 \; \forall n, \; m \neq n} \tag{1.4}$$

where $\boldsymbol{M}$ is a system model, $\boldsymbol{m = 1, 2, \dots, N}$.

This formula shows that the model classes can overlap with a zero probability, as a subset of the models common for two or more classes may obtain a nonzero prior probability and posterior probability only through one of the hypotheses [81].

It should be noted that while the choice of the prior $\boldsymbol{p(H_n)}$ is obvious assuming that all hypotheses are equally likely, the choice of the prior $\boldsymbol{p(\theta_n|H_n)}$ is quite difficult in the case of hypothesis testing [81].

Further, given $\boldsymbol{p(H_n|D(t_s))}$ and $\boldsymbol{p(\theta_n|H_n, D(t_s))}$ for some $\boldsymbol{t_s \geq 0}$ and $\boldsymbol{n = 1, 2, \dots, N}$, it is possible to calculate $\boldsymbol{p(H_n|D(t))}$ for $\boldsymbol{t > t_s}$. The assumption of the natural conditions of control is made [81]:

$$\boldsymbol{p(H_n|D(t)) = \frac{p\left(D_{t_S+1}(t)\middle|D(t_S), H_n\right) \cdot p(H_n|D(t_S))}{\sum_{m=1}^{N} p\left(D_{t_S+1}(t)\middle|D(t_S), H_m\right) \cdot p(H_m|D(t_S))},} \tag{1.5}$$

where $\boldsymbol{D_{t_S+1}(t)}$ are data observed at the unknown system output from time $\boldsymbol{t_S+1}$ to $\boldsymbol{t}$, variables $\boldsymbol{D(t)}$ and $\boldsymbol{D(t_S)}$ are data observed up to and including the time-index $\boldsymbol{t}$ and $\boldsymbol{t_S}$ respectively, hypothesis $\boldsymbol{H_n}$ is an unknown identification model with a certain structure and an order, term $\boldsymbol{p\left(D_{t_S+1}(t)\middle|D(t_S),H_n\right)}$ is a probabilistic description of the modelled system with the identification model given by hypothesis $\boldsymbol{H_n}$.

Then applying the chain rule to the above equation, the first factor in the numerator can be written in the following way [81]:

$$\boldsymbol{p\left(D_{t_S+1}(t)\middle|D(t_S),H_n\right)=\prod_{\tau=t_S+1}^{t} p\left(y_\tau\middle|u_\tau,D(\tau-1),H_n\right)\cdot p(u_\tau|D(\tau-1),H_n)}, \quad (\mathbf{1.6})$$

where $\boldsymbol{\tau}$ is a certain time interval, $\boldsymbol{\tau=t_0+1,\ t_0+2,\dots,t}$, $\boldsymbol{u_\tau}$ is the input in time $\boldsymbol{\tau}$, $\boldsymbol{y_\tau}$ is the output in time $\boldsymbol{\tau}$, $\boldsymbol{D(\tau-1)}$ are data observed up to and including time $\boldsymbol{\tau-1}$.

Moreover, if the natural control conditions are considered, then the following simplifications can be made [81]:

$$\boldsymbol{p_n(y_\tau|u_\tau,D(\tau-1))=p(y_\tau|u_\tau,D(\tau-1),H_n)}. \quad (\mathbf{1.7})$$

And finally the formula for computing the hypothesis probability takes the form [81]:

$$\boldsymbol{p(H_n|D(t))=\frac{\prod_{\tau=t_s+1}^{t} p_n(y_\tau|u_\tau,D(\tau-1))\cdot p(H_n|D(t_S))}{\sum_{m=1}^{N}\prod_{\tau=t_s+1}^{t} p_m(y_\tau|u_\tau,D(\tau-1))\cdot p(H_m|D(t_S))}}. \quad (\mathbf{1.8})$$

Further, under the assumption of the natural conditions of control it can be written [81]:

$$\boldsymbol{p_n(y_\tau|u_\tau,D(\tau-1))=\int p_n(y_\tau|u_\tau,D(\tau-1),\theta_n)\cdot p_n(\theta_n|D(\tau-1))d\theta_n}. \quad (\mathbf{1.9})$$

There the simplified notation is used [81]:

$$\boldsymbol{p_n(\theta_n|D(\tau-1))=p_n(\theta_n|D(\tau-1),H_n)}. \quad (\mathbf{1.10})$$

It is obvious that in order to test, which hypothesis is more likely to be true, it is necessary to estimate the probabilities of the models and on the basis of the probabilities to make a decision. The equation for computing the hypothesis probability leads to the fact that for the posterior probability ratio for any two of the $\boldsymbol{N}$ hypotheses it is valid [81]:

$$\boldsymbol{\frac{p(H_n|D(t))}{p(H_m|D(t))}=\prod_{\tau=t_s+1}^{t}\frac{p_n(y_\tau|u_\tau,D(\tau-1))\cdot p(H_n|D(t_S))}{p_m(y_\tau|u_\tau,D(\tau-1))\cdot p(H_m|D(t_S))}}. \quad (\mathbf{1.11})$$

The left-hand side is a so-called Bayes factor. By this factor our prior beliefs about the hypotheses are updated to yield the posterior beliefs, about which hypothesis is more likely [20, 81].

The entire probability distribution on the set of $\boldsymbol{N}$ hypotheses is uniquely determined by any $\boldsymbol{N-1}$ finite ratios for $\boldsymbol{n \neq m}$ and by the following condition [81]:

$$\sum_{k=1}^{N} \boldsymbol{p(H_k|D(t))} = \mathbf{1}. \qquad (\mathbf{1.12})$$

Using the likelihood function the posterior probability ratio equation can be rewritten as follows [81]:

$$\frac{\boldsymbol{p(H_n|D(t))}}{\boldsymbol{p(H_m|D(t))}} = \frac{\int \boldsymbol{L_{n(t)}(\theta_n)d\theta_n}}{\int \boldsymbol{L_{m(t)}(\theta_m)d\theta_m}} \cdot \frac{\boldsymbol{\epsilon_n}}{\boldsymbol{\epsilon_m}} \cdot \frac{\boldsymbol{p(H_n)}}{\boldsymbol{p(H_m)}}, \qquad (\mathbf{1.13})$$

where $\frac{\boldsymbol{\epsilon_n}}{\boldsymbol{\epsilon_m}} \cdot \frac{\boldsymbol{p_n(\theta_n)}}{\boldsymbol{p_m(\theta_m)}}$ is a prior distribution.

From the above formula it can be concluded that by the choice of a prior distribution the posterior probability of any of compared hypotheses can be heavily influenced. However, it is not so determining. With growing $\boldsymbol{t}$ the ratio of integrals over the likelihood diverges very fast, if the hypothesis $\boldsymbol{H_n}$ is true. It is the reason why the ratio of integrals begins dominating any reasonably chosen ration of $\frac{\boldsymbol{\epsilon_n}}{\boldsymbol{\epsilon_m}} \cdot \frac{\boldsymbol{p_n(\theta_n)}}{\boldsymbol{p_m(\theta_m)}}$. It means that with growing $\boldsymbol{t}$ the posterior probability of the true hypothesis is converging to one in any case. However, this property holds only for the large data sizes. For small or medium data sizes the choice of prior distributions should be made carefully [81].

The recursion for the real-time updating of the probability distribution on the hypotheses can be written from the general formula for computing the hypothesis probability for $\boldsymbol{t_s = t-1}$ [81]:

$$\boldsymbol{p(H_n|D(t))} = \frac{\boldsymbol{p_n\big(y_t|u_t, D(t-1)\big)}}{\boldsymbol{p\big(y_t|u_t, D(t-1)\big)}} \cdot \boldsymbol{p\big(H_n|D(t-1)\big)}, \qquad (\mathbf{1.14})$$

where $\boldsymbol{p_n\big(y_t|u_t, D(t-1)\big)}$ is a conditional probability distribution within the *n-th* hypotheses determined according to the formula (1.9). The denominator here is the ordinate of the overall predictive probability density from the previous step for the newly observed output [81]. It is a normalizing constant.

The last thing, which should be mentioned in this connection, is that the Bayes factors condition on the observed data. This fact gives benefits in increasing the flexibility in data collection and in the robustness of the inferences [20].

1.2. Hypothesis Testing about the Order of a Regression Model

Generally, in the Bayesian RLS regression model approaches including works [40-41, 47-48, 50, 81], several assumptions for performing hypothesis testing in regards to the least square computation are proposed. First of all, it was assumed that a stochastic system can be described by a parametrized system in the form of a conditional probability density function:

$$\boldsymbol{p(y_t|D(t-1);u_t,\theta,\omega_t) = k\cdot\omega_t^{\frac{1}{2}}\cdot\exp\left\{-\frac{\omega_t}{2}\cdot(y_t-\theta^T\cdot Z[N])^2\right\},} \qquad (1.15)$$

where $\boldsymbol{\theta}$ is a vector of unknown regression parameters, it is a random variable of size $\boldsymbol{N}$; $\boldsymbol{Z[N]}$ is a data vector of size $\boldsymbol{N}$ consisting of the delayed output values and input data $\boldsymbol{u_t}$, which directly influence the output $\boldsymbol{y_t}$; $\boldsymbol{k}$ is a normalizing constant; $\boldsymbol{\omega_t}$ is an unknown degree of accuracy, it is a random variable, which can be defined as a reciprocal to the variance $\boldsymbol{\sigma_t^2}$ [40-41, 47-48, 50, 81]:

$$\boldsymbol{\omega_t = \frac{1}{\sigma_t^2}.} \qquad (1.16)$$

Note that the upper index $^{\boldsymbol{T}}$ means the transpose of a vector.

Note also that the conditional probability density (1.15) corresponds to the description of a system by a regression model in the form [40-41, 47-48, 50, 81]:

$$\boldsymbol{y_t = \theta^T\cdot Z[N] + e_t,} \qquad (1.17)$$

where $\boldsymbol{e_t}$ is a sequence of random variables, which are mutually independent on the past measured data and on the last input and which have a normal distribution with a zero mean value and an unknown variance. This unknown variance is defined through the degree of accuracy $\boldsymbol{\omega_t}$ in equation (1.15) [40-41, 47-48, 50, 81].

The second important assumption is that the prior conditional probability density of the parameters $\boldsymbol{\theta}$, $\boldsymbol{\omega_t}$ for time $\boldsymbol{t = T, T+1}$, ... has the form of Gaussian-Wishart distribution [40-41, 47-48, 50, 81]:

$$\boldsymbol{p(\theta,\omega_t|D(t-1)) = k\cdot\omega_t^{\frac{\nu+N-2}{2}}\cdot\exp\left\{-\frac{\omega_t}{2}\cdot\begin{bmatrix}-\theta\\1\end{bmatrix}^T\cdot V_M[N+1]\cdot\begin{bmatrix}-\theta\\1\end{bmatrix}\right\},} \qquad (1.18)$$

where $\boldsymbol{V_M[N+1]}$ is a positive definitive symmetric matrix of size $\boldsymbol{N+1}$, $\boldsymbol{V_M[N+1] > 0}$; $\boldsymbol{\nu}$ is a real positive value, which represents the number of degrees of freedom in the Gaussian-Wishart distribution, $\boldsymbol{\nu > 0}$.

The important point is that the form of the conditional probability density of the parameters is replicated during the recursive update [40-41, 47-48, 50, 81]:

$$\boldsymbol{p\big(\theta\big|D(t)\big) = \frac{p(y_t|D(t-1);u_t,\theta)}{p(y_t|D(t-1);u_t)} \cdot p\big(\theta|D(t-1)\big).} \qquad (\mathbf{1.19})$$

It means that the update of the conditional probability density (1.15) can be fully described by the algebraic relations for the development of characteristics $\boldsymbol{V_M[N+1]}$, $\boldsymbol{\mathcal{V}}$. These characteristics constitute the sufficient statistics of the probability identification [40-41, 47-48, 50, 81].

$$\boldsymbol{V_M[N+1] = \begin{bmatrix} V_M[N] & V[N] \\ V^T[N] & \mathbf{v} \end{bmatrix},} \qquad (\mathbf{1.20})$$

where $\boldsymbol{V_M[N]}$ is a square symmetric positive definite matrix of size $\boldsymbol{NxN}$, $\boldsymbol{V[N]}$ is a column vector of size $\boldsymbol{N}$, $\boldsymbol{V^T[N]}$ is a transposed vector $\boldsymbol{V[N]}$ of size $\boldsymbol{N}$, $\mathbf{v}$ is a positive scalar.

Note that the lower notation $_{\boldsymbol{M}}$ under the letter means "matrix" to differentiate between a matrix $\boldsymbol{V_M[N]}$ and a vector $\boldsymbol{V[N]}$ in further discussion of the algorithm.

The update of the characteristics is as follows [40-41, 47-48, 50, 81]. To simplify the notation, instead of $\boldsymbol{(t|t)}$under the characteristics the upper line " $\bar{}$ " above the letter is used to show that the corresponding characteristics is after updating with $\boldsymbol{y_t}$. The index $\boldsymbol{(t|t-1)}$ is omitted under the characteristics, which means that the corresponding characteristics is before updating with $\boldsymbol{y_t}$ [40-41, 47-48, 50, 81]:

$$\boldsymbol{\bar{V}_M[N+1] = V_M[N+1] + Z[N+1] \cdot Z^T[N+1],} \qquad (\mathbf{1.21})$$

$$\boldsymbol{\bar{\mathcal{V}} = \mathcal{V} + 1,} \qquad (\mathbf{1.22})$$

where $\boldsymbol{Z[N+1]}$ is an extended data vector of size $\boldsymbol{N+1}$, i.e. a vector $\begin{bmatrix} \boldsymbol{Z[N]} \\ \boldsymbol{z_{N+1}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{Z[N]} \\ \boldsymbol{y_t} \end{bmatrix}$.

The a prior statistics $\boldsymbol{V_{0_M}}$ and $\boldsymbol{\mathcal{V}_0}$ are outputs from a priori probability density function $\boldsymbol{p(\theta|D(t_0)}$ and used for the start of the recursion.

The letter $\boldsymbol{N}$ gives the order of the system.

Another form of the statistics update is given in Appendix 1. This form has been used in the experimental part of the work. Besides, in Appendix 1 the algorithm used for programming is presented in details as it is used in a code. It is decided not to give it here due to its complexity and a great number of equations, which can be hard readable and which can complicate the understanding of the main point of the chapter.

The characteristics updates for further steps are performed with a forgetting factor. The most common forgetting techniques are exponential [47] and directional forgetting [50]. The simplest one from these two for the implementation is the exponential forgetting, which assumes the memory to be infinite. It gives small

weights to the old data and larger weights for the latest data points [76]. A weighting factor ranges from 0 to 1. Actually there is only one additional step in the exponential weighting RLS: the weighting of the covariance matrix. The benefit of this step is that it leads to the extremely simple RLS algorithms, which are stable, because both the old data and errors are wiped out by the exponential forgetting [76].

However, the exponential weighting can face a problem in case of ill-excited systems. If the input and output signals do not bring sufficient information, the previous information is gradually wiped out with the exponential forgetting. The solution is to discount the old data only when a new information is available, i.e. to make forgetting in the certain directions. This approach was offered by Kulhavý R. and was called the directional forgetting [50].

However, as far as the QRD RLS Lattice algorithm is supposed to be used for the noise cancellation for the hand detection applications and due to the fact that the latter algorithm does not use the directional forgetting, the exponential forgetting will be preferred in this work.

The characteristics updates using both the exponential and directional forgetting can be found in Appendix 1.

1.3. Types of the RLS Algorithms

This chapter gives a brief insight into types of the RLS algorithms to show their advantages and disadvantages. Finally, the reason of our choice of the QRD RLS Lattice algorithm used for hypothesis testing is given.

In [76], the adaptive filtering problems are introduced, the standard RLS algorithms are described and other RLS-based types of the algorithms, which have some specific, beneficial features, are derived.

The algorithms are often used in the applications, where a real-time processing is the requirement. In this case the signal processing device needs to be as fast as the sampling devices that produce new data in each time step. Thus, the new data should be used for re-computation and for the update of the previous information. The RLS algorithms do not perform parameter estimation and prediction error computation from the very beginning: only the data from the previous step are used for re-computation. It saves time and decreases the complexity [76].

The standard recursive least square algorithm is given by M. Moonen in the following form [76]:

$$\overline{\boldsymbol{\theta}}[\boldsymbol{N}] = \boldsymbol{\theta}[\boldsymbol{N}] + \boldsymbol{k} \cdot (\boldsymbol{z_{N+1}} - \boldsymbol{Z^T}[\boldsymbol{N}] \cdot \boldsymbol{\theta}[\boldsymbol{N}]) =$$

$$= \boldsymbol{\theta}[\boldsymbol{N}] + (\boldsymbol{V_M}[\boldsymbol{N}])^{-1} \cdot \boldsymbol{Z}[\boldsymbol{N}] \cdot (\boldsymbol{z_{N+1}} - \boldsymbol{Z^T}[\boldsymbol{N}] \cdot \boldsymbol{\theta}[\boldsymbol{N}]), \qquad (\mathbf{1.23})$$

where $(\boldsymbol{V_M}[\boldsymbol{N}])^{-1} = (\boldsymbol{U^T}[\boldsymbol{N}] \cdot \boldsymbol{U}[\boldsymbol{N}])^{-1}$ is an autocorrelation matrix of the filter input signal; $\boldsymbol{k} = (\boldsymbol{V_M}[\boldsymbol{N}])^{-1} \cdot \boldsymbol{Z}[\boldsymbol{N}]$ is a Kalman gain vector, which contributes to better performance of the algorithm and gives the direction, in which $\boldsymbol{\theta}[\boldsymbol{N}]$ should be modified [76].

From the above formula it is obvious that $\overline{\boldsymbol{\theta}}[\boldsymbol{N}]$ is computed from $\boldsymbol{\theta}[\boldsymbol{N}]$ and it uses only $\boldsymbol{O}(\boldsymbol{N^2})$ arithmetic operations [76].

However, the standard RLS algorithms are usually used only for the theoretical purposes, as far as they can be potentially unstable due to the numerical round-off errors. The problem consists in computation of a covariance matrix $\boldsymbol{V_M}[\boldsymbol{N}]$, which due to the numerical errors can lose its positive definiteness [76].

The stable variants of the RLS algorithms are based on the QR decomposition (QRD) of the matrix $\boldsymbol{U}[\boldsymbol{N}]$. The decomposition itself is made in the following form [76]:

$$\boldsymbol{U} = \boldsymbol{Q} \cdot \boldsymbol{R}, \qquad (\mathbf{1.24})$$

where $\boldsymbol{U}$ is a long matrix of size $\boldsymbol{LxN}$, where $\boldsymbol{L}$ comprises all measured data, $\boldsymbol{Q}$ is an orthogonal matrix of size $\boldsymbol{LxN}$, $\boldsymbol{R}$ is an upper triangular matrix of size $\boldsymbol{NxN}$ with positive diagonal elements.

The matrix $\boldsymbol{R}$ is also called as a Cholesky factor of the normal matrix of $\boldsymbol{U^T \cdot U}$ and it is valid that [76]

$$\boldsymbol{U^T \cdot U} = \boldsymbol{R^T \cdot R}. \qquad (\mathbf{1.25})$$

The compound matrix $\boldsymbol{U}[\boldsymbol{N+1}] = \begin{bmatrix} \boldsymbol{Z_1}[\boldsymbol{N}] & \dots & \boldsymbol{Z_L}[\boldsymbol{N}] \\ \boldsymbol{z_{N+1_{(1)}}} & \boldsymbol{z_{N+1_{(2)}}} \dots & \boldsymbol{z_{N+1_{(L)}}} \end{bmatrix}$ has the following QR decomposition [76]:

$$\left[\boldsymbol{U_{L-1}}[\boldsymbol{N+1}] \middle| \begin{matrix} \boldsymbol{Z_L}[\boldsymbol{N}] \\ \boldsymbol{z_{N+1_{(L)}}} \end{matrix}\right] = [\boldsymbol{Q}|\boldsymbol{q}] \cdot \begin{bmatrix} \boldsymbol{R} & \boldsymbol{z} \\ \boldsymbol{0} & \boldsymbol{\zeta} \end{bmatrix}, \qquad (\mathbf{1.26})$$

where matrix $\left[\boldsymbol{U_{L-1}}[\boldsymbol{N+1}] \middle| \begin{matrix} \boldsymbol{Z_L}[\boldsymbol{N}] \\ \boldsymbol{z_{N+1_{(L)}}} \end{matrix}\right]$ has size of $\boldsymbol{Lx(N+1)}$, $[\boldsymbol{Q}|\boldsymbol{q}]$ is an orthogonal matrix of size $\boldsymbol{Lx(N+1)}$, the last term is a triangular matrix of size $\boldsymbol{(N+1)x(N+1)}$ [76].

The QR decomposition can be made by using different methods, among which are the Givens, Householder and Gram-Schmidt methods.

The equation for a so-called square-root information RLS (SRI RLS), which avoids forming the product $\boldsymbol{U^T \cdot U}$, is given by M. Moonen in the following way [76]:

$$\begin{bmatrix} \overline{\boldsymbol{R}} & \overline{\boldsymbol{h}} \\ \boldsymbol{0} & * \end{bmatrix} \leftarrow \overline{\boldsymbol{Q}}^{\boldsymbol{T}} \cdot \begin{bmatrix} \boldsymbol{R} & \boldsymbol{h} \\ \boldsymbol{Z^T}[\boldsymbol{N}] & \boldsymbol{z_{N+1}} \end{bmatrix}. \qquad (\mathbf{1.27})$$

Taking into considerations all previous points, the equation for computing $\boldsymbol{\theta}$ looks as follows [76]:

$$\boldsymbol{\theta} = \boldsymbol{R^{-1}} \cdot \boldsymbol{h}, \qquad (\mathbf{1.28})$$

or for the next step of computation [76]:

$$\overline{\boldsymbol{\theta}} = \overline{\boldsymbol{R}}^{-1} \cdot \overline{\boldsymbol{h}}. \qquad (1.29)$$

Basically, the algorithm consists of two computational steps per time update. The first step is a triangular updating and the second one is a triangular back-substitution (optional). Due to the fact that both steps require $\boldsymbol{O}(\boldsymbol{N^2})$ computations, the algorithm itself has the computational complexity equal to $\boldsymbol{O}(\boldsymbol{N^2})$ [76].

One more benefit of this algorithm consists in the fact that it is possible to calculate the error signal without explicitly computing $\boldsymbol{\theta}$ at each time step. This is referred to as a residual extraction and it is used in many applications, where only the error computation is needed [76]. In this work this feature of the algorithm is considered to be of a great importance as far as only the computed errors will be used to eliminate the undesired responses from the environment, i.e. for the noise cancellation.

One more point to be also mentioned is a need to avoid the computation of the square-root, performed when making the QR decomposition with the Givens transformation. It is beneficial in the hardware implementations, because it prevents the computational bottleneck. To avoid the square-root computation, one can make a particular factorization of the R-matrix. This leads to the square-root free Givens rotations [76].

To summarize the advantages of the SRI RLS algorithms, its numerical stability and easiness to be applied on a sequential processor should be underlined. Besides, when there is only a need for a residual extraction and, therefore, the back-substitution can be skipped, the algorithm can be pipelined for the parallel processing.

However, without skipping the back-substitution and a weight vector computation, the algorithm can result in the data contraflow and cannot be easily pipelined. In this case the alternative algorithm can be used, which is based on the ‘inverse QRD updating and referred to the square-root covariance (SRC) algorithms by M. Moonen [76].

Though the square-root RLS algorithms have better numerical properties than the standard RLS algorithms, the computational complexity of $\boldsymbol{O}(\boldsymbol{N^2})$ per time update is quite large and there is a need to make it smaller in some applications.

There are a number of the fast RLS algorithms described in literature. Among them, for example, is a fast version of the QRD-updating based on a residual extraction algorithm called the QRD least squares lattice (QRD-LSL) algorithm. This algorithm is very useful when only a residual extraction is needed. In this case the back-substitution step is avoided and the algorithm can be easily pipelined. Because the QRD-LSL has only orthogonal transformations, it results in its good numerical properties and, therefore, in the long term stability [76].

Other fast algorithms based on the QRD-LSL include the Lattice algorithms without the orthogonal transformations. They are cheaper as far as their computational complexity is concerned, but could suffer from the numerical problems [76].

Another algorithm, which uses the orthogonal transformations, but has the same computational complexity as the QRD-LSL does, is a so-called "fast QR" algorithm [76].

As far as the computational complexity is concerned, the most efficient RLS Lattice algorithm requires ***24N*** operations per time update and can be pipelined for the increased rate of operations [76].

The above mentioned fast algorithms are good and suitable for the residual extraction problems. However, some applications require the weight computation. In this case the fast versions of the inverse updates based RLS algorithms can be applied. One of them is a so-called 'fast transversal filter' (FTF). The problem of this algorithm is that it cannot be used for continuous operations with very large amounts of data, because it suffers from the numerical instability. As far as the computational complexity is concerned, the FTF requires ***8N*** operations per time update, but it cannot be easily pipelined due to the data contraflow; thus, the computation cannot be accelerated by the parallel processing [76].

As far as in this work the noise cancellation is performed based on the residual extraction, the next chapter describes the incorporation of hypothesis testing into the QRD RLS Lattice algorithms, which was chosen due to its modular structure, reliability and computational speed.

1.4. QRD RLS Lattice Algorithm and Hypothesis Testing

Due to convenience in the implementation and the computational speed as it was shown in the previous section, the QRD RLS Lattice in the error feedback form is chosen for the implementation on the HW platform.

The RLS Lattice algorithm can be derived from the QRD algorithm, more detailed information of which can be found in [40-41, 47-48, 50, 81]. However, a short description of the most important equations is given in Appendix 2.

Due to its modular structure, the QRD RLS Lattice algorithm is suitable for the incorporation of the hypothesis estimation. In this case each module can perform the order update. It allows obtaining estimations of all orders during the computation process [84].

The hypothesis estimation is performed using (1.14). There are two stages of probability estimation. The first stage computes the order update. It means that the old probability estimates are updated by the new data during the first stage. In the second stage the normalization of the updated order estimates is performed. The second stage fulfills the forgetting of the hypothesis probability density function [84]. These stages can be incorporated into the QRD RLS Lattice algorithm.

However, to compute the probability estimates of the hypotheses, it is necessary to know $\boldsymbol{p(y_t|D(t-1), H_N)}$, which is calculated using (A.15):

$$\boldsymbol{p(y_t|D(t-1);\ H_N) = \pi^{-1/2} \cdot \frac{\Lambda^{((\vartheta-N)/2+1)}}{\overline{\Lambda}^{((\overline{\vartheta}-N)/2+1)}} \cdot \frac{|V[N]|^{1/2}}{|\overline{V}[N]|^{1/2}} \cdot \frac{\Gamma((\overline{\vartheta}-N)/2+1)}{\Gamma((\vartheta-N)/2+1)}}$$

The initial hypothesis probability density function is chosen in the form of the uniform distribution.

It is worth noting that the equation (A.15) is not used in its direct form in the algorithm, because it needs some approximations due to the numerical problems. The approximation of the equation and the update of probabilities are presented in Appendix 2.

The implementation of $\boldsymbol{N}$ QRD RLS Lattice estimations to test each hypothesis can be parallelized. For each estimation of an identification model by hypothesis testing, one QRD RLS Lattice instance can be computed in parallel. It can be implemented on the multi-core ARM A53 Cortex processor of the Xilinx Zynq Ultrascale+ device and on the FPGA accelerators [34-35].

1.5. FPGA, Tools and Techniques

The prototype of the device, where the algorithm will be running, will receive the signals from the ultrasound microphones. This part will be programmed in the FPGA programmable logic of the Xilinx Zynq Ultrascale+ device. Besides, as far as the algorithm itself will be also implemented on the FPGA programmable logic part of the hardware platform, it is not out of the point to give a definition of the FPGA and describe the tools and techniques used for the FPGA design development. The present section reveals these issues.

The term FPGA stands for a field-programmable gate array. It is a large-scale integrated circuit, which can be reprogrammed after being produced. The fact that the configuration of the device and its operation can be changed is obvious from the word combination “field-programmable”. The phrase “gate array”, on the other hand, explains what kind of internal architecture the device has to have the ability of being reprogrammed “in the field” [100].

The FPGA structure is shown in Figure 1.1.

The common FPGA architecture comprises an array of small blocks of programmable logic, reconfigurable or programmable interconnects, which wire the blocks together, and I/O blocks [6].

The logic blocks are usually called configurable logic blocks (CLB) or logic array block (LAB) depending on a vendor. Generally, a logic block includes a number of logical cells, consisting of up to 5-input lookup tables, a full adder cell, some multiplexers and a D-type flip-flop. The memory elements can be also represented by more complex blocks of memory instead of simple flip-flops. Besides, modern FPGA devices comprise some on-chip memory resources, such as, for example, Static Random Access

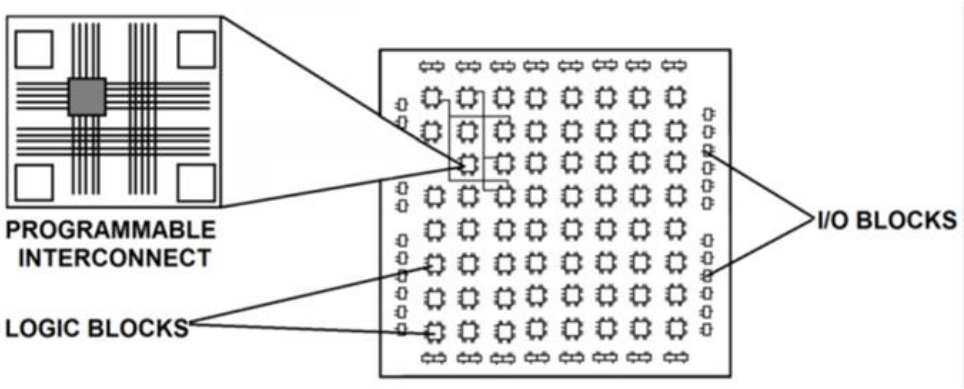

Figure 1.1: FPGA architecture [6]

Memory (SRAM). Thus, the local memory within each logic element can be combined with globally shared memory blocks [6].

The lookup table (LUT) is formed out of one or two single-bit registers and some logic elements such as clock enables and multiplexers. The LUTs represent small 8-bit RAMs, which can implement any combinational function of their inputs. The complexity of the function performed can be configured in the FPGA [6].

It is good to take in mind that routing is very crucial for the FPGA. It can influence performance of the whole design; therefore, a compromise between the programming flexibility and the area efficiency should be found when thinking over about the number of routing channels, which have to be placed in the FPGA. The routing channels can be made as high-speed long lines across the chips or as flexible local lines, connecting the separate blocks. The interconnections can run vertically and horizontally. There are also programmable switches in the FPGA architecture, which are shown as a gray square in Figure 1.1. The switches are implemented with MOSFET transistors and connect the orthogonal lines of interconnections according to the digital circuit designed in the FPGA. They can be SRAM-based, electrically erasable, or one-time-programmable [6].

In more complex FPGAs the logic blocks are combined with higher-level arithmetic and control structures, such as multiplexers and counters [6].

Moreover, because the FPGA-based applications have many different system-level interface requirements, the FPGAs usually contain configurable I/O blocks, which can be configured as TTL, CMOS, PCI and etc. The majority of FPGAs are also equipped with dedicated high-speed I/Os for clocks and global resets as well as PLLs and clock management schemes [6].

It should be also noted that besides basic features the modern FPGA families include also a higher level functionality. It results in reducing the required area and increasing speed in computation of the common embedded functions. The examples of such higher level functionality can be multipliers, generic DSP blocks, embedded processors, high speed I/O logic, embedded memories and etc. [6].

Nowadays the FPGAs also contain so-called IP cores such as processor cores, external memory controllers and etc. These cores have ASIC level performance and power

consumption, though they exist alongside the programmable fabric. This results in more free space for the application-specific logic [100].

The behavior of the FPGA is usually defined by using a hardware description language (HDL) such as VHDL or Verilog or by arranging blocks of pre-existing functions using block diagram-oriented design tools such as Vivado [104] from the Xilinx. The advantage of specifying the FPGA behavior with the HDL is in fact that it allows working with large structures, because it is possible to specify them numerically. On the other hand, schematic way of specifying the FPGA behavior is more user-friendly and better for visualization purposes [100].

However, the key for the success of the project development is to define, which parts of the design should be implemented on the hardware and which parts should be made in the form of the software on a traditional processor. The important problem in the early phases of the design is connected with an effective partitioning of the algorithm when applying the parallel programming techniques as well as with the mapping of the application to the hardware resources through the use of the automated compiler and hardware synthesis tools. These two points greatly influence the finally achieved system performance [100].

Speaking about the advantages of using the FPGA compared with the ASICs, one can emphasize its ability of being reconfigured after being manufactured, a partial re-configuration of a part of the design, the lower non-recurring engineering costs, the reduced development time, and a lower risk during the development. Besides, using the FPGA the designer can incrementally port and verify the algorithms previously prototyped in the software. Moreover, the FPGA can serve as a prototyping mechanism with further goal to implement the design on the ASIC-based platform [100].

It should be also noted that the use of the FPGA in the DSP domain is increasing. Talking about the DSP applications in respect to the FPGA, it is good to say that in many DSP applications the mixed processor design is preferred. In such a design the application's less-performance-critical components such as an operating system, a network stack and the user interface are placed on a host microprocessor such as the ARM, whereas more computationally intensive components are served by the dedicated hardware in the FPGA [100].

The main producers of the FPGA platforms are companies Xilinx and Altera. They provide the Windows and Linux design software, which is used to design, analyze, simulate and compile the designs [6].

During the present work the software tools from the Xilinx company are supposed to be used: Vivado Design Suite [104], Software Development Kit (SDK) [107] and Software Defined System on Chip (SDSoC) tools [108].

The Vivado Design Suite is a software tool produced by the Xilinx for synthesis and analysis of the HDL designs. It has additional features for the SoC development and high-level synthesis and enables C, C++ and SystemC programs to be directly targeted into the Xilinx devices without manually creating the Register-Transfer Level (RTL) [59].

The Vivado Design Suite has a library of predefined complex functions and circuits, which can be used for the design. These predefined circuits are called IP cores and they simplify the design process [59].

This software tool can be used for the hardware design preparation, which includes several phases. The first step is to create the hardware design using the IP Integrator and a list of the available IP cores, thus creating the RTL description in the VHDL or Verilog. The design has to be validated. The validation process helps to find the errors in the design that could prevent the hardware from working properly. The most frequent errors can appear in connections between the blocks or in the parameter settings for the individual blocks [59].

If the validation is successful, then a so-called HDL wrapper can be generated. It is basically a top-level description of the system. Then the synthesis process generates all source files for the IP blocks as well as any relevant constraint files and maps the design to a netlist. The netlist is translated to a gate level description. On this stage the simulation is made once again to confirm that the synthesis is fulfilled without errors [59].

The next step is the design implementation, where the netlist is placed and routed onto the FPGA device resources and a bitstream file (a binary file) with the configuration data for the implementation in the programmable logic is generated. The designer can validate the map, place and route the results using a timing analysis, a simulation and other verification methodologies. After it the hardware image is complete and the hardware platform can be exported to the SDK environment. The SDK supports creation of the software applications for the specified hardware platform [59].

One more tool, which will be possibly used during the present work, is the SDSoC environment. The SDSoC is a system level compiler, which targets a base platform and is capable to compile C/C++ functions into the programmable logic. It works one level above the Vivado HLS compiler. After analyzing a program and determining the data flow between the software and hardware functions, it generates an application specific SoC including a complete boot image with the firmware, operating system, and application executable. The Xilinx HLS compiles the transformed C/C++ to the HDL code. The HDL code and the corresponding cores are automatically packed into the IP-XACT format and serve as the input for the Vivado IP integrator. The SDSoC environment automatically generates the compatible data-mover IP-cores and the interface IP-cores for the programmable logic part of the Zynq Ultrascale+ device. This can result in the automated generation of a new SoC system with new HW accelerators, which replace the original SW-based system. The SDSoC supports compilation of the Xilinx versions of OpenCV libraries, which comprise different mathematical functions such as Gaussian, Median, Bilateral, Canny edge detection, SVM, LK Optical Flow and etc. [44, 107].

More detailed description of the design development using the mentioned software tools will be provided in the practical part of the work.

1.6. Results and Related Publications

During the research stage presented in the previous sections the analysis and comparison of the existing adaptive RLS algorithms were performed. The results of this stage of the research are the following released publically accessible Application note and Evaluation package:

1. Kadlec J., Likhonina R. Adaptive RLS Algorithms Reference Implementations. Application note, ÚTIA, 2017.

   Abstract: This software presents set of adaptive recursive least squares system identification algorithms based on the Bayesian extensions of real-time adaptive system identification as well as extending the existing recursive least square adaptive algorithms for estimation of time varying parameters in the applications of acoustic signal processing. The included reference adaptive algorithms are implemented in Matlab 2016b. Algorithms serve as „golden“ reference models for the embedded implementations on dedicated processors like Arm Cortex A9 and the FPGA programmable logic accelerators in devices the Xilinx Zynq. Algorithms are numerically robust. Algorithms are implemented in double precision floating point (64bit), single precision floating point (32bit) and in logarithmic arithmetic with precision 32bit and 19bit. This software also presents adaptive recursive least squares system identification algorithms taking advantage of dynamic normalization of the core of the algorithm into the guarantied range <1-, 1> for all variables. These algorithmic cores are suitable for the fixed-point implementation (14bit).

2. Evaluation package including .m scripts with the DSP algorithms pre-compiled as .mexw64 files for MATLAB R2016.b (or higher) and two standalone SW applications for Win7 64b or Win10 64b (for users without MATLAB). These standalone SW applications have been created by compilation of .m scripts and .mex functions in the MATLAB R2019b compiler toolbox.

   Both the application note and the evaluation package are available for downloading at http://sp.utia.cz/index.php?ids=results&id=dsp_1_6

Moreover, the tools and techniques for programming in the FPGA part of the Xilinx Zynq were investigated. The outputs of this investigation are the following:

1. Functional demonstrator of the camera-to-touchscreen device prepared using the Xilinx Vivado 2015.2 and SDK 2015.2 tools and showing how to get a full HD color image from the camera module with a higher resolution to a touch display with a smaller resolution and to move along the image by touching the screen.

2. Likhonina R., Kohout L., Kadlec J. Camera to Touchscreen Demonstration for MicroZed 7020 carrier board, Avnet 7-inch Zed Touch Display and Avnet Toshiba Industrial 1080P60 Camera Module. Application note, ÚTIA, 2016.

   Abstract: This application note describes a camera-to-touchscreen demonstrator, which has been created using MicroZed 7020 carrier board, Avnet 7-inch Zed Touch Display and Avnet Toshiba Industrial 1080P60 Camera Module.

The Camera Module sends full HD (1080p high-definition) image at 60 fps, which is processed by MicroZed 7020 carrier board and transferred to the 7-inch LCD display with active area 800x480 pixels. Thus, there is a part of full HD image displayed on the LCD display. The full image can be looked through moving along the active area by touching the screen.

The application note and the demonstrator are available for downloading at http://sp.utia.cz/index.php?ids=results&id=Avnet_TCM_LCD.

3. Likhonina R., Kohout L., Kadlec J. Camera-to-touchscreen design. *In: 6th International Workshop on Mathematical Models and their Applications (IWMMA'2017)*, Krasnojarsk, RU, 20171113.

Abstract: The present paper describes an FPGA design of a camera-to-touchscreen demonstrator that has been prepared using Xilinx Vivado 2015.2 and SDK 2015.2 tools. The demonstrator consists of MicorZed 7020 Carrier Board, Avnet 7-inch Zed Touch Display and Avnet Toshiba Industrial 1080P60 Camera Module. The camera transmits a full HD video signal at 60 frames per seconds to MicroZed 7020 board, which processes it and sends to the LCD display with active area of 800x480 pixels. As the display has smaller resolution, only a fragment of the whole video frame can be seen at once on the display, whereas the full image is stored in the memory. By touching the screen one can travel along the stored video frame and look through the whole image. The design can be used, for example, as a car rear view mirror monitor benefiting from touchscreen technologies.

The conference article is available at
http://library.utia.cas.cz/separaty/2017/ZS/likhonia-0484186.pdf

CHAPTER 2

Algorithm Implementation in the MATLAB R2019b Environment

This chapter describes an approach, which is supposed to be applied as a pre-processing stage for hand detection and gesture recognition problems.

It comprises three sections. The first section describes simulation results performed in the MATLAB R2019b simulation environment [70] and compares two algorithms – the QRD RLS algorithm and the QRD RLS Lattice algorithm – in terms of the computational accuracy and time.

In the second part of this chapter the experimental results with real data are performed. Firstly, it presents the simulation experiments based on the parameters obtained during the identification process with the real data. Such kind of a simulation is closer to the reality and presents a safe step to the experiments with the real data, which proves that the algorithms function in a reliable way. It can also serve as a golden model for further experiments on different devices. Finally, the results of experiments with the real data obtained from the device equipped with ultrasound transducers including the experiments both with the QRD RLS algorithm and the QRD RLS Lattice algorithm are discussed and the algorithms are compared again in terms of the computational accuracy and time.

The last two sections provide a short summary of the whole chapter and a comparative analysis of the outputs including contributions on this stage of the research and the related publications.

2.1. Simulation in MATLAB R2019b

This section provides a description of the simulation results as well as a comparison of the computation results of the QRD RLS algorithm and the QRD RLS Lattice algorithm. The computation is performed in the MATLAB R2019b environment on a PC.

2.1.1. Experiments with the QRD RLS Lattice Algorithm

The proposed approach, subject to a detailed description in this section, is based on a noise cancellation technique and uses the QRD RLS Lattice algorithm with the exponential forgetting (EF) [40-41, 47-48, 50, 81].

The basic concept of a hand recognition application is straightforward: the device equipped with a network of ultrasound transducers and an integrated pre-processing unit transmits ultrasound impulses, which are reflected from a hand and returned back to the system. The received responses and their characteristics such as, for example, time needed to come back to the system, are used to detect the hand presence and position or even the distance from the hand to the device [60-63].

However, a seemingly simple concept of a hand detection application faces a couple of challenges in the real environment as far as inevitable undesired responses from other objects in the environment apart from the hand can cause a significant problem for an identification process (see Fig. 2). In this case an undesired response can be considered to be a static response. Obviously it has to be removed or at least suppressed in the target signal, which is the subject for the next stage of the data processing [60-63].

According to [76], a noise cancellation method assumes two types of signals:

- one is the desired signal, which is composed both from the temporary present short distance reflection signal and from the reflection signal mixed with the environment;
- the other one is the reference ultrasound source signal.

It is also assumed that a stochastic nature of the environment can be modelled by an additive uncorrelated noise. Based on the static relation of the environment reflection signal and the ultrasound source signal, it is possible to reconstruct the temporary present short distance reflection signal as a prediction/filtration error of the adaptive QRD RLS Lattice algorithm. It is constructed as a residual from the model of the static environment. As it was stated in the previous chapter, the QRD RLS Lattice algorithm allows computing the error signal without explicitly calculating the estimates of the parameters of a regression model at each time step [76]. It is a key and very important feature, which is supposed to be used to deal with a hand detection problem and with the suppression of the undesired responses.

During a simulation process, it is assumed that the hand appears only for a short period of time. Therefore, the reflections from the hand represent an additional short period disturbance. In this particular case the noise cancellation technique is applied using the QRD RLS Lattice algorithm with the EF, which is extended with hypothesis testing. It allows estimating a model, which suits best for this or that situation given the incoming data. Figure 2.1 shows a block diagram of the modelling process [60-64].

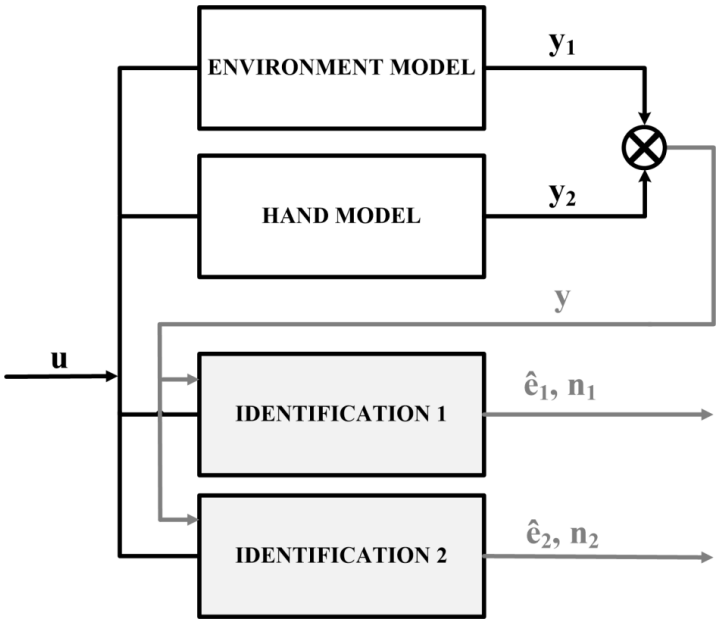

Figure 2.1: Block diagram

It is obvious from the diagram in Fig. 2.1 that the modelling process of a noise cancellation case comprises four main parts:

- the environment model producing the undesired noise and serving as a reference signal;
- the hand model simulating a hand, which appears for a short period of time and causes the additional disturbance;
- two identification blocks representing regression models of two different orders, one of which has a certain time delay.

Generally, a hand detection problem is modelled with so-called linear finite impulse response (FIR) based regression models [76]; thus, all four models – the environment and hand models, and two identification blocks – are the linear FIR models having a common input $\boldsymbol{u}$. Given the input data, the environment and hand models compute the outputs $\boldsymbol{y_1}$ and $\boldsymbol{y_2}$ respectively. Later on, these outputs are summed up to receive the output $\boldsymbol{y}$. The result of the summation $\boldsymbol{y}$ is sent to identification block 1 and identification block 2. The identification blocks perform parameter estimation of the hand model (if needed) recursively and calculate prediction and filtration errors [60-64].

As it was mentioned before, the hand reflection signal is reconstructed as a prediction/filtration error of the algorithm. Therefore, it is clear that the development of the prediction error $\boldsymbol{e}$ should estimate the development of the output $\boldsymbol{y_2}$ (see Fig. 2.1), as far as it is the hand, which causes the additional disturbance and, therefore, the increase of the prediction/filtration error. Using prediction resp. filtration errors, the hand appearance can be detected [60-64].

To make the detection even more precise and reliable, the algorithm is supplemented with the estimation of the probability of two hypotheses about one or another identification model suitable for the given situation. Thus, on its basis an appropriate identification model suited better to the given situation can be chosen. Basically,

due to the computation complexity and limited time for computation, only two identification models are considered (see Fig. 2.2).

Both models, subject to estimation, use the same incoming data. However, the first model has a higher order and learns using all available data during a specific time period. The second model has a smaller order and a certain time delay (TD) (see Fig. 2.2). It ensures that the second model works on the data where there is no hand presence possible as far as the hand is assumed to appear only in the area of the time delay. In this way a certain assumption about a possible distance of the hand from the device is made.

Using this principle, it can be assumed that if there is no hand in front of the device, the identification model with a higher order would have a higher probability; otherwise, the identification model of a smaller order with the time delay is supposed to suit better for an estimation process as far as it describes better and more accurately the given situation.

The assumption can be explained in the following way. As it was mentioned above, both models – a model of a higher order and a model of a smaller order – have the same input and are trained on the same data. However, the first model with a higher order is set in the way that it takes all available data and tries to perform the echo cancellation as precise as it is possible. It learns on the situation, when there is no hand, and creates a certain relation between its FIR parameters and the input/output data. Therefore, when there is no hand, it performs the filtration very well, but when the hand appears, the relation between the parameters of the model and the input/output data is strongly affected and the model has to learn again, so it becomes inappropriate for the given situation. However, it still computes the prediction/filtration errors based on its parameter estimates, which only start changing.

Contrarily, the second model with a smaller order is set in a way that it has the input time delayed data and it works with the input/output from the part, where it is assumed that there is no hand appearance possible. Due to the fact that it does not have all available data, it cannot compute the prediction/filtration errors in a right way, but the estimation process is always in the same situation, i.e. there is a constant

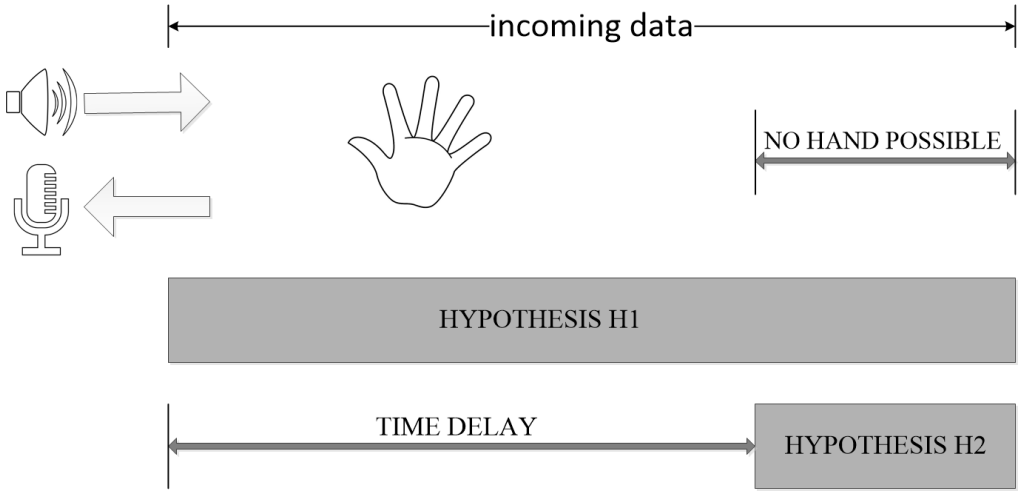

Figure 2.2: Hypothesis testing model [64]

problem to make estimation. When there is a hand, nothing will change for the second model, because the hand is out of the area where the model has its FIR parameters, so the relation between its FIR parameters and the input/output will not change for the second model, though there will be a higher dispersion on the output due to more noise related to the reflections from the hand, which are not compressed. Due to this fact, the second model with a smaller order and the time delay will have a higher probability when the hand appears in front of the device, because its parameters do not change. And in this way it will get a higher probability than the first model, which is greatly affected by the hand appearance.

Thus, using the output probabilities of the recursive model probability estimation, the hand presence in front of the device is clearly identified.

During investigation, a series of experiments using both time-invariant and time-variant environment models have been fulfilled. As far as their outputs are very similar, only several of them have been chosen to illustrate the challenges and the final results of the simulation (see Fig. 2.3-2.7).

Let's start with the time-invariant environment model.

The signal ***u***, the input to the environment and hand models as well as to two identification blocks, is built in a way to correspond to the input signal provided by the ultrasound device, which real data will be used in the experiments described in the next section. It is in the form of a set of chirps represented by a sinusoid wave with a period of 5 samples and 880 samples space between them (see Fig. 2.3).

Such kind of the input signal is considered to provide the ill-excitation of the system, which means that it affects the numerical sensitivity and complicates the estimation process; however, it enables to calculate the hand distance to the device, which represents further contribution of the present approach. It will be described and experimentally shown in the section devoted to the experiments using the real data from the ultrasound device.

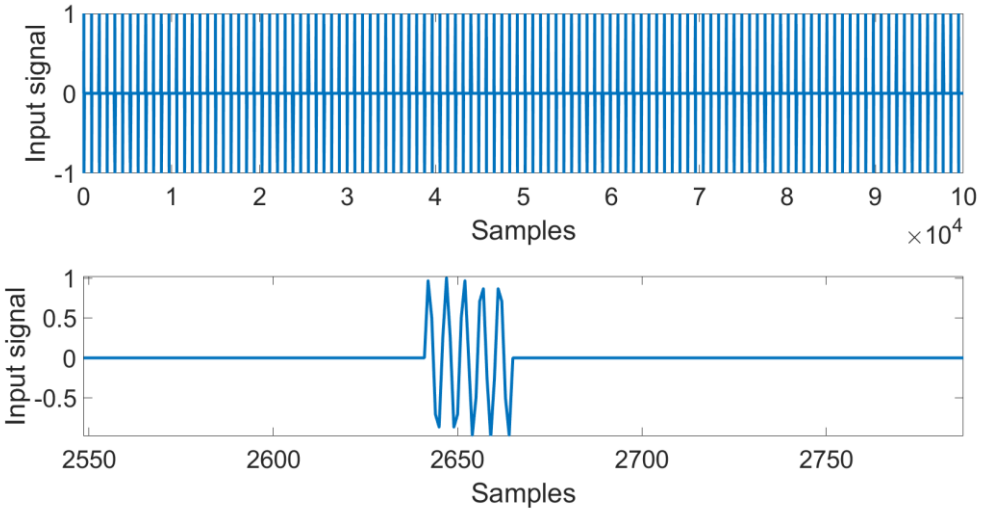

Figure 2.3: Input signal

The environment model is represented by a FIR model with order 880 to cover all available data (the inputs/outputs). It has constant random coefficients for the experiments in Fig. 2.4 and Fig. 2.5 and random coefficients with slowly changing values for the experiments in Fig. 2.6 and Fig. 2.7.

The hand model is represented by a FIR model having order 300. The hand absence is modelled by setting the coefficients in the columns of the model matrix to zero values. The columns of the matrix correspond to the time development of the signal. To simulate a short-term appearance of the hand, the coefficients of the hand model are set to non-zero values. To simulate a certain distance of the hand to the device, the first 50 rows and the last 150 rows of the matrix of the hand model are set to zero.

It should be also noticed that both the environment and hand models operate with the uncorrelated additional output noise to make the simulation closer to a real situation.

During the simulation, three cases of the hand appearance are analyzed:

- two short-term appearances at time step 10 000 and 50 000, both lasting for 2000 samples;
- a longer-term hand appearance at time step 80 000, lasting for 10 000 samples.

It is worth mentioning that to model the system, there should be made certain assumptions about the distance between the hand and the device, which is supposed to be known. Given it, the assumptions about the orders of the identification models and the time delay for hypothesis testing are made. Based on them, the implemented hypothesis order probability estimation identifies, which identification system is better for the parameter estimation for this or that period of the identification process.

Both identification models are also represented by the FIR models. The first model has the order set to 880 to ensure that it covers all available data during the one pulse of the input signal, which lasts for 880 samples. The order of the second model is 300. However, it takes only a limited number of the remote data, where the hand is unlikely to appear. Therefore, there is the time delay set to 580.

Due to the implemented hypothesis, the final results of estimation are not strongly dependent on a value of the EF factor; thus, one optimal value can be set for both the time-invariant and time-variant systems. However, a choice of the optimal value of the EF factor depends on the order of the system and should be made within certain limits given by the equation (A.79) in Appendix 2.

The value of the EF factor for the performed experiments is set to 0.99998.

The time scale for all experiments is 100 000 samples. Further, to make the simulation experiments closer to the experiments with the real ultrasound data, where the signals are transmitted with a frequency 40 kHz, but then it is sampled with a sampling frequency of 192 kHz, it is also assumed that the sampling frequency of 192 kHz is used, i.e. a maximum distance, from where the objects can be seen given the order of the model 880 is approximately 78.6 cm. Adding TD=580 samples to the second identification model, we, thereby, make an assumption that the hand can occur within the range of 0 – 51.8 cm from the device.

Now let us discuss the results of the experiments with the time-invariant (Fig. 2.4-2.5) and time-variant environment models (Fig. 2.6-2.7) in more details.

Figure 2.4 shows the results of the experiments given the time-invariant environment model. The prediction error development is shown in magenta (see Fig. 2.4-2.7). In the very beginning of the estimation process there is an increase of a prediction error. It can be explained by the fact that the system needs some time to learn to estimate the parameters correctly and to converge to the right values. The character of the hand output and, therefore, the development of the hypothesis in the place of the hand presence, i.e. its form "short peak-nothing-short peak", is conditioned by the character of the input signal, described above.

Figure 2.4 shows three peaks of the prediction error at time step 10 000, 50 000 and 80 000. It is the moment when the hand appears. It is also clear from Fig. 2.4 that after the hand disappears there is still an increase of the prediction error preserved for some time. Especially it is obvious for the third hand appearance, when it stays a longer period in front of the device. These slowly decreasing errors are caused by the fact that the value of the EF factor is close to 1 and the system adapts to a new situation very slowly. Therefore, the development of the prediction error and the output $\boldsymbol{y_2}$ (presented in blue), though it should be similar, differs from time to time (see Fig. 2.4). From this, it is obvious that based only on the prediction error results; it is hard to identify when the hand disappears, especially, in case of the long-term presence. By implementing hypothesis testing in the algorithm, the results can be improved and become more precise.

In Fig. 2.4 the identification model of the 880th order is shown by a red curve, while the identification model of the 300th order with the time delay (TD) of 580 samples is presented in a green curve.

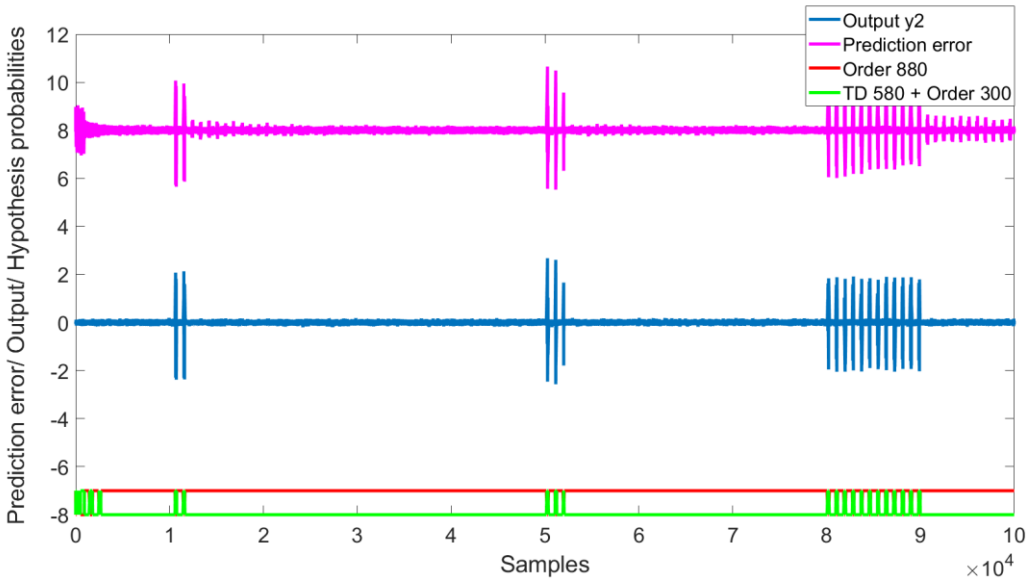

Figure 2.4: Simulation results: a time-invariant environment model

It is clear from the graphs in Fig. 2.4 that when the hand is present in front of the device, the identification model of a smaller order is winning. Contrarily, if there is no hand in front of the device, the identification model of a higher order is applied by the algorithm. It is obvious that the results are in correspondence with the assumption made in the beginning of this section, which states that the smaller order model should have a higher probability, if there is a hand in front of the device. Thus, hypothesis testing determines the exact moment of the hand appearance, which due to the nature of the input signal can be used for calculating the hand distance from the device.

To show the results in more details, a fragment for the second hand appearance is chosen (see Fig. 2.5).

To bring the simulation more in a line with the reality, in further series of experiments a time-variant environment model is used. In this case it is not static anymore and it simulates the situation when there are some slowly moving objects present in the environment.

Similarly to figures for the time-invariant environment model, the results for the time-variant environment model are presented in Fig. 2.6.

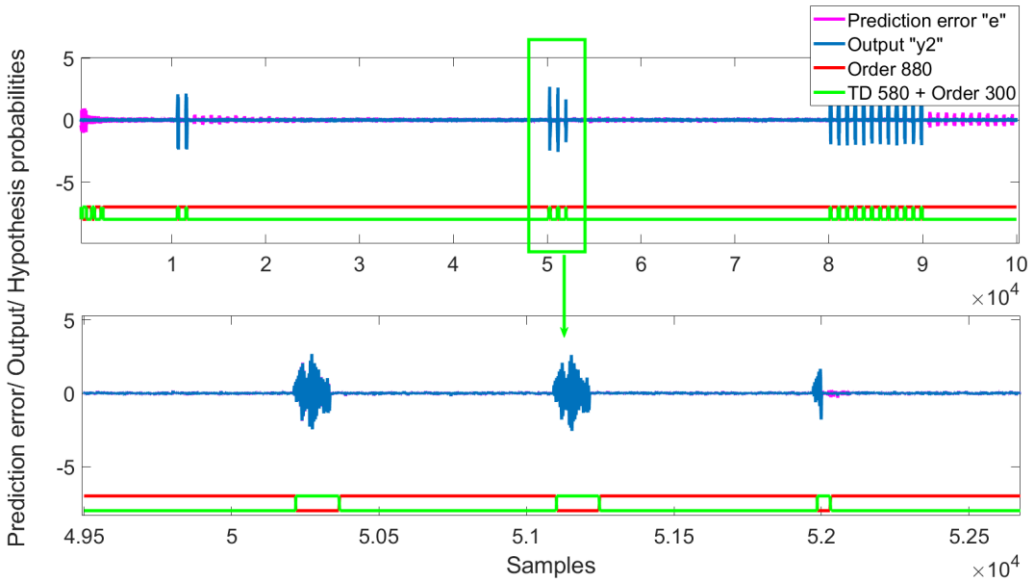

Figure 2.5: Simulation results: a time-invariant environment model (detailed fragment)

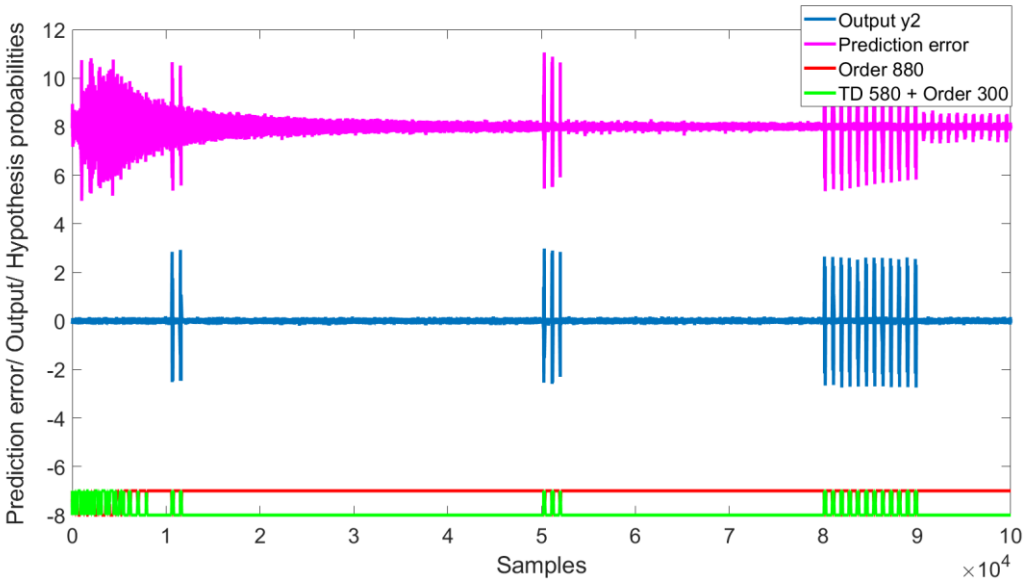

Figure 2.6: Simulation results: time-variant environment model

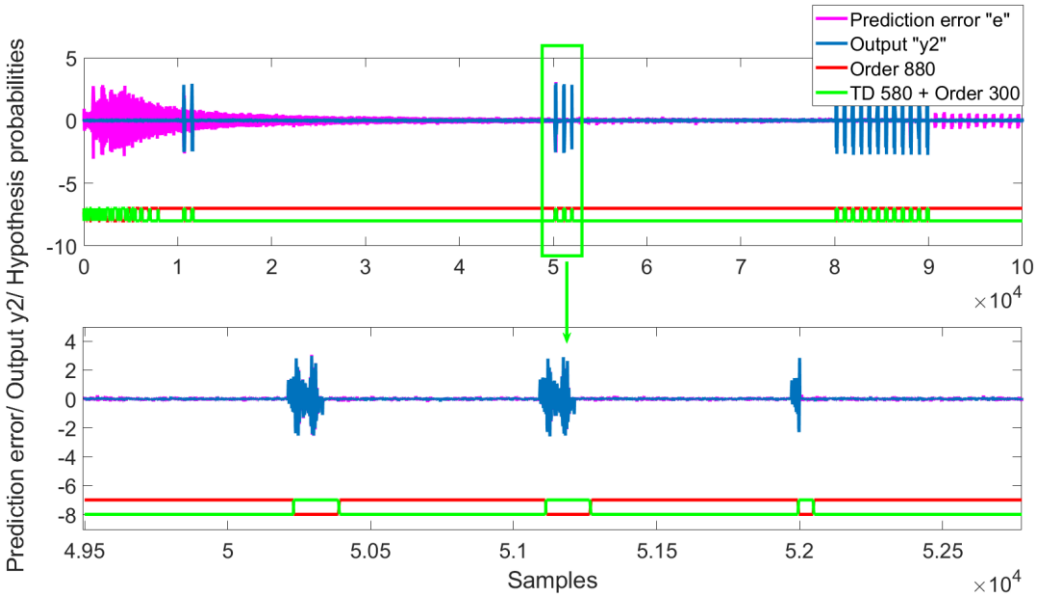

Figure 2.7: Simulation results: a time-variant environment model (detailed fragment)

All parameters are set exactly alike as in the previous experiments, except for the coefficients in the environment model, which are slowly changing-in-time random values.

It is clear from Fig. 2.6 that the prediction error in the very beginning is very high due to using the time-variant environment model and the first hand appearance is almost undetectable if using only the development of the prediction error. However, the hand is accurately recognized by the application of hypothesis testing. The results of estimation for the second and the third hand appearance are very precise as well.

More detailed results for the second hand appearance are shown in the fragment in Fig. 2.7.

From the graphs above it is obvious that the assumption about the smaller order model best suited for a description of the hand presence in front of the device is valid. Besides, this simulation model corresponds better to the real situation and, therefore, the assumption is supposed to be valid for the experiments with the real ultrasound data as well.

2.1.2. Comparison of Computation Results of the QRD RLS Algorithm and the QRD RLS Lattice Algorithm

To show the advantages of the QRD RLS Lattice algorithm chosen for the implementation on the hardware platform, let us compare the results of the experiments for the time-variant environment model for the QRD RLS algorithm and for the QRD RLS Lattice algorithm. The comparison is made in terms of the accuracy and computational time.

The settings of the experiments are the same as those described in the previous sections.

The results of the QRD RLS algorithm are shown in Figure 2.8, while the results of the QRD RLS Lattice algorithm are shown in Figure 2.9.

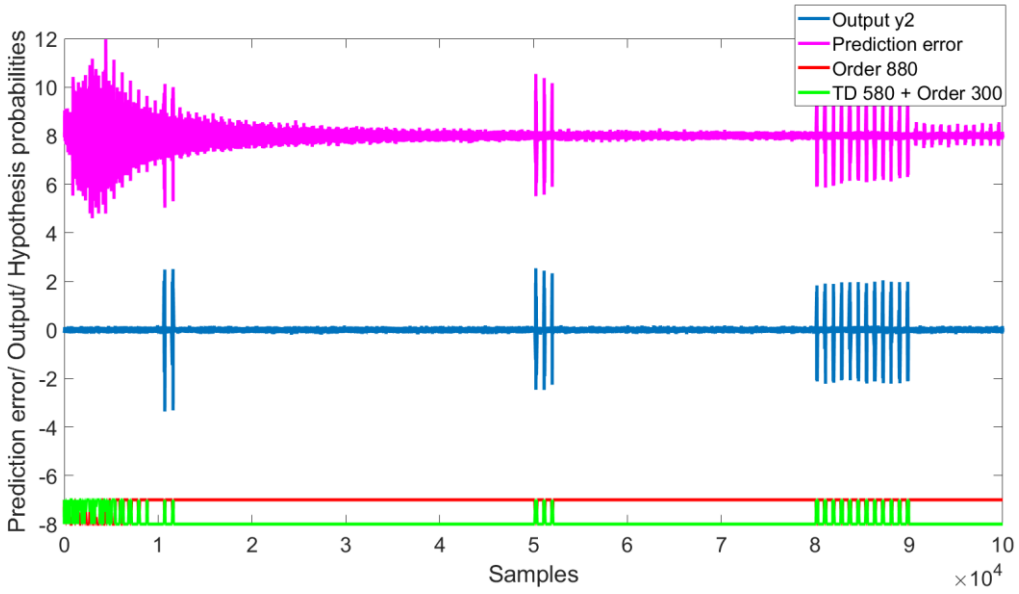

Figure 2.8: Results of the estimation process using the QRD RLS algorithm

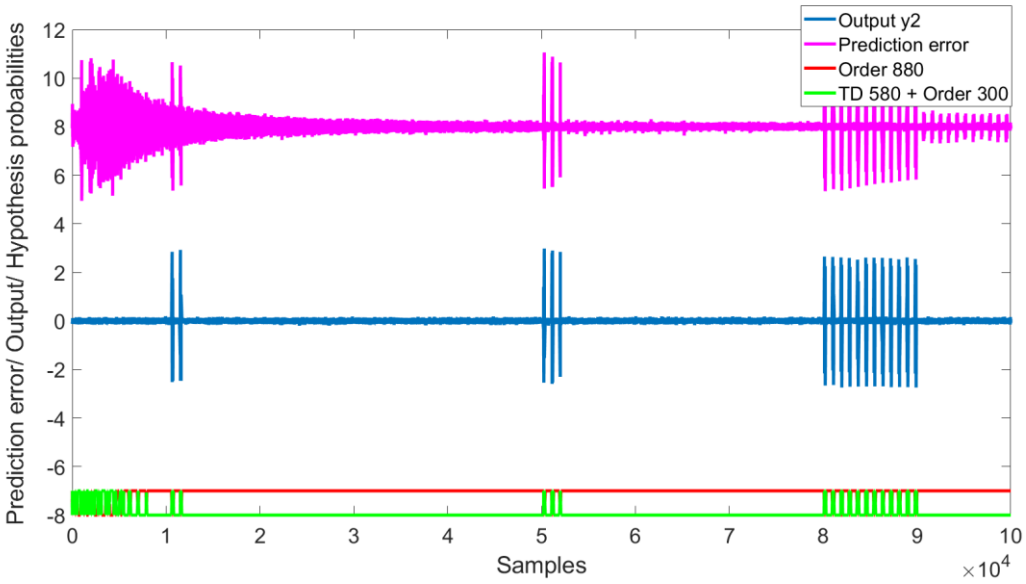

Figure 2.9: Results of the estimation process using the QRD RLS Lattice algorithm

It is obvious from the graphs in Fig. 2.8 and in Fig. 2.9 that the outputs for both the QRD RLS algorithm and the QRD RLS Lattice algorithm in terms of the quality of estimation and its accuracy are practically the same. However, as far as the computational time is concerned, the gap between two compared algorithms is quite large. For the QRD RLS Lattice algorithm, the computational time is within 15s on a PC with Intel® Core™ i7-4770 CPU, 3.5GHz: the variable initialization is made in the MATLAB R2019b environment and the algorithm itself is calculated in C code using .mexw64. However, for the QRD RLS algorithm, it constitutes 580s or approximately 10 min of computation.

Thus, given N=100000 and the number of operations for one computation step 34850, the QRD RLS Lattice algorithm requires approximately 58 MFLOP/s to be processed in real time. In the experiments the PC (single core) delivered 232 MFLOP/s for the QRD RLS Lattice algorithm in the DP performance, i.e. it can successfully compute the algorithm in the range of the real-time processing.

As for the QRD RLS algorithm, given N=100000 and the number of operations for one computation step 2301952, it requires approximately 3837 MFLOP/s to be processed in real time. However, in the experiments the PC (single core) delivered only 397 MFLOP/s for the QRD RLS algorithm in the DP performance.

According to the experiments performed during the simulation in the MATLAB R2019b environment, it can be concluded that the proposed approach to system identification using the QRD RLS Lattice algorithm is promising and it can provide the sufficiently precise outputs. It has its limitations in terms of making the assumptions about the distance between the hand and the device. These limitations should be carefully considered before the implementation of the algorithm.

However, it is also obvious that though the QRD RLS Lattice algorithm is fast in comparison with the QRD RLS algorithm, still it might be insufficient for the real-data processing and there might be a need to accelerate the performance

of the algorithm. The means for the algorithm acceleration will be discussed in the next chapters.

The next section describes the results obtained while using the real ultrasound data from the device equipped with the ultrasound transducers and microphones. Performing these experiments, we would be able to say if the computation speed of the algorithm is sufficient or if it needs the acceleration.

2.2. Experiments with Real Data

This section describes the experimental results with real data taken from a device equipped with ultrasound sensors. The results of the experiments using QRD RLS Lattice algorithm are provided in section 2.2.1, while comparison between QRD RLS algorithm and QRD RLS Lattice algorithm is described in section 2.2.2.

2.2.1. Experimental Results Using the QRD RLS Lattice Algorithm

The ultrasound data were provided by the device developed in ÚTIA. The detailed description of the device and its functions can be found in [85]. In the present section only a short description is provided.

Figure 2.10 shows the design of the ultrasound device and its components.

Three basic components of the hardware platform are represented by the TE0720 FPGA SoM module, the TE0706 carrier board and the ÚTIA evBoard v1.7.

The board comprises a linear microphone array, a dual piezo ultrasound speaker, a common clock distribution to all microphones and the ultrasound speaker driving the circuit. The microphone array consists of 32 digital microphones, one of which is used to provide the control signal. The distance between the microphone acoustic ports constitutes 3.8 mm. The frequency range of a microphone is from 0 to 80 kHz, while the array maximal frequency is 40 kHz [85].

It should be also noted that the ultrasound speaker works on 40 kHz frequency. The responses after it are sampled and decimated to obtain the Pulse-Code Modulation (PCM) samples, which sampling frequency is 192 kHz.

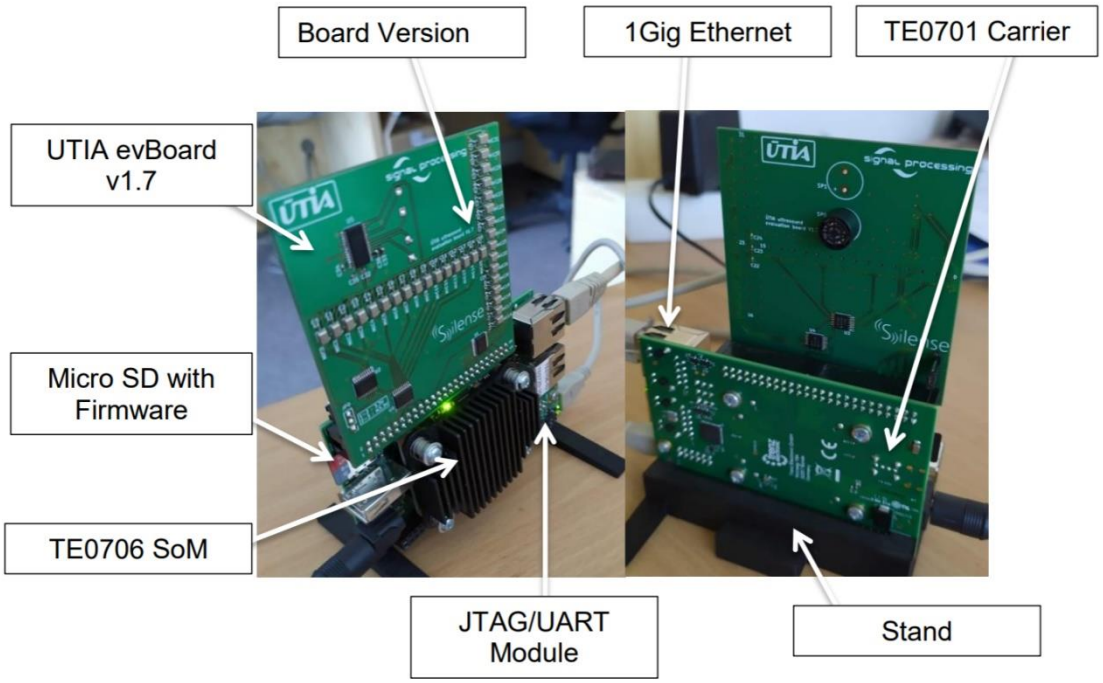

Figure 2.10: ÚTIA evBoard with the FPGA module and the carrier board [85]

The beam-former was designed in the FPGA part of the hardware module by the ÚTIA team. The block diagram is shown in Fig. 2.11.

There are three dark-green blocks on the left side from the memory block, which process the data from the microphones and store them in the memory [85]:

- block "Packetize", which adds end markers to the stream of the microphone data, i.e. on this stage the data are divided into blocks,
- block "Capture", which stores the data blocks to the double data rate (DDR) memory,
- block "Chirp Generator", which generates chirps. Using this block, the chirp frequency, its length and the number of reverse phase periods at the end of each chirp are adjustable [85].

Blue blocks on the right side from the memory block are responsible for adjustments of the microphones and the signal from the microphones. There are six blocks [85]:

- block "Adjust Phase", which adjusts the phase of the signal taking into consideration the steering direction of the beam-former. It is possible to set the initial angle and the number of sectors of the view field.
- block "Decimate and Sum", which define the microphones for beam-forming and the order and subsampling/decimation factor,
- block "BP Filter", which applies the bandpass filtering to the delay-and-sum (DAS) beam-former output,
- block "RMS Envelope", which uses a running window to compute the route mean square (RMS). Besides, it optionally subsamples the output. The window length and the subsampling factor are adjustable.

- block “Max Detection”, which is responsible for the detection of the position and the value of the global maximum in the output data,
- block “Conversion to Img”, which has two modes – Normalization on and off. When the Normalization mode is turned on, the compensation of the strength of the recorded ultrasound depending on the distance is performed by using the exponential function. When the Normalization mode is turned off, the minimal and maximal values are used for the conversion of the RMS envelope data to the value between 0-255 [85].

For more details refer to [85].

At this point let’s describe the output signal, which is subject to processing by the algorithms in this work.

As it was mentioned above, the output signal is obtained from 31 microphones situated on the ultrasound device. The ultrasound speaker sends chirps 600 times on the 40 kHz frequency, whereby in-between sending the preceding signal and the following one there is a certain waiting period or a delay, during which nothing is sent. Each signal has 880 PCM samples, which are obtained by sampling with the sampling frequency of 192 kHz.

The raw output signal is presented in Figure 2.12. It has 11520000 samples, lasts for 60s and after each pulse there is a certain time delay, which constitutes 18320 samples or 0.095s.

On the upper graph in Fig. 2.12 there is a raw uncompressed signal, where it is impossible to differentiate by a human eye if there is a hand. On the bottom graph the enlarged pulses are presented to show the waiting period between two measurements.

For illustration purposes, the signal was processed in a way that the large peaks, which correspond to the cross-talks, are removed, so that it became obvious, in which time moment the hand was present in front of the device (see Fig. 2.13).

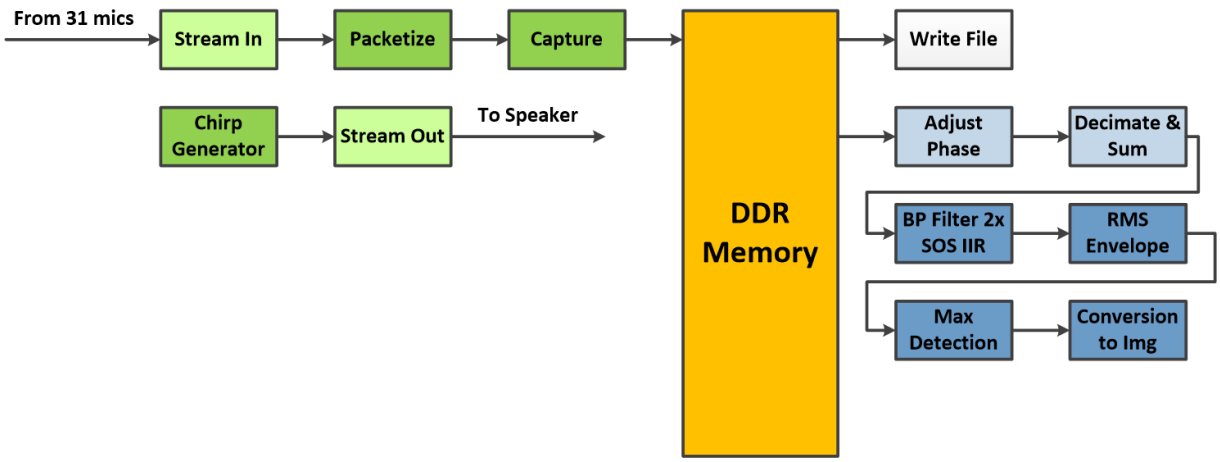

Figure 2.11: ÚTIA FPGA implementation of the beam-former accelerators [85]

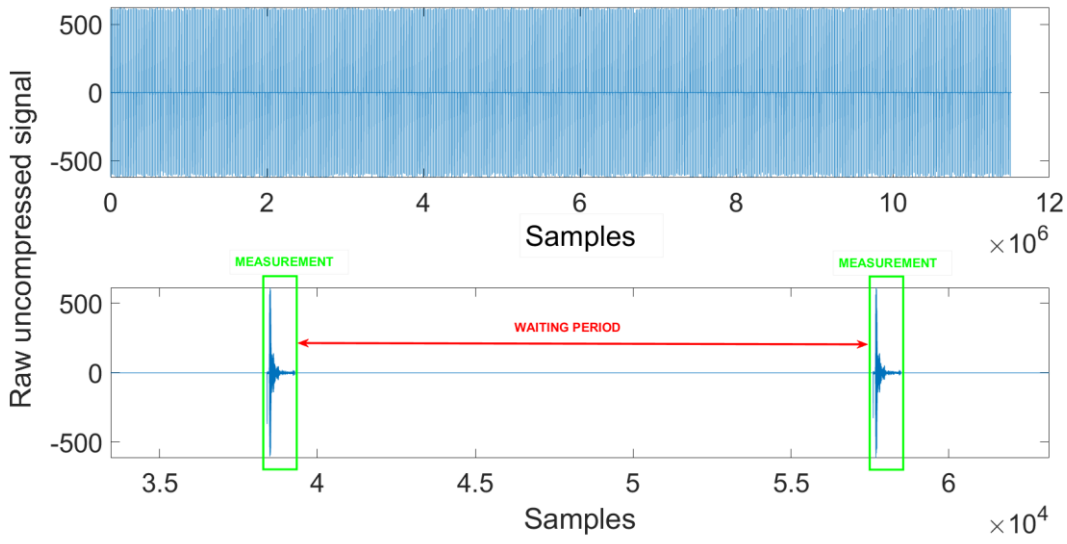

Figure 2.12: Raw uncompressed output signal from the ultrasound device [64]

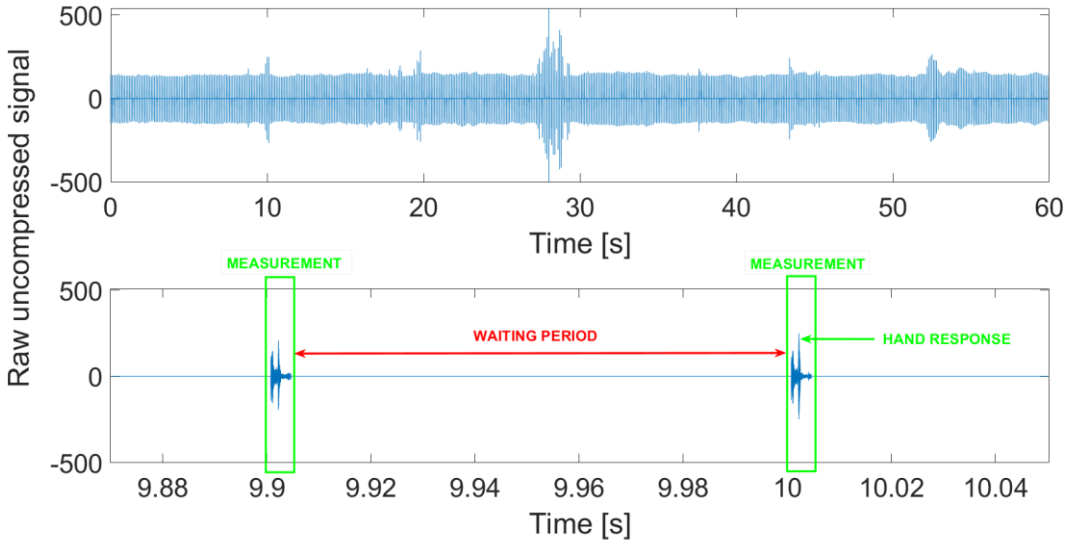

Figure 2.13: Raw uncompressed output signal from the ultrasound device (without cross-talks)

Besides, in Fig. 2.13 the time scale was converted to seconds to illustrate that the total measurement time was 60s.

The principle of functioning of the algorithm is in the way illustrated in Fig. 2.14.

In the beginning the algorithm works with a priori initial conditions. It performs estimation in stage S_1 up to the waiting period and in the end it has some outputs. Its outputs represent initial conditions for identification for the following stage, i.e. for stage S_2 (see Fig. 2.14). During the waiting period the algorithm stops and does not compute anything. When stage S_2 starts, the algorithm begins

its estimation process again, but with the inputs calculated in stage S_1. In this way it performs the identification process and computes all data up to stage S_n where the final results are computed. Due to the waiting period, practically the algorithm has more time for computation in each stage.

However, for the practical and computational purposes the raw signal from Fig. 2.12 is compressed by removing delays between the preceding and the following signals. The compressed raw signal is illustrated in Fig. 2.15.

It should be also noted that during the experiments described below the data from only one microphone are used. It differs from the current approach, which is used by the ÚTIA team for hand detection on the ultrasound device described above. In the current approach the data from all microphones are used and the beam-forming is made. In the approach presented in this section it is illustrated that even on the basis of the data taken from one microphone it is possible to detect the hand presence and the distance between the hand and the device.

If we look at the signal in more details (see Fig. 2.16), we can clearly see that in the bottom part of the figure, which illustrates a detailed graph for one pulse sent during the measurement, there is a high peak in the beginning when the speaker sends the signal (outlined in a red rectangle in Fig. 2.16). It is due to the fact that the device listens to itself. Then there are smaller peaks, which represent the resonance. The response from the hand is outlined in a green rectangle.

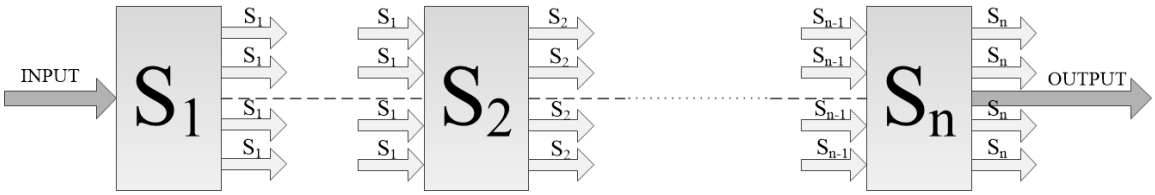

Figure 2.14: QRD RLS Lattice principle

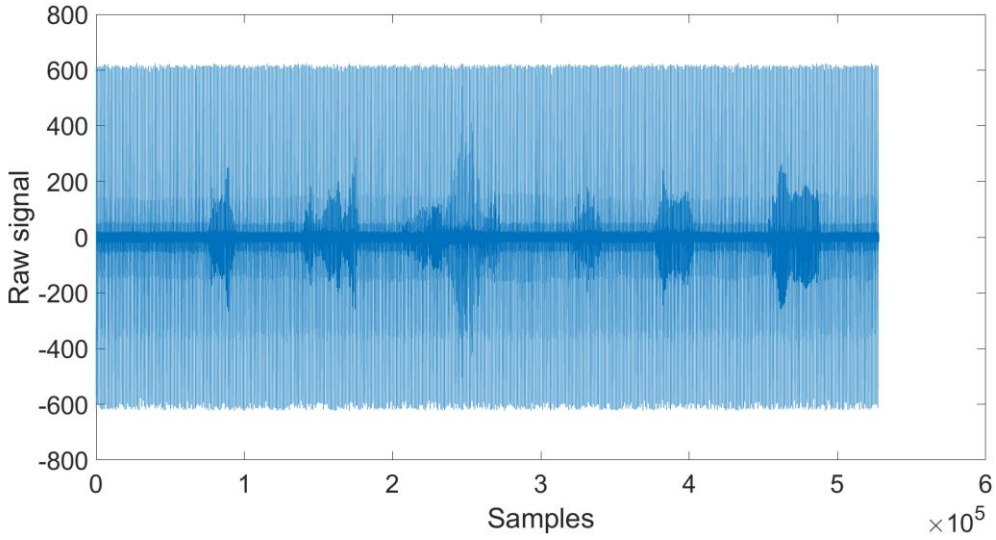

Figure 2.15: Raw compressed output signal from the ultrasound device

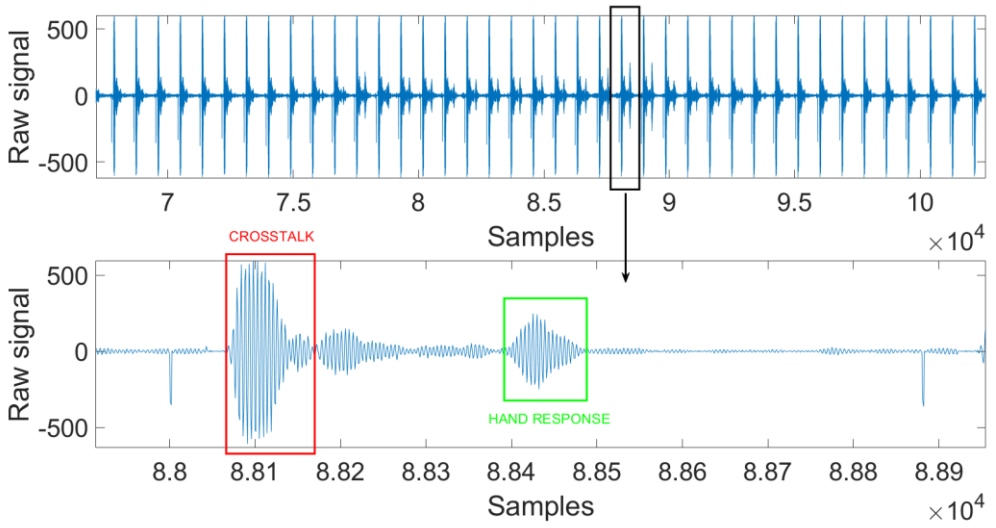

Figure 2.16: Fragment of the output signal

The input signal provided by the ultrasound speaker is a set of chirps represented by a sinusoid wave with a period of 5 samples, a sampling frequency 192 kHz, and 880 samples space between them. It is not possible to measure it directly; thus, the reconstruction of the input signal from the raw output signal was performed. As far as the microphone and the speaker are situated close to each other, it is possible to do it in a way that the reconstructed input signal is very similar to the real one.

As it was shown above, the first high peak in the raw output signal is actually the input signal listened out by the device (see Fig. 2.16 and Fig. 2.17). From the graphs, it is obvious that the peak occurs every time the speaker sends the pulse and, therefore,

it has a certain period of its occurrence. It can be used for the reconstruction of the input signal in the experiments with the real data as far as we know the beginning, the length of the input signal (140 samples), and the duration of one pulse (880 samples). Everything in-between the preceding peak and the next one are set to zero (see Fig. 2.17).

In this way the input signal is taken from the raw output signal and reconstructed in a separate input, which is used in the experiments with the Lattice identification hypothesis testing algorithm. The reconstructed input signal is shown in Fig. 2.18.

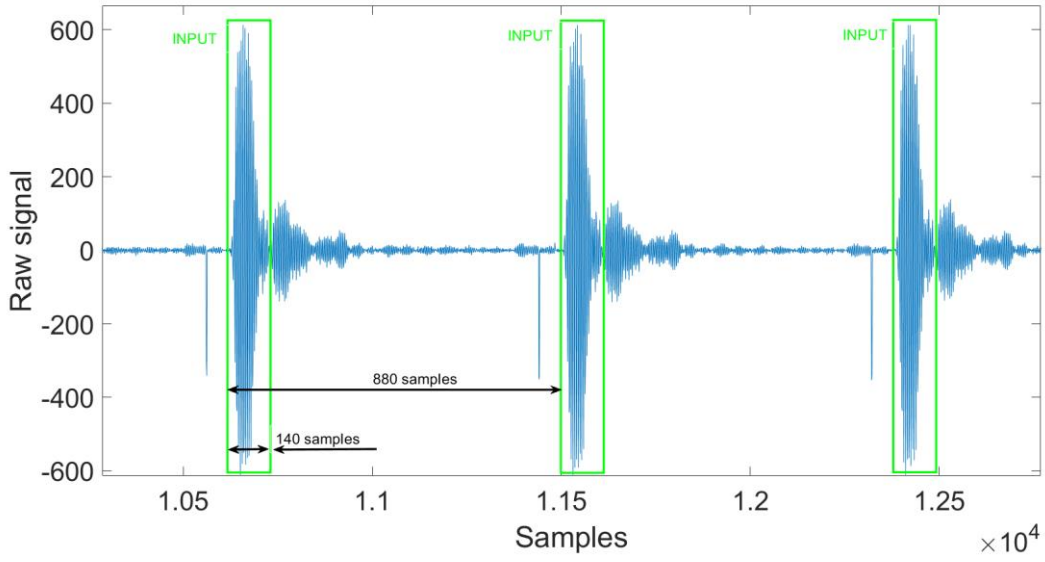

Figure 2.17: Input reconstruction

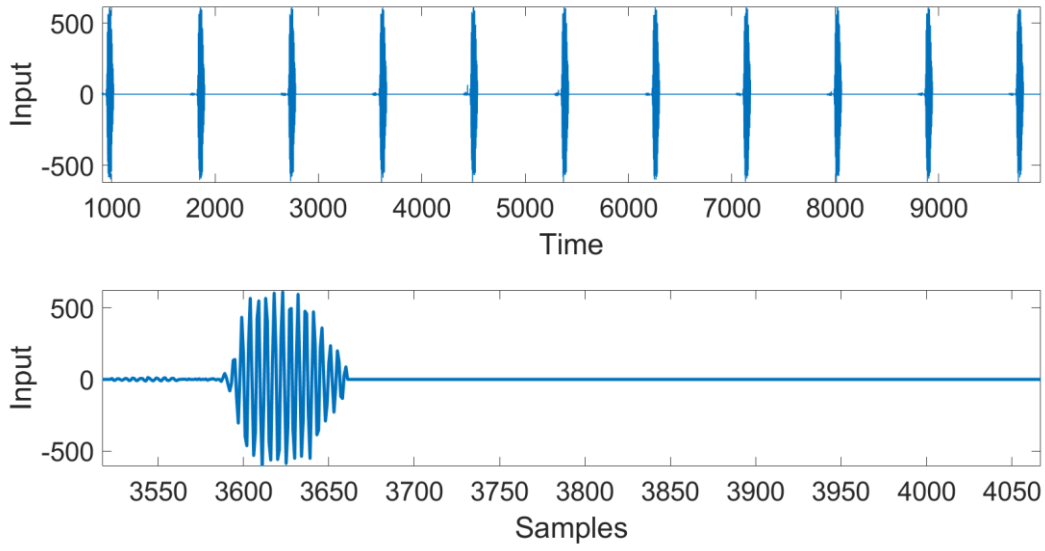

Figure 2.18: Input signal [64]

The raw output signal is modified in a way that the self-listened inputs (high peaks used for the reconstruction of the input signal) are replaced by zeros. It is possible to make it without affecting the validity of the experiments, because the period of its occurrence is known. The final output signal used in the experiments is illustrated in Fig. 2.19.

The higher peaks on the graph in Fig. 2.19 represent the responses from the hand, while the smaller peaks are signals coming back from other objects in the environment and considered to be noise. From the graph in Fig. 2.19, it is clear that during the measurements there are six hand appearances on different distances from the device. Moreover, the development of the third hand presence is different from the others. It is due to the fact that in this case the hand was moving forwards and backwards from the device.

The goal of the algorithms using hypothesis testing is to identify the presence of the hand and its distance from the device.

But as the first step before applying hypothesis testing on the real data, it is reasonable to bring the simulation more in a line with the reality and to ensure that the algorithms function in a way it has to function. In the simulation described in the previous section, the artificial models were used. They were based on the random parameters generated in the MATLAB R2019b environment. In case described below the simulation is based on the measured data and uses the real input signal.

As the first phase of performing the simulation, it is necessary to perform the identification process using the QRD RLS algorithm, which will compute parameters of the models. Taking one of the parameters when there is no hand and the other one when there is a hand, it is possible to reproduce the simulation close to the reality. These parameters represent the environment and hand models.

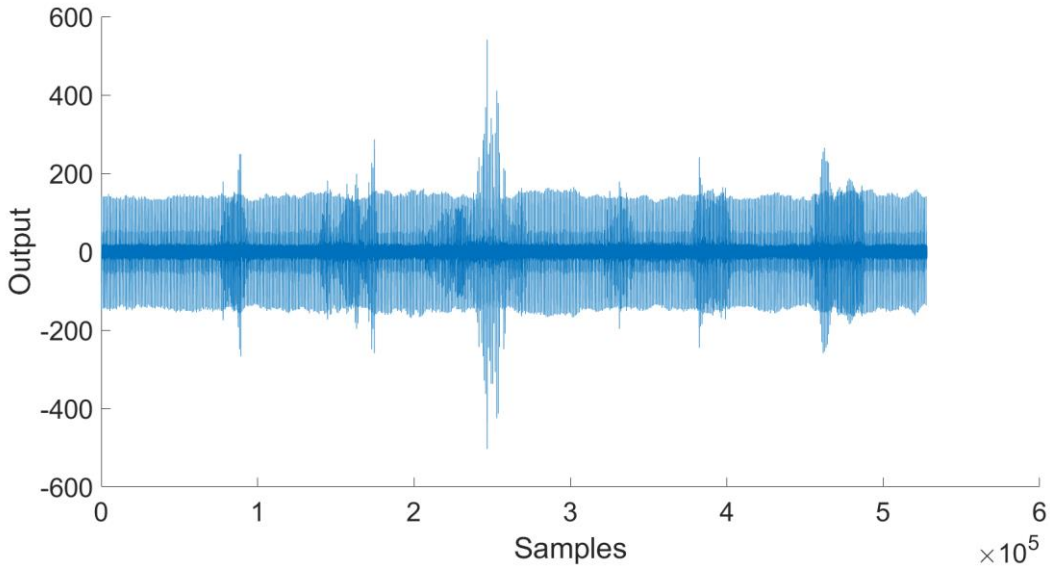

Figure 2.19: Output signal [64]

The parameters are taken in the area, where the identification process is stable. The difference of the simulation from the real situation is in the fact that the hand is always the same - constant and static - because it is simulated using one parameter only. Thus, its distance to the device is also constant. Otherwise, the simulation is as close to the reality as it is possible.

The principle of the simulation is similar as it was described in the section devoted to the simulation with the random values. There are four models: the environment model, the hand model and two identification models. This time, however, the order of the identification models are set to 768 and 256 with the time delay of 512 as it is supposed to be done in the experiments with the real data using hypothesis testing. The choice of the orders is explained in more details later on. The EF factor is set to 0.99999988.

The time period of the simulation is 528000 samples as it will be in the experiments with the real data with the application of hypothesis testing.

To make the simulation experiments close to the reality, there are also six occurrences of the hand simulated. They have more or less the same period of the presence in front of the device as in the experiments with the real data provided hypothesis testing. During the simulation one hand appearance is close to the preceding hand occurrence. It is made for the purpose to complicate the computation for the algorithm and to see if the algorithm is able to detect hand appearances closely coming one after another.

The results of the simulation are presented in Fig. 2.20.

It is obvious from the graph in Fig. 2.20 that the QRD RLS Lattice algorithm functions precisely enough in the situation when the parameters from the real data computation are used for the generation of the environment and hand models. It accurately switches between two hypotheses when the hand appears. It means that hypothesis testing can be safely applied to the real data computation, which is described below.

Let us remind first the basic concept of the algorithm for the real data computation. The idea is the same as it was in the case of testing the algorithm with the simulated data (see Fig. 2.21).

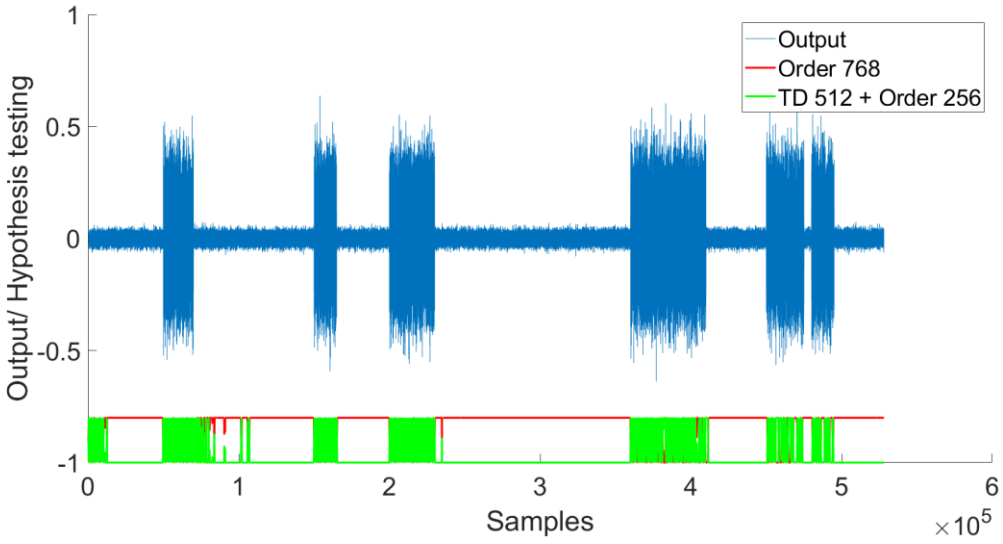

Figure 2.20: Estimation results using the QRD RLS Lattice algorithm (simulation with real data parameters)

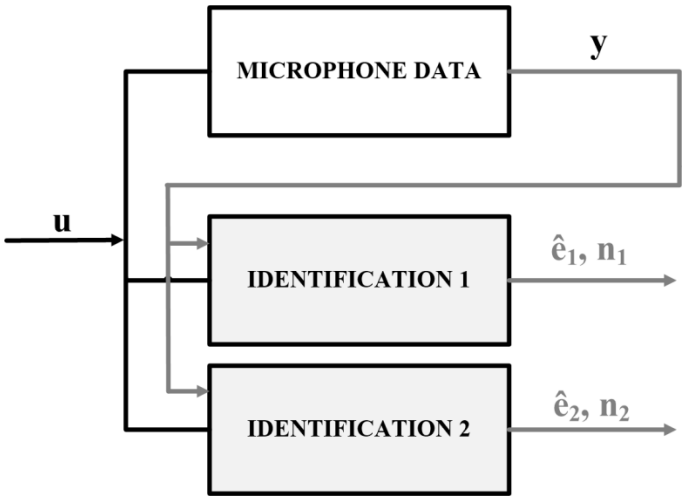

Figure 2.21: Block diagram

However, in this case there are three blocks only (see Fig. 2.21):

- the block representing the real data coming from a microphone,
- two identification blocks standing for the regression models of two different orders, one of which has a certain time delay.

The output of the algorithm of our interest is the estimated probability of hypotheses of the regression model: if there is a hand in front of the device, the algorithm should choose the regression model structure appropriate for this situation and different from the one in case there is no hand.

The identification models are set in the following way. The order of the first model is set to n_1=768. This order is enough to cover the available data for the one pulse

and at the same time it is good for pipelining into 3 or 6 processes, which will be described in the section about the optimization of the algorithm and its HW implementation.

The order of the second model is set to n_2=256 and there is a time delay of 512 samples. Thus, there is an assumption made that the second identification system makes estimation on the data, where there is no hand appearance possible, i.e. on the distance in the range from 0 cm to 46 cm. Besides, this order is also easily divided by 1 or 2 processes.

After fulfilling a set of the experiments, the optimal value of the EF factor was found, which is 0.99997. As far as the compressed output signal is used, the experiments work on N=528000 data samples.

Knowing the sampling frequency and the number of data samples, it is possible to calculate time, during which the algorithm has to perform the outputs to be able to process the data in real time.

$$\boldsymbol{T_{real} = \frac{N}{f_s} = \frac{528000}{192000} = 2.75s.} \qquad (\mathbf{2.1})$$

However, we also need to consider the fact that the real time of the measurement of the hand responses by the ultrasound device differs from it due to the waiting period before sending each signal by the speaker as it was described in the beginning of this section. Thus, considering this time of silence and the real size of the measured data, i.e. 11520000 samples, the real T_{real} = *60s*. It means that the algorithm has to perform estimation within 60s, in order it was possible to apply it for the real-time data processing applications. Moreover, considering the number of operations during one step of computation equal to 34850 and the number of steps N=528000, it is possible to calculate the number of operations per second, which are required to be performed. It is equal to approximately 307 MFLOP/s.

Figure 2.22 shows the results of computation.

On the graph in Fig. 2.22 there are the computed filtration errors (blue curves) and the hypothesis development during time (green and red curves in the bottom of the graph). In the beginning of the estimation process, the algorithm needs some time to estimate parameters in a right way. It is the learning stage of the algorithm. Therefore, there is some uncertainty, which identification model to choose. However, after the learning stage, i.e. after approximately 6000 samples, the estimation process converges to the correct values and the algorithm accurately recognizes the hand appearance by switching between two hypotheses.

In case when there is a hand, which causes the additional disturbance, the system switches into the identification model with a smaller order and with the time delay (green coloured). Contrarily, if there is no hand, the identification model with a higher order (red coloured) learnt on all available data has a higher priority. Thus, it is obvious from the graph that the assumption made in the beginning of the section is valid for the real ultrasound data the same as it was valid for both simulations.

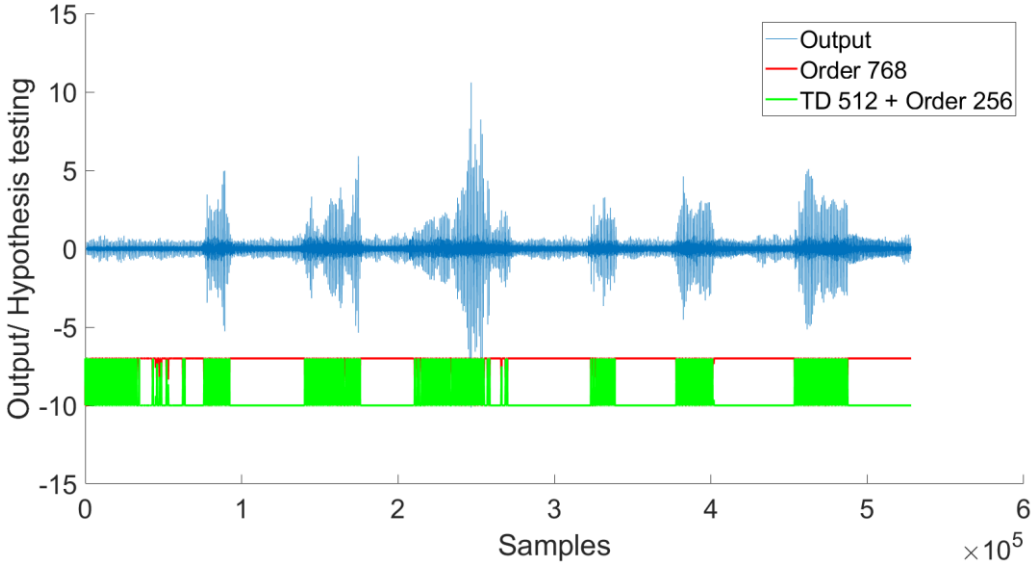

Figure 2.22: Hand detection [64]

As it was stated before, the filtration errors computed by the identification model trained on all available data, i.e. the identification model with a higher order, are accurately estimated and can be used along with the hypotheses to detect the hand presence (see Fig. 2.23).

The upper graph in Fig. 2.23 represents the output signal ***y***, while the bottom graph shows the development of the filtration error. It is clear from Fig. 2.23 that the signal is cleaned out from the unwanted responses from the environment and can be used for further data processing. The hypotheses help to define the precise moment the hand appears and disappears and in this way they allow to say from where further signal analysis should be started.

As it was mentioned above, due to the fact that the input signal is in the form of pulses (chirps) (see Fig. 2.24), it is possible to calculate the distance of the hand from the device.

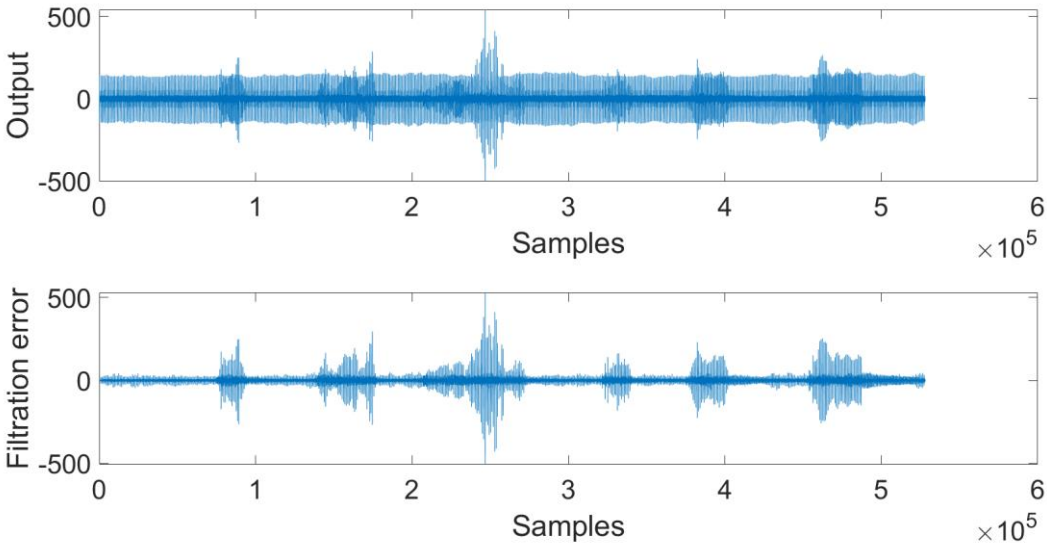

Figure 2.23: Hand detection (filtration errors)

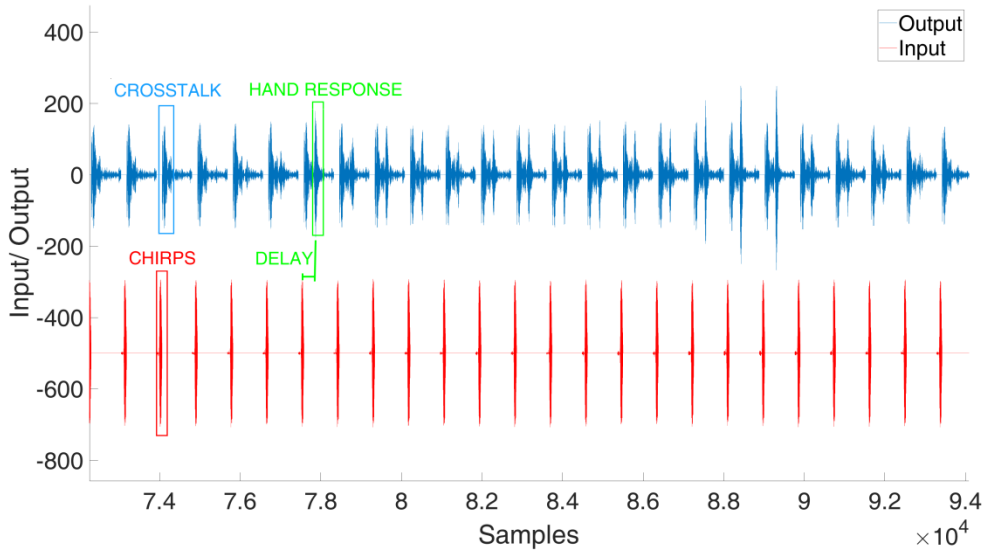

Figure 2.24: Distance computation

It is clear from the graph in Fig. 2.24 that there is a certain delay between the chirp and the hand response. Thus, knowing the number of samples between the input (the chirp) and the response from the hand, it is possible to calculate the hand distance from the device according to the well-known equation:

$$\boldsymbol{s = \frac{v \cdot t}{2}} \qquad (\mathbf{2.2})$$

where $\boldsymbol{s}$ is the distance [m]; $\boldsymbol{v}$ is the speed of the ultrasound in the air, which is 343 m/s; $\boldsymbol{t}$ is time [s].

In the equation time is divided by 2, because it should be taken into consideration that the signal goes from and back to the device.

Time can be calculated according to the following equation:

$$\boldsymbol{t = \frac{N}{f_s}}, \qquad (\mathbf{2.3})$$

where $\boldsymbol{N}$ is the number of samples between the input and a hand response, $\boldsymbol{f_s}$ is a sampling frequency [kHz], which is 192 kHz in our case.

The final results are presented in Fig. 2.25 and Fig. 2.26.

From the graph in Fig. 2.25 it is obvious that during the measurement five hand occurrences were more or less in the same position and their distance was approximately the same, i.e. approximately 40-45 cm. There are some minor changes in the position of Hand 2, which constitutes only several centimetres. However, in case of Hand 3 the changes in the position are more visible. They can be explained by the fact that during the measurement the operator was moving the hand towards and backwards from the device; therefore, its distance varies from 27 cm to 50 cm. Though the assumption about the maximal possible distance of the hand from the device, that is 46 cm, is not valid for Hand 3 as far as it was moving in the range from 27 cm to 50 cm, still the algorithm proves to be robust and hypothesis testing functioned reliable in this case too.

Moreover, from the graph in Fig. 2.25 it is easy to obtain the information about the duration, during which the hand was present in front of the device, because time is converted from samples to seconds. More detailed information about time as well as about the precise hand distance to the device is presented in Table 2.1.

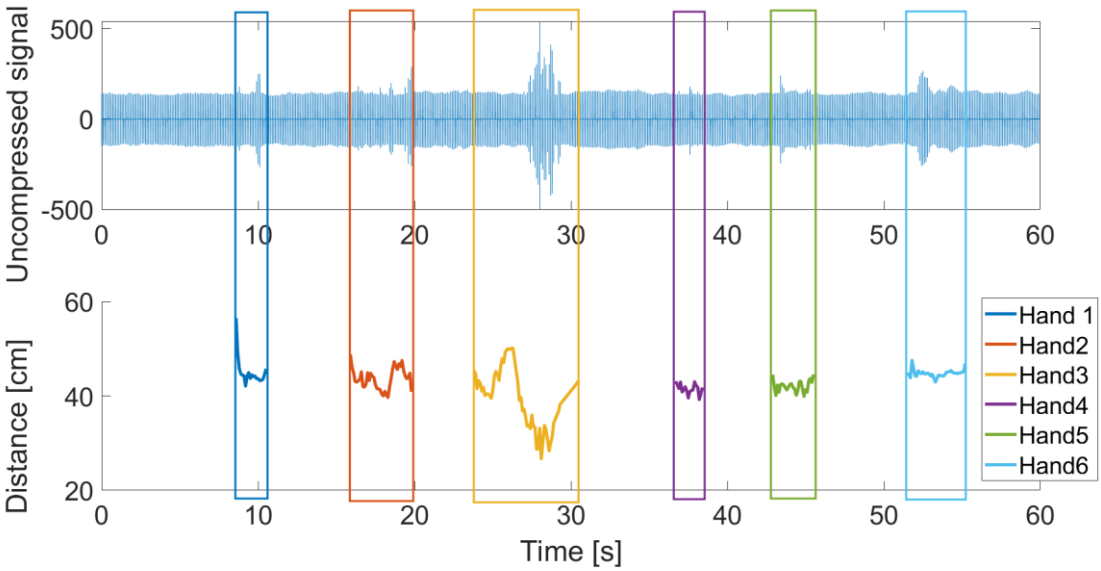

Figure 2.25: Hand distances (uncompressed signal)

However, because in Fig. 2.25 the signal is uncompressed and very long, it is not very visible from the graph, the moment the hand appears and disappears. Therefore, the graph illustrating the compressed output signal along with the results for the hand distance computation is presented in Fig. 2.26.

Figure 2.26 illustrates the compressed signal and the hypothesis development during the computation process along with the calculated distance of the hand to the device. In this case time is presented in samples.

Table 2.1 shows the precise values of the distance and time for each hand appearance [64].

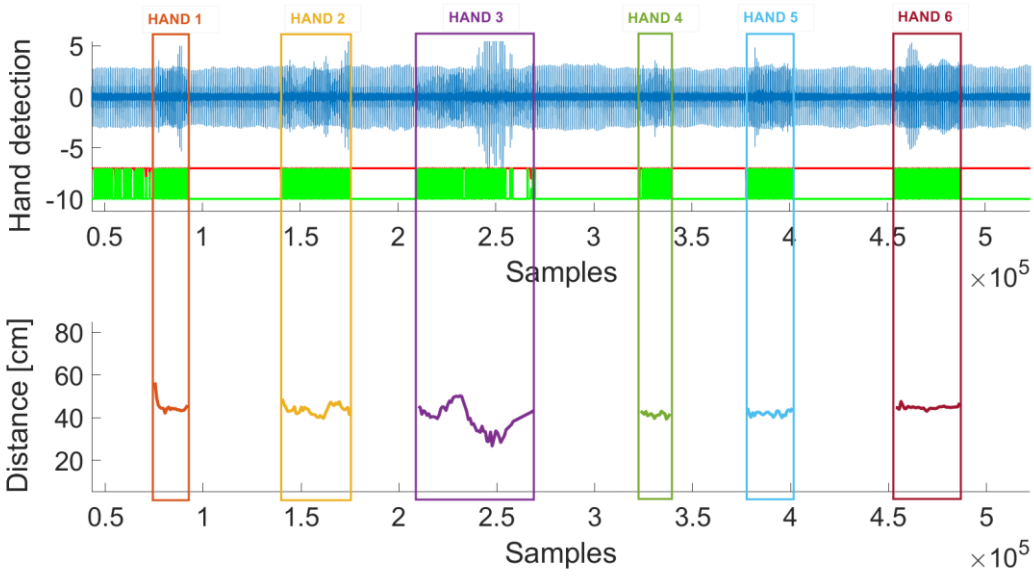

Figure 2.26: Hand distances (compressed signal) [64]

Table 2.1: Hand distances

Hand	Distance [cm]	Time [s]
Hand 1	44	1.9
Hand 2	40-45	3.9
Hand 3	27-50	6.7
Hand 4	41	1.7
Hand 5	40	2.6
Hand 6	45	3.7

Finally, it can be concluded that the experiments with the real data obtained from the ultrasound device show that the algorithm based on hypothesis testing functions reliably and precisely enough and can be used for dealing with a noise cancellation problem. Moreover, using this method, it is possible both to detect the hand and to calculate its distance from the device on the basis of the data only from one microphone. The distance calculation is considered to be the additional contribution to the solution of the hand detection problem.

2.2.2. Comparison of Computation Results of the QRD RLS Algorithm and the QRD RLS Lattice Algorithm

The ultrasound data were also used for the experiments using the QRD RLS algorithm to compare its performance with the algorithm chosen for the implementation on the hardware platform.

The results of the experiments using the QRD RLS algorithm are shown in Fig. 2.27, while the outputs of the QRD RLS Lattice algorithm are presented in Fig. 2.28.

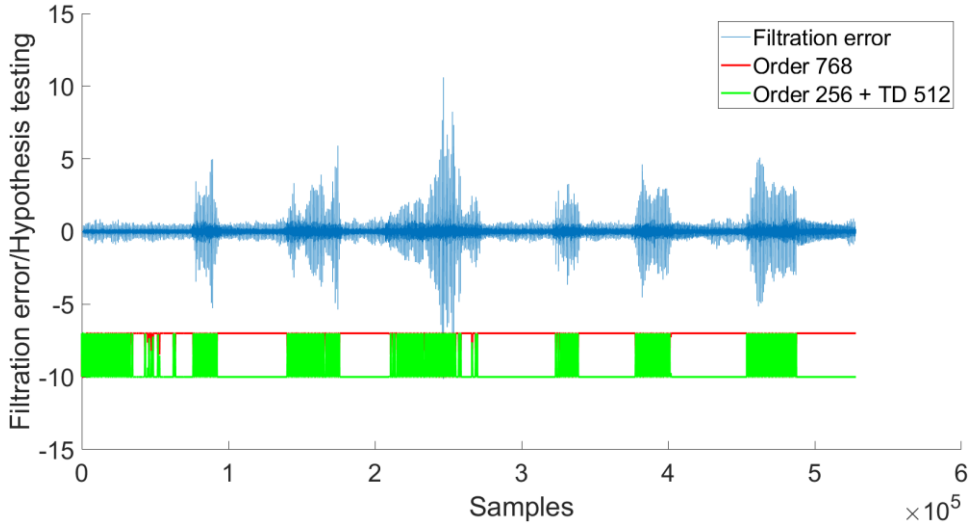

Figure 2.27: Estimation results using the QRD RLS algorithm

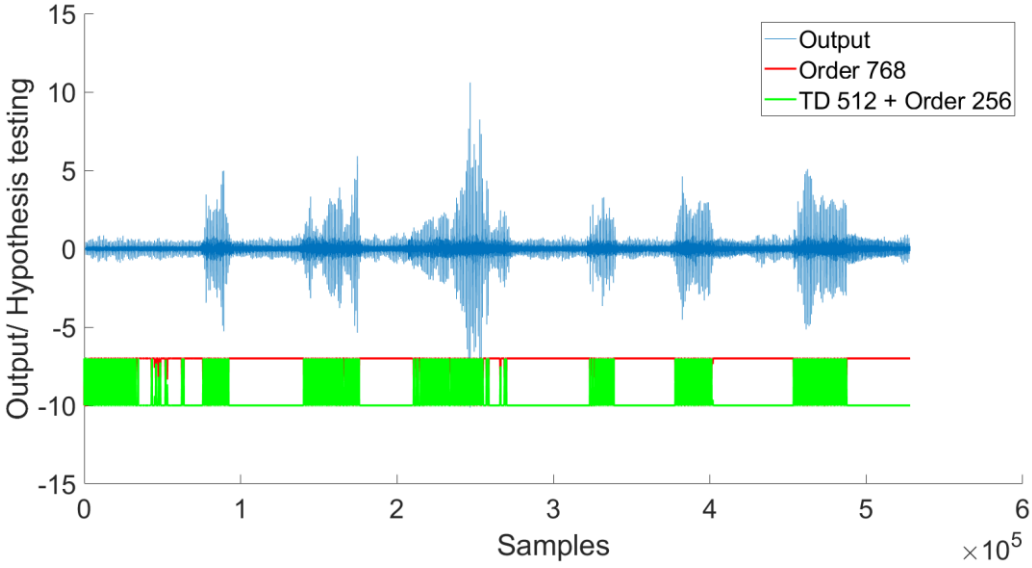

Figure 2.28: Estimation results using the QRD RLS Lattice algorithm

In the beginning of the estimation process both algorithms need some time for learning. After approximately 6000 samples, the estimation process converges to the right values and both algorithms function reliably and precisely recognizing the hand in places where it factually appears.

It is obvious from the graphs in Fig. 2.27 and in Fig. 2.28 that both algorithms have very similar results and are equally accurate. However, the advantages of the QRD RLS Lattice algorithm become more visible and relevant, when the computational time is compared.

The computation was made again on a PC with Intel® Core™ i7-4770 CPU, 3.5GHz: the variable initialization is made in the MATLAB R2019b environment and the algorithm itself is calculated in C code using .mexw64. For the QRD RLS Lattice algorithm, the time of computation considering N=528000 and the highest order n_1=768 is approximately 45s. For the QRD RLS algorithm with the same settings, it is approximately 2330s, i.e. 39 min of computation. It should be noted that the computation was performed on the single core.

As it was mentioned in the beginning, to use the algorithm in the real-time applications, the algorithm has to make computations within 60s. The QRD RLS algorithm has 2301952 operations for one step of computation. The QRD RLS Lattice algorithm has 34850 operations for one step of computation. To process the algorithms in real-time, a PC needs to deliver approximately 20257 MFLOP/s for the QRD RLS algorithm and approximately 307 MFLOP/s for the QRD RLS Lattice algorithm. During the experiments it was found out that given the described setup, the PC (single core) delivered approximately 522 MFLOP/s for the QRD RLS algorithm and approximately 409 MFLOP/s for the QRD RLS Lattice algorithm, computation in the DP arithmetic. It means that in case of the QRD RLS algorithm, it is approximately 39 times slower than it is required for the real-time processing. In case of the QRD RLS Lattice algorithm the PC (single core) manages to compute the algorithm in the range of the real-time processing.

However, it is assumed that the computational speed of the QRD RLS Lattice algorithm will be slowed down while being used on small platforms as far as they have a processor frequency only 1.05GHz and a programmable logic max. 240MHz. Thus, to process the data on a small platform in real time, the QRD RLS Lattice algorithm is not fast enough and still needs accelerating. The methods of the acceleration and the hardware implementation are proposed in the next chapters.

2.3. Discussion

This chapter is devoted to the experiments with the algorithms used for the noise cancellation, speaking more precisely, for a hand detection problem. The experiments are based both on simulations and on the real data from the ultrasound device.

The contribution of the algorithm on this stage is that it computes the prediction/filtration errors and makes the output signal cleaner for further data processing. Moreover, using hypothesis testing the moment of the hand appearance and disappearance can be precisely identified. Another benefit is in the fact that it allows computing the distance of the hand to the device. Thus, only on the basis of the data measured on one microphone, it is possible to obtain the valuable results such as the accurate detection of the hand presence and its distance to the device.

Besides, taking the parameters computed during the experiments with the real data, the simulation close to the reality was performed. It proved that hypothesis testing could be safely applied on the real data measured on the ultrasound device.

The chapter also provides a comparison between the QRD RLS algorithm and the QRD RLS Lattice algorithm and shows why the latter is supposed to be used for the implementation on the HW platform. One of the reasons is that the QRD RLS Lattice algorithm proves to be faster than the QRD RLS algorithm. Besides, the QRD RLS Lattice algorithm has a particular structure, which is easily pipelined and, thus, the algorithm can be accelerated and implemented on the HW platform.

Table 2.2 gives an insight into the computational time of both algorithms during different experiments.

The first number of MFLOP/s in Tab. 2.2 refers to MFLOP/s delivered by the PC (single core) given the described computation settings. The second number of MFLOP/s, which is compared with the first one, is MFLOP/s required for the algorithm to be computed in the range of the real-time processing.

It is obvious from Tab. 2.2 that the computation of the QRD RLS algorithm is very slow and unsatisfactory, especially in case of the real data processing, which constitutes approximately one hour of computation.

As far as the QRD RLS Lattice algorithm is concerned, its computation can be made in real time given the same or faster PC. It is valid both for the simulation and real data experiments.

However, it was a bit slower than it is required during the experiments concerning the simulation based on the real data parameters (the third set of the experiments). It can

Table 2.2: Comparison of algorithms in terms of their computational time

Type of experiments	QRD RLS algorithm		QRD RLS Lattice algorithm	
	Time [s]	MFLOP/s	Time [s]	MFLOP/s
Simulation	580	397 vs 3837	15	232 vs 58
Real data	2330	522 vs 20257	45	409 vs 307
Simulation using the real data parameters	----		71	259 vs 307

be explained by the fact that the inputs based on the real data parameters are computed for the purposes of the simulation. Contrarily, in the real time experiments the inputs are provided by the real measurements.

If compared with the simulation based on the random parameters, the time scale is much longer for the third set of experiments (the simulation with the real data parameters). It constitutes 528000 data samples vs 100000 data samples for the experiments with the random parameters. Therefore, the computational time for the third set of experiments is also longer than for the first one (see Tab. 2.2).

The next chapter describes the method of pipelining and parallel processing of the QRD RLS Lattice algorithm for the purposes of its acceleration and implementation on the HW platform.

2.4. Results and Related Publications

To summarize the main contributions on this stage of investigation, the following outputs should be mentioned:

1. A new approach for hand detection is proposed, where a specific structure of the regression models was designed in a way, which enables not only to eliminate the undesired responses from the objects in the environment, but also to determine the exact moment of the hand appearance.

   The structure of the regression models is proposed in the following way: one regression model has a higher order and analyses all incoming data, while the other regression model has a smaller order and a predefined time delay.

   The assumption is: if there is no hand in front of the device, the model with a higher order has a higher probability as far as the relations between its FIR parameters correspond to the incoming data. When the hand appears in front of the device, the model with a smaller order and the time delay has a higher probability, because the change does not affect the relations between its FIR parameters, whereas in case of the model with a higher order the relations of its FIR parameters are corrupted and do not correspond anymore to the incoming data.

2. The exact moment of the hand occurrence is determined due to incorporated hypothesis testing choosing between two structures of two regression models best appropriate for a description of the present situation.

3. Knowing the exact moment of the hand appearance and due to the nature of the input signal, which is in the form of chirps, it is possible to calculate the distance between the hand and the device.

4. This approach – a new structure of the regression models, hypothesis testing, the distance calculation – was implemented both for the QRD RLS algorithm and the QRD RLS Lattice algorithm for the purpose of a comparison and choosing the more suitable algorithm for the implementation on the HW platform.

5. The principle of the algorithm computation corresponds to the provided output signal, where there is a certain waiting period between the chirps. Thus, the algorithm functions in such a way that in the beginning the algorithm works with a priori initial conditions, then it performs estimation and saves state parameters for the next stage of computation. During the waiting period the algorithm does not compute anything. After the waiting period, it starts again, but using the state parameters saved in the previous stage of computation. Due to the waiting period, the algorithm practically has more time for computation on each stage.

Source codes are available in MATLAB R2019.b and SciLab:

- simulation package for modelling of echo cancellation for ultrasound hand-gesture recognition in form of MATLAB scripts or compiled Win 64bit applications (MATLAB 2018.b (or higher) or Win7 64bit or Win10 64bit are required): http://sp.utia.cz/index.php?ids=results&id=noise-cancellation
- QRD RLS Lattice algorithm :
    - simulation with parameters generated in the MATLAB environment:
        - time-invariant environment model
        - time-variant environment model
    - simulation with parameters calculated from the real ultrasound data
    - experiments with the real ultrasound data
        - hand detection
        - distance computation
- QRD RLS algorithm:
    - simulation with parameters generated in the MATLAB environment:
        - time-invariant environment model
        - time-variant environment model
    - experiments with the real ultrasound data
        - hand detection

QRD RLS algorithms are available
at https://zs.utia.cas.cz/index.php?ids=projects/storaige/disertace_Raissa_Likhonina

Publications related to the research topic are the following:

1. Likhonina R., Kadlec J. Noise cancellation using QRD RLS algorithms. Application note, ÚTIA, 2018.

   Abstract: This Application Note aims to simulate a noise cancellation problem with MATLAB tools. This is purposed for pre-processing process for final gesture recognition application. It also shows advantages and disadvantages of an approach used for a noise cancellation. In applications for gesture recognition the signals can reflect and be detected not only from a desired source (a hand), but from the environment as well, which creates undesired noise and hardens the process of precise gesture identification. Therefore, it is essential to eliminate the signals, which come from other static sources than a hand. For these purposes echo cancellation methods can be used. Echo cancellation is widely and successfully applied in telephony in a way of preventing echo from being created or removing it after it is already present. We will assume that a hand will appear just for a short time period and the additional reflections will act as an additional short period "disturbance". The echo cancellation in this specific case will be based on QRD algorithm with double precision arithmetic and exponential forgetting. The QRD algorithm is also called as an information filter without square root operations. It is based on QRD decomposition of the input/output information matrix. The recursively updated QRD factorization of the information matrix helps to avoid the problem with loss of positive definiteness of the information matrix due to rounding errors and, thus, provides a numerically stable solution. The exponential forgetting is used instead of directional forgetting to keep the perspective of reduction of the computation time by applying the QRD version of the Lattice algorithm. QRD Lattice works only with the exponential forgetting with a constant exponential forgetting factor.

   The application note together with the simulation package is available at http://sp.utia.cz/index.php?ids=results&id=noise-cancellation .

2. Likhonina R. QRD RLS algorithm for hand gesture recognition applications. *In: Proceedings of IWSSIP 2019*, pp. 195-201, Eds: Žagar Drago, Rimac-Drlje Snježana, Martinović Goran, Galić Irena, Vranješ Denis, Habijan Marija, International Conference on Systems, Signals and Image Processing 2019 (IWSSIP 2019), (Osijek, HR, 20190605), DOI: 10.1109/IWSSIP.2019.8787283.

   Abstract: The work is focused on algorithmic technique for detection of hand presence and distance from a hand to a device transmitting ultrasound signals. The described method is based on QRD Recursive Least Squares (RLS) algorithm with double precision arithmetic and exponential forgetting (EF). Modelling of a hand detection problem is based on linear Finite Impulse Response (FIR) based regression models and performed using MATLAB tools. The modelled system comprises an environment model, a hand model and an identification block. A series of experiments testing both time-invariant and time-variant environment models and time-variant hand models show the importance of a correct choice of the EF factor. The experiments prove the accuracy of the algorithm and the possibility to calculate a distance

from the hand to the device. The final version of the algorithm is supposed to be implemented on the embedded Xilinx Zynq device equipped with a microphone and ultrasound transducers.

The article is available at http://library.utia.cas.cz/separaty/2019/ZS/likhonina-0505584.pdf

3. Likhonina R. Hand gesture recognition based on ultrasound technology: pre-processing stage. *In: Proceedings - Research monograph: 2019 8th Mediterranean Conference on Embedded Computing (MECO),* pp. 354-360, Eds: Stojanović Radovan, Jóźwiak Lech, Jurišić Dražen, Lutovac Budimir, Mediterranean Conference on Embedded Computing - MECO'2019 /8./, (Budva, ME, 20190610), DOI: 10.1109/MECO.2019.8760063.

Abstract: This paper describes an approach, which can be used as a pre-processing stage for a hand detection and gesture recognition problem. The approach is based on noise cancellation using QRD Recursive Least Squares (RLS) algorithm with double precision arithmetic and exponential forgetting (EF). The paper discusses algorithmic techniques and presents experiments showing how it is possible to calculate the distance between a hand and a device. A series of experiments were performed. During them a time-variant environment regression model and a time-variant hand model as well as different values of the EF factor were used.

The article is available at http://library.utia.cas.cz/separaty/2019/ZS/likhonina-0505586.pdf

4. Likhonina R. Hand detection algorithm: pre-processing stage. *In: Proceedings of the 17th international conference on informatics in control, automation and robotics,* pp. 695-701, Eds: Gusikhin Oleg, Madani Kurosh, Zaytoon Janan, ICINCO 2020 (17th international conference on informatics in control, automation and robotics), (online conference, FR, 20200707), DOI: 10.5220/0009885206950701.

Abstract: The present work describes a new approach to hand detection based on QRD Recursive Least Squares (RLS) Lattice algorithm and probabilistic approach to system identification. The described method is supposed to be used as a pre-processing stage for a hand gesture recognition application based on ultrasound technology. The approach includes a noise cancellation concept and uses linear Finite Impulse Response (FIR) based regression models in MATLAB environment. Within the algorithm the hypothesis testing technique is implemented. The work shows the results of computation using real data from an ultrasound device. The final version of the algorithm is supposed to be implemented on the embedded Xilinx Zynq device.

The article is available at http://library.utia.cas.cz/separaty/2020/ZS/likhonina-0532163.pdf

CHAPTER 3

Algorithm Optimisation on a PC

The present chapter is devoted to the optimization of the QRD RLS Lattice algorithms and to the implementation of its pipelined version on a PC. It also analyses the obtained results from the viewpoint of computational time and the number of operations per second for different pipe-line versions of the algorithm. In the end the main contributions of this stage of the research and the related publications are discussed.

3.1. Batch Version of the Algorithm

As it was described in the previous section, though the algorithm manages to calculate the outputs within 60s, which corresponds to the duration of the real data obtained from the ultrasound device; still there is a need to optimize and accelerate it as far as it is supposed to be used on a hardware platform with a small memory footprint and with a lower processor frequency.

The version of the algorithm, which was used in the previous experiments, functions in a way that all variables needed for computation, i.e. the inputs, are initialized and saved in the global memory in the MATLAB R2019b environment [70], while the computation is made in the C code using .mexw64 files. Each time step the programme needs to take the variable, to allocate the memory for it, to make computation and to return the variable back. It takes time and slows down the computation process.

In a batch version of the algorithm the computation is performed in a way that during the initialisation, all necessary inputs are prepared in the MATLAB R2019b environment and after that they are copied to the locally allocated memory, which is not moved and which is not allocated all the time. The C functions have pointers on the data and work with the local data only. The programme takes all variables prepared in advance, computes the algorithm and returns the outputs to MATLAB R2019b. After it the hypotheses are calculated. In this way the time of computation can be reduced.

After fulfilling these changes, the computational time is substantially reduced from 45s of a non-batch version to 19s of a batch version of the algorithm, which is almost two times.

The next change, which has to be done in the structure of the algorithm, is to perform computation in the single precision (SP) arithmetic. The point is that the Xilinx Zynq Ultrascale+ hardware platform is best suited for computing the floating point (FP)

operations, which results in the worse accuracy, but in the higher speed. The previous version of the algorithm uses the double precision (DP) arithmetic both during computation in the MATLAB R2019b environment and in C code using .mexw64 files. Therefore, the next step is to use the SP FP operations instead, while computing the algorithm in C code and to control if the accuracy of the outputs is still satisfactory for our purposes.

It is obvious from Fig. 3.1 that the hand presence is detected precisely by switching between the model of a higher order (red coloured), when there is no hand, and the model of a smaller order (green coloured), when the hand appears. Thus, the results obtained during performing the computation in the SP FP arithmetic are very similar to those made in the DP arithmetic and can be considered to be accurate.

As far as the computational time is concerned, it does not decrease very much and constitutes approximately 16s (compared to 19s in the DP arithmetic).

This small decrease of the computational time can be explained by the fact that all inputs are calculated in the MATLAB R2019b environment in DP. After that they are converted into variables in SP FP and the algorithm is computed in C code using .mexw64 files. After the main computation is made, the outputs in SP FP are converted into the outputs in DP and returned to MATLAB R2019b. The DP-SP-DP conversion performed before and after the main computation slows down the process. However, when the algorithm is mapped on the ARM Cortex A53 device, all operations will be made in SP FP and, therefore, it is supposed that the computational time will be potentially decreased.

To be able to implement the algorithm on the ARM Cortex A53 device and to use the benefits of the parallel computation, there is a need to perform several steps.

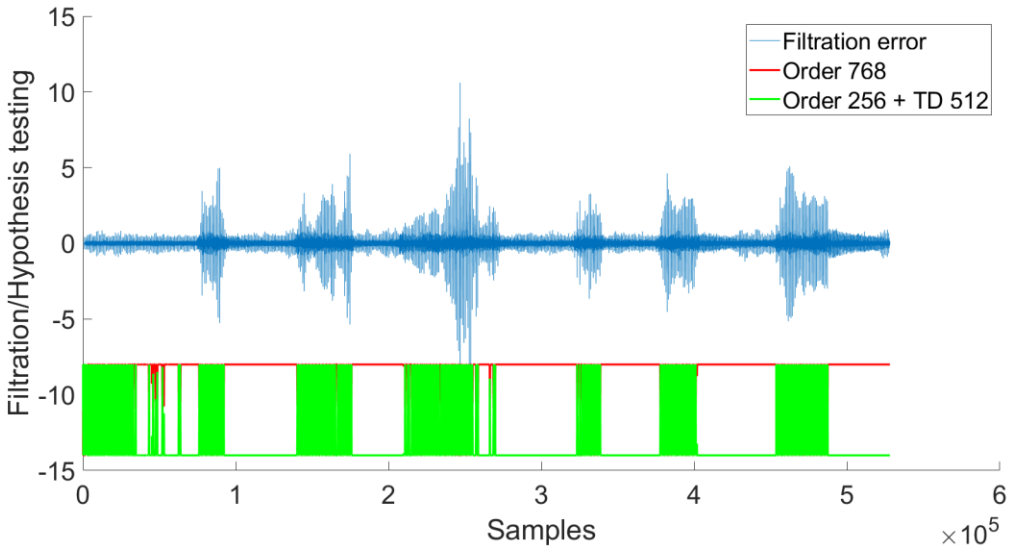

Figure 3.1: QRD RLS Lattice algorithm – single precision arithmetic

As the first step, it is necessary to pipeline it, i.e. to divide it into a number of processes, where the output of one process will be the input for the next one. The systolic structure of the QRD RLS Lattice algorithm allows doing it more or less easily.

Such kind of processes can be executed sequentially or in parallel. The latter is the final goal of the algorithm modification in this chapter.

3.2. Pipelining and Parallel Processing

Since it is necessary to optimize and to accelerate the QRD RLS Lattice algorithm to make it suitable for the implementation on the ARM Cortex A53 device, a pipelining and a parallel processing comes into consideration.

It should be noted that the pipelining and parallel techniques come along with each other. It means that if the algorithm can be pipelined, it can also be processed in parallel.

The pipelining is a technique where a problem is divided into several stages. Each stage is separately executed and is connected with other stages. The block diagram in Fig. 3.2 illustrates the process of the pipelining [27, 98].

It is clear from the block diagram in Fig. 3.2 that each stage S_1-S_n has an input register, which contains the data necessary for processing. The output of the preceding stage serves as the input to the following stage. The computation process in this case is made sequentially from S_1 to S_n, where in the last stage the final computation results are available.

It should be noted, however, that in case with the QRD RLS Lattice algorithm, which has a systolic array structure, the time is fixed. The computation begins from order 0 and runs up to the pre-defined order ***n***, where variables of a higher order are fully dependent on the variables of a lower order. It is not possible to cut the algorithm at any time, but it is possible to define a certain boundary, where the state variables can be saved and moved to the next stage of computation starting from the order ***n+1***. It is also possible to compute the vector of such state variables for each time step ***t, t+1, t+2…t+n*** and to save them for the next stage of the computation process.

This systolic property of the algorithm enables to do the pipelining and parallel processing, which will be described in more details in the next sections of this chapter.

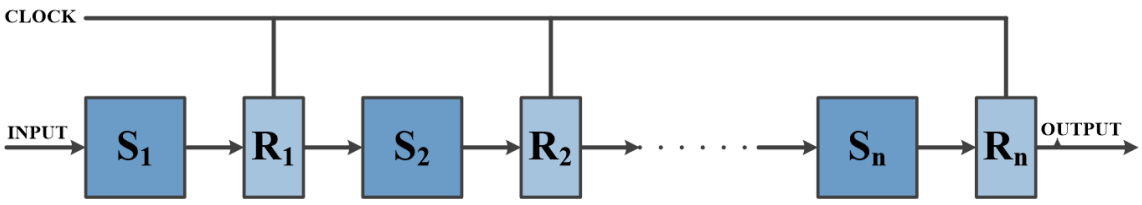

Figure 3.2: Block diagram of the pipelining process [98]

The purpose of the pipelining is to reduce the throughput of the system and make the computation possibly faster [27]. Due to the increase of the system throughput, the power consumption can be also reduced. All this is very critical for the implementation of the algorithm on the Xilinx Zynq Ultrascale+ hardware platform.

However, there are a number of risks connected with the pipelining, which should be considered while performing it.

One of the risks associated with the pipelining is possible conflicts between instructions using the data. Thus, it is important to ensure that the following instruction is not allowed to access the data if the preceding one is working on them [27, 98].

This issue leads to the problem of the data dependency, when the results of the following stage are dependable on the outputs of the previous one [27, 98].

The parallel processing differs from the pipelining in a way that during the pipelining the independent stages are executed in an interleaved manner (see Fig. 3.2), while in the parallel processing it is achieved by duplicating the hardware (see Fig. 3.3) [27].

It means to build a parallel processing structure, the Single-Input-Single-Output (SISO) system needs to be converted into a Multiple-Input-Multiple-Output (MIMO) system [27].

The block diagram in Fig. 3.3 shows that in the parallel processing a problem is also divided into several smaller independent parts, or blocks as it was made in the pipelining; however, they are processed concurrently by multiple processors. The processors communicate via a shared memory [80].

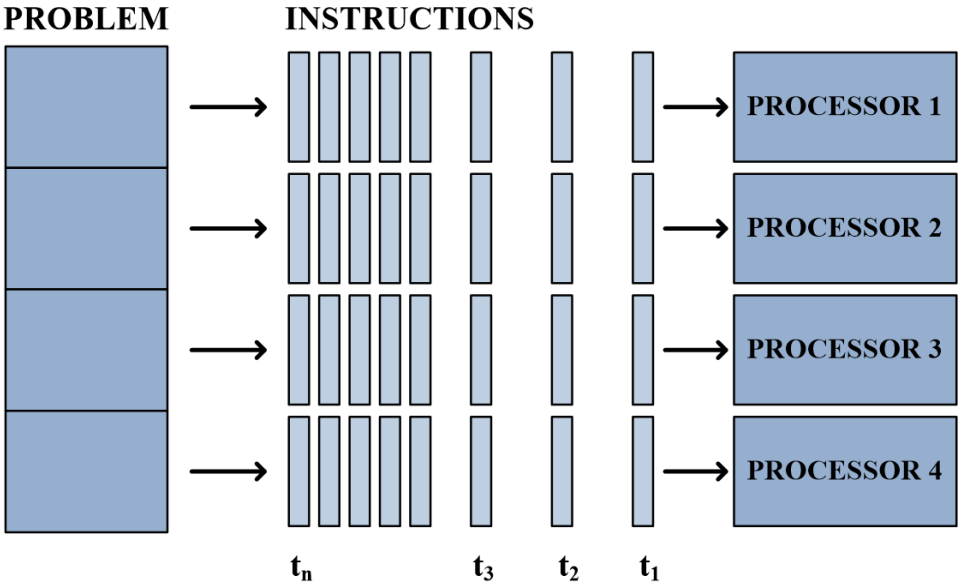

Figure 3.3: Block diagram of the parallel processing [80]

It is also shown in Fig. 3.3 that each part is further divided into the sequences of instructions. The execution of the instructions is coordinated by the control unit. After all stages are executed, the computation results are combined as a part of the whole algorithm [80].

As far as several inputs can be processed at the same time, the sampling rate is reduced. It decreases the power consumption and speeds up the computational time [80].

Thus, the pipelining and parallel processing techniques can be combined to make the QRD RLS Lattice algorithm faster and to decrease the power consumption. Due to its systolic structure, it is possible to divide the algorithm under consideration into several stages for processing it in an interleaved manner. Further on, these stages can be processed in parallel on multiple processors. The next section describes the mentioned approach in more details.

3.3. Parallel Computing Toolbox in MATLAB R2019b

To perform the parallel computation there is a special toolbox in MATLAB R2019b, which is called the Parallel Computing Toolbox. It aims at dividing a large computationally and data intensive problem into smaller parts, after which these parts can be computed using multicore processors. The toolbox comprises high-level constructs such as parallel for-loops, special array types, and parallelized numerical algorithms. It enables to run programmes in both interactive and batch modes. The applications are executed on so-called workers, which run locally. The workers represent the MATLAB R2019b computational engines [71].

To compute in parallel, it is necessary to use “parfor loop”, which allows independent iterations to run in parallel on multicore central processing units (CPUs). The parallel pools are automatically created by “parfor”. The file dependencies are automatically managed [71].

The safest way to test this tool with the algorithm under consideration is to compute two identification models of different orders simultaneously. It can be done without large changes in the algorithm as far as there is no dependency between the models. There is a need only to provide both models with the input data (see Fig. 3.4).

By applying the MATLAB R2019b Parallel Computing Toolbox, two identification models M_1 (the identification model of a higher order) and M_2 (the identification model of a smaller order) are computed as two processes on two cores in parallel.

For these purposes MATLAB creates two server MATLABs on two cores of a PC, i.e. MATLAB worker 1 and MATLAB worker 2 in Fig. 3.4. It copies all necessary input data for the model M_1 and for the model M_2 to both of its workers. After receiving the necessary inputs, both workers process the data independently and simultaneously. As soon as the computation is fulfilled, the results are sent to the main programme, where, based on the outputs, hypothesis testing is performed.

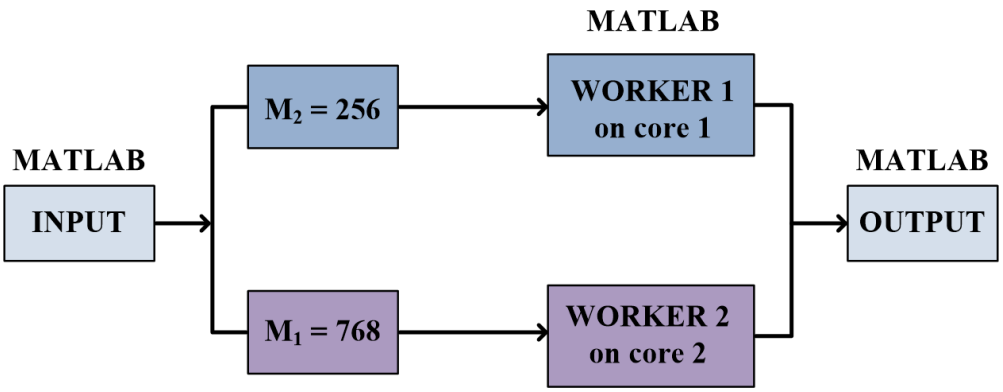

Figure 3.4: Parallel processing in MATLAB R2019b

The final computation results are the same as they were discussed in the section about computing the real data using the QRD RLS Lattice algorithm.

As far as the computational time is concerned, it is 16s for the computation in DP and 14s for the computation in SP FP on the same PC described in the previous sections. If compared with the previous results, which was 19s for the computation in DP and 16s for the computation in SP FP, it is obvious that the algorithm is not much faster than it was before without using the Parallel Computing Toolbox. It is due to the fact that there is still a very large volume of the data, which has to be duplicated for the parallel processes. Thus, there is a need to divide the problem into smaller parts to reduce the amount of the data for each core of the processor.

To accelerate the algorithm, each identification model should be pipelined into more parts. There are several versions of the pipelining and parallel processing considered and analysed in this section in terms of the computational time and the number of operations per seconds. These versions are the following:

1. Two processor cores:
    a. one process of 256 for M_2; one process of 768 for M_1,
    b. one process of 256 for M_2; three processes for M_1, each of 256.

The first case is to process the data on two cores of the processor. It has two possibilities. The first one is to process two identification models, - one of which is of order 256 and another one is of order 768, - on two separate processor cores as it was described above, which does not speed the computation very much.

The second possibility is to divide computation for M_1 into 3 parts, each of which will be 256 (see Fig. 3.5).

Thus, there will be one process T_1=256 for M_2 and three processes T_2, T_3, T_4 for M_1, each of which is of size 256. The process T_1 is processed on one processor core and the processes $T_2 \ldots T_4$ are processed on another processor core sequentially.

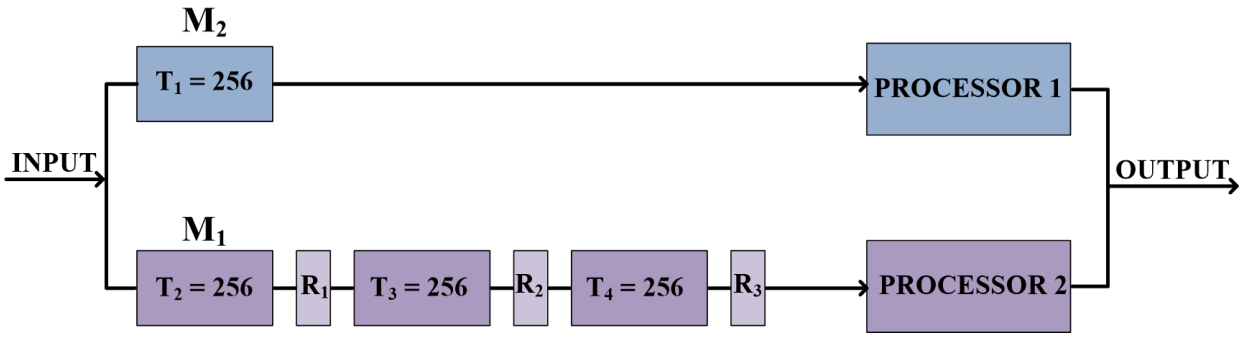

Figure 3.5: Parallel processing (2 processor cores)

2. Four processor cores:
    a. one process of 256 for M_2 processed by one processor core; three processes, each of size 256, for M_1 are processed by three processor cores.
    b. two processes, each of 128, for M_2 processed by one processor core; six processes, each of 128, for M_1 processed by three processor cores.

As far as M_1 was already pipelined into three smaller parts, it is logically to try to compute each part in parallel (see Fig. 3.6).

Thus, in case of four processor core processing, there is one processor core, which computes the identification model M_2 of order 256 and there are other three processor cores, which perform the computation of three processes, each of 256, for the identification model M_1 of order 768. The difference with the two processor core version is that in case of four processor core computation, all processes can be processed in parallel.

The second variant of four processor core version of the computation has both models M_1 and M_2 pipelined into smaller parts, each of which is equal to 128. The block diagram of this case is shown in Fig. 3.7.

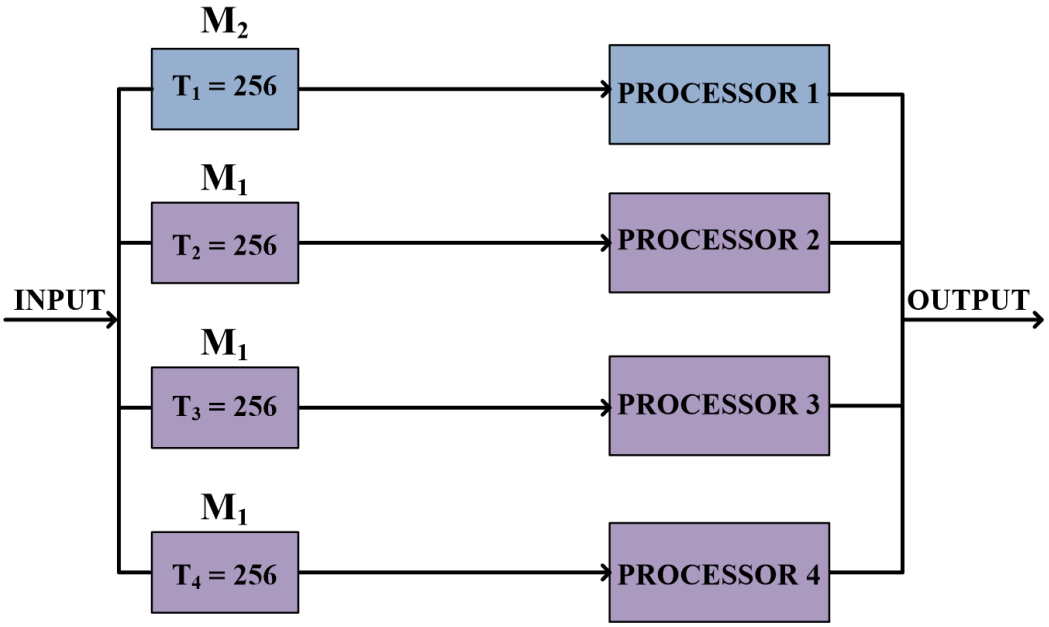

Figure 3.6: Parallel processing (4 processor cores)

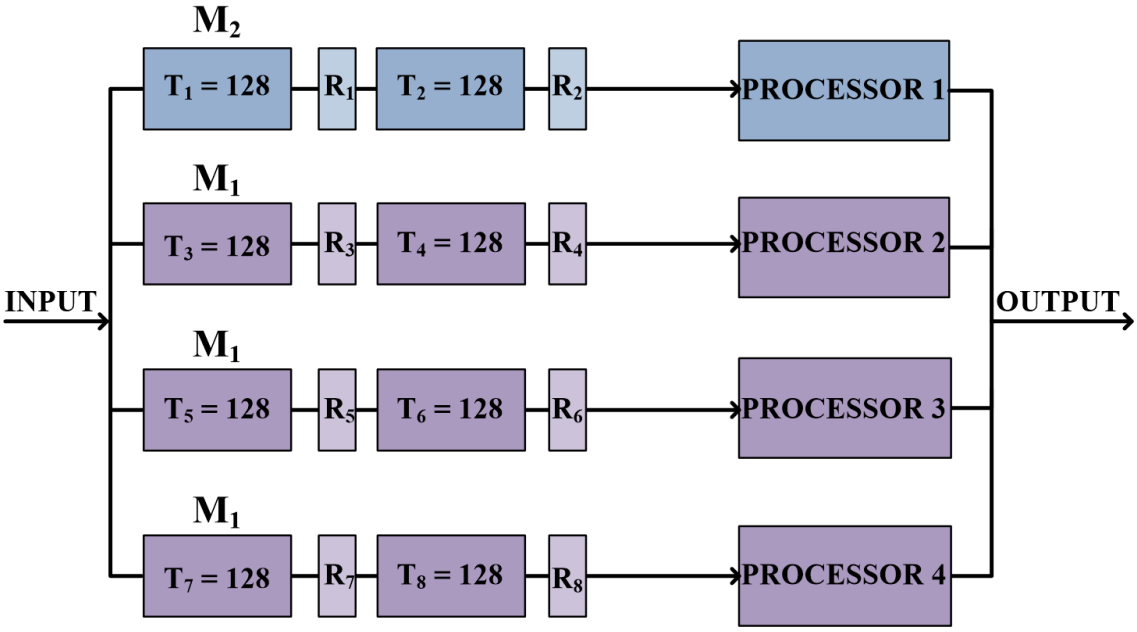

Figure 3.7: Parallel processing (4 processor cores, 8 processes)

It is obvious from the block diagram that M_2 is divided into two processes T_1 and T_2, each of which has size of 128. Both processes are computed on one processor core sequentially.

M_1 is divided into six processes $T_3 \ldots T_8$ and can be processed on three processor cores. Each processor core can process two processes. Thus, processor 2 will process T_3 and T_4 sequentially, processor 3 will process T_5 and T_6 sequentially and finally

processor 4 will process T_7 and T_8 sequentially. In the final implementation the processor cores can execute their operations in parallel.

This variant of four processor core computation leads to the last case considered in this section – eight processor core processing.

3. Eight processor cores: two processes, each of 128, for M_2 processed by two processor cores; six processes, each of 128, for M_1 processed by six processor cores.

As far as both identification models are already pipelined and the total number of processes is 8, they can be logically executed on the eight core PC (see Fig. 3.8).

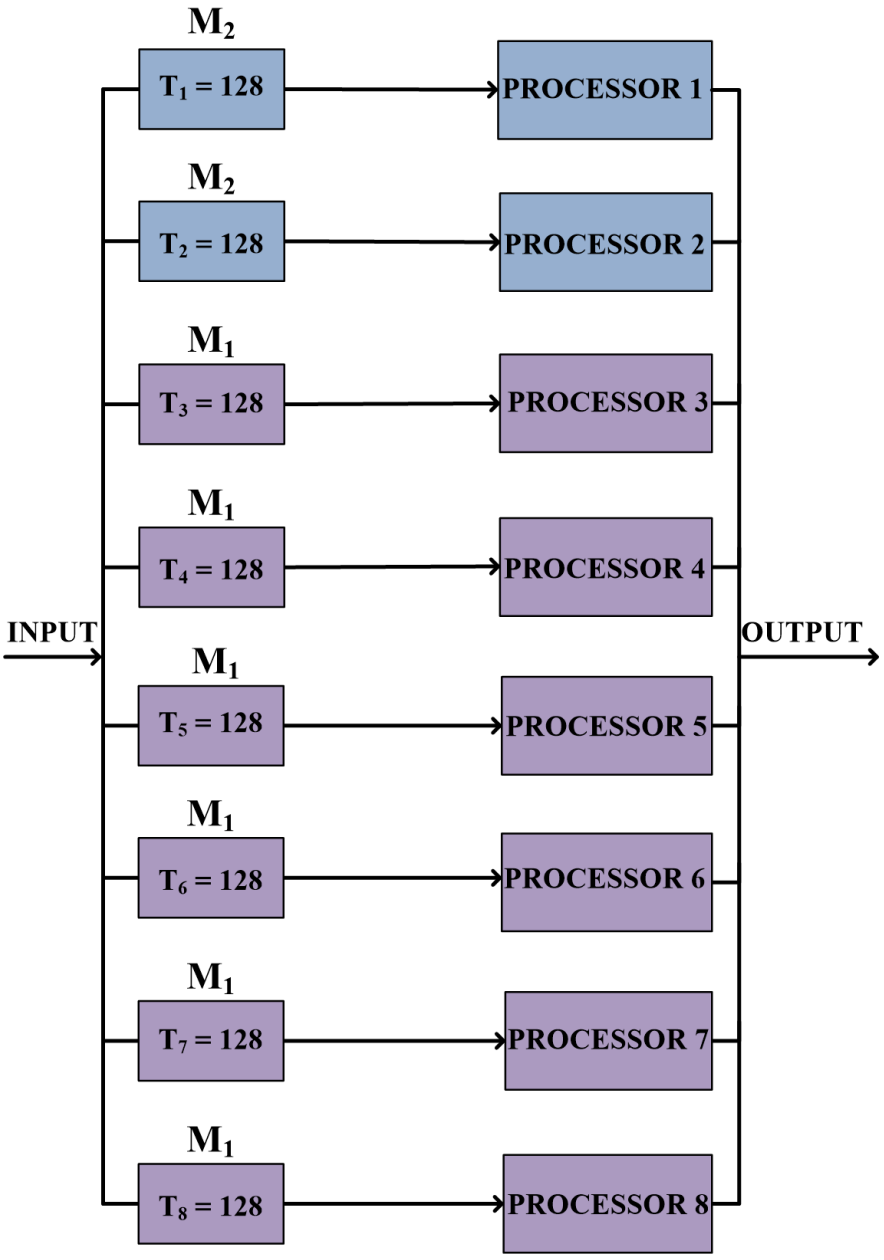

Figure 3.8: Parallel processing (8 processor cores)

Figure 3.8 illustrates that each thread of size 128 is executed on a separate processor core. Thus, the computation can be parallelized in this way.

However, to make cores run simultaneously, the algorithm should be divided not only in terms of its size, but also in terms of time. It is necessary to prescribe time for each process, to decide, which data from the preceding stage of computation will be the inputs for the next stage of the algorithm and to ensure the data independency while processing separate parts of the algorithm.

The matrix of the state parameters in our case, i.e. the outputs, which will be the inputs for the next stage of computation, comprises 9 elements. These parameters should be transferred from one stage of the computation process to another as it is shown in Fig. 3.9.

In terms of time the algorithm time step, i.e. N=528000 samples, was divided into 10, 20, 50, 100 and 200. Due to the pipelining and parallel processing, there is no need to copy the whole matrix of the input data for each process, but only the data this or that process is supposed to work on at that certain moment (see Fig. 3.10). It decreases the amount of the data each processor should deal with.

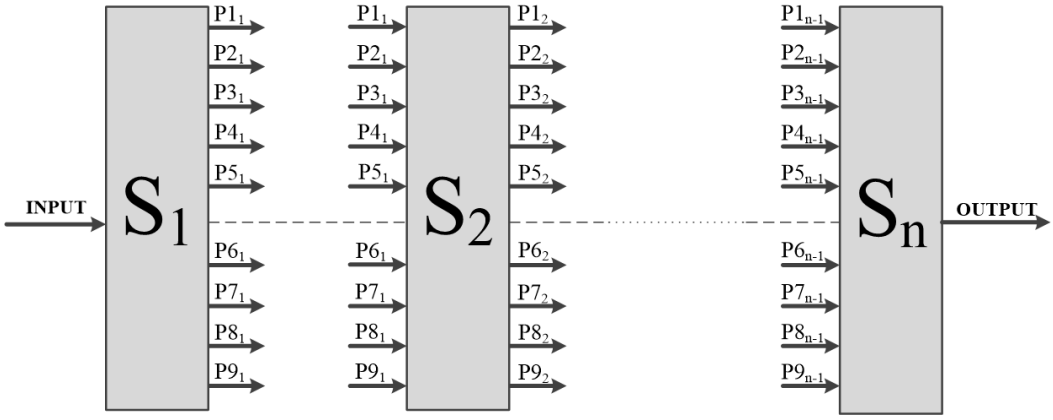

Figure 3.9: State parameter transmission

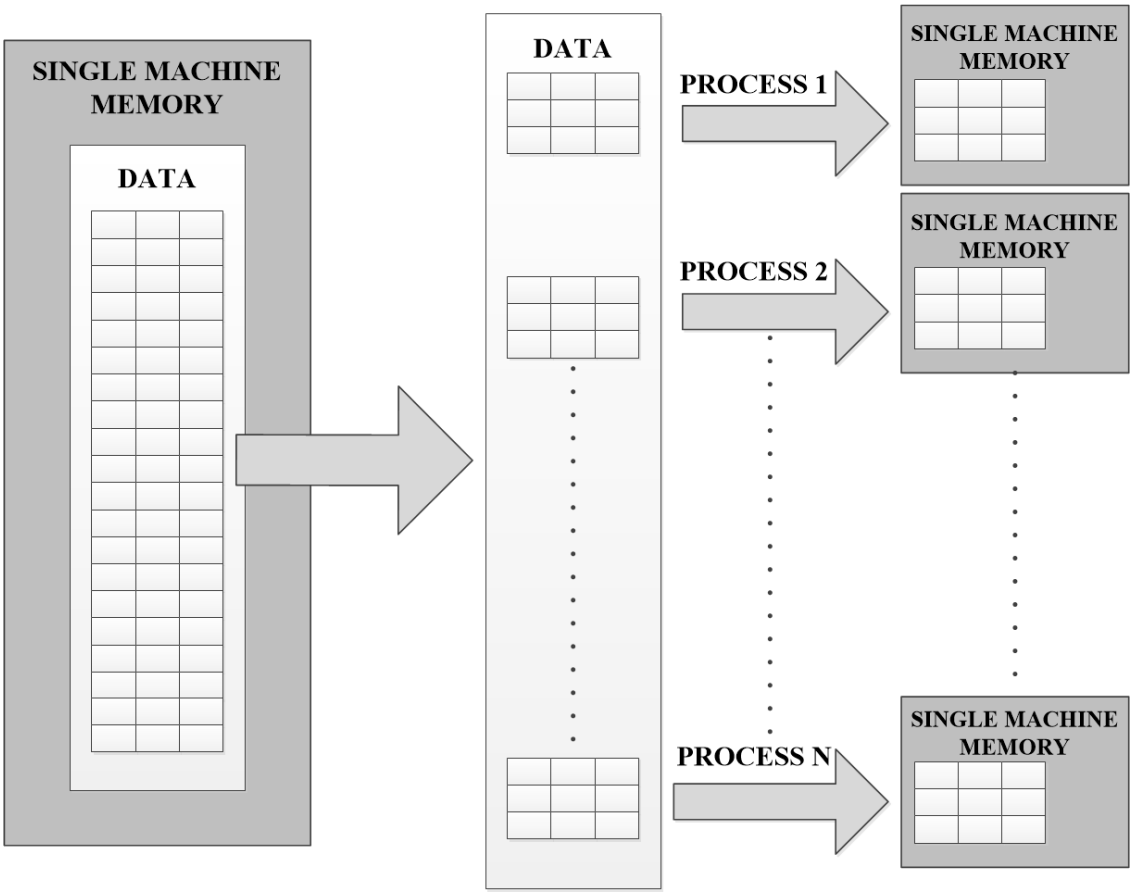

Figure 3.10: Data transfer for parallel processing

It should be noticed that the smaller step is, the faster the algorithm should be. It is logical to divide time into smaller steps. However, it is not always the case, because the data communication is also increasing. It results in slowing down the computation process. So, there is a need to analyse, which is more appropriate for the situation under consideration, and to find a golden middle. The results of experiments of the parallel processing are illustrated in Tables 3.1-3.5 and in Fig. 3.11-3.12.

Let us remind that in order the algorithm could be used for the real time applications, it should compute outputs within 60s and deliver at least 307 MFLOP/s. Table 3.1 shows the number of MFLOP per second for each version of the pipelined processing given N=528000.

Figure 3.11 shows the computational time needed for processing the algorithm in different versions of the parallel implementation when time step N is divided into smaller parts. The graph in Fig. 3.11 illustrates the example of the computational time received while computing in the SP FP arithmetic.

Table 3.1: Number of operations per second (for N=528000)

Number of processors	MFLOP/s
1	307
2 (1 processor)	307
2 (2 processors)	307
4	316
8	316

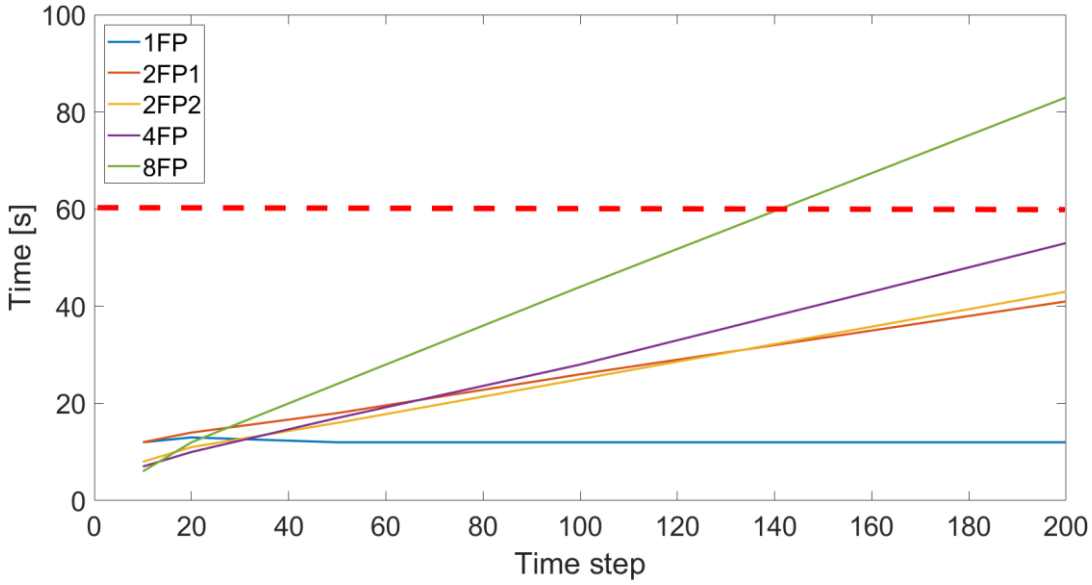

Figure 3.11: Time needed for the algorithm computation given a different number of processors

It is obvious that the fastest version of the algorithm for the final results of computation is when time step ***N*** is divided by 10. The time of computation is increasing with the increase of the amount of the data needed to be processed at each stage. However, up to ***N*** divided by 140 (for 8 cores), the algorithm is computed within required 60s.

More detailed results for the computation both in the DP and SP FP arithmetic are presented in Tab. 3.2 and Tab. 3.3 respectively.

From the tables it is clear that when ***N*** is divided by 200, for the eight processor core versions the final computational time is not sufficient as far as it constitutes 83s and 81s for the computation in DP and SP FP respectively. For other versions of the parallel processing, it is still within 60s.

However, we should also consider the computational time for each separate output, which is presented in the last columns of the tables. Thus, for the fastest version of the algorithm, i.e. ***N***/10, it is 6s. It means that only once in 6s the hypotheses can obtain the necessary data and decide if there is a hand or there is no hand in front of the device. It is obvious that for the real-time applications this case is not appropriate.

Table 3.2: Computational time (DP arithmetic) (for N divided into smaller parts)

Time step	Time [s]	Time [s]	Time [s]	Time [s]	Time [s]	Outputs [s]
	1DP	2DP1	2DP2	4DP	8DP	
10	16	15	10	8	7	**6**
20	16	16	12	11	13	**3**
50	16	21	18	18	25	**1.2**
100	16	29	27	30	45	**0.6**
200	16	44	45	54	**83**	**0.3**

Table 3.3: Computational time (SP arithmetic) (for N divided into smaller parts)

Time step	Time [s]	Time [s]	Time [s]	Time [s]	Time [s]	Outputs [s]
	1FP	2FP1	2FP2	4FP	8FP	
10	12	12	8	7	6	**6**
20	13	14	11	10	12	**3**
50	12	18	16	17	24	**1.2**
100	12	26	25	28	44	**0.6**
200	12	41	43	53	**81**	**0.3**

In the second case ***N*** is divided by 20, i.e. that the outputs are available every 3s, which is also not enough for the real-time applications.

For ***N*** divided by 50, it constitutes 1s, which is already close to what is required. And only when ***N*** is divided by 100 or a higher number, it gives a less than 1s period of the identification process, where the data needed for hypothesis testing are provided and, thus, the hand presence is detected.

Figure 3.12 shows the number of operations per second, which each analyzed version of the algorithm requires. It is clear from the graph that the number of operations per second is decreasing with the higher amount of the data needed to be duplicated from MATLAB for performing the parallel processing given a higher number of processors, which are available. Thus, the least MFLOP/s is for the eight processor core version of the algorithm given ***N***/200.

More detailed information both for the DP and SP FP computation is given in Tab.3.4 and Tab.3.5.

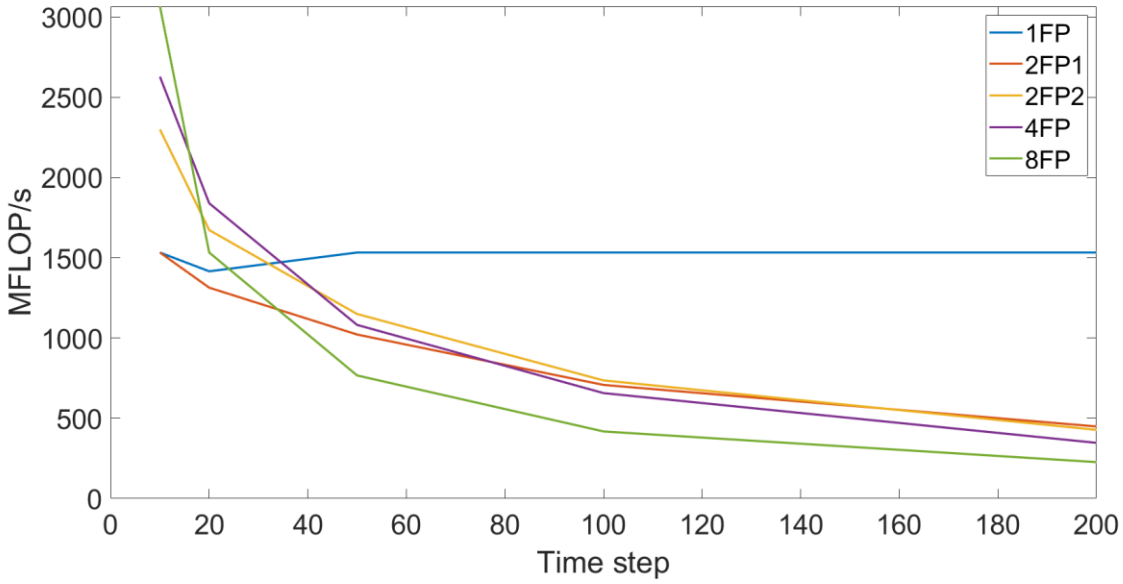

Figure 3.12: Number of operations per second given a different number of processors

Table 3.4: Number of operations per second for the algorithm in the DP arithmetic

Time steps	MFLOP/s	MFLOP/s	MFLOP/s	MFLOP/s	MFLOP/s	Outputs [s]
	1DP	2DP1	2DP2	4DP	8DP	
10	1150	1227	1840	2629	2629	**6**
20	1150	1150	1533	1840	1416	**3**
50	1150	876	1022	1082	736	**1.2**
100	1150	635	682	657	409	**0.6**
200	1150	418	409	347	**222**	**0.3**

Table 3.5: Number of operations per second for the algorithm in the SP FP arithmetic

Time steps	MFLOP/s	MFLOP/s	MFLOP/s	MFLOP/s	MFLOP/s	Outputs [s]
	1FP	2FP1	2FP2	4FP	8FP	
10	1533	1533	2300	2629	3067	**6**
20	1416	1314	1673	1840	1533	**3**
50	1533	1022	1150	1082	767	**1.2**
100	1533	708	736	657	418	**0.6**
200	1533	449	428	347	**227**	**0.3**

Because hypothesis testing has to be in real time and because the algorithm is supposed to be processed on more than two cores, the parallel version of the algorithm given $\boldsymbol{N}$ divided by 100 is the best variant from the described cases so far.

Thus, the four processor PC delivered the SP FP performance for the QRD RLS Lattice algorithm constitutes 657 MFLOP/s, while for eight processors it is only 418 MFLOP/s. In both cases it is enough for the real-time processing.

According to the experiments presented above, it is clear from Tab. 3.4 and Tab. 3.5 that the MATLAB R2019b Parallel Toolbox does not parallelize well as far as the processing on eight processors is slower than the processing on four processors (44s vs 28s respectively). It can be explained by the fact that MATLAB needs to duplicate all necessary variables and functions to all its created workers. It means that given $\boldsymbol{N}$ divided by 10, MATLAB copies data 10 times for each its worker. With the higher division number and the higher number of created workers, the data duplication, i.e. the communication overhead, is increasing and it slows down the computation process significantly. Therefore, in this particular case parallel processing was not very efficient. However, it should be noted that all possible steps for the optimization of the algorithm on a PC including converting from DP to SP, using a batch structure, the pipelining and parallel processing, were made.

The next step is to implement one of the pipelined versions of the algorithm on the ARM Cortex A53 device. As the golden model, the four processor core version is considered to be used as far as the Xilinx Zynq Ultrascale+ has the quad-core ARM Cortex A53 processor.

In this case the algorithm acceleration is supposed to be substantial, because the data communication will not be so demanding. There will be one shared memory, where the data will be stored. To process the algorithm in parallel, a number of threads will be created. They will work with the exact area of the memory, so there will be no need to duplicate all data to each thread. Only the state parameters are required to be copied. It will decrease the load of the data communication.

However, in case the HW implementation does not give satisfactory results in terms of the computational speed, there will be a need to use the accelerators in the FPGA logic of the device. Then, the version of the algorithm with more cores will come into consideration.

3.4. Results and Related Publications

The outputs and contributions of the research on the present stage of the algorithm development are the following:

1. The QRD RLS Lattice algorithm is pipelined for parallel processing on 2, 4 and 8 processor cores using the MATLAB R2019b Parallel Toolbox.
2. The algorithm is able to give the complete development of all necessary variables including hypothesis probabilities every 0.6s. It means that it enables to reconstruct what the hand did in the time period of 0.6s. It can be beneficial for the applications when on the basis of the hand distance from the device simple gestures can be identified.

The following source codes are available (MATLAB R2019b or a higher version and the MATLAB Parallel Toolbox are required)
at https://zs.utia.cas.cz/index.php?ids=projects/storaige/disertace_Raissa_Likhonina:

- QRD RLS Lattice algorithm - batch version:
  - double precision arithmetic
  - single precision arithmetic
- QRD RLS Lattice algorithm – 1 processor core
  - 1 core, 1 process, double precision arithmetic
  - 1 core, 1 process, single precision arithmetic
- QRD RLS Lattice algorithm – 2 processor cores
  - 2 processor cores, 2 processes, double precision arithmetic
  - 2 processor cores, 2 processes, single precision arithmetic
  - 2 processor cores, 4 processes, double precision arithmetic
  - 2 processor cores, 4 processes, single precision arithmetic
- QRD RLS Lattice algorithm – 4 processor cores
  - 4 processor cores, 4 processes, double precision arithmetic
  - 4 processor cores, 4 processes, single precision arithmetic
  - 4 processor cores, 8 processes, double precision arithmetic
  - 4 processor cores, 8 processes, single precision arithmetic
- QRD RLS Lattice algorithm – 8 processor cores
  - 8 processor cores, 8 processes, double precision arithmetic
  - 8 processor cores, 8 processes, single precision arithmetic

Publications related to the research topic are the following:

1. Likhonina R., Uglickich E. Hand detection application based on QRD RLS Lattice algorithm and its implementation on Xilinx Zynq Ultrascale+. *In: Neural Network World*, 32(2), pp. 73-92, 2022, 10.14311/NNW.2022.32.005.

   Abstract: The present paper describes hand detection application implemented on Xilinx Zynq Ultrascale+ device, comprising multi-core processor ARM Cortex A53 and FPGA programmable logic. It uses ultrasound data and is based on adaptive QRD RLS Lattice algorithm extended with hypothesis testing. The algorithm chooses between two use-cases: (1) “there is a hand in front of the device” vs (2) “there is no hand in front of the device”. For these purposes a new structure of the identification models was designed. The model presenting use-case (1) is a regression model, which has the order sufficient to cover all incoming data. The model responsible for use-case (2) is a regression model, which has a smaller order than the model (1) and a certain time delay, covering the maximal distance where the hand can possibly appear. The offered concept was successfully verified using real ultrasound data in MATLAB optimized for parallel processing and implemented in parallel on four cores of ARM Cortex A53 processor. It was proved that computational time of the algorithm is sufficient for applications requiring real-time processing.

   The article is available at nnw.cz/obsahy22.html

CHAPTER 4

Algorithm Implementation on the Xilinx Zynq Ultrascale+ Cortex A53 ARM 4 Cores, 1.05 GHz Platform

This chapter describes the implementation of the QRD RLS Lattice algorithm on the HW platform from the Trenz Electronic, a German provider of the development services in the electronics industry.

The first section is devoted to the device used for the implementation, while the following sections present the implementation steps and the ways of the algorithm acceleration. Finally, the main outputs of the research on this stage of the development are discussed.

4.1. Trenz Electronic Platform Description

For the purposes of the implementation of the QRD RLS Lattice algorithm, the Trenz Electronic TE0808 SoC and the Trenz Electronic TEBF0808 carrier board are used. A short description of the platform together with the implementation steps is presented below.

The Trenz Electronic TE0808 is an industrial MPSoC module, which comprises the Zynq UltraScale+ ZU9EG-ES1, the four core ARM processor of frequency 1.05 GHz, the programmable logic max. 240 MHz, 64-bit DDR4 (max. 4GB), dual SPI boot Flash in parallel (512MB maximum), user I/Os, B2B connectors.

It is of size 52x76 mm and it requires 3.3V power supply. The TE0808 module equipped with the components is illustrated in Fig. 4.1 in more details [34].

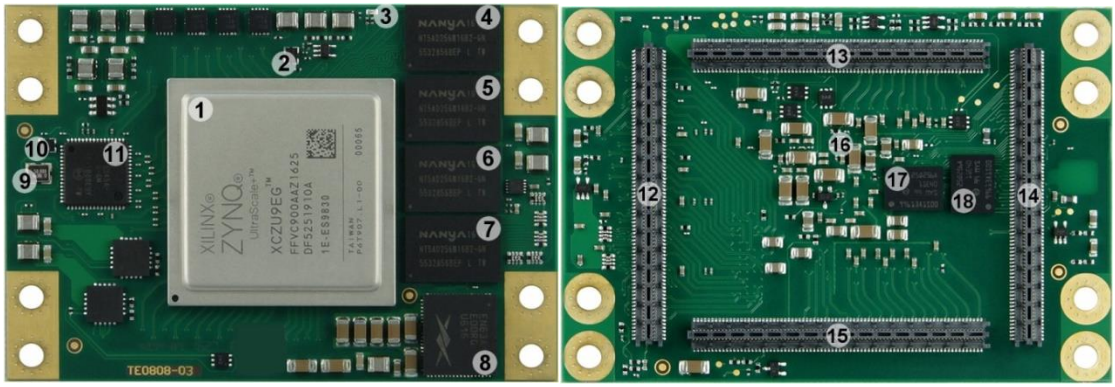

Figure 4.1: Trenz Electronic TE0808 MPSoC module [34]

1 - Xilinx Zynq UltraScale+ XCZU9EG MPSoC, 4-7 – 256Mx16 DDR4-2400 SDRAM, 17-18 - 256Mb serial NOR Flash memory.

Here only several components of the module are mentioned. For more details about the TE0808 MPSoC module components, please, refer to [34].

The Trenz Electronic TEBF0808 carrier board is a baseboard, which is used for the module described above. It comprises on-board components, which serve for testing and evaluating the modules compatible with this board. The board can be fitted into a PC enclosure. Fig. 4.2 illustrates the carrier board and its components [35].

The TEBF0808 carrier board has several JTAG interfaces, which serve for programming both the System Controller CPLDs and the Zynq Ultrascale+ MPSoC. It is also equipped with two System Controller CPLDs, which is the central system management unit. They control signals, which are logically linked by the implemented logic of the CPLD firmware. The CPLD firmware generates the output signals to control the system, the on-board peripherals and the interfaces. The System Controller also monitors the power-on sequence and displays the programming state of the FPGA module [35].

The TEBF0808 carrier board has also several EEPROMs for configuration and general user purposes, an embedded MMC memory with a memory density of 32Gb (4GB) and an on-board Gigabit Ethernet PHY [35].

The carrier board size is 170mm x 170 mm [35].

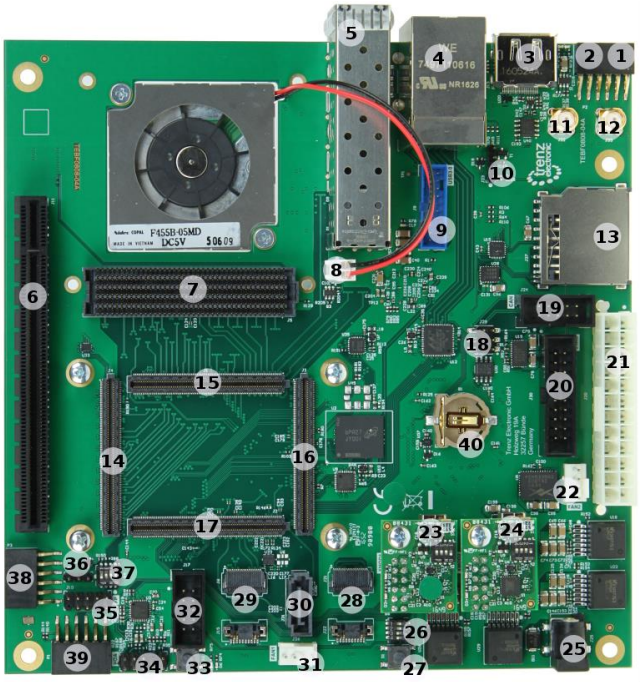

Figure 4.2: Trenz Electronic TEBF0808 carrier board [35]

7 – FMC HPC connector, 13 – MMC Card socket, 14-17 – a place for a module.

Here only several parts of the carrier board are mentioned. For more detailed information about the TEBF0808 carrier board, please, refer to [34]

The working prototype of the device used for the algorithm computation is presented in Fig. 4.3.

As it is shown in Fig. 4.3, the prototype consists of the Trenz Electronic platform, a computer, a display where the computation results can be viewed, a ventilator to cool the Xilinx Zynq Ultrascale+ Cortex A53 ARM processor and other accessories.

Figure 4.3: Prototype for the QRD RLS Lattice algorithm computation

The board is connected via the Ethernet cable to a mini computer UMAX U-Box N41 with Intel Celeron Quad Core N4100 (Gemini Lake) 2.4 GHz, Intel UHD Graphics 600, 4GB DDR4 RAM, 64GB eMMC. The prototype device allows performing the algorithm computation, viewing the results on the display and making necessary changes of the host SW by cross-compilation of the host code for the ARM processor. Besides, it is portable and, thus, it enables a certain level of flexibility and convenience while working with it.

4.2. Algorithm Implementation on the Xilinx Zynq Ultrascale+ Cortex A53 ARM Processor, 4 Cores, 1.05 GHz

The first step of the implementation is to use a batch version of the algorithm, which is not parallelized yet and is computed on one core. It is a safe way to implement the algorithm on the Xilinx Zynq Ultrascale+ Cortex A53 ARM device and to ensure that it functions in a way it is supposed to do.

Initially, the TE0808 is equipped with the SciLab SW interpret, which is similar to MATLAB R2019b [70] and which allows computing the algorithm under consideration on the HW platform.

During the experiments it was shown that the computational time was very long and constituted approximately 196s for DP and 189s for SP FP for the data length of 60s.

Moreover, this time is only for computing two identification models and does not consider hypothesis testing. Due to some specificity of the computation of some functions in the SciLab SW interpret, the hypothesis computation lasts very long: 729s and 727s in DP and SP FP respectively.

Even taking the computational time for computing the identification models only, it is much than twice slower than it is necessary for the real-time processing. For a comparison, the computational time on a PC for a batch version of the algorithm was 19s.

The slowdown of the computational time was expectable as far as the Xilinx Zynq Ultrascale+ Cortex A53 ARM processor has only 1.05 GHz processor frequency compared with 3.5 GHz of the PC, Intel® Core™ i7-4770 CPU.

It is also necessary to note that for the QRD RLS Lattice algorithm the ARM device delivers only 94 MFLOP/s (for DP) and 98 MFLOP/s (for SP) instead of 307 MFLOP/s required for the real-time computation. It is obvious that the acceleration of the algorithm is needed.

To optimize and to accelerate the algorithm under consideration, it is necessary to fulfil a number of steps.

Firstly, it is reasonable if the Xilinx Zynq Ultrascale+ Cortex A53 ARM device does not use the SciLab SW interpret for generation of the input values. Instead, so-called header files, .h, can be created, where all input data will be saved. These header files are then copied to the local memory of the Xilinx Zynq Ultrascale+ Cortex

A53 ARM device and the processor can work with the necessary data addressing to its local memory. It should be kept in mind, however, that the local memory is limited in its size and, therefore, it is necessary to ensure that there is enough space for the data needed for the computation.

After the data are copied, the Xilinx Zynq Ultrascale+ Cortex A53 ARM processor runs appropriate functions, which compute the algorithm. The computation was presented both in the DP and SP FP arithmetic to compare the results.

Differently from the data processing in MATLAB R2019b, in case of SP FP computation, there is no need to convert DP into SP in the beginning and in the end of the computation process. It may potentially accelerate the process of computation too, but not substantially.

For the verification purposes, the output values computed in MATLAB R2019b were saved as .h files and used as the reference values while verifying the outputs computed on the Xilinx Zynq Ultrascale+ Cortex A53 ARM device.

The computational time for the algorithm in the DP and SP arithmetic constitutes approximately 200s and 188s respectively; however, in this case including the hypothesis testing computation. It means the Xilinx Zynq Ultrascale+ Cortex A53 ARM device delivers only 92 MFLOP/s (for DP) and 98 MFLOP/s (for SP) instead of 307 MFLOP/s required for the real-time computation.

Secondly, as far as the Xilinx Zynq Ultrascale+ Cortex A53 ARM device under consideration has four processor cores, it means there is a great potential already on this stage of the development that the algorithm can be processed in parallel and the computational time can be decreased substantially, i.e. approximately up to 3.5x. It could be sufficient for the real-time processing.

However, there are a number of challenges, which should be taken into consideration while making the implementation.

The first challenge is in the data distribution and data storage. The parallel processing means that the processors should communicate the data locally. Thus, it is necessary to think about how large a separate stage of the algorithm can be to ensure that all necessary data are located in the local memory.

The second challenge is to control that the algorithm parallelization functions in a way it should function and the obtained results are correctly computed. During the algorithm implementation, it is possible to stop the computation process at any time except for processing a microinstruction and to see what there is in the beginning and in the end of a computation step. Thus, in this way it is possible to develop and to debug the whole computation process.

The algorithm optimization at this stage of development is performed in two main steps. The first step is to divide the QRD RLS Lattice algorithm into smaller parts and to make computation run in parallel. When succeeded, the next step is to perform hypothesis testing also in parallel and in a way that their outputs, i.e. the probabilities for each model, are available at time step corresponding to the real-time processing.

In the beginning, the algorithm was split into two parts. One core makes the computation for a long identification model (M_1, T_1 = 768 for H_1), whereas the other core computes a regression model of a smaller order (M_2, T_2 = 256 for H_2). Again, the computation was made both in DP and SP FP and the outputs were verified with the results obtained in MATLAB R2019b.

According to the experiments, this step allows decreasing the computational time from 200s to 151s for DP and from 188s to 139s for SP. It means that the Xilinx Zynq Ultrascale+ Cortex A53 ARM device delivers 122 MFLOP/s (for DP) and 133 MFLOP/s (for SP FP) on this stage of the algorithm optimization process. It is not enough for the real-time processing, but already on this stage the algorithm can serve as a golden model, which proves that it functions correctly and which can be used as a sample for further optimization of the algorithm.

The next step is to implement the four core pipelined version of the algorithm on the Xilinx Zynq Ultrascale+ Cortex A53 ARM device using the computational resources of all four processors.

Again, the input parameters are copied to the local memory via header files prepared in advance. Once they are in the local memory, one core computes the QRD RLS Lattice algorithm with order n_2=256 and time delay TD=512 for a hypothesis H_2, which has a higher probability when the hand appears in front of the device.

Other three cores compute the QRD RLS Lattice algorithm with order n_1=768 for a hypothesis H_1, which states that there is no hand in front of the device. However, the second algorithm is split into three threads T_2=T_3=T_4=256 and each part is processed separately by a separate core.

The results of computation were verified with the reference model provided by MATLAB R2019b.

The experiments show that the computational time is substantially reduced while using the four core pipelined version of the algorithm. It constitutes approximately 58s for DP and 55s for SP. It means that the Xilinx Zynq Ultrascale+ Cortex A53 ARM device delivers 318 MFLOP/s for the DP version of the algorithm and 335 MFLOP/s for the SP FP version of the algorithm. This is already sufficient to process the data in real time.

Figure 4.4 compares the computational time and MFLOP/s for one core (1), two core (2) and four core (4) versions of the algorithm on the Xilinx Zynq Ultrascale+ Cortex A53 ARM device. The example is given for the SP FP arithmetic and for computing with the data blocks equal to 528, i.e. there are the outputs ready for hypothesis testing each 60ms, which corresponds to 1000 results in 60s.

Table 4.1 shows the computational time for four cores given the data divided into smaller blocks, where ***ns*** is a division factor [64].

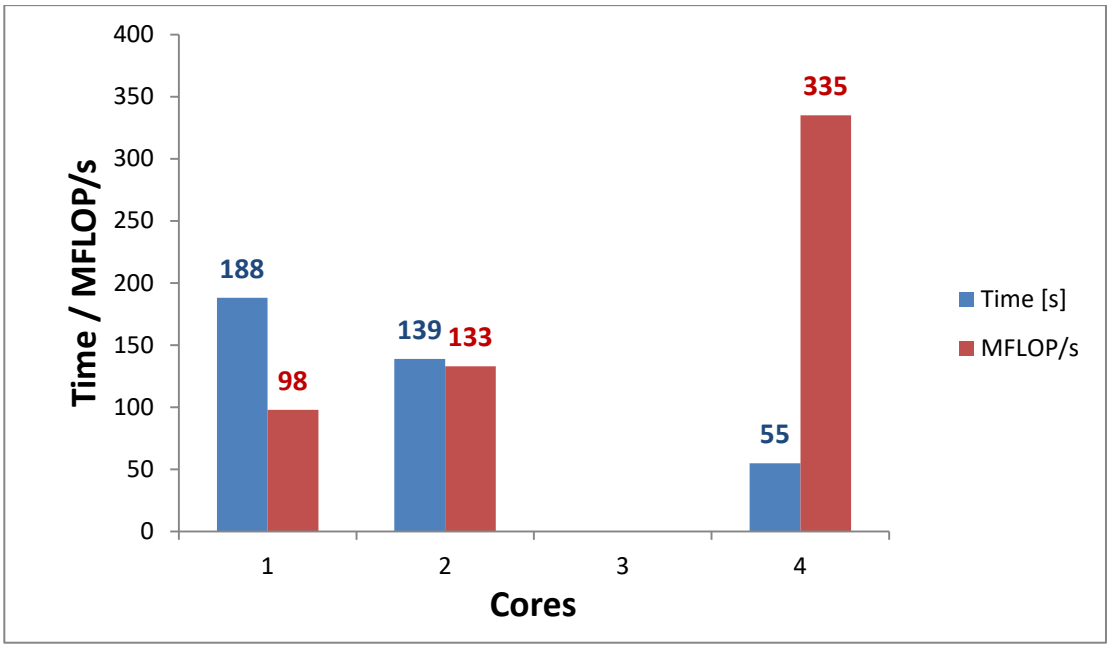

Figure 4.4: Computational time and MFLOP/s for different core versions of the algorithm (SP FP arithmetic, ns=1000)

Table 4.1: Computational time for different division factors [64]

ns	Time [s] for DP	Time [s] in SP
10	66.57	62.74
20	62.18	62.70
50	59.40	56.40
100	58.57	55.77
200	58.22	55.23
1000	**57.51**	**54.85**
2000	57.66	55.25
4000	57.83	55.33
8000	57.89	55.13
16000	58.82	55.56

It is obvious from the table that the best computational time is achieved when the data are divided by a division factor ***ns*** = 1000 (outlined in green).

To make the comparison clearer, Figure 4.5 represents an example of the computational time for the four core version of the algorithm for the SP FP arithmetic.

From the graph in Fig. 4.5 it is also clear that the computational time is decreasing by making smaller data blocks purposed for the computation and reaches its best value at ***ns*** = 1000, i.e. a block contains 528 samples at each step of the computation process.

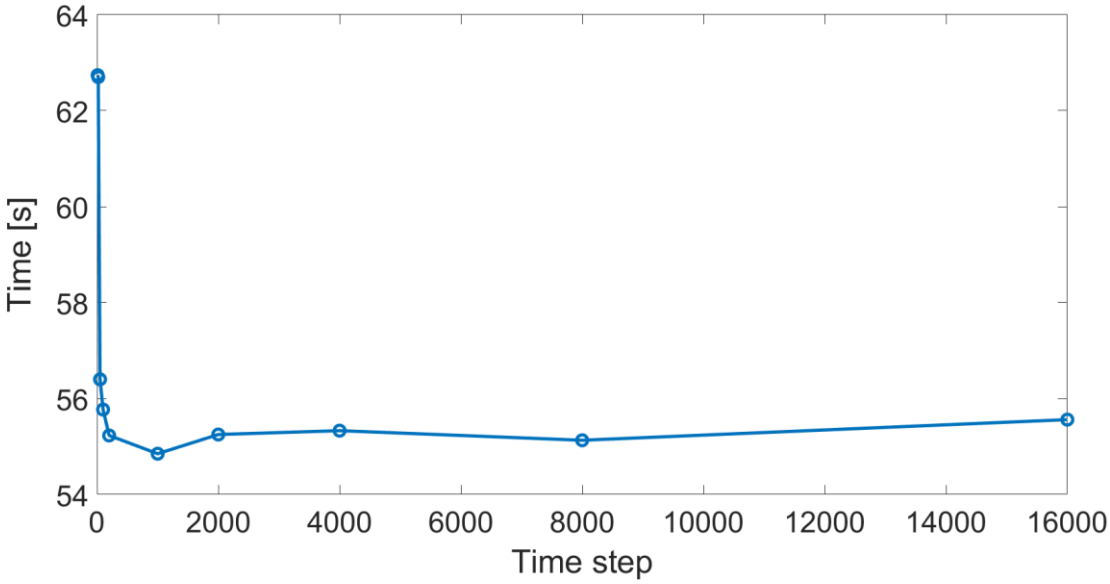

Figure 4.5: Computational time for four core version of the algorithm given SP FP arithmetic [64a]

However, after this value it begins increasing again as far as the communication also increases. Thus, a division factor ***ns*** = 1000 is considered to be optimal for this case of the data processing.

In the version of the algorithm described above each processor has its own local set of variables and there is a need to move the data back and forth between the processors to ensure that the following process has all necessary data for the next stage of the computation. This has been a source of slowing down in the computation process. Another way to ensure that each core has the valid data for the next step of the data processing is to apply a so-called cache ping-pong technique.

Let us remind that the longer algorithm M_1 is divided into three threads T_2=T_3=T_4=256. To avoid a large data amount movement back and forth between the processors, it is possible to create a pair of threads for each part of the algorithm, $2T_a$, $3T_a$, $4T_a$ and $2T_b$, $3T_b$, $4T_b$, which are fully identical. However, there are twice as much registers, which require twice as much memory and which are accessed as group A and group B in turns. Thus, when group A is ready with the computation, the data are saved in the memory and group B can access them to compute the next step. Applying this method, we will ensure that the threads will have the appropriate data for each computational step without the additional copy of the data. It eliminates the need to wait until the valid data are copied by the single ARM core from T_2 to T_3 or T_3 to T_4 respectively. The principle of the ping-pong technique is schematically presented in Fig. 4.6.

The experiments proved that the method functions well in terms of the accuracy and avoidance of the data conflicts. However, it did not succeed to decrease the computational time, which remained approximately 58s and 55s for the DP and SP FP arithmetic respectively.

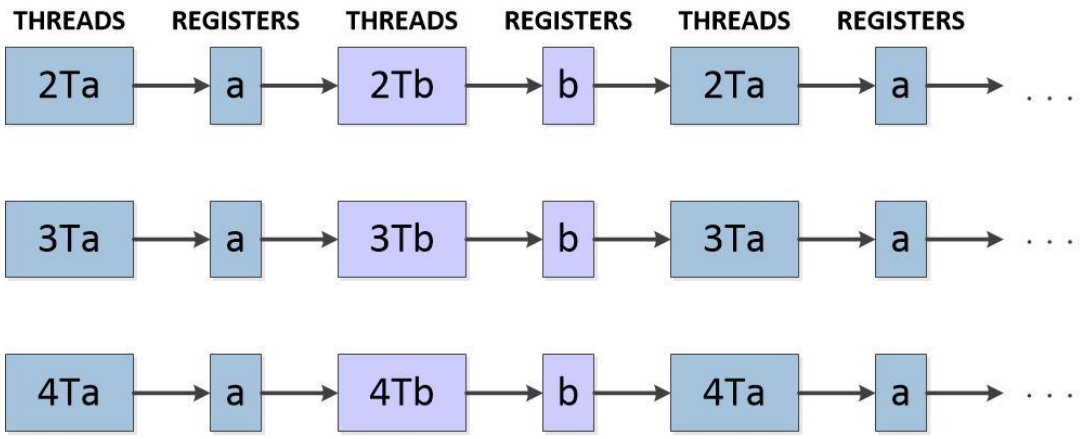

Figure 4.6: Ping-pong data sharing

It can explained by the fact that the data transfer is so fast that it does not contribute greatly to the time decrease, because the number of variables in this case is growing and it potentially slows down caching by a frequent cache-miss. However, this technique is supposed to be used on the stage of the QRD RLS Lattice algorithm implementation in the FPGA part of the Xilinx Zynq Ultrascale+ Cortex A53 ARM device.

The last step of the optimization at this point is to incorporate hypothesis testing into the parallelized algorithm and to see if the computational time increases and if the algorithm needs further acceleration or not.

As it was mentioned above, the optimal division factor ***ns*** is equal to 1000. It gives the best results in terms of the computational time and it ensures that there will be the appropriate data for hypothesis testing each 60ms, which is more than sufficient for the real-time processing applications. It should be also noted that the algorithm provides the whole information about the identification process including the probabilities of each model for 528 samples every 60ms. It helps to reconstruct what the hand did during this period of time.

Thus, the hypothesis computation should be applied in a way that the algorithm gives the probabilities of each model every 60ms. For these purposes, two additional threads were created: T_{h1} and T_{h2}. Thread T_{h1} computes all necessary values for hypothesis H_1 and thread T_{h2} – for hypothesis H_2. They run independently of each other after all necessary data needed for the computation are received from thread T_1 of model M_2 (for hypothesis H_2) and thread T_4 of model M_1 (for hypothesis H_1).

After threads T_{h1} and T_{h2} finish their computation at a certain time step, the probabilities of M_1 and M_2 are calculated. This process is running each 60ms. The whole computational time increases, but not critically. After incorporating the hypothesis computation, it constitutes precisely 58s for the DP version of the algorithm and 55.24s for the SP FP version of the algorithm. It means that the optimization of the algorithm was successfully fulfilled and it reaches its main goal: the algorithm can be applied for the real-time processing applications.

If compared with the computation of the QRD RLS Lattice algorithm in MATLAB R2019b with the Parallel Toolbox, the Xilinx Zynq Ultrascale+ Cortex A53 ARM implementation has the computational time from 156.27s to 67.13s (1 processor to 4 processors) approximately, whereas in the MATLAB environment it takes from 12.4s to 6.2s (1 CPU core to 4 CPU cores) approximately to perform the whole computation of the algorithm for the computation in 10 steps.

However, with the increasing number of the processors and with the increasing value of the computation steps, the computational time in MATLAB with the Parallel Toolbox is increasing, while in the case of the Xilinx Zynq Ultrascale+ Cortex A53 ARM implementation it is decreasing. Thus, for the Xilinx Zynq Ultrascale+ Cortex A53 ARM implementation with four processor cores computing in 1000 time steps, the computational time reaches 54.85s, whereas in the MATLAT with four processor cores and the computation in 1000 time steps it constitutes approximately 231s.

The reduced performance of the MATLAB Parallel Toolbox is due to the overhead related to the frequent copy of the data and the start of the parallel processes. More detailed comparison for one CPU, two CPU and four CPU implementations in terms of the computational time and MFLOP/s is provided in Tab. 4.2 and Tab. 4.3.

Table 4.2: Comparison of the computational time for the MATLAB and ARM implementations (SP FP arithmetic, for N divided into smaller parts)

Time step (ns)	MATLAB Time [s]	ARM Time [s]	MATLAB Time [s]	ARM Time [s]	MATLAB Time [s]	ARM Time [s]	Outputs [s]
	1FP	1FP	2FP	2FP	4FP	4FP	
10	12.41	156.27	9.90	139.06	6.20	67.13	**6**
20	12.51	156.44	12.41	138.97	9.59	63.09	**3**
50	12.38	156.30	18.08	138.86	16.76	56.79	**1.2**
100	12.34	156.19	27.15	139.02	28.41	56.16	**0.6**
200	12.35	156.25	44.60	138.79	53.05	55.62	**0.3**
1000	12.37	156.09	156.23	138.91	231.31	54.85	**0.06**
2000	12.78	187.92	284.50	139.12	421.28	55.25	**0.03**

Table 4.3: Comparison of the number of operations per second for the MATLAB and ARM implementations (SP FP arithmetic, for N divided into smaller parts)

Time step (ns)	MATLAB MFLOP/s	ARM MFLOP/s	MATLAB MFLOP/s	ARM MFLOP/s	MATLAB MFLOP/s	ARM MFLOP/s	Outputs [s]
	1FP	1FP	2FP	2FP	4FP	4FP	
10	1484	118	1861	133	2971	274	**6**
20	1472	118	1484	133	1921	292	**3**
50	1488	118	1019	133	1099	324	**1.2**
100	1493	118	679	133	648	328	**0.6**
200	1492	118	413	133	347	331	**0.3**
1000	1489	118	118	133	80	336	**0.06**
2000	1441	98	65	132	44	333	**0.03**

The rows in green in Tab. 4.2 and Tab. 4.3 are the computational time and MFLOP/s respectively given the time step ***ns*** = 1000 for different versions of the algorithms implemented both in the PC MATLAB R2019b with the Parallel Toolbox and on the ARM cores. In case of the computation for 1000 time steps the outputs are available each 60ms.

4.3. Algorithm Implementation in the FPGA Programmable Logic

The previously described steps of the optimization and implementation of the QRD RLS Lattice algorithm on the Xilinx Zynq Ultrascale+ Cortex A53 ARM processor proved to be successful. However, the computational time is very close to the upper threshold for the real-time processing. Thus, for less powerful SoC platforms the algorithm will function slower and the efficiency of the proposed solution will decrease.

Therefore, in this section there is a try to convert the algorithm to the FPGA part of the Trenz Electronic platform to accelerate the computation and to reach the increased performance in respect to the real-time processing.

In the present chapter the ways of the algorithm transformation to the FPGA are described and a comparison with the previously described solutions in terms of the computational time and MFLOP/s is presented.

4.3.1. FPGA Accelerators

In the previously described SW experiments and implementations the Xilinx Zynq Ultrascale+ ZU09-EG-ES1 device has been used. This device has been located on the Trenz Electronic TE0808 MPSoC with the Cortex A53 4-core ARM, 1.05 GHz [34-35].

The Xilinx Zynq Ultrascale+ ZU09-EG-ES1 is an evaluation sample of the device. It requires the Xilinx Vivado 2017.4 [104] and SDSoC 2017.4 design tools [107-108]. The SDSoC 2017.4 design tools are supported by Xilinx only up to the version 2019.1. Starting from the Xilinx tools release 2019.2, the Xilinx decided to support a new Vitis toolchain [103].

Therefore, the perspective development of the HW accelerators mapping to the HW acceleration required to migrate from the Xilinx SDSoC toolchain to the Xilinx Vitis acceleration flow, starting from the Xilinx 2019.2 tools.

It is the reason why in this section, we will target:

- the Xilinx Zynq Ultrascale+ ZU09-EG-1E device on the TE0808-09EG-1E module with 2GBytes of DDR4 memory and the 4-core A53 ARM processor (1.05 GHz),
- the Xilinx Zynq Ultrascale+ ZU15-EG-1E device on the TE0808-15EG-1E module with 2GB of DDR4 memory and the 4-core A53 ARM processor (1.05 GHz).

The Xilinx Zynq Ultrascale+ ZU09EG device supports only BRAMs, whereas the Xilinx Zynq Ultrascale+ also supports URAMs.

The Xilinx Zynq Ultrascale+ ZU09EG device is a direct replacement of the ZU09-EG-ES1 device. It can implement eight HW Data Processing Units (DPUs) in the programmable logic. These units use BRAMs (1K x 32b).

The Xilinx Zynq Ultrascale+ ZU15EG device can also implement eight HW DPUs in the programmable logic. These units use 112 URAMs (4K x 64b) and also some BRAMs (1K x 32b). The HW module works with 4GB 64-bit DDR4 SDRAM, 128MB SPI Boot Flash (dual parallel) [102].

Both devices are supported by the Xilinx High Level Synthesis (HLS 2019.2) and by the Xilinx Vitis 2019.2 acceleration design flow. This is a software platform developed by the Xilinx and released first in year 2019. The Vitis acceleration flow design methodology enables portability from a platform to a platform (standard Xilinx development boards or custom boards) and also a platform reuse, i.e. it is possible to swap different acceleration applications with the same platform [109].

The ZU09-EG-ES1, ZU09-EG-1E and ZU15-EG-1E have the same speed grade of the programmable logic and the same performance of the 4-core 1.05 GHz A53 ARM processor. Therefore, the SW implementation results made in the previous chapters can be compared with the implementation results in the FPGA obtained in this chapter.

The HW version of accelerators, which can be used for the computation of a parallelized version of the QRD RLS Lattice algorithm, uses eight Single Instruction Multiple Data (SIMD) DPUs developed in ÚTIA, the signal processing department. The detailed information about these HW accelerators can be found in [42, 45]. In this section only a short description is presented.

The internal structure of the Zynq Ultrascale+ SoC with eight accelerators is illustrated in Fig. 4.7.

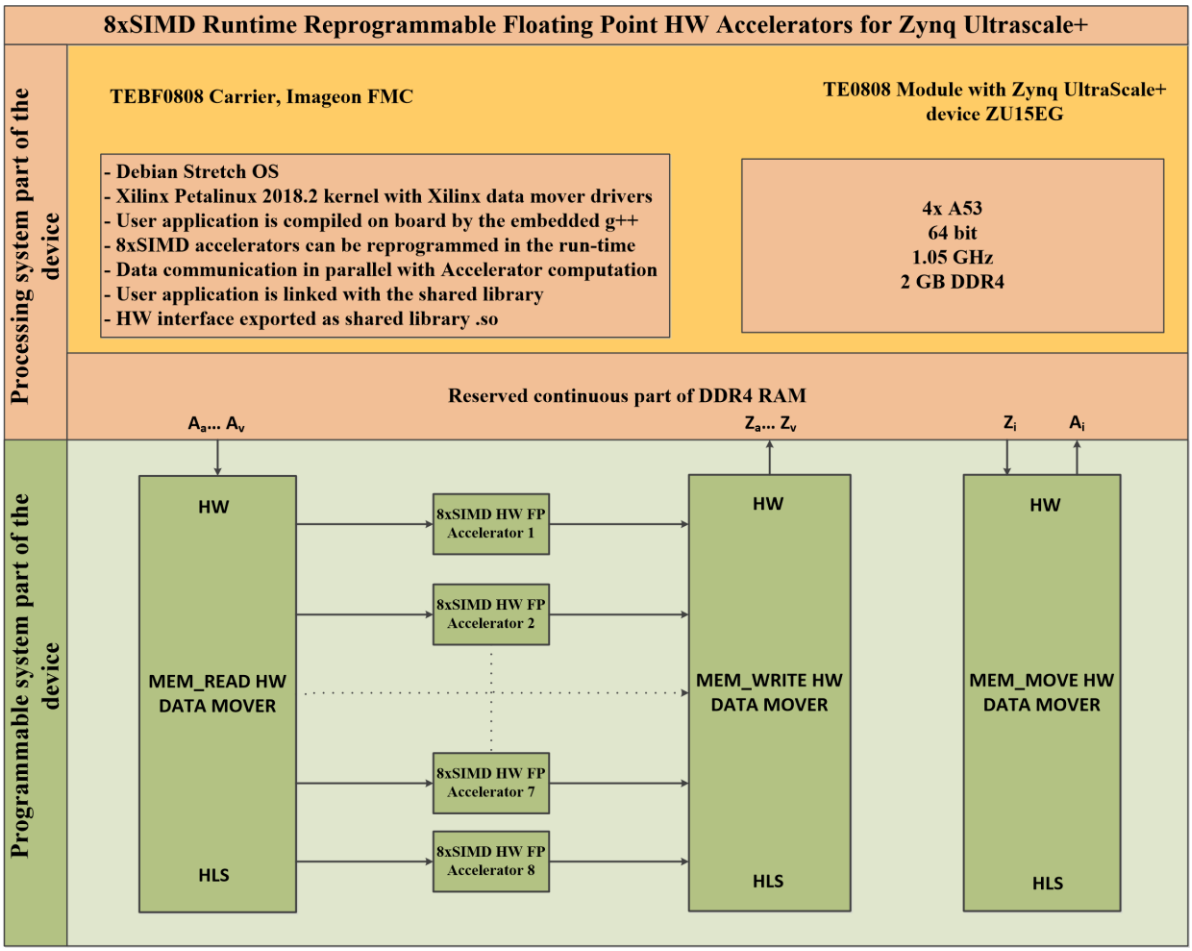

Figure 4.7: Zynq Ultrascale+ SoC with eight 8xSIMD HW accelerators [45]

The HW acceleration is represented by eight accelerators and a direct memory access (DMA) input/output [45].

There is a possibility to develop the SW firmware, which uses the accelerator. This firmware can be compiled by the user application running in Petalinux or Debian OS on the A53 processor [42, 45].

The Debian OS can be configured for an automatic boot of the X11 Desktop after the Power On. It contains the SciLab SW interpret with a graphical GUI. The SciLab can be used autonomously without a PC [42, 45].

The HW of the floating point accelerators is fixed. The reconfiguration can be made by reprogramming the firmware code. The firmware defines the function of the programmable finite state machine (FSM) inside the accelerator and the function of the communication logic [42, 45].

The input comprises the programme firmware data, the configuration registers for the scalar control and the floating point single precision data. Both the program firmware data and the floating point single precision results are transmitted via the AXI stream interface from the ARM processor [42, 45].

The outputs have two parts: dedicated interrupt lines indicating the end of the data movement operation, and the floating point single precision result data, which are received in the data buffers in DDR4 memory via the AXI stream interface generated in the SDSoC [42, 45] or in the Vitis [103] acceleration flow.

Data communication is implemented as an AXI-stream. The connectivity is represented with the AXI stream input, the AXI stream output and the AXI-lite configuration registers [42, 45].

The AXI stream input has input FIFO 512x256b and supports the AXI stream side channel indicating the last transferred word sent to the component via the DMA transaction from the ARM processor [42, 45].

The AXI stream output has output FIFO 512x256b and supports the output side channel indicating the last transferred word sent from the component via the DMA to the ARM processor [42, 45].

The interfaces of the accelerators include the data streaming I/O, which is AXI-S 256b at 240 MHz and the firmware programme VLIW 128b at 240 MHz. [42].

The 8xSIMD HW accelerator firmware is a simple sequence of VLIW vector instructions, which is stored in the accelerator programme memory. The accelerator does not support for-loops, if-else, and similar constructs as well as it does not perform checking overflow/underflow in the floating point operations [42, 45].

The firmware can be first defined in the host software and then downloaded via the streaming interface to the accelerator. It is re-programmable in run-time by the data streaming. The computation and data streaming can be performed in parallel [42, 45].

The accelerator has two parts. One side is responsible for the data communication and the other is responsible for the computation. The computation and stream-based data communication can be overlapped. This is controlled by the user-space host software running on the ARM core. It can be used for the run-time reconfiguration by loading a new VLIW instruction sequence to the accelerator programme memory while the computation is in progress [42, 45].

The architecture allows designing a time configurable set of SP FP data-flow operations driven by the predefined state machines. Possible variants are the following [42, 45]:

- 8xSIMD_v10 Operations: SP FP Vector ADD, SUB, MUL, DIV operations,
- 8xSIMD_v20 Operations: (1) + SP FP MAC (chained multiply and add operation),
- 8xSIMD_v30 Operations: (1) + (2) + SP FP vector by vector dot product operation,
- 8xSIMD_v40 Operations: (1) + (2) + (3) + SP FP extended vector by vector dot product operation.

The data communication support HW is determined at the design time and cannot be changed at the runtime. The HW data movers are generated by the Xilinx

Vitis 2019.2 [42, 45]. In the application described below the DMA HW interrupt based API is used.

As far as the memory of the accelerator is concerned, the data are stored in dual-ported blocks. Speaking more precisely, the 8xSIMD HW accelerator has 12 dual-ported 4096x64b URAMs Blocks (0…11) in case of ZU15EG device or 48 dual-ported 1024x32b BRAMs in case of ZU09EG device, which are used as 24 Data RAM 1024x32b $A_1…A_8$, $B_1…B_8$ and $Z_1…Z_8$.

The 8xSIMD HW accelerator has two 4096x128b Blocks (12, 13), which are used as programme RAM 4096x128b $P_1…P_3$ (see Tab. 4.4) [42, 45]. The ZU15EG uses two URAM memory blocks and the ZU09EG uses 16 BRAM memory blocks for a programme.

Thus, for the algorithm implementation purposes there is an environment with the four core A53 ARM, which has Linux running on the ARM core and the HW acceleration, i.e. eight 8xSIMD accelerators in the programmable logic part of the device [42].

The golden model MATLAB .mex functions are used to verify both the results of the 4-core ARM SW implementation and those of the HW acceleration part. The results obtained from MATLAB, from SW implementation and from HW implementation are then compared in terms of computational time and number of operations per second.

Table 4.4: Internal block rams of accelerators [42, 45]

SIMD A 32 bit	Block 64 bit	SIMD B 32 bit	Block 64 bit	SIMD Z 32 bit	Block 64 bit	VLIW prog	Block 64 bit
A1	0	B1	4	Z1	8	P1	12
A2		B2		Z2		P2	
A3	1	B3	5	Z3	9	P3	13
A4		B4		Z4		P4	
A5	2	B5	6	Z5	10		
A6		B6		Z6			
A7	3	B7	7	Z7	11		
A8		B8		Z8			

4.3.2. Algorithm Implementation in the FPGA Logic Part of the Device

This chapter describes the computation of two FIR filters based on the QRD RLS Lattice algorithm and their probabilities in the programmable logic part of the Xilnix Zynq Ultrascale+ device. The computation results are compared and verified with the golden model from MATLAB R2019b. The implementation is performed in a hardware pipeline and benefits from the parallel processing. The communication uses a ping-pong sharing technique, the principle of which was described in the previous chapters.

Let us remind the design used for the four core ARM computation.

There are four cores working in parallel and computing two regression models, M_1 and M_2. M_1 is of order 768, M_2 has a smaller order of 256 and a time delay of 512. Both models are internally systolic arrays. This allows the algorithm implementation on four SW threads – T_1=256 for the computation of M_2 and T_2=T_3=T_4=256 – for the computation of M_1. It means that the system order for a large Lattice is 768 and the system order for a small Lattice is 256 (see Fig. 3.6).

In order to cut the QRD RLS Lattice algorithm into several parallel processed, system-order related stages, it is necessary to define, to store and to transfer the outputs, which will be the inputs for the next stage of the computation. They should be transferred from one stage of the computation process to another stage. Nine input/output variables were defined (see Fig. 3.9).

Besides, it should be kept in mind that there is a certain phase of a run-up of the algorithm before the parallel computation and a certain phase of a wind-up of the algorithm after the parallel computation.

The run-up phase is needed to ensure that all cores will have data for starting the computation in parallel. The wind-up phase finishes the computation and gradually frees the parallel computed cores (see Fig. 4.8).

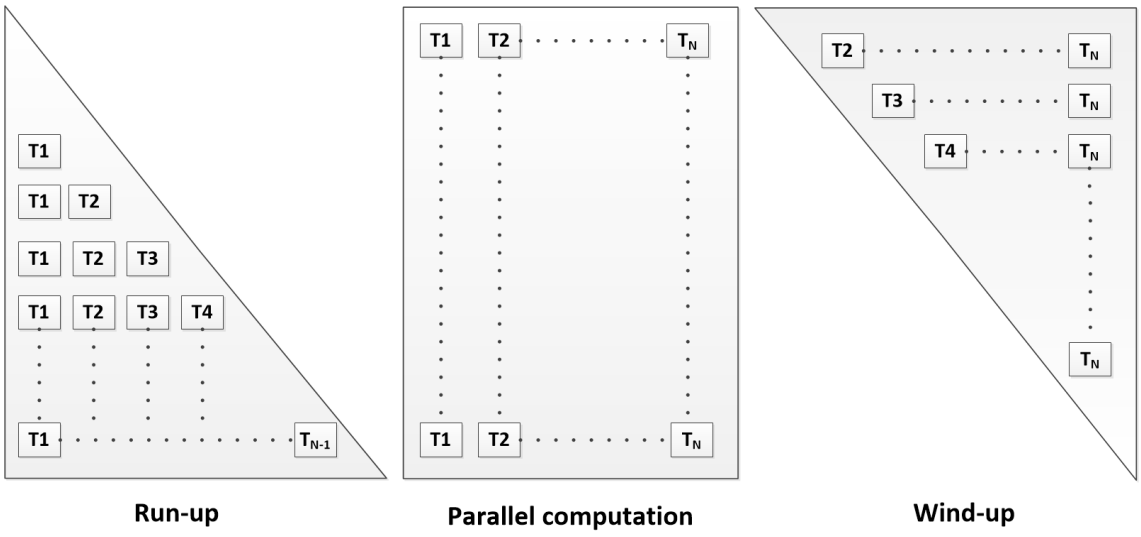

Figure 4.8: Run-up – parallel computation – wind-up

In case of the ARM processor the algorithm is divided only into four stages, while the ARM processor has only four cores and a large memory to keep a large amount of data there. The computational capabilities of four cores were fully used and, thus, the computation itself was very efficient. Besides, the ARM processor works at 1.05 GHz frequency. As the result the computational time of the algorithm was enough for the real-time data processing.

However, in case of the FPGA implementation there are certain limitations. Firstly, the local memories of the HW device are small and it is impossible to move such a large amount of data there. Secondly, it works at 240 MHz frequency.

On the other side, there are eight accelerators in the FPGA. Each accelerator has eight layers, i.e. 8x8=64. It means there are 16x more threads, which can process the data in parallel. It gives a great potential to accelerate the algorithm if it is specially optimized for the HW application.

To fulfil it, the first step is to divide the algorithm into smaller parts in order to meet memory limitations of the HW device and to use fully all eight DPUs and eight layers in them.

For these purposes, the algorithm is divided in a way that each stage computes four orders at each time step. It means a smaller identification model M_2 is divided into 64 pthreads (64 pthreads x 4 orders = 256 order) and a larger identification model M_1 is divided into 192 pthreads (192 pthreads x 4 orders = 768 order). It was the maximal possible amount of data, which can be put in the local memories of the device in the present HW design.

This solution was firstly proposed and verified in the SW on the ARM processor. The SW version can be executed in pthreads started from the user host application. The pthreads, similarly as it was in the previous implementation, run in parallel under control of the ARM Linux kernel and on four cores of the ARM A53 processor. However, the computational time considerably increases from 55s to approximately 120s. This increase is due to the following reasons:

(a) the communication traffic between pthreads substantially increased,
(b) the overhead related to starting and stopping pthreads increased,
(c) the volume of the computation in each pthread decreased by the move from 4 pthreads to 256 pthreads.

It results in slowing down of the SW version of the algorithm.

The computation process for the FPGA implementation of the algorithm is schematically presented in Fig. 4.9. The ARM part is presented in yellow colour, while the FPGA logic part is coloured in green.

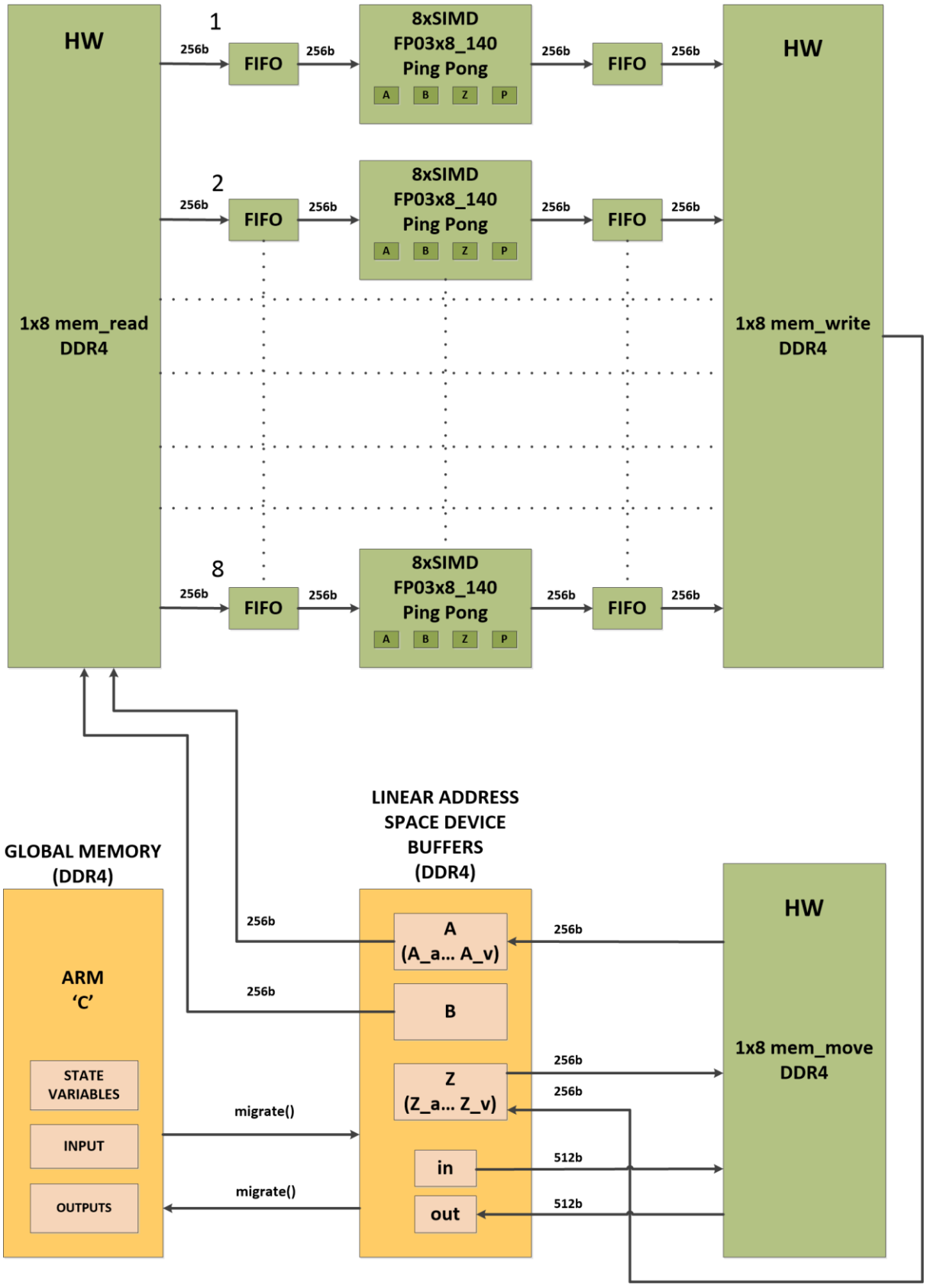

Figure 4.9: Block diagram of the computation process

As it is obvious from Fig. 4.9 the design works with three HW data movers, – memory_read, memory_write and memory_move, - and eight 8xSIMD FP03x8 accelerator HW IPs. All kernels are interfaced using OpenCL C++ API and compiled to the HW by the Xilinx Vitis 2019.2 toolchain.

Before starting the computation process, the SW OpenCL utility functions find a HW image in the HW archive on the SD card and verify what kind of devices exist and what kind of functions they can fulfil.

Moreover, the parameters should be defined and the registers should be set up in the SW to launch the HW kernels and to start the current stage of computation in the HW cores. The registers are set up via the AXI-lite to make the HW kernels know the amount of data and their destination. After it all kernels are ready to be started.

The memory_read kernel sends the input data from DDR4 device buffers A to eight 8xSIMD HW accelerators sequentially. It works on 240 MHz. The communication width is 256b, i.e. 8x32 words.

The memory_write kernel sends the output data from eight 8xSIMD HW accelerators to device buffers in the DDR4. It also works on 240 MHz. The communication width is 256b, i.e. 8x32 words.

The memory_move kernel connects the QRD RLS Lattice algorithm stages by moving the outputs from the output data buffers Z in the DDR4 to the input data buffers A in the DDR4.

In the ARM host SW the OpenCL function “migrate” is used to migrate the data from the host Global Memory into the Linux part of the DDR4 to the linear address space device buffers in the DDR4.

The device buffers A, B and Z contain the data for all eight 8xSIMD HW accelerators. The data in A, B and Z device buffers are divided into 8 parts, i.e. $A_1 \ldots A_8$, $B_1 \ldots B_8$, $Z_1 \ldots Z_8$, for eight 8xSIMD HW accelerators.

The device buffers A are used for the inputs, whereas the device buffers Z are used for the outputs.

The device buffers B are used for the internal state of both Lattice filters (lattice coefficients), which are then stored and updated inside of eight 8xSIMD HW accelerators. There is no need to copy them to the DDR4 device buffers during the algorithm computation.

There are four memory block types inside each 8xSIMD accelerator: A, B, Z and P. Memories A, B, Z are the data blocks, where $A_1 \ldots A_8$ is for the inputs, $B_1 \ldots B_8$ is for the state variables and $Z_1 \ldots Z_8$ is for the outputs. These are the local memories in the FPGA, which are not accessible by the ARM processor.

P is a programme block, where the instructions for the computation are saved.

The composition of blocks A, B, Z, P is presented in Tab. 4.4.

The data blocks and the programme block are composed with URAMs in case of the ZU15EG device and with BRAMs in case of the ZU09EG device.

Eight 8xSIMD HW accelerators need in total 112 URAMs, i.e. 12 for A, B, Z blocks + 2 for P block for each DPU. Each DPU has four URAMs, which are of size 4095x64b. It is possible to work with URAMs for A, B, Z blocks as with two halves of 32b. Besides, for A, B, Z blocks only 1K Word is needed, remaining space is not used. However, 4K Words is the minimal increment for URAM (see Tab. 4.4).

The programme block P consists of two URAMs, i.e. it is 4096x128b. The number of Very Large Instruction Words (VLIW) for the Lattice programme is 2774 VLIW. The maximal limitation for a programme is 4K Words.

In case of the ZU15EG device the data blocks are composed of BRAMs and use 3x64 memories, each memory is 1024x32b. The programme block uses 2x8 memories, each of 4096x64b.

The host ARM application forms the VLIW program instructions in the DDR4 memory as two 64b words. The C programme defines sequences of VLIW programme instructions in the DDR4 memory and writes them to the eight 8xSIMD HW accelerator programme memories. The programme is autonomously executed by the accelerators. Thus, all eight 8xSIMD HW accelerators have the same program P and compute the same algorithms, but on the separate stages, i.e. using different data in blocks A, B and Z.

In the experiments there are 528000 data samples in total. They are divided into 2000 time steps, i.e. each computation step has to process 264 time samples. There is no enough space in the 8xSIMD internal memory to move data for all 264 time samples. Only 12 samples can be processed as a batch. Therefore, each 264 data sample block is further divided into 22 data sub-blocks 12x22=264.

The data sub-blocks are coming in a form of 8x9x4x12 blocks, where 8 is the number of SIMD layers of each 8xSIMD HW, 9 is the number of I/O variables needed for each time sample, 4 is a vector length related to the number of threads computed in each layer of 8xSIMD accelerator and 12 is the number of time steps.

The data from the device buffers are coming for 4 threads for each layer of each accelerator. Also, there are 22 blocks of data for $A_1 \ldots A_8$ inputs and $Z_1 \ldots Z_8$ outputs in the device buffers, i.e. from **_a** to **_v**. It allows benefiting from the ping-pong sharing technique.

It means when A_a is being computed, A_b can be simultaneously prepared for the next step of the computation. When A_b is being computed, A_c is being prepared, etc.

After A_a is received via the output FIFOs from the accelerator URAMs or BRAMs, it is written to the output memory buffer in the DDR4. Simultaneously A_b is being processed inside of eight 8xSIMD accelerators and A_c is being copied from the memory buffer in the DDR4 via the input FIFOs to the accelerator URAMs or BRAMs.

The HW accelerators receive the data blocks for the computation via the input FIFOs. It is necessary to send 9x12=108 words for one thread or 108x4=432 words for 4 threads at each computational batch of each 8xSIMD HW accelerator computation layer. The input and output FIFOs of each 8xSIMD HW accelerator have depth 512 of 256b wide words. These input and output FIFOs are separate for each 8xSIMD HW accelerator.

The 8xSIMD HW accelerators are programmed as the Lattice computation kernels to execute 256 threads. Computation is started from the user host application. It is supposed that all 192+64=256 threads will run in parallel. However, to launch the parallel computation of all 256 threads at the same time, a run-up phase should be done first (see Fig. 4.8). It is performed in the SW on the ARM processor.

After computing the first 191 steps, there are already data available for 192 threads of a larger identification model M_1 and for 64 threads of a smaller identification model M_2. Thus, the parallel computation of all 256 threads can be started from time step 192. The parallel computation is fulfilled by the SW pthreads or by the HW accelerators.

As it was mentioned before, each 8xSIMD HW accelerator has 8 layers inside. The first 6 layers of each accelerator, i.e. $L_1 \ldots L_6$, compute the parts of the QRD RLS Lattice of order 768, while the last 2 layers, i.e. $L_7 \ldots L_8$, compute the parts of the QRD RLS Lattice of order 256 (see Fig. 4.10).

Each 8xSIMD HW accelerator layer computes 4 threads by the execution of the sequences of vector instructions with a vector length equal to 4.

Each thread computes 4 order-related steps of the algorithm and, therefore, the QRD RLS Lattice algorithm with order 768 = 4*192 and the QRD RLS Lattice algorithm with order 256 = 4*64 are computed.

Eight 8xSIMD HW accelerators work in parallel under the control of the pre-loaded firmware. The accelerators perform the processing of a batch of 12 input data samples for both Lattice filters samples and generate 12 output data samples for both Lattice filters. There are registers on each of the accelerators, which prescribe the address: from where to where the inputs/outputs should be moved.

In parallel to the HW computation, the ARM processor SW prepares new input data for the processing of the next 12 data samples and finally waits for an interrupt. At the end of each batch of 12 data samples, the HW accelerators generate an interrupt. It serves for synchronisation of the SW with the HW processing.

As it was mentioned above, during the computation the accelerators are able to copy the data via the FIFOs from/to the DDR4 device buffers. This leads to the possibility to compute and to transmit the data in parallel. It is made to provide a larger efficiency, i.e. to use the computational and communication resources of the accelerators as much as possible.

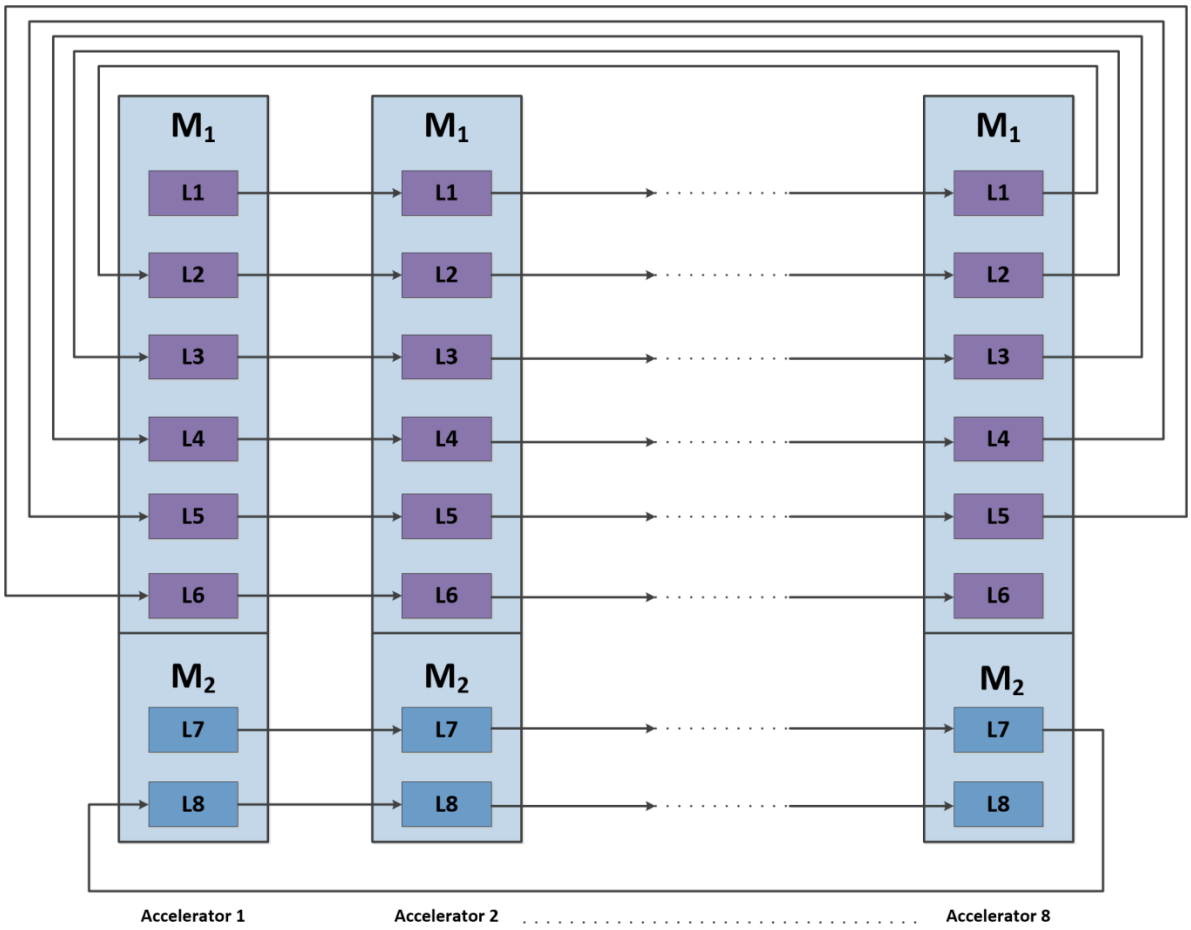

Figure 4.10: 8 SIMD HW accelerator layers

It is possible due to the fact that the RAM inside the programmable logic has two addresses, i.e. it is dual-ported as it was described above. The memory is divided into two parts: one part is for reading and writing for the present step (a computation part) and the second part is for the data preparation for the next step (a communication part).

The communication is able to read and to write one 256b wide word in single clock cycle (240 MHz). The data needed for the computation should be divided in a way that in one part the computation is performed and in the other part the communication (input and output) is made in parallel.

This copy to/from the accelerator in parallel with the accelerator execution of its sequence of VLIW instructions requires avoiding data race-conditions. This has to be avoided by the user application, by writing to the data which are not used for writing by the currently executed sequence of the VLIW instructions [42].

Therefore, the HW computation should be shorter than communication. When it is not, then it is necessary to use the SW hand-shake of the ARM with the HW (it is supported for the array of up to eight 8xSIMD accelerators) to ensure that the computation programme batch is finished before a new data transmission is started.

When the first outputs are ready, they are sent to the linear address space device buffers in the DDR4 via the output FIFOs and memory_write kernel (see Fig. 4.10). The data

outputs can be reused for the next step of the computation or they can be migrated to the Linux host Global Memory part of the DDR4, where the data can be accessible to the ARM processor host SW application.

It should be noted that eight 8xSIMD HW accelerators are independent. The computation layers are not connected with each other (see Fig. 4.10).

However, it is necessary to transmit the Lattice-order-state related input/output variables between the HW accelerator units for the next step of computation (see Fig. 3.9). It means that a part of the outputs stored in the device buffer Z have to be transmitted on the right place as the inputs of the device buffer A for the next stage of computation.

In the first stage of the HW accelerator development, it was made in this way:

- The outputs from Z were migrated from the device buffer in the DDR4 back to the ARM host Global Memory in the DDR4.
- Copying from Z to A in the host Global Memory was performed by the ARM in the SW.
- After it, the new inputs in A were migrated from the host Global Memory back to the linear address device space buffers.

The process of copying to and from between the host Global Memory and the linear address space device buffers was slow and due to it the computational time was approximately 140s.

To make it faster and more efficient, the Vitis HLS toolchain was used to create the single HW memory_move kernel for all 8xSIMD HW accelerators (see Fig. 4.11).

The memory_move kernel has access to both linear address space device buffers A and Z. It has also a mechanism for reading and writing. The goal of the memory_ move kernel is to take the outputs from the device buffer Z and to move them on a certain address in the device buffer A. The communication data path is 256b wide (to accommodate 8x32=256b).

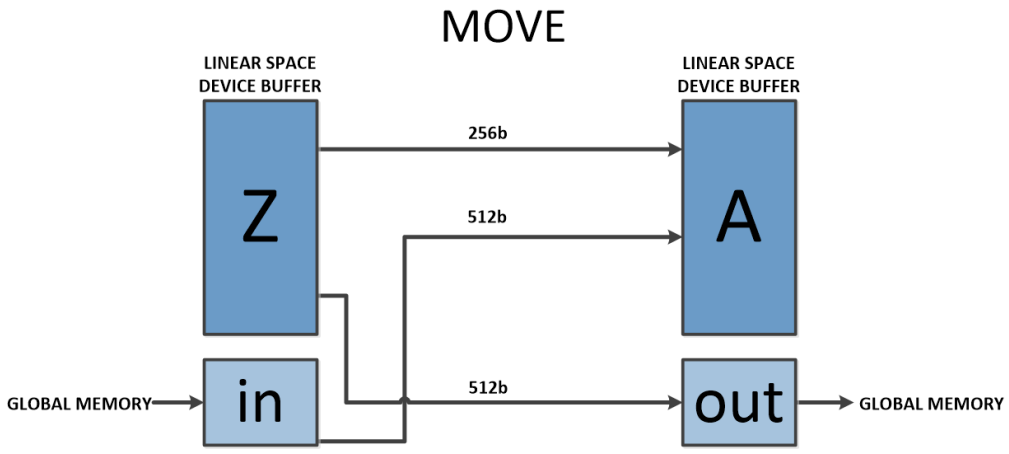

Figure 4.11: MOVE kernel

Besides, there are two small buffers “in” and “out”. The “in” buffer serves for giving the input data for each time step of the computation. The “out” buffer keeps the outputs of each step of the computation, which are migrated then to the host Global memory. The communication data path is 512b wide (to accommodate 9x32=288b > 256b). The remaining bits of each 512b wide word are unused.

Thus, the memory_move kernel performs the HW supported data communication, i.e. the connectivity in a way that the outputs of the algorithm, which are in the device buffer Z in the DDR4, are connected in a right way to the new inputs, which are in the device buffer A in the DDR4.

Briefly speaking, the memory_move kernel makes an application-specific data connection of the outputs into the inputs for all eight 8xSIMD HW accelerators.

Figure 4.12 illustrates the principle of data communication for one 8xSIMD in more details.

The ARM part is coloured in yellow, while the FPGA logic part is in green and comprises three HW kernels, one 8xSIMD HW accelerator and the input/output FIFOs.

As it was stated before, the data are divided into 22 blocks, i.e. from A_a to A_v for the inputs and from Z_a to Z_v for the outputs. When the first block A_a, which contains 12 data samples, are processed by the 8xSIMD unit, the output Z_a is sent to the device buffer. The memory_move kernel takes the output Z_a and transforms it into the input A_b, which is then sent for processing by the 8xSIMD unit. In this way the data communication is supported by HW (see Fig. 4.12).

It is good to remind that during the computation the ping-pong communication technique is applied and the parallel computation of the Lattice stages in all eight 8xSIMD HW accelerators is performed in parallel with data movement. All operations are performed 2000 times to complete computation of all 528000 data samples.

To resume, in the present design there are three tasks performed in parallel (see Fig. 4.9):

- memory_read for 8 accelerators,
- computation for 8 accelerators and 8 layers in each accelerator, i.e. 8x8=64,
- memory_write for 8 accelerators.

However, the process of the computation is still slowed down by the time overheads in the SW part.

Let us remind that in the algorithm computation there are two for-cycles:

- the outer one computes 2000 time steps, i.e. 12*22*2000=528000 data samples,
- the inner one processes 264 time steps, i.e. 12*22=264 data samples.

The average time needed for performing one step of the inner for-cycle is shown in Fig. 4.13.

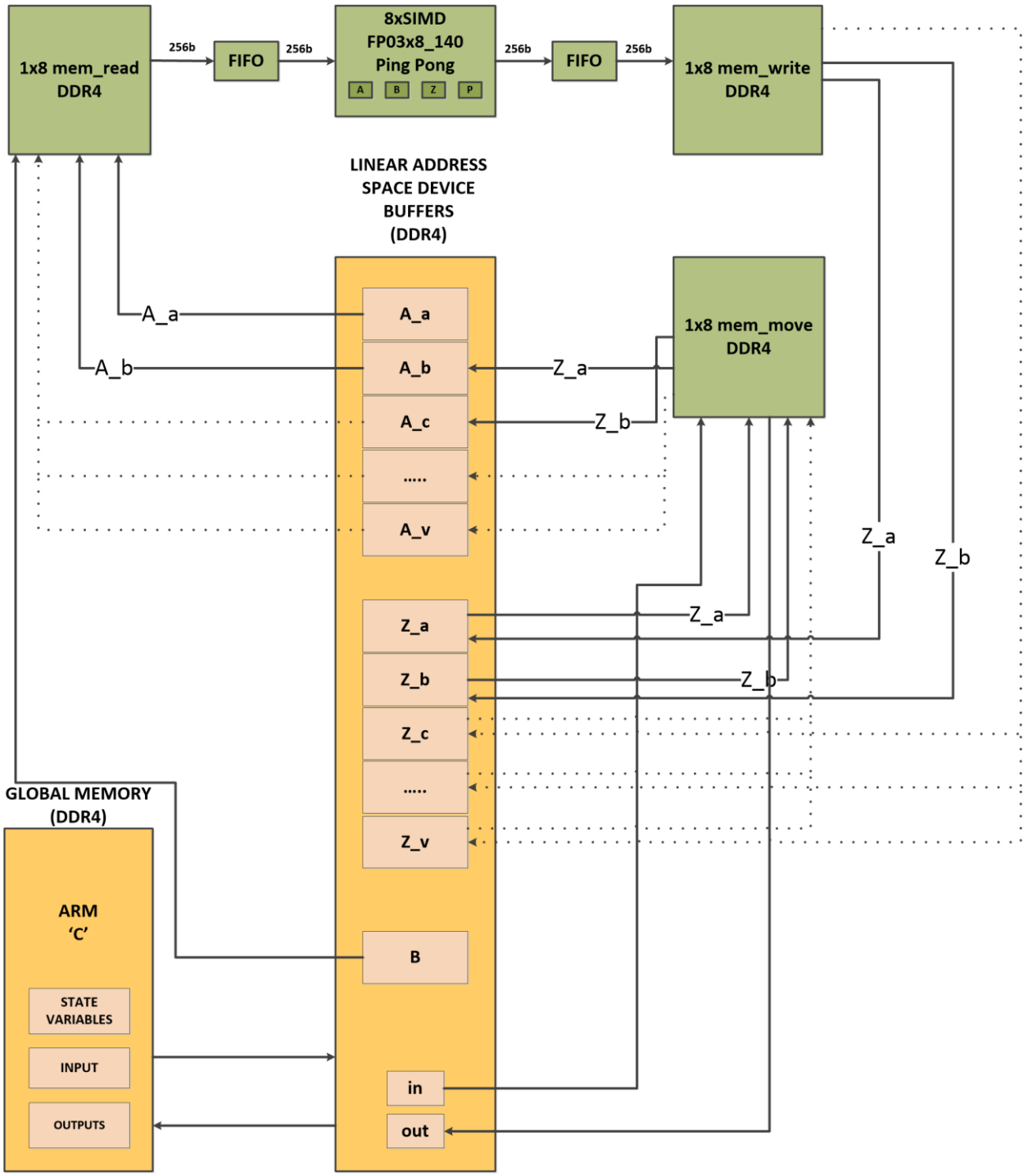

Figure 4.12: Data sharing via memory_move

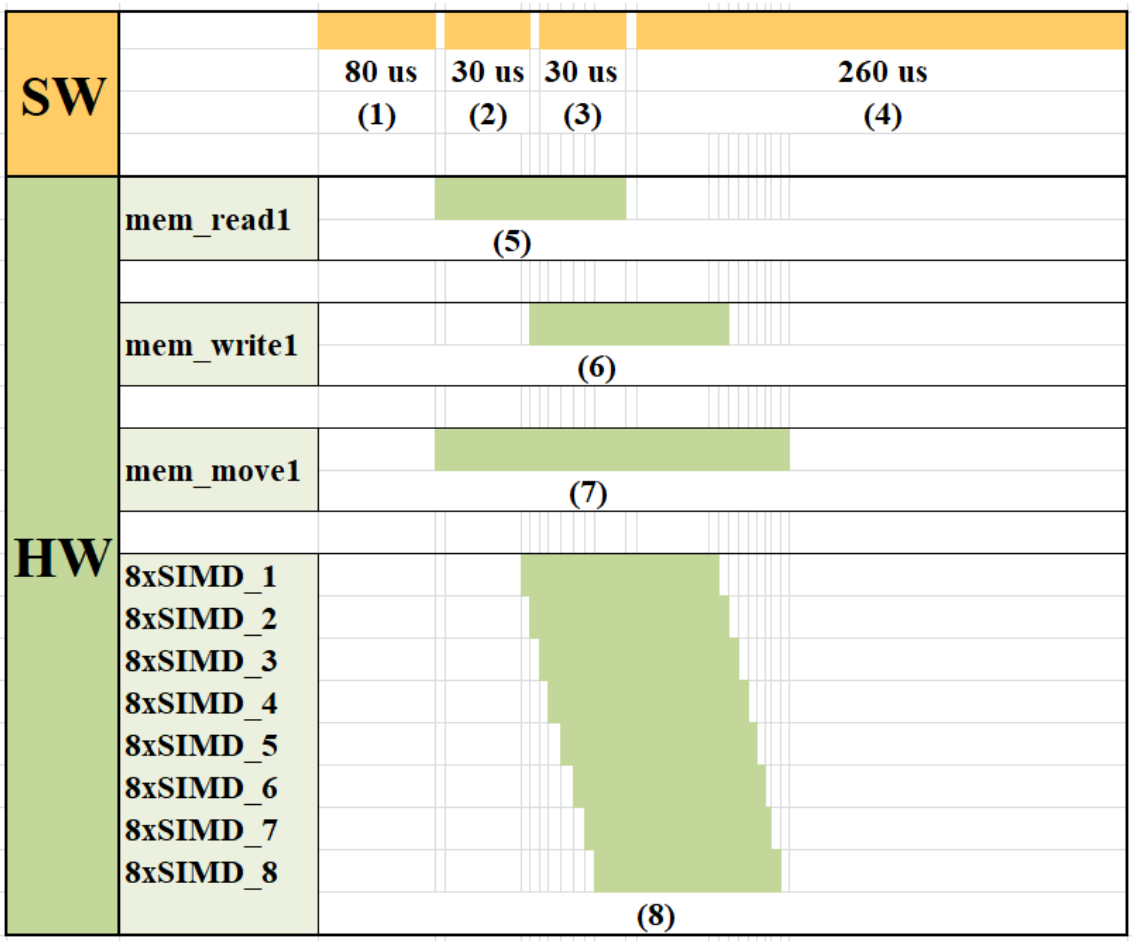

Figure 4.13: Computation of one step of the inner for-cycle

Figure 4.13 illustrates the following processes:

(1) defining parameters and starting the current stage of computation in the HW cores (average time),
(2) migrating the inputs for the next stage of computation (average time),
(3) migrating the results from the previous stage of computation (average time),
(4) waiting for the end of the HW supported data movement and the HW computation (average time),
(5) mem_read1 HW data mover for 8 8xSIMD (DDR4 device buffer A to 8x AXI-S),
(6) mem_write1 HW data mover for 8 8xSIMD (8x AXI-S to DDR4 device buffer Z),
(7) mem_move1 HW data mover for 8 8xSIMD (device buffers Z and I/O to device buffer A),
(8) eight 8xSIMD HW accelerators computing 12 time steps in 256 Lattice threads.

It is obvious from Fig. 4.13 that there is a large overhead caused in the SW part. The first step of a parameter definition and starting the current stage of the computation process in the HW cores (1) lasts 80μs. The migration of the inputs for the next stage of computation (2) and the migration of the outputs from the previous stage

of computation (3), which are performed in parallel with the computation process, last 30μs each.

The HW kernels are launched in the ARM SW by calling to the Xilinx XRT run-time support. After the HW kernels are ready with their work, they should be finished by the ARM. For these purposes, the ARM creates three processes (4) for three kernels. These processes wait for the interrupts from the HW kernels to finish them. Creating these processes lasts 260μs in the SW in average. This overhead is related to the current implementation of the OpenCL and XRT API.

However, in the future development, it will be possible to reduce its impact by making the HW computation more efficient, e.g. by providing the 8xSIMD HW accelerators with a larger amount of data for the computation or by making the programme batch longer (for more details see Future prospects).

After the HW accelerated computation process is over and the outputs are migrated to the host Global Memory, the results of the algorithm computation are verified and compared with the golden SW model. The results are proved to be identical.

The computational time reaches in average 16s for the HW accelerated QRD RLS Lattice algorithm, it means

- **3.4x acceleration** is achieved in comparison to the 4 thread SW implementation on the four core A53 processor,

- **7.5x acceleration** is achieved in comparison to the 256 thread SW implementation on the four core A53 processor.

The comparison of the optimal implementations of the algorithm on

- PC Intel® CoreTM i7-4770 CPU in MATLAB R2019b,
- Xilinx Zynq Ultrascale+ Cortex A53 4 core ARM processor, 4 threads,
- Xilinx Zynq Ultrascale+ Cortex A53 4 core ARM processor, 256 threads,
- Eight 8xSIMD HW accelerators, 256 threads,

are presented in Fig 4.14 and Fig. 4.15.

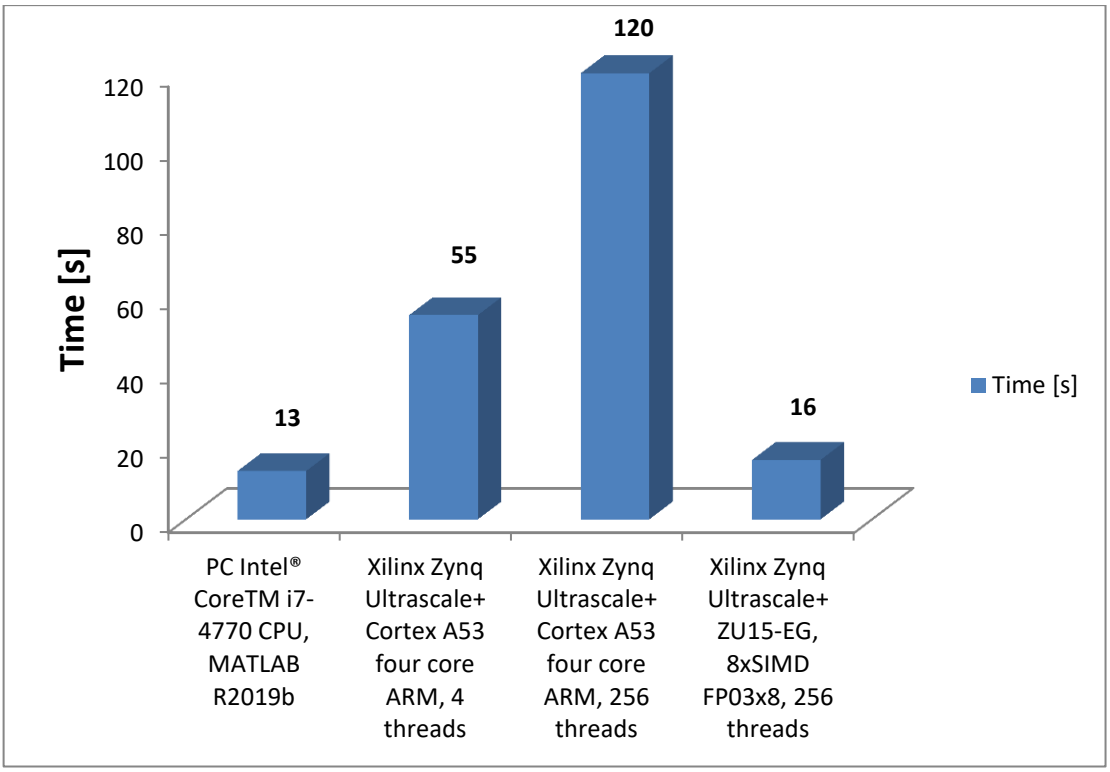

Figure 4.14: Performance comparison in terms of the computational time

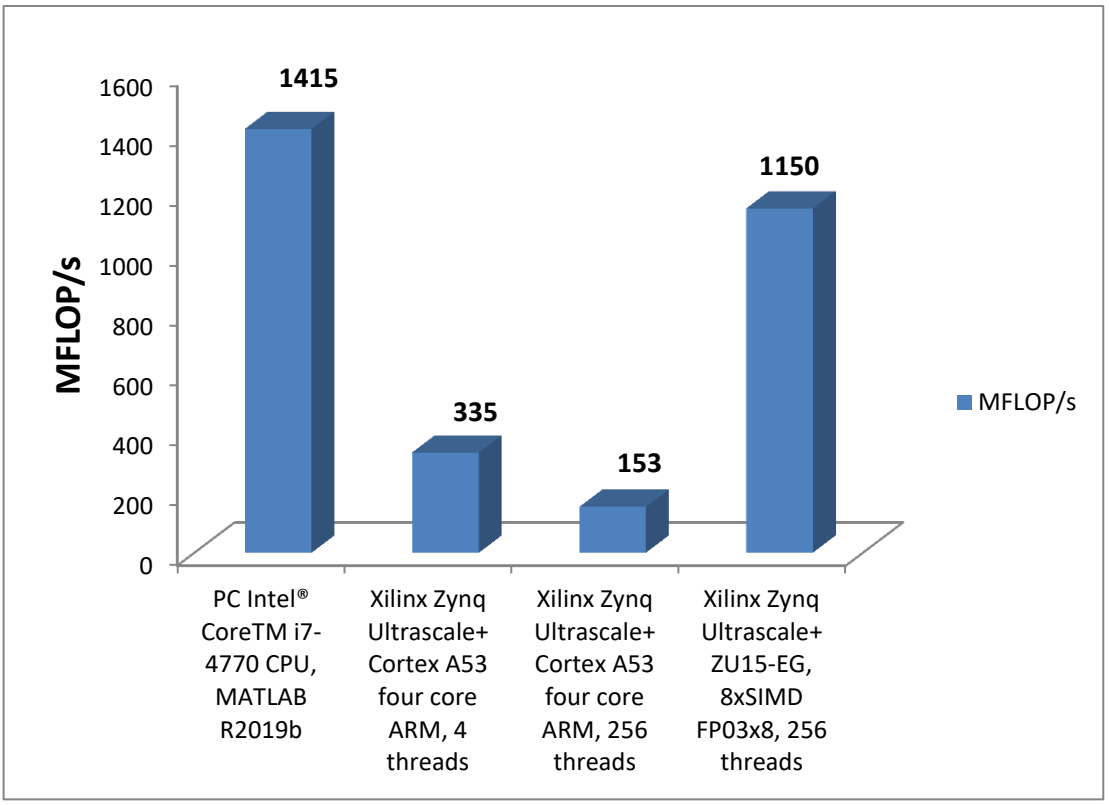

Figure 4.15: Performance comparison in terms of MFLOP/s

Let us remind that the QRD RLS Lattice based filters require 34850 floating point operations for each time step. The operations include +, -, *, /. From 34850 floating point operations there are 8200 floating point division. These operations are computed by eight layers of the 8xSIMD HW accelerators on the ZU09EG or ZU15-EG. The accelerators are controlled by the single SW thread running on the ARM A53.

The SW optimized (-O3) four-thread implementation of the algorithm on the Xilinx Zynq Ultrascale+ Cortex A53 ARM processor was executed on four A53 cores with 1.05 GHz clock frequency. It gives approximately 55s of the computational time.

However, the SW implementation of the algorithm on the Xilinx Zynq Ultrascale+ Cortex A53 ARM processor with the same number of threads as the HW implementation has, i.e. with 256 threads, gives only 120s for the parallel processing. The reason is that the ARM processor has only four cores and it needs to share 256 tasks between these four cores. It results in slowing down the computation process.

The HW accelerators on the Zynq system with eight 8xSIMD HW accelerators running at 240 MHz accelerate the SW implementation with 4 threads **3.4x** and the SW implementation with 256 threads **7.5x**.

It is also clear from Fig. 4.14 and Fig. 4.15 that using the HW accelerators the performance is more or less similar to the single-thread optimized implementation on PC Intel® CoreTM i7-4770 CPU in MATLAB R2019 (13s on PC vs 16s in the FPGA).

The FPGA version of the algorithm is a bit slower; however, it is good to keep in mind that the PC works at 3.5 GHz, whereas the eight 8xSIMD HW accelerators work at 240 MHz, i.e. on **16.7x** smaller frequency.

It is also good to note that for the present HW implementation it delivers about 1 GFLOP/s (see Fig. 4.15). However, theoretically the HW device might deliver up to 16 GFLOP/s peak.

The reason of delivering less GFLOP/s is that the HW implementation does not use the whole time for the computation during the overheads described. Besides, a half of the instructions inside are just “copying”, which is an operation, but it does not deliver GFLOPs. Moreover, there is some loss in GFLOPs due to the sequential streaming to the 8xSIMD by the memory_read kernel and due to the sequential streaming out from the 8xSIMD by the memory_write kernel.

For the future development, it is expected that it will be possible to reach up to 4 GFLOP/s on a larger Zynq Ultrascale+ ZU15 device by increasing the amount of data processed by the 8xSIMD or by increasing the size of the programme, or by increasing the number of order-related steps inside of each layer of the 8xSIMD. Each of the variants has its limitations. However, the better efficiency can be achieved.

The algorithm under consideration was also implemented using the identical HW design on the ZU09-EG device to prove the portability of the HW design described above and to show that it will really give the similar computational time. The SIMD units were configured with the smallest memories and are still compatible with the host SW.

The computational time for this implementation is similar to the implementation on the ZU15-EG, i.e. it constitutes in average 16s.

The power consumption of the platform under consideration was measured using the Agilent MSO6034A MegaZoom III Technology 300-MHz Oscilloscope, the voltage probe Agilent 10073C and the current probe Agilent N2783A with the amplitude accuracy ±1.05 of reading ±10 mA at 23 °C ± 3 °C.

Figure 4.16 shows the values of the current (yellow curve), voltage (green curve) and power (magenta curve) when there is no computation, while Fig. 4.17 illustrates the course of the current (yellow curve), voltage (green curve) and power (magenta curve) when the algorithm is being computed.

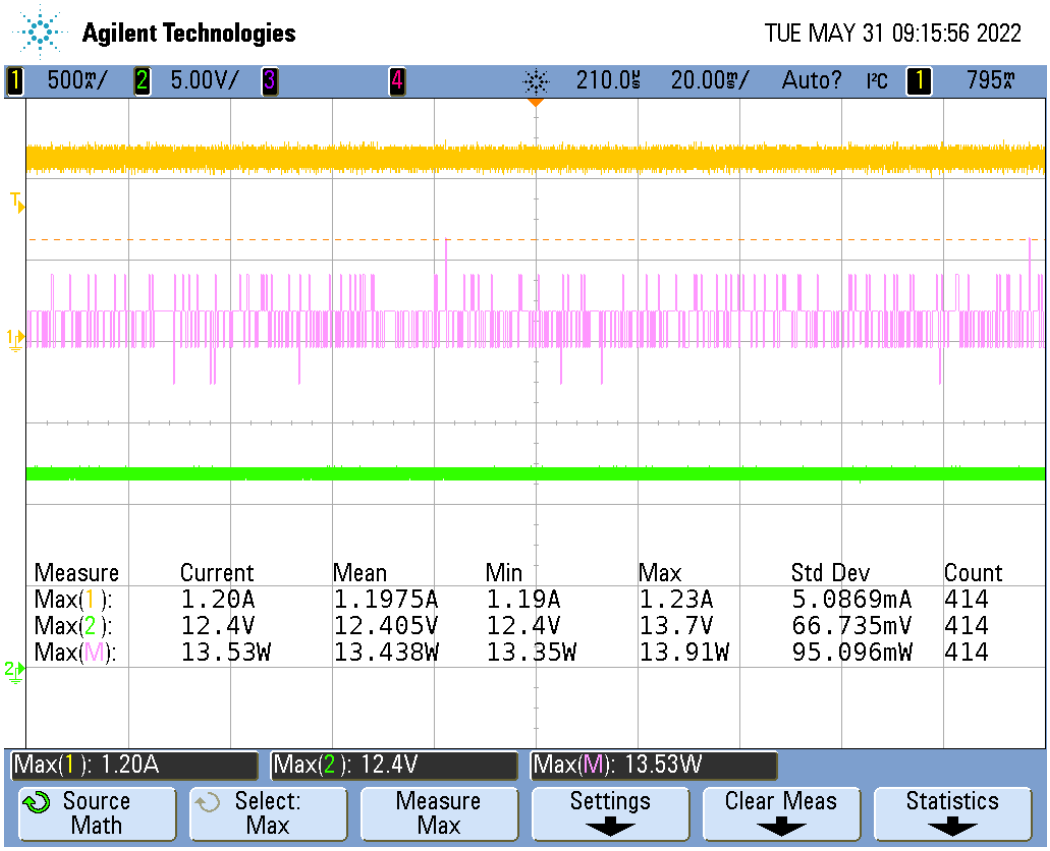

Figure 4.16: Current, voltage and power in a stand-by mode

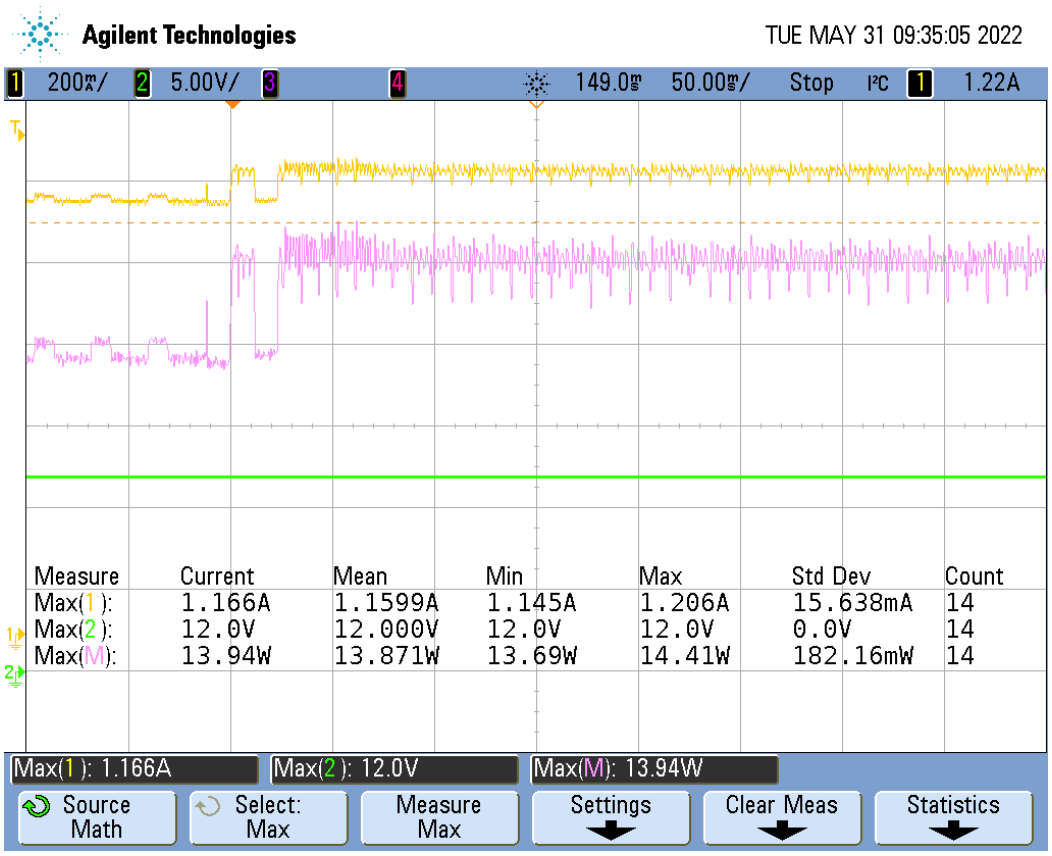

Figure 4.17: Current, voltage and power during the computation in SP

The power for a stand-by mode constitutes max. 13.53W. The power during the algorithm computation is max. 13.94W. Thus, the difference in the power consumption between the stand-by and computation modes is 0.41W. The small increase of the power while starting the computation is due to the fact that the Xilinx Zynq Ultrascale+ Cortex A53 ARM is set in a way that even in a stand-by mode it runs on the full power and does not decrease either clock frequency or voltage.

4.3.3. Portability to Different Platforms

There are several Xilinx devices, where the proposed design can be used without large changes (see Tab. 4.5).

Table 4.5: Compatible Xilinx Zynq devices

Device	Characteristics and limitations
ZU9-EG-ES	An evaluation version supported only with the old toolchain Vivado 2017.4 and SDSoC 2017.4, which are no longer supported by the Xilinx. It has the Cortex A53 4x core ARM processor, 8x DPUs, BRAMs. The HW design should be made in the SDSoC, but it will have the identical memory_read, memory_write and memory_move DMA and A, B, Z, P blocks. There could be a problem in data communication, because it uses 256b AXIS. The SDSoC 2017.4 is supporting integration of only 32b AXIS interfaces.
ZU9-EG	The same platform as the previous one, but not an evaluation version. It is already supported with the Xilinx Vitis 2019.2 and has the HLS Acceleration Flow. It has the Cortex A53 4x core ARM processor, 8x DPUs, BRAMs, 256b and 512b AXIS. The proposed design is compatible with the platform, but blocks A, B, Z and P should be made of BRAMs instead of URAMs. For the implementation of the algorithm data blocks A, B, Z were designed as 3x64 memories, each of 1024x32b, the programme block P has 2x8 memories, each of 4096x64b. There are also 2x8 FIFOs, each of 512x256b. From the point of view of the SW and firmware the design is identical and from the point of view of the OpenCL it is compatible. During the experiments it was proved that the implementation of the algorithm has the same acceleration as the ZU15-EG.
ZU15-EG	A platform, which is used for the described HW implementation of the QRD RLS Lattice algorithm. It has the Cortex A53 4x core ARM processor, 8x DPUs, URAMs, BRAMs and the HLS Acceleration Flow. It is supported with the Xilinx Vitis 2019.2. It has a great potential for further acceleration of the algorithm when enlarging the memory data blocks, the programme block and FIFOs. It also gives a possibility for a more complex design with 2x HW to solve two different tasks.

Table 4.5: Compatible Xilinx Zynq devices (continuation)

ZU04-EG	A small chip. It has only 3x DPUs, BRAMs. In respect to the considered algorithm, it is possible to use 2x DPUs on this chip, but each layer of each DPU will compute 16 threads instead of 4 threads. It can have the Cortex A53 4x core ARM processor or the Cortex A53 2x core ARM processor. The computation in the SW part with the 2x core ARM processor will take more time, but the computation from the viewpoint of the OpenCL will be identical, while it is actually a relation between the HW and one core of the ARM processor. In case of this platform the acceleration is supposed to be smaller.
ZU07-EG (ZCU104)	A university reference module. It has the Cortex A53 4x core ARM processor, 6x DPUs, URAMs and BRAMs. The SW and firmware will be identical to those for the ZU15-EG.

4.4. Results and Related Publications

The last stage of the research work provides the following contributions:

1. The parallel implementation of the QRD RLS Lattice algorithm on four cores of the ARM Cortex A53 processor of the Xilinx Zynq Ultrascale+ device, which succeeds to compute the algorithm using the real ultrasound data within required 60s and delivers 335 MFLOP/s, i.e. it is suitable for the real-time applications. Besides, the algorithm provides the complete information about the identification process including the probabilities for every 528 samples each 60ms. It means that the reconstruction of the hand behavior is possible for a simple gesture identification based on the calculated distance between the hand and the device.

2. The FPGA implementation of the QRD RLS Lattice algorithm using eight 8xSIMD HW accelerators including memory_read, memory_write and memory_move HLS HW kernels for data moving processes. The implementation computes 256 threads in parallel and performs the data communication between eight 8xSIMD HW accelerators. The ping-pong communication technique is applied. The algorithm provides the complete information about the identification process for every 264 samples (each 30ms). The computational time reaches 16s, i.e. it delivers 1150 MFLOP/s. In comparison with the implementation on four cores of the Cortex A53 ARM processor, 4 threads, and with the implementation on four cores of the Cortex A53 ARM processor, 256 threads, the HW acceleration reaches **3.4x** and **7.5x** respectively. The computational time of the FPGA implementation is already

comparable with the computation on the single threaded PC Intel® CoreTM i7-4770 CPU in MATLAB R2019b, which constituted 13s for the best optimized version of one core delivering outputs each 30ms.

The implemented HW/SW system and the related Poster has been presented by the ÚTIA team at the Embedded World conference, Nurnberg, 21-23.6.2022. It is available at http://storaige.eu/utia-at-demonstrators-at-the-embedded-world-2022-conference/

The following source code is available (the implementation on the Xilinx Zynq Ultrascale+ Cortex A53 ARM processor)
at https://zs.utia.cas.cz/index.php?ids=projects/storaige/disertace_Raissa_Likhonina:

- QRD RLS Lattice algorithm – one core:
  - double precision arithmetic
  - single precision arithmetic
- QRD RLS Lattice algorithm – two cores core:
  - double precision arithmetic
  - single precision arithmetic
- QRD RLS Lattice algorithm – four cores core:
  - double precision arithmetic, hypothesis testing
  - single precision arithmetic, hypothesis testing
  - single precision arithmetic, ping pong sharing technique
- QRD RLS Lattice algorithm – 256 pthreads:
  - double precision arithmetic
- Implementation in the FPGA part of Xilinx Zynq Ultrascale+ device:
  - QRD RLS Lattice algorithm – 256 threads in parallel (including SW single precision implementation)

Publications related to the research topic are the following:

1. Likhonina R., Uglickich E. Hand detection application based on QRD RLS Lattice algorithm and its implementation on Xilinx Zynq Ultrascale+. *In: Neural Network World*, 32(2), pp. 73-92, 2022, 10.14311/NNW.2022.32.005.

   Abstract: The present paper describes hand detection application implemented on Xilinx Zynq Ultrascale+ device, comprising multi-core processor ARM Cortex A53 and FPGA programmable logic. It uses ultrasound data and is based on adaptive QRD RLS Lattice algorithm extended with hypothesis testing. The algorithm chooses between two use-cases: (1) “there is a hand in front of the device” vs (2) “there is no hand in front of the device”. For these purposes a new structure of the identification models was designed. The model presenting use-case (1) is a regression model, which has the order sufficient to cover all incoming data. The model responsible for use-case (2) is a regression model, which has a smaller order than the model (1) and a certain time delay, covering the maximal distance where the hand can possibly appear. The offered concept was successfully verified using real ultrasound data in MATLAB optimized

for parallel processing and implemented in parallel on four cores of ARM Cortex A53 processor. It was proved that computational time of the algorithm is sufficient for applications requiring real-time processing.

The article is available at nnw.cz/obsahy22.html

2. Kadlec J., Likhonina R. DTRiMC tool for TE0808-09-EG-ES1 module on TEBF0808 carrier board. Application note, ÚTIA, 2021.

Abstract: Evaluation package for the Design Time Resource integration of Model Composer DTRiMC tool. It serves for integration of eight 8xSIMD, FP03x8, floating-point, run-time-reconfigurable accelerators for Zynq Ultrascale+ TE0808-09EG-ES1 module on TEBF0808 carrier board. It provides SW projects and two designs containing the HW design bitstreams and API interface for SW developer in form of shared linux libraries. The SW developer can program ARM host application in C and compile by gcc compiler or in C++ and use the g++ compiler. User can use the Xilinx SDK for compilation and debug of provided SW projects on a PC (Linux or Windows 10, 64bit). The "make" utility can be also used for compilation of host applications directly on the embedded Zynq Ultrascale+ ZU09-EG-ES1 system. All designs presented in this evaluation package contain four independent twins of serial connected FP03x8 accelerators in the programmable logic part of the device. The HW data movers supporting the data communication are represented for the SW developer as shared C/C++ library with simple SW API. The API is identical for several alternatives of HW data movers. The evaluation package includes 8xSIMD FP32 accelerators with HW license enabling only restricted number of operations. If these licensed operations are all used, user has to reset complete system. This will enable to use the licensed count of operations again.

The application note is available at
http://sp.utia.cz/index.php?ids=results&id=2017_4_te0808_fp03x8_4x2_ila_mulf64_DTRiMC

CONCLUSION

The present work is devoted to the QRD RLS algorithms and the implementation of a chosen algorithm on the HW platform from Trenz Electronic. The platform comprises the multi-core processor ARM Cortex A53 and FPGA programmable logic. The work is supported by the European project called SILENSE, which aimed at using ultrasound technology for different kind of applications in automotive, smart home, wearables and other domains.

The algorithm in this work is supposed to solve a hand detection problem using noise cancellation techniques.

After making a research of the state of the art in this field, the novelty was stated out, which can be summarized as follows:

- though the algorithms function well on large PCs, still there exists a problem of their implementation on small area chips with small memory footprints;
- there is a gap in the research area what the algorithm implementation for hand detection applications based on ultrasound is concerned;
- the presented research uses noise cancellation techniques based on the RLS algorithm to pre-process incoming ultrasound data by removing undesired ultrasound responses from the target signal, subject to use for hand detection applications;
- hypothesis testing is applied in a context different from previous research [40-41, 47-48, 50, 81]: it is used to identify the structure of a regression model and to choose a particular identification model, which corresponds better to a real-time situation;
- the approach described in the work enables to compute distance between the hand and the device.

The main goals of the work were defined as follows:

- to develop a numerically robust adaptive signal processing algorithm of recursive identification of regression models for ultrasound signals, performing noise cancellation and computing hand distance from the device,
- to implement the algorithm on embedded hardware platform, using the data processed from a microphone,
- possibly, to apply the developed algorithm for tracking of hand movement-based gestures.

The 2D tracking of hand movement-based gestures was not in the scope of the work and was not implemented. Instead, it focuses on gestures based on 1D distance measurement.

Before the very algorithm implementation, the work described mathematical tools and techniques needed to achieve stated goals. The theoretical description included the recursive Bayesian approach to system identification, types of the RLS algorithms, incorporation of hypothesis testing, FPGA techniques and tools used for the HW implementation.

The research had several stages to achieve the final goal of the algorithm implementation on the HW platform. The first large stage was to modify the existing algorithm and to incorporate hypothesis testing so that it was appropriate for making the experiments in the MATLAB R2019b environment. For these purposes, from the family of the RLS algorithms the QRD RLS Lattice algorithm was chosen.

The choice was conditioned by the inner structure of the algorithm, which allowed pipelining and parallel processing. This property was essential during the algorithm implementation on the HW platform.

Besides, in the course of the experiments the QRD RLS Lattice algorithm proved to be much faster in comparison with the QRD RLS algorithm - 45s for the QRD RLS Lattice algorithm via 2330s for the QRD RLS algorithm while processing the real data from an ultrasound microphone – and to deliver more MFLOP/s. It meant that the QRD RLS Lattice algorithm after its optimization might be fast enough to work on a small HW platform and to process the data in real time.

During the first stage of investigation, the experiments with simulated data as well as the experiments with real data from an ultrasound microphone were performed. The experiments showed that using only prediction/filtration errors for detecting the hand was not sufficient. Therefore, the algorithm was incorporated with hypothesis testing. For these purposes, a special structure of regression models was proposed. Thus, two identification models corresponding to two different use-cases (“there is a hand in front of the device” and “there is no hand in front of the device”) were designed. It was assumed that a regression model with a higher order describes the situation when there is no hand in front of the device, while a regression model with a smaller order has a higher probability when there is a hand in front of the device. In this way hypothesis testing applied to two regression model structures helps to make results of hand detection more accurate.

The orders of the models were not chosen occasionally. The model with a higher order processes all incoming data at each time step. The model with a smaller order works only with a certain amount of incoming data. Besides, the model with a smaller order has a certain time delay, which defines the distance, after which the appearance of the hand is not already possible. The choice of the order was also conditioned by further steps of the research, i.e. pipelining and parallel processing.

On this stage of the algorithm development, the experiments showed that hypothesis testing helped to make results of hand detection more accurate. The assumptions made about the structure of the regression models were also fully proved by the experiments. Moreover, an additional value of the developed algorithm was the possibility to compute the distance between the hand and the device.

Another output of this stage of investigation was the simulation in MATLAB R2019b, which was made close to the real situation by applying parameters calculated by the QRD RLS algorithm during the identification process using real ultrasound data.

During the first stage of the research, it was also stated that in order the QRD RLS Lattice algorithm could be used in the real-time applications, it should compute the outputs from the provided ultrasound data within 60s. It means that it should deliver at least 307 MFLOP/s.

The second big stage of the research was the algorithm optimisation on a PC. For these purposes a batch version of the algorithm was created and pipelining and parallel processing techniques were used. The computation was made in the MATLAB R2019b environment using the Parallel Computing Toolbox. Different versions of the algorithm were presented: for two, four and eight core processing. The experiments showed that all versions computed accurately, but parallel processing in MATLAB R2019b in this particular case did not function as required, because it did not accelerate the computation process, so that the eight core version computed slower than four core version of the algorithm: 45s vs 30s respectively for time step equal to 100. It was explained by the complexity of the data communication process.

Also, on this stage of the research both the double precision and single precision floating point versions of the algorithm were investigated. As it was assumed the single precision version was faster, but not substantially: 28s (single precision floating point arithmetic) vs 30s (double precision arithmetic) for the four core version of the algorithm with time step equal to 100.

Though the computational time seemed to be sufficient for the real-time applications on this stage of the development, but it was necessary to remember that at this point the algorithm was computed on a PC with Intel® Core™ i7-4770 CPU, 3.5 GHz. However, the HW platform, which was used on further stage of investigation, has a processor frequency of only 1.05 GHz and a programmable logic max. 240 MHz. It meant that the computational time might be insufficient.

During the experiments it was stated that for further research stage the four core version of the algorithm will be used as a golden model as far as the ARM processor has four cores. Besides, the time step should be at least 100 to ensure that the hypotheses have data every 0.6s and, thus, they are able to provide the outputs in real time with only 0.6s delay.

After the second stage of the research was successfully fulfilled and the results were verified, the QRD RLS Lattice algorithm was implemented on the four cores of the ARM processor in a way that the hypotheses had appropriate data for further computation every 60ms, i.e. the optimal time step in the case of the implementation on the HW platform proved to be 1000.

The whole computational time on the third stage of the algorithm development constituted 58s (double precision arithmetic) and 55.24s (single precision arithmetic). As far as the real data measurement was 60s, the algorithm implementation on the HW platform was considered to be fast enough to process data in real time and to deliver the results every 60ms.

In terms of MFLOP/s, the ARM device delivered 317 MFLOP/s for the double precision version of the algorithm and 333 MFLOP/s for the single precision floating point version of the algorithm. In the beginning it was stated that in order the algorithm was able to compute in real time, it was needed to ensure at least 307 MFLOP/s. Thus, this requirement was fulfilled as well.

As the last step the QRD RLS Lattice algorithm was implemented in the FPGA part of the Xilinx Zynq Ultrascale+ device, on the ZU09EG and ZU15EG. There are eight 8xSIMD HW accelerators, each of which has 8 layers, i.e. there are 64 parallel computing data paths. The HW accelerator data paths can process a separate part of the algorithm in parallel at each time step. The larger identification model was divided into 192 parts, while the smaller identification model has 64 parts. The first six layers of each accelerator process threads for the larger identification model, the two remaining layers of each accelerator process threads for the smaller identification model. Each layer processes 4 threads, each thread computes 4 orders. The system order of the first Lattice filter is 768. The system order of the second Lattice filter is 256. Data - the inputs, state variables and outputs - are saved in the linear address space device buffers in the DDR4memory.

The HW accelerator local memories for the data and for the programme are composed from URAMs 4096x64b for the ZU15EG device or BRAMs 1024x64b for the ZU09EG device, the FIFO is composed from BRAMs, 512x256b. There are eight HW blocks, which fulfil the algorithm processing including the computation and data communication. The HW supported data communication is performed by the HLS HW unit memory_read, HLS HW unit memory_write and HLS HW unit memory_move. The computation is performed by eight 8xSIMD HW accelerators. The computational time is 16s. It delivers 1150 MFLOP/s.

It means a **3.4x** acceleration in comparison with the optimized 4-pthread SW implementation on the Xilinx Zynq Ultrascale+ Cortex A53 4 core ARM device is achieved.

It is a **7.5x** acceleration in comparison with the optimized 256-pthread SW implementation on the Xilinx Zynq Ultrascale+ Cortex A53 4 core ARM device.

It should be also noted that each modification of the algorithm and each small step of its implementation both in MATLAB R2019b and on the HW platform on every stage of the development were always accompanied with verification tests to ensure that the results were equal to the reference values.

To summarize the main outputs and contributions of the research, the following should be mentioned:

- Hypothesis testing is based on identification of the structure of a regression model, after which a decision-making process is performed. The structure of each regression model is chosen in a way that it corresponds to a certain situation: a regression model with a larger order analyses all incoming data and has higher probability when there is no hand in front of the device; while a regression model with a smaller order works with a limited amount of data. It has a certain time delay and a higher probability in the moment

the hand appears in front of the device. On the basis of the probabilities of the identification models, a decision is made. In comparison to [76], this approach uses recursive Bayesian identification. However, differently from [40-41, 47-48, 50, 81], Bayesian identification is applied specifically for achieving a specific goal – hand detection and distance determination – as it was described previously.

- The algorithm allows determining the distance between the hand and the device. It is due to the form of the input signal (chirps) used in the experiments. Using the distance computation, simple types of gestures can be identified.
- The simulation in the MATLAB R2019b environment was made close to the reality by using parameters calculated during the identification process using the real ultrasound data.
- The developed algorithm was implemented both in the MATLAB R2019b environment and in C code using real data from an ultrasound microphone. Pipelining and parallel processing techniques were applied.
- The QRD RLS Lattice algorithm combined with hypothesis testing was mapped on the embedded quad-core ARM Cortex A53 processor.
- The computation of the developed algorithm was conditioned by the specificity of the HW platform, its computational resources. It was required to compute 528000 data samples within 60s. Due to the pipelining and parallel processing technique, this goal was achieved and the algorithm implementation on the HW platform can be used for the real-time processing applications.
- The FPGAs are prepared for ultrasound microphones by the ÚTIA team and can be used in the same Xilinx Zynq Ultrascale+ application.
- The FPGA implementation of the QRD RLS Lattice algorithm is performed on eight 8xSIMD HW accelerators prepared in ÚTIA. It was implemented and tested on the Xilinx ZU09EG and ZU15EG. The computational time is 16s and a **3.4x** acceleration is achieved in comparison to the 4 pthread SW implementation on the 4-core A53 ARM processor. A **7.5x** acceleration is achieved in comparison to the 256 pthread SW implementation on the 4-core A53 ARM processor.
- The power consumption is low and constitutes 13.53W in a stand-by mode and 13.94W for the single precision floating point computation.

To conclude, the proposed method for hand detection based on the noise cancellation technique and using the QRD RLS Lattice algorithm functions reliably and accurately and fulfils its goals. The algorithm implementation on the HW platform functions as reliably and sufficiently accurately as the version of the algorithm on a PC does and can be used for the real-time applications. Therefore, the research and investigation results can be considered achieving their goals successfully in the scope defined in the beginning of the work.

POTENTIAL APPLICATIONS

Touchless technologies are a natural stage of HMI development. In recent years they have become more and more frequently used and more diverse. According to Grand View Research, Asia Pacific region, mainly China, India and Japan, dominated the market in 2017 and it is supposed that the situation will remain the same at least up to 2025. In Europe the dominance in this market is held by Germany, U.K. and France. It is forecasted that the gesture recognition market will reach $30.6 billion in 2025 [9, 12, 32].

The reason why people are inclined more and more to use gesture recognition and hand tracking applications are their intuitive and user-friendly mastering and easy usage. Not the least is better ergonomics of the devices. Besides, touchless technologies also contribute to more safety, e.g. what automotive and health care applications are concerned. In the light of the coronavirus pandemic, the hygiene concerns are also becoming more and more acute.

The major areas where gesture recognition and hand tracking technologies are used nowadays are automotive, consumer electronics and healthcare fields [9, 12, 32]. The algorithm under analysis can be potentially used anywhere where noise cancellation for such kind of applications is required. It means it can be used in [12, 32]:

1. automotive including lighting system, biometric access, HUD (heads-up display), music and incoming calls control, etc.,
2. healthcare including sign language, lab & operating rooms, checking imagery with simple gestures without touching the display, etc.,
3. consumer electronics including smart home applications, smart TV, gaming consoles, smartphones, etc.,
4. others, e.g. educational hubs, hospitality, advertisement & communication, etc.

FUTURE PROSPECTS

As far as Xilinx ZU15-EG has enough URAMs, it is possible in the future to enlarge the amount of data sent and received at each time step. For these purposes, it is necessary to enlarge memory blocks A, B, Z as well as a programme block P and FIFO. In this case the P device buffer will be 8K Words long and 128b wide, device buffers for A, B, Z will be 2K Words and FIFO 1K Word. It will result in working with a larger amount of data (24x9 instead of present 12x9) and in less communication between units. Thus, it will lead to higher acceleration, possibly to the 5x acceleration in comparison with performance of the ARM 4 core processor implementation. However, such kind of a design will not be compatible with Xilinx ZU09-EG due to the size of the latter platform.

Another way leading to even a higher acceleration is to compute two tasks in parallel. For example, it is possible to have four hypotheses instead of 2 or to have two variants of the algorithm with different forgetting factors. It means the amount of communicated data will increase from 24x9 to 24x18. In this case the P device buffer will remain the same, i.e. 8K Words long and 128b wide; device buffers A, B and Z will be 4K Words each and FIFO 2K Words. The HW units will be fully used and the best efficiency will be achieved. It is supposed that the acceleration will be at least 10x in comparison with the 4 thread implementation in SW or at least 30x in comparison with the 256 thread SW implementation.

The presented HW implementation delivers approximately 1 GFLOP/s for Zynq Ultrascale+ ZU09 (with BRAMs) as well as for the ZU15 device (with only partially utilized URAMs).

In the future development, we will focus on increasing of the amount of computing (batch size) for eight 8xSIMD units. We would like to reach up to the 4 GFLOP/s performance in case of the large Zynq Ultrascale+ ZU15 device for the HW accelerated Lattice filter. This performance increase will be possible due to the optimally utilised URAM memories and large FIFO memories and due to the extension of HW computing batch length. The fixed SW overhead related to the XRT and OpenCL API will have a relatively smaller impact on the final performance of HW accelerated Lattice algorithm in comparison to the current implementation.

The gained knowledge related to the implementation of the Lattice algorithm will be applied to HW accelerations of other DSP algorithms with an internal systolic array structure.

Further work in this direction could include making a device prototype, which would comprise both ultrasound microphones and beam-former FPGA design and a hand detection application on one HW platform. This prototype could be equipped with the display, which would provide graphic output of the computation and allow tracking of hand movement-based gestures.

BIBLIOGRAPHY

[1] ALESSANDRINI M., MARCHI DE L., SPECIALE N. Recursive Least Squares adaptive filters for ultrasonic signal deconvolution. *Conference: International Symposium on Circuits and Systems (ISCAS 2008).* Seattle, Washington, USA: May, 2008, doi: 10.1109/ISCAS.2008.4542073.

[2] ALI KH. J., Mohammad H. A., WALI M. H. Implementation of a recursive data of adaptive QRD-RLS algorithm using HDL coder. *In: ResearchGate:* March, 2020 [online].
Available at https://www.researchgate.net/publication/339613298_Implementation_of_A_Recursive_Data_of_Adaptive_QRD-RLS_Algorithm_Using_HDL_Coder

[3] APOLINARIO J., DINIZ P. S. R. Fast QRD-RLS algorithms. In: QRD-RLS adaptive filtering. Springer: December, 2009, pp. 1-27,
doi: 10.1007/978-0-387-09734-3_4.

[4] ARDALAN S. Floating-point error analysis of recursive least-squares and least-meansquares adaptive filters. *In: ICASSP '86. IEEE International Conference on Acoustics, Speech, and Signal Processing,* 1986, pp. 513-516, doi: 10.1109/ICASSP.1986.1169030.

[5] ARDALAN S., ALEXANDER S. Fixed-point roundoff error analysis of the exponentially windowed RLS algorithm for time-varying systems. *In: IEEE Transactions on Acoustics, Speech and Signal Processing,* Vol. 35, No. 6, pp. 770-783, June 1987, doi: 10.1109/TASSP.1987.1165207.

[6] AUGUSTO SOARES, J. What is FPGA? How does that works? *Quora,* 2017 [online]. Available at https://www.quora.com/What-is-FPGA-How-does-that-works

[7] BENESTY J., GAENSLER T. New Insights into the RLS Algorithm. *In: EURASIP journal on advances in signal processing*, March, 2004(3), doi: 10.1155/S1110865704310188

[8] BENZIANE M., BOUAMAR M., MAKDIR M. Simple and efficient double-talk-detector for acoustic echo cancellation. *Traitement du Signal,* Vol. 37, No. 4, pp. 585-592, doi: https://doi.org/10.18280/ts.370406

[9] BONDALAPATI K., PRASANNA V. K. Reconfigurable computing systems. *In: Proceedings of the IEEE,* Vol. 90, No. 7, pp. 1201–1217, July 2002, doi: 10.1109/JPROC.2002.801446.

[10] BOPPANA D., DHANOA K., KEMPA J. FPGA based embedded processing architecture for the QRD-RLS algorithm. *In: 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines,* 2004, pp. 330-331, doi: 10.1109/FCCM.2004.34.

[11] BOTTOMLEY G. E., ALEXANDER S. T. A novel approach for stabilizing recursive least squares filters. *IEEE Transactions on Signal Processing,* Vol. 39, Issue 8, August 1991, pp. 1770-1779, doi: https://doi.org/10.1109/78.91147.

[12] BUBNIUK, N. Hand Tracking and Gesture Recognition Using AI: Applications and Challenges. *Intellias: Intelligent Software Engineering [online]*, August 14, 2020. Available at https://www.intellias.com/hand-tracking-and-gesture-recognition-using-ai-applications-and-challenges/

[13] CHANDRAKASAN A. P., BRODERSEN R. W. Low power digital CMOS design. Springer New York, NY, Edition 1, 1995,
doi: https://doi.org/10.1007/978-1-4615-2325-3.

[14] CHEN CHAU-SHEN, HWANG TING TING, LIU C. L. Low power FPGA design – a re-engineering approach. *In: Proceedings of the 34th Design Automation Conference*, 1997, pp. 656–661, doi: 10.1109/DAC.1997.597226.

[15] CIOFFI J. M., KAILATH T. Fast recursive-least-squares transversal filters for adaptive filtering. *IEEE Transactions on Acoustics Speech and Signal Processing,* V. 32, Issue 2, May 1984, pp. 304-337, doi: 10.1109/TASSP.1984.1164334.

[16] CIOFFI J., KAILATH T. Windowed fast transversal filters adaptive algorithms with normalization. *In: IEEE Transactions on Acoustics, Speech, and Signal Processing,* Vol. 33, No. 3, pp. 607=625, June 1985,
doi: 10.1109/TASSP.1985.1164585.

[17] CIOFFI J. The fast householder filters RLS adaptive algorithm RLS adaptive filter. *In: International Conference on Acoustics, Speech and Signal Processing*, Vol. 3, pp. 1619–1621, 1990, doi: 10.1109/ICASSP.1990.115735.

[18] CIOFFI J. Limited-precision effects in adaptive filtering. *In: IEEE Transactions on Circuits and Systems,* Vol. 34, No. 7, pp. 821-833, July 1987,
doi: 10.1109/TCS.1987.1086209.

[19] CONSTANTINIDES G. A., CHEUNG P. Y. K., LUK W. Synthesis and optimization of DSP algorithms. Springer New York, NY, 2004, doi: https://doi.org/10.1007.b116503.

[20] CUMMING, G. The new statistics: why and how. *Psychological Science,* 25(1), 2014.

[21] DINIZ P. S. R. Adaptive filtering: algorithms and practical implementation. Third edition. Springer, Boston, MA: 2008, doi: 10.1007/978-0-387-68606-6.

[22] DINIZ P. S. R. Adaptive Lattice-based RLS algorithms. *In: Adaptive Filtering*. Fourth edition. Springer, Boston, MA: 2013, doi: 10.1007/978-1-4614-4106-9_7.

[23] DINIZ P. S. R., SIQUEIRA M. G. Finite and infinite-precision properties of QRD-RLS algorithms. *In: QRD-RLS Adaptive Filtering.* Springer, Boston, MA: December, 2009, doi: 10.1007/978-0-387-09734-3_9.

[24] DJIGAN V. I. RLS adaptive filtering algorithms based on parallel computations. *Radioengineering: Proceedings of Czech and Slovak Technical Universities and URSI Committers*, January, 2005.

[25] DOUGLAS S. C., ZHU QUANHONG, SMITH K. F. A pipelined architecture for LMS adaptive FIR filter architecture without adaptation delay. *In: IEEE Transactions on Signal Processing*, Vol. 46, No. 3, pp. 775-779, March 1998, doi: 10.1109/78.661345.

[26] ECSEL Research and Innovation Actions (RIA). *Proposal Outline: (Ultra)Sound Interfaces and Low Energy iNtegrated Sensors.* SILENSE: Calls, 2016.

[27] EKLUND C. Pipelining and Parallel Processing. *Nokia Research Center*, FIN-00045 Nokia Group, October 13, 1999. Available at http://www.netlab.tkk.fi/opetus/s38220/f99/rpt99f2.pdf

[28] FANG F., CHEN T., RUTEBNBAR R. A. Floating-point bit-width optimisation for low-power signal processing applications. *In: 2002 IEEE International Conference on Acoustics, Speech, and Signal Processing,* 2002, pp. III-3208-III-3211, doi: 10.1109/ICASSP.2002.5745332.

[29] FABRE P., GUEGUEN C. Improvement of the fast recursive least-squares algorithms via normalization: a comparative study. *In: IEEE Transactions. On Acoustics, Speech, and Signal Processing,* Vol. 34, Issue 2, April 1986, pp. 296-308, doi: 10.1109/TASSP.1986.1164813.

[30] FARHANG-BOROUJENY B. Adaptive filters theory and applications. *John Wiley & Sons Inc.*, 1998

[31] GOSLIN G. Using Xilinx FPGAs to design custom digital signal processing devices. *In: Proceedings of the DSPX 1995 Technical Proceedings*, pp. 565–604, 1995, 12NOV94.

[32] Grand View Research. Gesture Recognition Market Size, Share & Trends Analysis Report by Technology (Touch-based, Touchless), by Industry (Automotive, Consumer Electronics, Healthcare), and Segment Forecasts, 2019-2025. *Report ID: GVR-2-68038-019-4.* January, 2019. Available at https://www.grandviewresearch.com/industry-analysis/gesture-recognition-market

[33] GUPTA V. K., CHANDRA M., SHARAN S. N. Noise minimization from speech signals using RLS algorithm with variable forgetting factor. *In: Research Journal of Applied Sciences, Engineering and Technology.* 4(17): 3102-3107, August, 2012.

[34] HARTFIEL J. TE0808 TRM. *In: Trenz Electronic Wiki* (last modified on June 01, 2020). Available at https://wiki.trenz-electronic.de/display/PD/TE0808+TRM

[35] HARTFIEL J. TEBF0808 TRM. *In: Trenz Electronic Wiki* (last modified on March 12, 2019). Available at https://wiki.trenz-electronic.de/display/PD/TEBF0808+TRM

[36] HAYES M. H. Statistical digital signal processing and modelling. NY: June, 1996. ISBN: 978-0-471-59431-4

[37] HSIEH S. F., LIU K. J. R., YAO K. A unified approach for QRD-based recursive leastsquares estimation without square roots. *In: IEEE Transactions on Signal Processing*, Vol. 41, No. 3, pp. 1405–1409, March 1993, doi: 10.1109/78.205742.

[38] HORITA E., MIYANAGE Y. Numerically stable RLS algorithms for time-varying signals. *In: Wiley Online Library, Electronics and Communications in Japan (Part III: Fundamental Electronic Science),* Vol. 82, Issue 4, pp. 26-37, January 1999, doi: https://doi.org/10.1002/(SICI)1520-6440(199904)82:4<26::AID-ECJC4>3.0.CO;2-B

[39] IGLESIAS M. E. Implementation of QRD-RLS algorithm on FPGA. Application to noise canceller system. *In: IEEE Latin America Transactions*, Vol. 9, No. 4, pp. 458-462, July, 2011, doi: 10.1109/TLA.2011.5993728.

[40] KADLEC J. Průběžná pravděpodobnostní identifikace autoregresního modelu s neznámým řádem. *In: Analýza, syntéza a rozpoznávání řeči,* ČSVTS, Praha 1985

[41] KADLEC J. Pravděpodobnostní identifikace regresního modelu v pevné řádové čárce. Praha, 1986

[42] KADLEC J. Eight FP03x8 accelerators for TE0808-09-EG-ES1 module on TEBF0808 carrier board. Application note, ÚTIA, 2021.

[43] KADLEC, J., LIKHONINA, R. Adaptive RLS algorithms reference implementations with custom arithmetic. Application note: ÚTIA AV ČR, v.v.i., 2017

[44] KADLEC, J., POHL, Z., STEVEN VAN DER VLUGT, JÄÄSKELÄINEN, P., KOSKINEN, L. Algorithms, design methods, and many-core execution platform for low-power massive data-rate video and image processing. Almarvi, 2016

[45] KADLEC, J., POHL, Z., KOHOUT, L. Two serial connected evaluation versions of FP03x8 accelerators for TE0820-03-4EV-1E module on TE0701-06 carrier board. Application note: ÚTIA AV ČR, v.v.i., 2019

[46] KARKOOTI M., CAVALLARO J. R., DICK C. FPGA implementation of matrix inversion using QRD-RLS algorithm. *In: Conference Record of the Thirty-Ninth Asilomar Conference on Signals, Systems and Computers*, pp. 1625-1629, January, 2005, doi: 10.1109/ACSSC.2005.1600043.

[47] KÁRNÝ M. Bayesian estimation of model order. *In: Problems of Control and Information Theory*, Vol. 9, No. 1, pp. 33-46, 1980, MR0561805.

[48] KÁRNÝ M., BÖHM J., GUY T. V., JIRSA L., NAGY I., NEDOMA P., TESAŘ L. Optimized Bayesian dynamic advising. *In: Theory and Algorithms.* Springer, London: 2006.

[49] KIM J. Y., LEE Y. I., KIM K. J., NAM S. W., KO C. C. Utilization of a correlation RLS algorithm for nonlinear echo cancellation. *In: WSEAS Transactions on Circuits and Systems,* Vol. 5, No. 4, pp. 435-441, April, 2006. Available at http://scholarbank.nus.edu.sg/handle/10635/57763

[50] KULHAVÝ R.: Směrové zapomínání a průběžná identifikace systémů s pomalu se měnícími parametry. Výzkumná zpráva č. 1170, ÚTIA ČSAV 1983

[51] KUNG S. Y. VLSI array processing. Prentice.Hall; 1988.

[52] LEE D., MORF M., FRIDLANDER B. Recursive least-squares ladder estimation algorithms. *In: IEEE Transactions on Circuits and Systems,* Vol. 28, No. 6, pp. 467-481, June 1981, doi: 10.1109/TCS.1981.1085020.

[53] LEUNG H., HAYKIN S. Stability of recursive QRD-LS algorithms using finite-precision systolic array implementation. *In: IEEE Transactions on Acoustics, Speech, Signal Processing,* Vol. 37, No. 5, pp. 760-763, May 1989, doi: 10.1109/29.17570.

[54] LEV-ARI H., KAILATH T., CIOFFI J. Least squares adaptive lattice and transversal filters: a unified geometric theory. *In: IEEE Transactions on Information Theory,* Vol. 30, No. 2, pp. 222-236, March 1984, doi: 10.1109/TIT.1984.1056882.

[55] LI TS., TIAN K., LI WX. Method for improving RLS algorithms. *In: Journal of Marine Science and Application,* Vol. 6, No. 3, pp. 68-70, September, 2007, doi: 10.1007/s11804-007-5077-x.

[56] LI ZHU, LI CHAO. LMS and RLS algorithms comparative study in system identification. *In: 2011 International Conference on Multimedia Technology,* pp. 5428-5430, 2011, doi: 10.1109/ICMT.2011.6002287.

[57] LIGHTBODY G. High performance VLSI architectures for recursive least squares adaptive filtering. PhD Thesis Queen's University Belfast, 1999.

[58] LIGHTBODY G., WOODS R., WALKE R. Design of a parameterizable silicon intellectual property core for QR-based RLS filtering. *In: IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* Vol. 11, No. 4, pp. 659–678, August 2003, doi: 10.1109/TVLSI.2003.816142.

[59] LIKHONINA, R., KOHOUT, L., KADLEC, J. Camera-to-touchscreen design. *Proceedings of 6th International Workshop on Mathematical Models and their Applications (IWMMA'2017),* pp. 94-99, 6th International Workshop on Mathematical Models and their Applications (IWMMA'2017), Krasnoyarsk, 2017.

[60] LIKHONINA R., KADLEC, J. Noise Cancellation Using QRD RLS Algorithms. Application Note: ÚTIA AV ČR, v.v.i., 2018.

[61] LIKHONINA R. Hand Gesture Recognition Based on Ultrasound Technology: Pre-processing Stage. *2019 8th Mediterranean Conference on Embedded Computing (MECO),* pp. 1-6, 2019, doi: 10.1109/MECO.2019.8760063.

[62] LIKHONINA, R. QRD RLS Algorithm for Hand Gesture Recognition Applications. *2019 International Conference on Systems, Signals and Image Processing (IWSSIP),* pp. 195-200, 2019, doi: 10.1109/IWSSIP.2019.8787283.

[63] LIKHONINA R. Hand detection algorithm: pre-processing stage. *In: Proceedings of the 17th international conference on informatics in control, automation and robotics,* pp. 695-701, ICINCO 2020 (17th international conference on informatics in control, automation and robotics), 2020, doi: 10.5220/0009885206950701.

[64] LIKHONINA R., UGLICKICH E. Hand detection application based on QRD RLS Lattice algorithm and its implementation on Xilinx Zynq Ultrascale+. *In: Journal "Neural Network World*", 32(2), pp. 73-93, 2022, doi: 10.14311/NNW.2022.32.005.

[65] LIU K. J. R., HSIEH S. F., YAO K. Recursive LS filtering using block Householder transformations. *International Conference on Acoustics, Speech, and Signal Processing,* Vol. 3, pp. 1631–1634, 1990,
doi: 10.1109/ICASSP.1990.115739.

[66] LIU K. J. R., HSIEH S. F., YAO K. Systolic block Householder transformations for RLS algorithm with two-level pipelined implementation. *In: IEEE Transactions on Signal Processing,* Vol. 40, No. 4, pp. 946–958, April 1992, doi: 10.1109/78.127965.

[67] LJUNG S., LJUNG L. Error propagation of recursive least squares adaptation algorithms. *In: IFAC Proceedings Volumes, Elsevier,* Vol. 17, Issue 2, pp. 677-681, July 1984, doi: https://doi.org/10.1016/S1474-6670(17)61049-8.

[68] LJUNG S., LJUNG L. Error propagation properties of recursive least squares adaptation algorithms. *In: Automatica, Elsevier,* Vol. 21, Issue 2, pp. 157-167, March 1985, doi: https://doi.org/10.1016/0005-1098(85)90110-4.

[69] MARTINEK R., KAHANKOVA R., NEDOMA J., FAJKUS M., SKACEL M. Comparison of the LMS, NLMS, RLS, and QR-RLS algorithms for vehicle noise suppression. *In: ICCMS 2018, Proceedings of the 10th International Conference on Computer Modeling and Simulation,* pp. 23-27, January, 2018, doi: 10.1145/3177457.3177502.

[70] MATLAB. (9.7.0.1261785). (R2019b), Natick, Massachusetts: The MathWorks Inc., 2019.

[71] MathWorks, Inc.: Parallel Computing Toolbox, 1994-2020. Available at https://uk.mathworks.com/products/parallel-computing.html

[72] MATSUBARA K., NISHIKAWA K., KIYA H. Pipelined LMS adaptive digital filter based on look-ahead delayed LMS algorithm. *In: IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing,* Vol. 46, No. 1, pp. 51-55, January 1999, doi: 10.1109/82.749082.

[73] MCWHIRTER J. G., KADLEC J., WALKE R. L. Normalised givens rotations for recursive least squares processing. *VLSI Signal Processing, VIII*, pp. 313–332, 1995, doi: 10.1109/VLSISP.1995.527502.

[74] MEYER-BAESE U. Digital signal processing with field programmable gate arrays. Springer Berlin, Heidelberg, 2014, doi: https://doi.org/10.1007/978-3-642-45309-0.

[75] MOHAMMAD M., AL-SCHEBEILLI S.. Finite-precision analysis: Fast QR-decomposition algorithm. *In: The 10th IEEE International Symposium on Signal Processing and Information Technology,* 2010, pp. 161-165, doi: 10.1109/ISSPIT.2010.5711765.

[76] MOONEN M. Introduction to adaptive signal processing. K. U. Leuven, Leuven, Belgium: 1999.

[77] MUNJAL A., AGGARWAL V., SINGH G. RLS algorithm for acoustic echo cancellation. *In: Proceedings of the 2nd National Conference on Challenges and Opportunities in Information Technology (COIT-2008) RIMT-IET,* Mandi Gobindgarh, March, 2008. Available at https://www.researchgate.net/publication/228901207_RLS_algorithm_for_acoustic_echo_cancellation.

[78] NISHIKAWA K., KIYA H. Pipeline implementation of gradient-type adaptive filters. *In: Wiley Online Library, Electronics and Communications in Japan (Part III: Fundamental Electronic Science),* Vol. 84, Issue 5, pp. 33-42, January 2001, doi: https://doi.org/10.1002/1520-6440(200105)84:5<33::AID-ECJC4>3.0.CO;2-5.

[79] NISHIKAWA K., KIYA H. Novel implementation technique of RLS algorithm for improving throughput of adaptive filters. *In: 10th European Signal Processing Conference,* pp. 1-4, 2000. Available at https://ieeexplore.ieee.org/document/7075545/metrics#metrics.

[80] OmniSci. Parallel Computing Definition, 2020. Available at https://www.omnisci.com/technical-glossary/parallel-computing

[81] PETERKA V. Bayesian approach to system identification. *In: Trends and Progress in System Identificaion* (P. Eykhoff, ed.), IFAC Series for Graduates, Research Workers and Practicing Engineers, Pergamon Press, 1981.

[82] PILLAI R. V. K., AL-KHALILI D., AL-KHALILI A. J., SHAH S. Y. A. A low power approach to floating point adder design for DSP applications. *In: Journal of VLSI Signal Processing.* Vol. 27, Issue 3, pp. 195–213, 2001, doi: 10.1023/A:1008140025773.

[83] PIRSCH P. Architectures for digital signal processing. Wiley; 1998.

[84] POHL Z., TICHY M., KADLEC J. Implementation of the Least-Squares Lattice with order and forgetting factor estimation for FPGA. *In: EURASIP Journal on Advances in Signal Processing*, 2008, 394201, doi: 10.1155/2008/394201.

[85] POHL, Z., KOHOUT, L.. UTIA Evaluation Board v1.7-v1.8 Beamforming Demo. *Application Note:* UTIA AV CR, v.v.i., 2019

[86] PORAT B., FRIDLANDER B., MORF M. Square-root covariance ladder algorithm. *IEEE International Conference on Acoustics, Speech, and Signal Processing,* pp. 877-880, 1981, doi: 10.1109/ICASSP.1981.1171192.

[87] PORAT B., KAILATH T. Normalized lattice algorithms for least squares FIR system identification. *In: IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol. 31, No. 1, pp. 122-128, February 1983,
doi: 10.1109/TASSP.1983.1164012.

[88] RAGHUNATH K. J., PARHI K. K. Finite-precision error analysis of QRD-RLS and STAR-RLS adaptive filters. *In: IEEE Transactions Signal Processing,* Vol. 45, No. 5, pp. 1193-1209, May 1997, doi: 10.1109/78.575694.

[89] RAWAT D., KUMAR A., JOSHI S., KESTWAL M. Ch. Comparison and simulation of adaptive equalizer of LMS, RLS algorithm using Matlab. ResearchGate: March, 2013.

[90] REGALIA P. A. Numerical stability properties of a QR-based fast least squares algorithm. *In: IEEE Transactions on Signal Processing,* Vol 41, No. 6, pp. 2096-2109, June, 1993, doi: 10.1109/78.218139.

[91] SAKAI H. Recursive leaset-squares algorithms of modified Gram-Schmidt type for parallel weight extraction. *In: IEEE Transactions on Signal Processing,* Vol. 42, No. 2, pp. 429-433, February 1994, doi: 10.1109/78.275620.

[92] SAMSON L., REDDY V. V. Fixed point error analysis of normalized ladder algorithm. *IEEE International Conference on Acoustics, Speech, and Signal Processing,* pp. 1752-1755, 1982, doi: 10.1109/ICASSP.1982.1171689.

[93] SETHY PRABIRA. Noise cancellation in adaptive filtering through RLS algorithm using TMS320C6713DSK. *In: International Journal of Electronics and Communication Engineering and Technology (IJECET),* Vol. 3, Issue 1, January-June, 2012, pp. 154-159. Available at
https://www.researchgate.net/publication/272175478_NOISE_CANCELLATION_IN_ADAPTIVE_FILTERING_THROUGH_RLS_ALGORITHM_USING_TMS320C6713DSK.

[94] SHANBHAG, N. R., PARHI, K. K. Pipelined adaptive digital filters. Springer New York, NY, 1994, doi: https://doi.org/10.1007/978-1-4615-2678-0.

[95] SILENSE. About the project [online]. Available at https://silense.eu/project

[96] SLOCK D. T. M., KAILATH T. (1991) Numerically stable fast transversal filters for recursive least least-squares adaptive filtering. *In: Golub G. H., Van Dooren P. (eds) Numerical Linear Algebra, Digital Signal Processing and Parallel Algorithms. NATO*

ASI Series (Series F: Computer and Systems Sciences), Vol. 70, pp. 605-615, Springer, Berlin, Heidelberg, doi: 10.1007/978-3-642-75536-1_49.

[97] StorAIge. Embedded storage elements on next MCU generation ready for AI on the edge, 2022. Available at https://storaige.eu/

[98] Studytonight. Education simplified. Pipelining, 2020. Available at https://www.studytonight.com/computer-architecture/pipelining

[99] SUKHUMALCHAYAPHONG S., BENJANGKAPRASERT C. Variable forgetting factor RLS algorithm for adaptive echo cancellation. *In: 14th International Conference on Control, Automation and Systems (ICCAS 2014)*, 2014, pp. 971-974, doi: 10.1109/ICCAS.2014.6987926.

[100] THIBAULT S., PELLERIN D. Practical FPGA programming in C. 1st edition, Prentice Hall, 2005, ISBN-13: 978-0-13-154318-8.

[101] THIRIPURASUNDARI C., SUMATHY V. Efficient FPGA architecture for RLS algorithm based adaptive beam forming for smart antenna system. *Asian Journal of Information Technology*. Vol. 15, Issue 16, pp. 3108-3124, January, 2016, doi: 10.3923/ajit.2016.3108.3124.

[102] Trenz Electronic. UltraSOM+ MPSoC Module with Zynq UltraScale+ ZU15EG and mounted Heat Spreader. Product description [online]. Available at https://shop.trenz-electronic.de/en/TE0808-05-BBE21-AK-UltraSOM-MPSoC-Module-with-Zynq-UltraScale-ZU15EG-and-mounted-Heat-Spreader

[103] Vitis v2019.2 (64-bit). Xilinx Vitis Development Environment. Xilinx, Inc., 1986-2019.

[104] Vivado v2017.4 (64-bit). Xilinx Inc., 2010 Logic Drive, San Jose, CA 95124.

[105] WOLF W. FPGA-based system design. Prentice-Hall, New Jersey: 2004.

[106] WOODS R., MCALLISTER J., LIGHTBODY G., YI Y. FPGA-based implementation of signal processing systems. Wiley, 2008.

[107] Xilinx. SDSoC environment user guide. UG1027 (v2017.2), pp. 6-9. August 2017 [online]. Available at https://www.xilinx.com/cgi-bin/docs/rdoc?v=2017.2;d=ug1027-sdsocuser-guide.pdf

[108] Xilinx SDK v2017.4 (64-bit). Xilinx Software Development Kit. Release Version: 2017.4. Eclipse contributors and others (2000, 2009), Apache Software Foundation, Xilinx Inc., 2009-2017.

[109] Xilinx. Vitis Unified Software Platform Overview [online]. Available at https://www.xilinx.com/products/design-tools/vitis/vitis-platform.html

[110] XU J., ZHOU W., GUO Y. A simplified RLS algorithm and its application in acoustic echo cancellation. *In: 2nd International Conference on Information Engineering and Computer Science,* 2010, pp. 1-4, doi: 10.1109/ICIECS.2010.5678354.

APPENDICES

APPENDIX 1

Hypothesis Testing about the Order of a Regression Model

Here it is shown that instead of using several algorithms with two or more different orders, it is possible in some cases to use one algorithm with so-called nested orders.

Using the properties of the nested orders it can be shown that

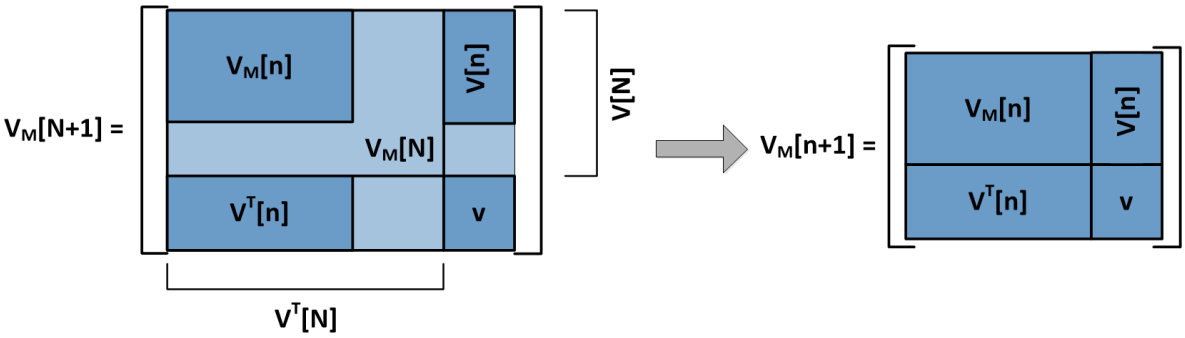

Figure A.1: Example of nested order

It means that relation (2.21) can be rewritten as follows for $\boldsymbol{n = 1, \dots, N}$:

$$\overline{\boldsymbol{V}}_{\boldsymbol{M}}[\boldsymbol{n+1}] = \boldsymbol{V}_{\boldsymbol{M}}[\boldsymbol{n+1}] + \begin{bmatrix} \boldsymbol{Z}[\boldsymbol{n}] \\ \boldsymbol{z}_{N+1} \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{Z}[\boldsymbol{n}] \\ \boldsymbol{z}_{N+1} \end{bmatrix}^{\boldsymbol{T}}, \qquad (\boldsymbol{A.1})$$

where $\boldsymbol{z}_{N+1}$ represents the output $\boldsymbol{y}_t$.

The following equations are presented for an order $\boldsymbol{n}$ nested in $\boldsymbol{N}$, for $= \boldsymbol{1, \dots, N}$. However, they are valid for order $\boldsymbol{N}$ too as it is obvious from Fig. A.1.

Using this decomposition and completion to the full squares, the conditional probability density can be written in the following form [41]:

$$\boldsymbol{p[\theta, \omega_t | D(t-1)] = k \cdot \omega_t^{\frac{\nu+n-2}{2}} \cdot \exp\left\{-\frac{\omega_t}{2} \cdot \Lambda\right\} \cdot}$$

$$\boldsymbol{\cdot \exp\left\{-\frac{\omega_t}{2} \cdot (\theta - \theta[n])^T \cdot C[n] \cdot (\theta - \theta[n])\right\}}, \qquad (\boldsymbol{A.2})$$

where

$$\boldsymbol{C}[\boldsymbol{n}] = \boldsymbol{V}_{\boldsymbol{M}}^{-\mathbf{1}}[\boldsymbol{n}], \tag{A.3}$$

$$\boldsymbol{\theta}[\boldsymbol{n}] = \boldsymbol{C}[\boldsymbol{n}] \cdot \boldsymbol{V}[\boldsymbol{n}], \tag{A.4}$$

$$\boldsymbol{\Lambda} = \mathbf{v} - \boldsymbol{V}^{\boldsymbol{T}}[\boldsymbol{n}] \cdot \boldsymbol{C}[\boldsymbol{n}] \cdot \boldsymbol{V}[\boldsymbol{n}], \tag{A.5}$$

$$\boldsymbol{\omega} = \boldsymbol{\mathcal{V}} \cdot \boldsymbol{\Lambda}^{-\mathbf{1}}. \tag{A.6}$$

Vector $\boldsymbol{\theta}[\boldsymbol{n}]$ enables to express the conditional mean value by the probability distribution of the unknown regression parameters. It is a vector of regression coefficient estimates [40-41, 47-48, 50, 81].

$\boldsymbol{\Lambda}$ is a scalar, which enables to express the conditional mean value by the distribution of unknown parameters. It is a residue after completion to the full squares [40-41, 47-48, 50, 81].

$\boldsymbol{k}$ is a normalizing constant, while $\boldsymbol{\omega_t}$ is an unknown degree of accuracy [40-41, 47-48, 50, 81].

Further, it will be shown how the update of the characteristics is performed. To simplify notation, instead of $(\boldsymbol{t}|\boldsymbol{t})$under the characteristics the upper line " ¯ " above the letter is used to show that the corresponding characteristics is after updating with $\boldsymbol{y_t}$. The index $(\boldsymbol{t}|\boldsymbol{t}-\mathbf{1})$ is omitted under the characteristics, which means that the corresponding characteristics is before updating with $\boldsymbol{y_t}$ [40-41, 47-48, 50, 81].

$$\overline{\boldsymbol{V}}_{\boldsymbol{M}}[\boldsymbol{n}] = \boldsymbol{V}_{\boldsymbol{M}}[\boldsymbol{n}] + \boldsymbol{Z}[\boldsymbol{n}] \cdot \boldsymbol{Z}^{\boldsymbol{T}}[\boldsymbol{n}], \tag{A.7}$$

$$\overline{\mathbf{v}} = \mathbf{v} + \boldsymbol{z}_{\boldsymbol{N+1}}^{\mathbf{2}}, \tag{A.8}$$

$$\overline{\boldsymbol{V}}[\boldsymbol{n}] = \boldsymbol{V}[\boldsymbol{n}] + \boldsymbol{Z}[\boldsymbol{n}] \cdot \boldsymbol{z}_{\boldsymbol{N+1}}, \tag{A.9}$$

$$\overline{\boldsymbol{C}}[\boldsymbol{n}] = \boldsymbol{C}[\boldsymbol{n}] - \frac{\boldsymbol{C}^{\boldsymbol{T}}[\boldsymbol{n}] \cdot \boldsymbol{Z}[\boldsymbol{n}] \cdot \boldsymbol{Z}^{\boldsymbol{T}}[\boldsymbol{n}] \cdot \boldsymbol{C}[\boldsymbol{n}]}{\mathbf{1} + \boldsymbol{\xi}}, \tag{A.10}$$

$$\overline{\boldsymbol{\theta}}[\boldsymbol{n}] = \boldsymbol{\theta}[\boldsymbol{n}] + \frac{\boldsymbol{C}^{\boldsymbol{T}}[\boldsymbol{n}] \cdot \boldsymbol{Z}[\boldsymbol{n}]}{\mathbf{1} + \boldsymbol{\xi}} \cdot \boldsymbol{e}, \tag{A.11}$$

$$\overline{\boldsymbol{\Lambda}} = \boldsymbol{\Lambda} + \frac{\boldsymbol{e} \cdot \boldsymbol{e}}{\mathbf{1} + \boldsymbol{\xi}}, \tag{A.12}$$

where

$$\boldsymbol{\xi} = \boldsymbol{Z}^{\boldsymbol{T}}[\boldsymbol{n}] \cdot \boldsymbol{C}[\boldsymbol{n}] \cdot \boldsymbol{Z}[\boldsymbol{n}], \tag{A.13}$$

$$\boldsymbol{e} = \boldsymbol{z}_{\boldsymbol{N+1}} - \boldsymbol{\theta}^{\boldsymbol{T}}[\boldsymbol{n}] \cdot \boldsymbol{Z}[\boldsymbol{n}] = [-\boldsymbol{\theta}^{\boldsymbol{T}}[\boldsymbol{n}], \mathbf{1}] \cdot \begin{bmatrix} \boldsymbol{Z}[\boldsymbol{n}] \\ \boldsymbol{z}_{\boldsymbol{N+1}} \end{bmatrix}, \tag{A.14}$$

$\boldsymbol{e}$ is a prior prediction error, $\boldsymbol{z_{N+1}}$ is the output $\boldsymbol{y_t}$. It is the last element in vector $\boldsymbol{Z[N+1]}$.

Note also that the density $\boldsymbol{p(y_t|D(t-1), H_n)}$ has the following form [40-41, 47-48, 50, 81]:

$$\boldsymbol{p(y_t|D(t-1);\ H_n) = \pi^{-1/2} \cdot \frac{\Lambda^{\left(\frac{\vartheta-n}{2}+1\right)}}{\bar{\Lambda}^{\left(\frac{\bar{\vartheta}-n}{2}+1\right)}} \cdot \frac{|V_M[n]|^{\frac{1}{2}}}{|\bar{V}_M[n]|^{\frac{1}{2}}} \cdot \frac{\Gamma\left(\frac{\bar{\vartheta}-n}{2}+1\right)}{\Gamma\left(\frac{\vartheta-n}{2}+1\right)}}, \qquad (A.15)$$

where $\boldsymbol{\vartheta}$ is the number of data samples accumulated in $\boldsymbol{V_M[n]}$, $\boldsymbol{|V_M[n]|}$ is a determinant of matrix $\boldsymbol{V_M[n]}$, $\Gamma(\cdot)$ is a Gamma function.

This probability density has a Student distribution with $\boldsymbol{\vartheta - n + 2}$ degrees of freedom [40-41, 47-48, 50, 81].

Using the analytical relations, there is no need to work with the whole probability density, which is demanding on computer resources. The certain values of $\boldsymbol{y_t}$ and $\boldsymbol{u_t}$ can be obtained to calculate the probability of the hypothesis according to the equation (1.14).

It means that it can be implemented inside one algorithm with nesting orders, each hypothesis representing one of the nested orders within a large order. However, it is valid if the exponential forgetting is used. Otherwise, it is necessary to have several algorithms of different orders and compute them separately, which is supposed to be implemented in the present work.

Note also that the summation of the computed probabilities has to be equal to one.

For the characteristics updates for further steps the forgetting factor is used. The main requirement when implementing the forgetting technique is to keep parameters and the degree of accuracy unchanged by the forgetting factor, i.e.

$$\boldsymbol{\theta_{(t|t-1)} = \theta_{(t-1|t-1)}}, \qquad (A.16)$$

$$\boldsymbol{\omega_{(t|t-1)} = \omega_{(t-1|t-1)}}. \qquad (A.17)$$

The equations for the characteristics updates for further steps of computation using the exponential forgetting are the following [40-41, 47]:

$$\boldsymbol{\bar{\mathcal{V}} = \alpha \cdot \mathcal{V}}, \qquad (A.18)$$

$$\boldsymbol{\bar{V}_M[n+1] = \alpha \cdot V_M[n+1]}, \qquad (A.19)$$

$$\boldsymbol{\bar{V}[n] = \alpha \cdot V[n]}, \qquad (A.20)$$

$$\boldsymbol{\bar{V}_M[n] = \alpha \cdot V_M[n]}, \qquad (A.21)$$

$$\boldsymbol{\bar{\mathrm{v}} = \alpha \cdot \mathrm{v}}, \qquad (A.22)$$

$$\overline{\boldsymbol{C}}[\boldsymbol{n}] = \frac{\mathbf{1}}{\boldsymbol{\alpha}} \cdot \boldsymbol{C}[\boldsymbol{n}], \qquad (\boldsymbol{A.23})$$

$$\overline{\boldsymbol{\theta}}[\boldsymbol{n}] = \boldsymbol{\theta}[\boldsymbol{n}], \qquad (\boldsymbol{A.24})$$

$$\overline{\boldsymbol{\Lambda}} = \boldsymbol{\alpha} \cdot \boldsymbol{\Lambda}, \qquad (\boldsymbol{A.25})$$

where $\boldsymbol{\alpha}$ is a forgetting factor ranging from 0 to 1.

It is obvious from these relations that the exponential forgetting keeps the mean value of parameters [40-41, 47-48, 50, 81]:

$$\boldsymbol{E}[\boldsymbol{\theta_{t+1}}, \boldsymbol{\omega_{t+1}} | \boldsymbol{D}(\boldsymbol{t})] = \boldsymbol{E}[\boldsymbol{\theta_t}, \boldsymbol{\omega_t} | \boldsymbol{D}(\boldsymbol{t})]. \qquad (\boldsymbol{A.26})$$

The relations using the directional forgetting look as follows [40-41, 50]:

$$\overline{\boldsymbol{\mathcal{V}}} = \boldsymbol{\alpha} \cdot \boldsymbol{\mathcal{V}},$$

$$\overline{\boldsymbol{\Lambda}} = \boldsymbol{\alpha} \cdot \boldsymbol{\Lambda}.$$

for $\xi > 0$:

$$\overline{\boldsymbol{V}}_{\boldsymbol{M}}[\boldsymbol{n}] = \boldsymbol{V}_{\boldsymbol{M}}[\boldsymbol{n}] - \boldsymbol{\delta} \cdot \boldsymbol{Z}^{\boldsymbol{T}}[\boldsymbol{n}], \qquad (\boldsymbol{A.27})$$

$$\overline{\boldsymbol{V}}[\boldsymbol{n}] = \boldsymbol{V}[\boldsymbol{n}] - \boldsymbol{\delta} \cdot \boldsymbol{z_{N+1}} \cdot \boldsymbol{Z}[\boldsymbol{n}], \qquad (\boldsymbol{A.28})$$

$$\bar{\mathbf{v}} = \boldsymbol{\alpha} \cdot \mathbf{v} - \boldsymbol{\delta} \cdot \boldsymbol{z}_{\boldsymbol{N+1}}^{\mathbf{2}} + (\mathbf{1} - \boldsymbol{\alpha}) \cdot \boldsymbol{V}^{\boldsymbol{T}}[\boldsymbol{n}] \cdot \boldsymbol{C}[\boldsymbol{n}] \cdot \boldsymbol{V}[\boldsymbol{n}], \qquad (\boldsymbol{A.29})$$

$$\overline{\boldsymbol{C}}[\boldsymbol{n}] = \boldsymbol{C}[\boldsymbol{n}] + \frac{\boldsymbol{C}^{\boldsymbol{T}}[\boldsymbol{n}] \cdot \boldsymbol{Z}[\boldsymbol{n}] \cdot \boldsymbol{Z}^{\boldsymbol{T}}[\boldsymbol{n}] \cdot \boldsymbol{C}[\boldsymbol{n}]}{(\mathbf{1} \backslash \boldsymbol{\delta}) - \boldsymbol{\xi}}, \qquad (\boldsymbol{A.30})$$

$$\overline{\boldsymbol{\theta}}[\boldsymbol{n}] = \boldsymbol{\theta}[\boldsymbol{n}]. \qquad (\boldsymbol{A.31})$$

for $\xi \to 0+$:

$$\overline{\boldsymbol{V}}_{\boldsymbol{M}}[\boldsymbol{n}] = \boldsymbol{V}_{\boldsymbol{M}}[\boldsymbol{n}], \qquad (\boldsymbol{A.32})$$

$$\overline{\boldsymbol{V}}[\boldsymbol{n}] = \boldsymbol{V}[\boldsymbol{n}], \qquad (\boldsymbol{A.33})$$

$$\bar{\mathbf{v}} = \boldsymbol{\alpha} \cdot \mathbf{v} + (\mathbf{1} - \boldsymbol{\alpha}) \cdot \boldsymbol{V}^{\boldsymbol{T}}[\boldsymbol{n}] \cdot \boldsymbol{C}[\boldsymbol{n}] \cdot \boldsymbol{V}[\boldsymbol{n}], \qquad (\boldsymbol{A.34})$$

$$\overline{\boldsymbol{C}}[\boldsymbol{n}] = \boldsymbol{C}[\boldsymbol{n}], \qquad (\boldsymbol{A.35})$$

$$\overline{\boldsymbol{\theta}}[\boldsymbol{n}] = \boldsymbol{\theta}[\boldsymbol{n}], \qquad (\boldsymbol{A.36})$$

where

$$\boldsymbol{\xi} = \boldsymbol{Z^T}[\boldsymbol{n}] \cdot \boldsymbol{C}[\boldsymbol{n}] \cdot \boldsymbol{Z}[\boldsymbol{n}], \qquad (\boldsymbol{A.37})$$

$$\boldsymbol{\delta} = \frac{\boldsymbol{1-\alpha}}{\boldsymbol{\xi}}, \qquad (\boldsymbol{A.38})$$

$$\boldsymbol{\bar{z}_{N+1}} = \boldsymbol{\theta^T}[\boldsymbol{n}] \cdot \boldsymbol{Z}[\boldsymbol{n}], \qquad (\boldsymbol{A.39})$$

$\boldsymbol{\bar{z}_{N+1}}$ is a posterior prediction of $\boldsymbol{z_{N+1}}$

Note that in case of the directional forgetting, the nesting of the orders is not valid anymore. Therefore, to estimate parameters for different order models, it is necessary to apply separate algorithms for each model.

APPENDIX 2

QRD RLS Lattice Algorithm and Hypothesis Testing

As it is mentioned in Chapter 1, the RLS Lattice algorithm can be derived from the QRD algorithm. The derivation of the algorithm is based on the below equations multiplied with a data vector [40-41, 47-48, 50, 81].

For the purposes of more readability, the notation ***["size"]*** of vectors and matrices is omitted in this chapter. Instead, notation ***("order/step of computation")*** is applied to note if the values are from the present or from the previous step of computation are used.

$$\widehat{\boldsymbol{\theta}}_{\boldsymbol{f}}(\boldsymbol{n}) = \begin{bmatrix} \widehat{\boldsymbol{\theta}}_{\boldsymbol{f}}(\boldsymbol{n}-\mathbf{1}) \\ \mathbf{0} \end{bmatrix} + \begin{bmatrix} -\boldsymbol{\theta}_{\boldsymbol{bz}}(\boldsymbol{n}) \\ \mathbf{1} \end{bmatrix} \cdot \boldsymbol{\Lambda}_{\boldsymbol{z}}^{-\mathbf{1}}(\boldsymbol{n}) \cdot \boldsymbol{K}(\boldsymbol{n}), \qquad (\boldsymbol{A}.\mathbf{40})$$

$$\boldsymbol{\theta}_{\boldsymbol{b}}(\boldsymbol{n}) = \begin{bmatrix} \mathbf{0} \\ \boldsymbol{\theta}_{\boldsymbol{bz}}(\boldsymbol{n}) \end{bmatrix} + \begin{bmatrix} \mathbf{1} \\ -\widehat{\boldsymbol{\theta}}_{\boldsymbol{f}}(\boldsymbol{n}-\mathbf{1}) \end{bmatrix} \cdot \boldsymbol{\Lambda}_{\boldsymbol{z}}^{-\mathbf{1}}(\boldsymbol{n}) \cdot \boldsymbol{K}(\boldsymbol{n}), \qquad (\boldsymbol{A}.\mathbf{41})$$

$$\overline{\boldsymbol{\theta}}_{\boldsymbol{bz}}(\boldsymbol{n}) = \boldsymbol{\theta}_{\boldsymbol{b}}(\boldsymbol{n}-\mathbf{1}),\ \boldsymbol{n=2,\ldots N.} \qquad (\boldsymbol{A}.\mathbf{42})$$

Note that $\widehat{\boldsymbol{\theta}}_{\boldsymbol{f}}$ is a vector of autoregression coefficients in a forward direction, $\boldsymbol{\theta}_{\boldsymbol{b}}$ is a vector of regression coefficients in a backward direction calculated from the decomposition of matrix $\boldsymbol{V}_{\boldsymbol{M}}[\boldsymbol{N}+\mathbf{1}]$, $\boldsymbol{\theta}_{\boldsymbol{bz}}$ is a vector of regression coefficients in a backward direction calculated from the decomposition of matrix $\boldsymbol{V}_{\boldsymbol{M}}[\boldsymbol{N}]$, $\boldsymbol{\Lambda}_{\boldsymbol{z}}$ are diagonal elements of matrix $\boldsymbol{D}$ in the matrix decomposition $\boldsymbol{V}_{\boldsymbol{M}}^{-\mathbf{1}}[\boldsymbol{N}] = \boldsymbol{U}[\boldsymbol{N}] \cdot \boldsymbol{D}[\boldsymbol{N}] \cdot \boldsymbol{U}^{\boldsymbol{T}}[\boldsymbol{N}]$, where $\boldsymbol{U}$ is an upper triangular matrix with units on the main diagonal, $\boldsymbol{D}$ is a diagonal matrix with positive elements on the main diagonal. $\boldsymbol{\Lambda}$ are scalars in $\boldsymbol{N}$ different decompositions of $\boldsymbol{V}_{\boldsymbol{M}}[\boldsymbol{N}+\mathbf{1}] = \boldsymbol{L}[\boldsymbol{N}+\mathbf{1}] \cdot \boldsymbol{D}[\boldsymbol{N}+\mathbf{1}] \cdot \boldsymbol{L}^{\boldsymbol{T}}[\boldsymbol{N}+\mathbf{1}]$, where $\boldsymbol{L}$ is a low triangular matrix with units on the main diagonal. $\boldsymbol{K}$ stands for coefficients of response.

After this step, relations between prediction (resp. filtration) errors in forward and backward directions can be obtained. On the basis of these variables, the update of parameters $\boldsymbol{\Lambda}_{\boldsymbol{z}}$, $\boldsymbol{\Lambda}$, $\boldsymbol{K}$ is provided [40-41, 47-48, 50, 81].

The relations for the updates of the parameters $\boldsymbol{\Lambda_z}$, $\boldsymbol{\Lambda}$, $\boldsymbol{K}$ are presented below [40-41, 47-48, 50, 81]:

$$\overline{\boldsymbol{K}}(\boldsymbol{n}) = \boldsymbol{K}(\boldsymbol{n}) + \frac{\boldsymbol{h_z}(\boldsymbol{n}) \cdot \boldsymbol{e}(\boldsymbol{n-1})}{\boldsymbol{1 + \xi(n-1)}}, \qquad (\boldsymbol{A.43})$$

$$\overline{\boldsymbol{K}}(\boldsymbol{n}) = \boldsymbol{\alpha^2} \cdot \overline{\boldsymbol{K}}(\boldsymbol{n}), \qquad \boldsymbol{for\ n = 1, 2, \dots, N}, \qquad (\boldsymbol{A.44})$$

$$\overline{\boldsymbol{\Lambda}}_{\boldsymbol{z}}(\boldsymbol{n}) = \boldsymbol{\Lambda_z}(\boldsymbol{n}) + \frac{\boldsymbol{h_z^2}(\boldsymbol{n})}{\boldsymbol{1 + \xi(n-1)}}, \qquad (\boldsymbol{A.45})$$

$$\overline{\boldsymbol{\Lambda}}_{\boldsymbol{z}}(\boldsymbol{n}) = \boldsymbol{\alpha^2} \cdot \overline{\boldsymbol{\Lambda}}_{\boldsymbol{z}}(\boldsymbol{n}), \qquad \boldsymbol{for\ n = 1, 2, \dots, N}, \qquad (\boldsymbol{A.46})$$

$$\overline{\boldsymbol{\Lambda}}(\boldsymbol{n}) = \boldsymbol{\Lambda}(\boldsymbol{n}) + \frac{\boldsymbol{e^2}(\boldsymbol{n})}{\boldsymbol{1 + \xi(n)}}, \qquad (\boldsymbol{A.47})$$

$$\overline{\boldsymbol{\Lambda}} = \boldsymbol{\alpha^2} \cdot \overline{\boldsymbol{\Lambda}}, \qquad \boldsymbol{for\ n = 0, 1, \dots, N}, \qquad (\boldsymbol{A.48})$$

$\boldsymbol{h_z}$ are prediction errors computed in a backward direction from the decomposition of matrix $\boldsymbol{V_M[N]}$, $\boldsymbol{e}$ are prediction errors in a forward direction.

Let us remind also that for the initialization step the following equations are valid [40-41, 47-48, 50, 81]:

$$\boldsymbol{h_1 = y_{t-1}; e_0 = y_t;\ \ \xi_0 = 0}. \qquad (\boldsymbol{A.49})$$

To update parameters (A.43)-(A.48), it is necessary to know the forward prediction errors $\boldsymbol{e}$ and backward prediction errors $\boldsymbol{h_z}$. They are defined as follows [40-41, 47-48, 50, 81]:

$$\boldsymbol{e}(\boldsymbol{n}) = \boldsymbol{y_t} - \widehat{\boldsymbol{\theta}}_{\boldsymbol{f}}(\boldsymbol{n}) \cdot \boldsymbol{Z}(\boldsymbol{n}), \qquad (\boldsymbol{A.50})$$

$$\boldsymbol{h_z}(\boldsymbol{n}) = \boldsymbol{y_{t-n}} - \boldsymbol{\theta_{bz}^T}(\boldsymbol{n}) \cdot \boldsymbol{Z}(\boldsymbol{n-1}). \qquad (\boldsymbol{A.51})$$

The interaction between the forward and backward prediction errors of order ***n-1*** and ***n*** without using vectors $\widehat{\boldsymbol{\theta}}_{\boldsymbol{f}}$, $\boldsymbol{\theta_{bz}}$ is given by the equations [40-41, 47-48, 50, 81]:

$$\boldsymbol{e = y_t - \widehat{\theta}_f(n) \cdot Z(n) = e(n-1) - K(n) \cdot \Lambda_z^{-1}(n) \cdot h_z(n)\ for\ n=1,2,\dots,N} \quad (\boldsymbol{A.52})$$

where $\boldsymbol{e_0 = y_t}$.

$$\boldsymbol{h_z}(\boldsymbol{n}) = \boldsymbol{y_{t-n}} - \boldsymbol{\theta_{bz}^T}(\boldsymbol{n}) \cdot \boldsymbol{Z}(\boldsymbol{n-1}) = \boldsymbol{h}(\boldsymbol{n-1}), \qquad \boldsymbol{for\ n=1,2,\dots,N,} \quad (\boldsymbol{A.53})$$

where $\boldsymbol{h}$ are backward prediction errors computed from the decomposition of matrix $\boldsymbol{V_M[n+1]}$ and $\boldsymbol{h_0 = y_{t-1}}$.

$$\boldsymbol{h(n) = h_z(n) - K(n) \cdot \Lambda^{-1}(n-1) \cdot e(n-1)}, \qquad \boldsymbol{for\ n=1,2,\ldots,N,} \quad (\boldsymbol{A.54})$$

where $\boldsymbol{e_0 = y_t}$.

Thus, the equations (A.52)-(A.54) define the prediction errors $\boldsymbol{e}$, $\boldsymbol{h_z}$, which are necessary for updating parameters $\boldsymbol{\Lambda_z}$, $\boldsymbol{\Lambda}$, $\boldsymbol{K}$ according to the equations (A.43)-(A.48). It should be also noted that in this case there is no need to update matrix $\boldsymbol{V_M[N]}$ [40-41, 47-48, 50, 81].

To summarize, the algorithm in the form suitable for programming is presented below.

Note that $\boldsymbol{e_u}$ stands for a prediction error calculated from the inputs $\boldsymbol{u_t}$, while $\boldsymbol{e_y}$ represents a prediction error calculated using the outputs $\boldsymbol{y_t}$ (see Fig. A.2).

Start of the algorithm in time $\boldsymbol{t_0 = N+1}$

$\boldsymbol{\delta_0 > 0}, \qquad \boldsymbol{\delta_0 \to 0}$

$\boldsymbol{\alpha^2 \epsilon (0,1>}$

$\Lambda(\boldsymbol{n}) = \boldsymbol{\delta_0}, \qquad \boldsymbol{for\ \ n = 0,1,\ldots,N}$

$\Lambda_{\boldsymbol{z}}(\boldsymbol{n}) = \boldsymbol{\delta_0}\ , \quad \boldsymbol{for\ \ n = 1,2,\ldots,N}$

$\boldsymbol{K(n) = 0}, \qquad \boldsymbol{for\ \ n = 1,2,\ldots,N}$

Computation in time $\boldsymbol{t \geq t_0}$

$\boldsymbol{\bar{\Lambda}_0 = \alpha^2 \cdot (\Lambda_0 + y_t^2)}$

$\boldsymbol{\bar{\nu} = \alpha^2 \cdot (\nu + 1)}$

$\boldsymbol{\xi_0 = 0}$

$\boldsymbol{e_{u_{(0)}} = h_0 = u_t}$

$\boldsymbol{e_{y_{(0)}} = y_t}$

Cycle for ***n=1, 2, …, N***

$$\boldsymbol{h_z(n) = h \quad (n-1)}$$

$$\boldsymbol{\overline{K}_u(n) = \alpha^2 \cdot (K_u(n) + \frac{h_z(n) \cdot e_u(n-1)}{1 + \xi(n-1)})}$$

$$\boldsymbol{\overline{\Lambda}_z(n) = \alpha^2 \cdot (\Lambda_z(n) + \frac{h_z^2(n)}{1 + \xi(n-1)})}$$

$$\boldsymbol{h(n) = h_z(n) - K(n) \cdot \Lambda^{-1}(n-1) \cdot e_u(n-1)}$$

$$\boldsymbol{e_u(n) = e_u(n-1) - K_u(n) \cdot \Lambda_z^{-1}(n) \cdot h_z(n)}$$

$$\boldsymbol{\xi(n) = \xi(n-1) + \Lambda_z^{-1}(n) \cdot h_z^2(n)}$$

$$\boldsymbol{\overline{\Lambda}(n) = \alpha^2 \cdot (\Lambda(n) + \frac{e_u^2(n)}{1 + \xi(n)})}$$

$$\boldsymbol{\overline{K}_y(n) = \alpha^2 \cdot (K_y(n) + \frac{h_z(n) \cdot e_y(n-1)}{1 + \xi(n-1)})}$$

$$\boldsymbol{e_y(n) = e_y(n-1) - K_y(n) \cdot \Lambda^{-1}(n) \cdot h(n)}$$

Algorithm 1: Lattice with prediction errors [40]

Instead of $\boldsymbol{\xi(n)}$ it is possible to make updating in the following form [40-41, 47-48, 50, 81]:

$$\boldsymbol{\sigma^2(n) = 1 + \xi(n).} \qquad (\boldsymbol{A.55})$$

For ***n=0***

$$\boldsymbol{\sigma_0^2 = 1,} \qquad (\boldsymbol{A.56})$$

$$\boldsymbol{\sigma^2(n) = \sigma^2(n-1) + h_z^2(n) \cdot \Lambda_z^{-1}(n).} \qquad (\boldsymbol{A.57})$$

It is worth mentioning that the algorithm computes all necessary parameters for calculating the probability density function $\boldsymbol{p(y_t|D(t-1); n_t)}$ [40-41, 47-48, 50, 81].

The structure of the QRD RLS Lattice algorithm can be shown in the following graph [40]:

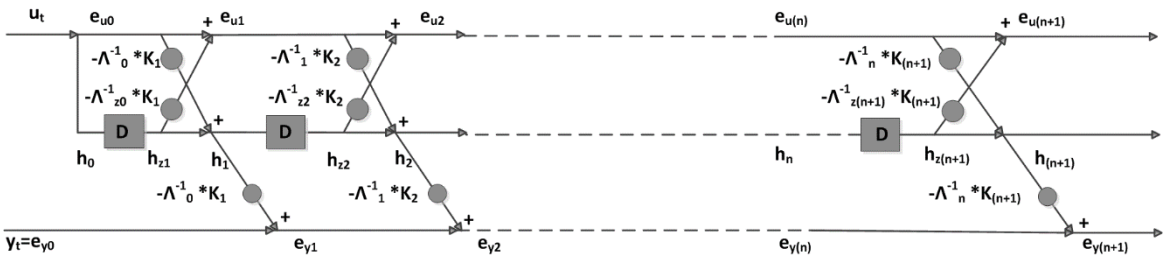

Figure A.2: QRD RLS Lattice algorithm [40]

 means multiplication with a constant,

 stands for "memory cell", with the help of which the following equation is performed: $\boldsymbol{h_z(n) = h\ \ (n-1)}$, where $\boldsymbol{h\ \ (n-1)}$ is from the previous time step.

Algorithm 1 can be changed to obtain the filtration errors instead of the prediction errors. This change allows deriving the normalized forms of the algorithms performing the computation in range ***(-1; 1)*** [40-41, 47-48, 50, 81], which decreases the computational complexity and ensures the numerical stability of the algorithm.

The equations, which define relations between the filtration and prediction errors, are as follows [40-41, 47-48, 50, 81]:

$$\bar{\boldsymbol{e}}(\boldsymbol{n}) = \boldsymbol{e}(\boldsymbol{n}) \cdot \left(\mathbf{1} - \bar{\boldsymbol{\xi}}(\boldsymbol{n})\right), \qquad (\boldsymbol{A.58})$$

$$\bar{\boldsymbol{h}}_{\boldsymbol{z}}(\boldsymbol{n}) = \boldsymbol{h_z}(\boldsymbol{n}) \cdot \left(\mathbf{1} - \bar{\boldsymbol{\xi}}(\boldsymbol{n-1})\right). \qquad (\boldsymbol{A.59})$$

It is also valid:

$$\mathbf{1} + \boldsymbol{\xi}(\boldsymbol{n}) = \frac{\mathbf{1}}{\mathbf{1} - \bar{\boldsymbol{\xi}}(\boldsymbol{n})}\,. \qquad (\boldsymbol{A.60})$$

The detailed description of the derivation of the equations can be found in [40-41, 47-48, 50, 81].

The algorithm can be written in the following form [40]:

Start of the algorithm in time $\boldsymbol{t_0 = N + 1}$

$\boldsymbol{\delta_0 > 0}, \qquad \boldsymbol{\delta_0 \to 0}$

$\boldsymbol{\alpha^2 \epsilon (0, 1 >}$

$\overline{\Lambda}\ \ (\boldsymbol{n}) = \boldsymbol{\delta_0}, \qquad \boldsymbol{for\ n = 0, 1, \dots, N}$

$\overline{\Lambda}_{\boldsymbol{z}}(\boldsymbol{n}) = \boldsymbol{\delta_0}\ , \qquad \boldsymbol{for\ n = 1, 2, \dots, N}$

$\overline{\boldsymbol{K}}\ \ (\boldsymbol{n}) = \boldsymbol{0}, \qquad \boldsymbol{for\ n = 1, 2, \dots, N}$

Computation in time $\boldsymbol{t \geq t_0}$

$\overline{\overline{\Lambda}}_{\boldsymbol{0}} = \boldsymbol{\alpha^2 \cdot \overline{\Lambda}_0 + y_t^2}$

$\overline{\overline{\boldsymbol{\nu}}} = \boldsymbol{\alpha^2 \cdot \bar{\nu} + 1}$

$\overline{\overline{\boldsymbol{\xi}}}_{\boldsymbol{0}} = \boldsymbol{0}$

$\overline{\overline{\boldsymbol{e}}}_{\boldsymbol{u0}} = \overline{\overline{\boldsymbol{h}}}_{\boldsymbol{0}} = \boldsymbol{u_t}$

$\overline{\overline{\boldsymbol{e}}}_{\boldsymbol{y0}} = \boldsymbol{y_t}$

Cycle for ***n=1, 2, …, N***

$\overline{\overline{\boldsymbol{h}}}_{\boldsymbol{z}}(\boldsymbol{n}) = \overline{\boldsymbol{h}}_{\boldsymbol{z}}(\boldsymbol{n-1})$

$$\overline{\overline{\boldsymbol{K}}}_{\boldsymbol{u}}(\boldsymbol{n}) = \boldsymbol{\alpha^2 \cdot \overline{K}_u(n)} + \frac{\overline{\overline{\boldsymbol{h}}}_{\boldsymbol{z}}(\boldsymbol{n}) \cdot \overline{\overline{\boldsymbol{e}}}_{\boldsymbol{u}}(\boldsymbol{n-1})}{\boldsymbol{1 - \overline{\overline{\xi}}(n-1)}}$$

$$\overline{\overline{\Lambda}}_{\boldsymbol{z}}(\boldsymbol{n}) = \boldsymbol{\alpha^2 \cdot \overline{\Lambda}_z(n)} + \frac{\overline{\overline{\boldsymbol{h}}}_{\boldsymbol{z}}^{\boldsymbol{2}}(\boldsymbol{n})}{\boldsymbol{1 - \overline{\overline{\xi}}(n-1)}})$$

$$\bar{\bar{h}}(n) = \bar{\bar{h}}_z(n) - \bar{\bar{K}}_u(n) \cdot \bar{\bar{\Lambda}}^{-1}(n-1) \cdot \bar{\bar{e}}_u(n-1)$$

$$\bar{\bar{e}}_u(n) = \bar{\bar{e}}_u(n-1) - \bar{\bar{K}}_u(n) \cdot \bar{\bar{\Lambda}}_z^{-1}(n) \cdot \bar{\bar{h}}_z(n)$$

$$\bar{\bar{\xi}}(n) = \bar{\bar{\xi}}(n-1) + \bar{\bar{\Lambda}}_z^{-1}(n) \cdot \bar{\bar{h}}_z^2(n)$$

$$\bar{\bar{\Lambda}}(N+1|n) = \alpha^2 \cdot \bar{\Lambda}(N+1|n) + \frac{\bar{\bar{e}}_u^2(N+1|n)}{1 - \bar{\bar{\xi}}(n)}$$

$$\bar{\bar{K}}_y(n) = \alpha^2 \cdot \bar{K}_y(n) + \frac{\bar{\bar{h}}_z(n) \cdot \bar{\bar{e}}_y(n-1)}{1 - \bar{\bar{\xi}}(n-1)}$$

$$\bar{\bar{e}}_y(n) = \bar{\bar{e}}_y(n-1) - \bar{\bar{K}}_y(n) \cdot \bar{\bar{\Lambda}}^{-1}(n) \cdot \bar{\bar{h}}(n)$$

Algorithm 2: Lattice algorithm with filtration error [40]

Again instead of $\bar{\bar{\xi}}(n)$**,** it is possible to make updating:

$$\bar{\sigma}^2(n) = 1 - \bar{\xi}(n). \qquad (A.61)$$

For ***n=0***:

$$\bar{\sigma}_0^2 = 1, \qquad (A.62)$$

$$\bar{\sigma}^2(n) = \bar{\sigma}^2(n-1) - \bar{h}_z^2(n) \cdot \bar{\Lambda}_z^{-1}(n). \qquad (A.63)$$

The main difference between ***Algorithm 1*** and ***Algorithm 2*** is in the fact that ***Algorithm 2*** works with parameters $\bar{\Lambda}_z$, $\bar{\Lambda}$, $\bar{K}$, which are already updated by the filtration errors $\bar{e}$ and $\bar{h}_z$ before using them in the equations. This property allows normalizing variables in the algorithm. For example, due to the equations for updating $\bar{\Lambda}$, it is obtained [40-41, 47-48, 50, 81]:

$$\bar{\Lambda}(n) = \alpha^2 \cdot \bar{\Lambda}(n) + \frac{\bar{e}^2(n)}{1 - \bar{\xi}(n)}, \qquad (A.64)$$

$$0 < 1 - \bar{\xi}(n) \leq 1, \qquad (A.65)$$

$$-1 < \bar{e}(n) \cdot \bar{\Lambda}^{-\frac{1}{2}}(n) < 1. \qquad (A.66)$$

As it was mentioned in Chapter 1, the QRD RLS Lattice algorithm is suitable for incorporation of the hypothesis estimation.

Let us remind the equation for estimation of the hypothesis probability (1.14):

$$\boldsymbol{p(H_n|D(t)) = \frac{p_n(y_t|u_t, D(t-1))}{p(y_t|u_t, D(t-1))} p(H_n|D(t-1)),} \qquad \boldsymbol{for\ n = 1, \dots, N.}$$

There are two stages of probability estimation. The first stage, which is presented by the numerator in (1.14), computes the order update. In the second stage, which is presented by the denominator of (1.14), the normalization of the updated order estimates is performed [84]. These stages can be incorporated into the QRD RLS Lattice algorithm.

It is obvious that to compute the probability estimates of the hypotheses, it is necessary to know $\boldsymbol{p(y_t|D(t-1), H_n)}$, which is calculated as follows (A.15):

$$\boldsymbol{p(y_t|D(t-1);\ H_n) = \pi^{-1/2} \cdot \frac{\Lambda^{((\vartheta-n)/2+1)}}{\overline{\Lambda}^{((\overline{\vartheta}-n)/2+1)}} \cdot \frac{|V[n]|^{1/2}}{|\overline{V}[n]|^{1/2}} \cdot \frac{\Gamma((\overline{\vartheta}-n)/2+1)}{\Gamma((\vartheta-n)/2+1)}.}$$

The equation for updating $\boldsymbol{\vartheta}$ can be written in the following form [84]:

$$\boldsymbol{\overline{\vartheta} = \alpha\vartheta + 1.} \qquad (\boldsymbol{A.67})$$

Let us remind also that the above equation is valid only for the regression model inherent to the hypothesis $\boldsymbol{H}$ as a conditional probability density function (1.15) [84]:

$$\boldsymbol{p(y_t|D(t-1), \Theta, H_n) = \frac{\omega_t}{\sqrt{2\pi}} e^{-(\frac{\omega_t}{2})(y_t - \theta_t^T Z[n])^2}.}$$

The part of the QRD RLS Lattice algorithm, which shows computation of incorporated hypothesis estimation, is given in the following form:

Initialization:

$\boldsymbol{j}$ is a number of models, $\boldsymbol{n}$ is the order of a model.

$$\boldsymbol{p_{i,-1} = \frac{1}{(j+1)}} \quad \forall \boldsymbol{i}$$

For *i=1:j*

$$\boldsymbol{\vartheta_{lim} = \frac{1}{1-\alpha^2}}$$

$$\boldsymbol{e1_i = \frac{\alpha\vartheta_{lim} - n_i}{2} + 1}$$

$$e2_i = \frac{\vartheta_{lim} - n_i}{2} + 1$$

$$\tau_i = \bar{\vartheta} + 1 - n_i = \alpha\vartheta - n_i$$

// Approximation of division of gamma functions

$$g_{i1} = \left(\frac{\tau_i + 1}{\tau_i}\right)^{\tau_i/2}$$

$$g_{i2} = \left(\frac{\tau_i}{2}\right)^{1/2}$$

$$g_{i3} = e^{-1/2} \cdot g_{i1} \cdot g_{i2}$$

$$g_i = \pi^{-1/2} \cdot g_{i3}$$

// Computation of other components of the equation (2.40)

$$ft_i = \left(\frac{\Lambda_i}{\bar{\Lambda}_i}\right)^{\left(\frac{\vartheta_i - n_i}{2} + 1\right)} \cdot \frac{1}{\bar{\Lambda}_i^{1/2}}$$

$$st_i = \frac{1}{\sigma_i^2}$$

$p_i(y_t|D(t-1), H_i)=ft_i*st_i*g_i$;

fi_hypo=1e-20*(1.0/j);

$p(H_i)=p_i(y_t)\cdot p(H_i)$;

$p(Hs)=\sum_{i=0}^{j} p(H_i)$

$p(H_i)=(p(H_i)+fi\_hypo)/(p(Hs)+fi\_hypo*j)$;

end

Algorithm 3: Hypothesis estimation

Let us discuss ***Algorithm 3*** in more details.

It is obvious from ***Algorithm 3*** that before the first iteration, it is necessary to define the initial hypothesis probability density function. It is chosen in the form of the uniform distribution:

$$\boldsymbol{p_{i,-1} = \frac{1}{(j+1)} \quad \forall i,} \qquad (\boldsymbol{A.68})$$

where $\boldsymbol{p_i}$ is the probability of model $\boldsymbol{i}$ of order $\boldsymbol{n}$ at time $\boldsymbol{t}$. Value ***t=-1*** represents the initialization step.

Initialization of other characteristics is given as follows [84]:

$$\boldsymbol{\vartheta_{lim} = \lim_{n\to\infty} \vartheta \quad = \frac{1}{1-\alpha}} \qquad (\boldsymbol{A.69})$$

$$\boldsymbol{e1 \quad = \frac{\alpha\vartheta_{lim} - n}{2} + 1} \qquad (\boldsymbol{A.70})$$

$$\boldsymbol{e2 \quad = \frac{\vartheta_{lim} - n}{2} + 1} \qquad (\boldsymbol{A.71})$$

$$\boldsymbol{g \quad = \pi^{-1/2} \frac{\Gamma(e2 \ )}{\Gamma(e1 \ )}} \qquad (\boldsymbol{A.72})$$

In ***Algorithm 3*** equation (A.15) is not used in its direct form due to the numerical problems.

The problematic part is the first division $\frac{\boldsymbol{\Lambda^{((\vartheta-n)/2+1)}}}{\boldsymbol{\bar{\Lambda}^{((\bar{\vartheta}-n)/2+1)}}}$ in the equation (A.15). It should be rewritten to avoid the potential numerical underflow or overflow in case of the implementation in the floating point with a limited data range. As far as (A.67), the equation can be re-arranged into the following form:

$$\boldsymbol{\frac{\Lambda^{\left(\frac{\vartheta-n}{2}+1\right)}}{\bar{\Lambda}^{\left(\frac{\bar{\vartheta}-n}{2}+1\right)}} = \left(\frac{\Lambda}{\bar{\Lambda}}\right)^{\left(\frac{\vartheta-n}{2}+1\right)} \cdot \frac{1}{\bar{\Lambda}^{\frac{1}{2}}}} \qquad (\boldsymbol{A.73})$$

The division of determinants in the second part of equation (A.15) can also cause an algorithm failure and should be reformulated. Using relations in [40-41, 47-48, 50, 81], it can be rewritten as follows:

$$\boldsymbol{\frac{|V[n]|^{1/2}}{|\bar{V}[n]|^{1/2}} = \prod_{n=1}^{N} \frac{\Lambda(n)}{\bar{\Lambda}(n)} = \frac{1}{1+\xi} = \frac{1}{\sigma^2} \qquad for\ n = 1, 2, \dots, N} \qquad (\boldsymbol{A.74})$$

Moreover, there is a need to perform the division of gamma functions in a better way. In [40-41, 47-48, 50, 81] for the approximation of the division of gamma functions, the Stirling equation is used:

$$\Gamma\left(\frac{\boldsymbol{x}}{\boldsymbol{2}}\right)=\sqrt{\boldsymbol{2\pi}}\boldsymbol{e}^{-\frac{\boldsymbol{x}}{\boldsymbol{2}}}\left(\frac{\boldsymbol{x}}{\boldsymbol{2}}\right)^{(\boldsymbol{x}-\boldsymbol{1})/\boldsymbol{2}}\quad \boldsymbol{x}>\boldsymbol{0} \tag{$\boldsymbol{A.75}$}$$

If the numerator of argument of gamma function in (A.15) is denoted as:

$$\boldsymbol{\tau_n=\vartheta+1-n=\alpha\vartheta-n} \tag{$\boldsymbol{A.76}$}$$

Then for the division of gamma function, it is obtained:

$$\frac{\Gamma\left(\frac{\boldsymbol{\tau_n+1}}{\boldsymbol{2}}\right)}{\Gamma\left(\frac{\boldsymbol{\tau_n}}{\boldsymbol{2}}\right)}=\overline{\boldsymbol{g}}(\boldsymbol{n})=\boldsymbol{e}^{-\frac{\boldsymbol{1}}{\boldsymbol{2}}}\left(\frac{\boldsymbol{\tau_n+1}}{\boldsymbol{\tau_n}}\right)^{\frac{\boldsymbol{\tau_n}}{\boldsymbol{2}}}\cdot\left(\frac{\boldsymbol{\tau_n}}{\boldsymbol{2}}\right)^{\frac{\boldsymbol{1}}{\boldsymbol{2}}} \tag{$\boldsymbol{A.77}$}$$

where $\boldsymbol{\tau_0=\overline{\vartheta}+1}$.

Summarizing all approximations, which have been made above, the equation (A.15) can be rewritten in the following way:

$$\boldsymbol{p(y_t|D(t-1);\ H_n)=\pi^{-\frac{1}{2}}\cdot\frac{\Lambda^{\left(\frac{\vartheta-n}{2}+1\right)}}{\overline{\Lambda}^{\left(\frac{\overline{\vartheta}-n}{2}+1\right)}}\cdot\frac{|V[n]|^{\frac{1}{2}}}{|\overline{V}[n]|^{\frac{1}{2}}}\cdot\frac{\Gamma\left(\frac{\overline{\vartheta}-n}{2}+1\right)}{\Gamma\left(\frac{\vartheta-n}{2}+1\right)}=}$$

$$\boldsymbol{=\pi^{-1/2}\cdot\left(\frac{\Lambda}{\overline{\Lambda}}\right)^{\left(\frac{\vartheta-n}{2}+1\right)}\cdot\frac{1}{\overline{\Lambda}^{\frac{1}{2}}}\cdot\frac{1}{\sigma^2}\cdot e^{-\frac{1}{2}}\left(\frac{\tau+1}{\tau}\right)^{\frac{\tau}{2}}\cdot\left(\frac{\tau}{2}\right)^{\frac{1}{2}}} \tag{$\boldsymbol{A.78}$}$$

Moreover, it should be noted that there are certain limits, within which the value of the forgetting factor $\boldsymbol{\alpha^2}$ can be chosen. These limits are conditioned by the maximum order of the model and can be written as follows:

$$\boldsymbol{1-\frac{1}{N+3}<\alpha^2<1} \tag{$\boldsymbol{A.79}$}$$

For detailed information about the derivation of the above formula as well as a definition of time, from when it is possible to start the recursive updating of the probability density function of the order of a regression model, please, refer to [40-41].

After the values of $\boldsymbol{p_{t-1}}$ have been updated, the normalization step follows to obtain updated probabilities $\boldsymbol{p_t}$. The probability density function is given by the equation for hypothesis testing and is extended with the forgetting of the hypotheses [84]:

$$\boldsymbol{p_i} = \frac{\boldsymbol{p_i^d} + \boldsymbol{\varphi}}{\sum_{\boldsymbol{i=0}}^{\boldsymbol{j}}(\boldsymbol{p_i^d} + \boldsymbol{\varphi})} \qquad (\boldsymbol{A.80})$$

where $\boldsymbol{p_i^d}$ is an updated, but not normalized probability of model $\boldsymbol{i}$ of order $\boldsymbol{n}$, for $\boldsymbol{i=1,\ldots,j}$. The value $\boldsymbol{\varphi}$ is the forgetting factor of the hypotheses.

The sum of updated probabilities $\boldsymbol{p_i^{sd}}$ can be calculated as [84]:

$$\boldsymbol{p_i^{sd}} = \sum_{\boldsymbol{i=0}}^{\boldsymbol{j}}(\boldsymbol{p_i^d} + \boldsymbol{\varphi}) = (\boldsymbol{j+1})\boldsymbol{\varphi} + \sum_{\boldsymbol{i=0}}^{\boldsymbol{j}} \boldsymbol{p_i^d} \qquad (\boldsymbol{A.81})$$

Then, the value of $\boldsymbol{p_i^d}$ is calculated using the update of $\boldsymbol{p_i^d}$ from its initial value $\boldsymbol{p_{-1}^d} = (\boldsymbol{j+1})\boldsymbol{\varphi}$.