Implementation of the RN Method on FPGA using Xilinx System Generator for Nonlinear System Regression

Intissar SAYEHI
University of Tunis Elmanar, Faculty of Mathematical, Physical and Natural Sciences of Tunis
Laboratory of Electronics and Microelectronics, (E. µ. E. L), FSM, Monastir, Tunisia

T. Saidani and B. Bouallegue
University of Monastir,
Faculty of Sciences of Monastir
Laboratory of Electronics and Microelectronics, (E. µ. E. L), Tunisia

Okba TOUALI
University of Monastir,
National Engineering School of Monastir, Tunisia
Laboratory LARATSI, ENIM, Monastir, Tunisia

Mohsen MACHHOUT
University of Monastir, Faculty of Sciences of Monastir
Laboratory of Electronics and Microelectronics, (E. µ. E. L), Tunisia

***Abstract*—In this paper, we propose a new approach aiming to ameliorate the performances of the regularization networks (RN) method and speed up its computation time. A considerable rapidity in totaling calculation time and high performance were accomplished through conveying difficult calculation charges to FPGA. Using Xilinx System Generator, a successful HW/SW Co-Design was constructed to accelerate the Gramian matrix computation. Experimental results involving two real data sets of Wiener-Hammerstein benchmark with process noise prove the efficiency of the approach. The implementation results demonstrate the efficiency of the heterogeneous architecture, presenting a speed-up factor of 40-50 orders of time, comparing to the CPU simulation.**

Keywords*—Machine learning; Reproducing Kernel Hilbert Spaces (RKHS); regularization networks; FPGA; HW/SW Co-simulation; systolic array architecture; PT326; Wiener-Hammerstein benchmark

I. INTRODUCTION

In the last decade, Kernel methods [1] like Support Vector Machine (SVM), Regularization Networks (RN) and Kernel Principle Component Analysis (KPCA) [2] have become typical to perform nonlinear systems identification. Comparing with the traditional method, such as Neural Networks [3], [4], Voltera series [5], and the Kernel methods present an attractive alternative. They are well founded in a rigid mathematical structure of Reproducing Kernel Hilbert Spaces (RKHS) [6], [7], it overcomes convex optimization problems. Furthermore, they are complete nonlinear regressors that necessitate simply reasonable computational complexity.

Kernel methods like Support Vector Machines (SVM) [8] proved a high efficiency in various fields because it reveals some disadvantages that have to be tackled appropriately in each appliance especially for big data sets. Recently, several learning algorithms as the regularization networks (RN) are inspired from the support vector machine and affected from the need of reaching algorithms simpler to implement by simplifying the quadratic programming QP problem in training SVMs, which can be hard to solve.

The RN is most promising theoretically and practically but suffers from the equality between the number of model parameters and observations.

In this paper, an efficient Regularization Network model was contributed for identifying nonlinear systems based on random observations. A successful FPGA HW/SW Co-Design for accelerating the Gramian matrix computation was developed. Moreover, to avoid the information redundancy in the training data set, an efficient statistical method was employed to extract the useful information describing the most frequently occurring observations.

An application to a known challenging nonlinear system proves the rapidity and the low-resource-consuming hardware of this model for modeling in RKHS space.

The paper is structured as: Firstly, the background of this work is presented and discussed the different categories of SVM implementations on FPGA board and its inefficiencies and weakness. Then, we briefly evoke some basic concepts from learning theory for identification of nonlinear systems in reproducing Kernel Hilbert Space (RKHS). After discussing RKHS proprieties and the representer theorem, the regularization networks is described as a machine learning. Then we present the designing tools exploited for the HW/SW Co-design. After describing the move from RN algorithm to RN model and the statistical Data Preprocessing method, we introduce the acceleration of Gramian matrix computation and we discuss the co-simulation performances. For better understanding, the basic principles of systolic array architecture and the serial multiplication were explained with simple examples. Finally, we validate the work on a challenging nonlinear system: the Wiener-Hammerstein benchmark with process noise. We deal with the main results

concerning time and error. Finally, we conclude with some comments and perspectives.

II. Background and Related Work

Kernel methods have become powerful tools for classification and regression tasks due to its capability to be trained from past examples and continually adapt to new situations. In term of performances and aptitude to generalization the Support Vector Machine (SVM) excels the other Kernel method. Unfortunately the high computational cost of the SVM running time is critically reliant on the training dataset size and the problem's dimension. Also the quadratic programming (QP) techniques are a severe and computationally expensive task. There were much software like Sequential Minimum Optimization (SMO) and SVMLIGHT [9] have been proposed to resolve these problems analytically but don't give an enormous amelioration for real-time embedded systems. Consequently, special hardware architectures are ordered to convene limitations as inadequate resources exploitation plus little power consumption. That's motivates researchers to implement this method on programmable device to accelerate the computation time especially in case of online training.

The embedded digital systems like microcontroller, Digital Signal Processors (DSPs) or Field Programmable Gate Arrays (FPGAs) permit attaining greater resource-performance relation, but necessitating a careful implementation design. The FPGAs are potent and greatly parallel processing and allows a great flexibility and efficiency for different applications. Lately they have showing considerable performance against the General Purpose Processors (GPPs) for a lot of purpose like machine learning algorithms [10], [11]. In addition, Graphics Processing Unit (GPU) presents a further proposal for elevated performance computing [12]. Comparing the FPGA and GPU implementations of diverse algorithms and applications was the subject of many studies [13], [14]. In the majority of times, FPGAs confirmed greater performance. Even though GPUs profit from lower cost and shorter development time prejudiced against to FPGAs, they are inferior to FPGAs in terms of power consumptions. Next, we reviewed existing and new practices in hardware implementations aiming efficient implementations of the SVM model on FPGA. It could be approximately classed in two major groups. The first one called FPGA hardware accelerator which implemented only one phase: training or validation phase. The second group enclosed the FPGA implementations of SVM for classification and regression.

A. FPGA: Hardware Accelerator

The training phase of the SVM algorithm has attracted a community of investigators to exploit hardware accelerators aiming a diminution in whole training time. J. Filho, *et al*. [15] proposed a dynamically reconfigurable SVM architecture that supports different sizes of training datasets. A modular architecture was designed through the SMO algorithm to obtain dynamic reconfiguration. The authors employed the hardware-friendly Kernel function proposed in [16] and so the Coordinate Rotation Digital Computer (CORDIC) algorithm for Kernel computations. The platform exploited was Xilinx Virtex-IV (XC4VLX25). The proposed reconfigurable architecture attained 22.38% area economy with good enough reconfiguration time punishment. To study the consequence of fixed-point data representation on accuracy and classification mistake, three diverse learning benchmarks were implemented and accomplished speeding up factors of more than 12.53 times quicker than the software implementation for the entirety training time.

L. Martinez, *et al*. [17] designed a heterogeneous architecture to accelerate SVM training phase. To reduce the dot-product computation time, these operations were affected by the hardware coprocessor of Xtreme DSP Virtex- IV whereas the hierarchy of SMO algorithm was implemented in GPP. This application was a classification of the ADULT dataset by the linear Kernel function. The expected coprocessor design reached an acceleration of 178.7x comparing software results. In another method, the SVM was trained offline on software and then the trained data were imported for exploitation for online classification on hardware (FPGA board). There was a variety of techniques using different implementations methods. The authors in [18] were proposed an embedded hardware SVM implementation on FPGA board: Xilinx Virtex-5, Spartan-3E. Thanks to the hardware friendly Kernel function [16], the hardware design was made easier and simpler targeting satellite onboard applications. In the same way, the multiplication process was substituted by simple shift operations that verified lower resources exploitation of 167 slices. This hardware design proved its efficiency in Satellite onboard application based on NASA database.

B. FPGA platform for both SVM classification and regression

It was an intelligent idea to use the same platform for different task: classification and regression.

The work of authors in [19] presented an excellent design for an elevated performance and low resource consumption for support vector classification and regression. The proposed architecture has been considered as general use for embedded applications, where the number of support vectors and the resolution of the parameters can be arranged. In addition, there is not a limit to the dimension of the input vectors and the number of support vectors but the size of the FPGA. The performance of this design was tested for a multi classification problem on a basic COIL database and for regression problem on sinus cardinal function. In both cases, the average error rate for the hardware is between 0% and 0.02 %, which means that the SVM gives better results when using the hardware then MATLAB.

An additional hardware architecture for SVM algorithm was offered for classification and regression problems [20] and established on the hardware friendly Kernel [16]. A tree structure founded on common Sum of Absolute Differences (SAD) unit was used for diminishing clock cycles. Beginning simulation study was executed on the accuracy of input parameters by selecting fixed-point arithmetic, caring the same classification accuracy level with no failure.

The designers were aiming a diminution in hardware complication and power consumption through executing SVM on FPGA with different ways instead of conventional

algorithm. They presented a different approach to surmount this difficulty but the SVM still a complex method especially when solving the quadratic programming problem which is computationally expensive mission. In this work we suggest to implement a Kernel method inspired from SVM which is the regularization networks (RN). It is simpler and easier to implement and gives similar performances.

In next paragraph, we present the theoretical basis of this method and its advantages.

III. Modeling Nonlinear System in Reproducing Kernel Hilbert Space (RKHS)

A. Overview of Statistical Learning Theory (SLT)

The principle of the Statistical Learning Theory [21] is to find such function f modeling a system from a set of observations

$O = \{(x_i , y_i)\}$, i =1..N composed of inputs x_i and outputs y_i . This function has to reproduce the comportment of the system by minimizing the functional risk.

$$R(f) = \int_{X,Y} V(y, f(x))P(x, y)dxdy \quad (1)$$

The term V(y,f(x)) is named cost function. It determines the variation among system output yi and the estimated output f(x). The couple (X, Y) is composed of a random vectors and (xi, y i) are independents samples. The risk R(f) cannot be expected caused by ignoring P(x, y). To resolve that difficulty we have to diminish the following term:

$$R_{emp}(f) = \frac{1}{N}\sum_{i=1}^{N} V(y_i, f(x_i)) \quad (2)$$

However the frank minimization of $R_{emp}(f)$ in the functions space H don't provide better estimation of R(f) minimization and may leads to over fitting. As a solution, Vapnik advanced the theory of structural risk minimization (SRM). It punishes the empirical risk through a function estimating the complexity of reserved model. This conducts to minimizing the restriction definite by this equation:

$$\min_{f \in H} D(f) = \frac{1}{N}\sum_{i=1}^{N} V(y_i, f(x_i)) + \lambda \quad (3)$$

Where the first term measure how well the function (f) fits the given data and the second term is the squared norm of (f) in the RKHS space H, which controls the complexity (smoothness) of the solution. The parameter λ is the regularization parameter that balances the tradeoff between the two terms.

The more significant is the solution regularity and not the value of λ. while it is not obvious to minimize the restraint (3) on any random function space H, whatsoever is it with finite or infinite dimension. Consequently, to conquer this trouble, the space H will be regarded as a RKHS.

B. Reproducing Kernel Hilbert Space (RKHS) and the representer theorem

We assume that X a random variable is estimated in the space $E \subset \mathbb{R}^d$ and we expect the existence of a function K named Kernel function $K : E^2 \to \mathbb{R}$ which is symmetric and positive definite. Accordingly, there is [1] a function $\phi : E \to H$ that:

$$K(x, x')) = \langle \phi(x), \phi(x') \rangle_H \quad (4)$$

H is the Reproducing Kernel Hilbert Space (RKHS) [7] of Kernel K. Such space acquired distinguishing properties:

$$\forall x \in E \text{ and } f \in H \quad \langle K(x,.), f \rangle_H = f(x) \quad (5)$$

- *Thanks to representer theorem [22] the resolution of the optimization difficulty offered by (3) in this space is specified by:*

$$f_{opt} = \sum_{i=1}^{N} a_i K(x_i, .) \quad (6)$$

There are many types of Kernel functions which can be considered as:

1) Linear Kernel

$$K(x, x')) = x \times x' \quad (7)$$

2) Polynomial Kernel

$$K(x, x')) = (1 + \langle x, x' \rangle)^{\eta} \quad (8)$$

Where, $\eta \in \mathbb{R}^*$ and $\langle x, x' \rangle$ is an Euclidian scalar product.

3) Radial Basis Function (RBF) Kernel

$$K(x, x')) = e^{\frac{\|x - x'\|^2}{2\sigma^2}} \quad (9)$$

Where, σ is a real positive parameter.

4) Sigmoid Kernel

$$k(x, y) = \tanh(\alpha . x^T . y + c) \quad (10)$$

The slope alpha and the intercept constant c are two adjustable parameters in the sigmoid Kernel.

C. Learning Machine: Regularization networks (RN)

Machine is one of the most recent research areas of data mining. The algorithm exploited to approximate the parameters a_i in (5) is entitled learning machine like regularization network (RN) [23]. The exploited algorithm to calculate approximately the parameters a_i is the regularization network (RN). Compared to other Kernel method that optimize the parameters iteratively like support vector regression (SVR) the RN takes less time and offer excellent performances in term of generalization ability. As exposed, the optimization problem (3) can be resolved thanks to the Kernel trick:

$$\|f\|_H^2 = \sum_{i=1}^{N}\sum_{j=1}^{N} a_i a_j K(x_i, x_j) \quad (11)$$

The cost function to be minimized by the RN is:

$$V(y_i, f(x_i)) = (y_i - f(x_i))^2 \quad (12)$$

The optimal function given by (5), where the sequence $\{a_i\}$ is:

$$a_i = \sum_{i=1}^{N} (G + \lambda)_{ij}^{-1} y_j \quad (13)$$

Where, $G \in \mathbb{R}^{N \times N}$ is the Gramian matrix associated to the Kernel function K , $G_{ij} = (K(x_i, x_j)), i, j = 1, ..., N$ and Y is the output vector. On the other hand, in matrix form:

$$A = (G + \lambda NI)^{-1} Y, A = (a_1, ..., a_N)^T, Y = (y_1, ..., y_N)^T \quad (14)$$

To simplify the understood of this method, the next section describes the move from RN algorithm to RN model and presents the designing tools and with explanation of the different components of the HW/SW Co-design.

IV. Proposed HW/SW Co-Simulation Method

A. Designing Tools

The used tools are MATLAB R2013a with Simulink from MathWorks [24], System Generator 14.7 for DSP and ISE 14.7 from Xilinx. The System Generator runs within the Simulink as simulation environment, which is part of MATLAB mathematical package. Simulink is an interactive software for modeling, simulating, and analyzing dynamical linear and nonlinear systems in continuous time, sampled time, or a hybrid of the two Systems. Thanks to the incorporation of MATLAB and Simulink, we can simulate, analyze, and revise our models in either environment at any point.

Xilinx System Generator [25] provides a set of Simulink blocks special for several hardware operations that could be implemented on various Xilinx FPGAs. These blocks can be used to simulate the functionality of the hardware system using Simulink environment. One of the advantages of Xilinx System Generator is the capability of generating HDL code directly from your designs.

Xilinx System Generator employs fixed-point format to describe all numerical values in the system and it provides some blocks to transform data provided from the software side of the simulation environment (Simulink) and the hardware side (System Generator blocks). This is an essential concept to understand throughout the design process using Xilinx System Generator. In the next section, we explain the steps of RN algorithm and how we transform it to a model that facilitates the hardware implementation.

B. Regularization Networks: from algorithm to model

An algorithm is a predetermined set of rules for conducting computational steps that produce a computational effect. Whereas, a model is a framework for expressing algorithms build from mathematical equations that is suitable for a hardware implementation. The development of a model in such way affords a simply understood system analysis for the models customers. Fig. 1 presents the conceptual model of the RN.

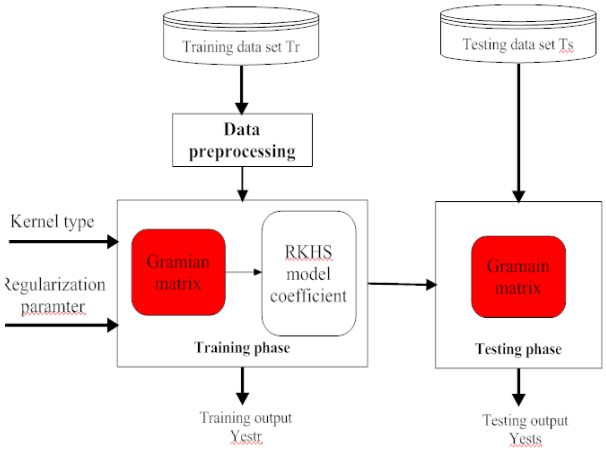

Fig. 1. Conceptual model of regularization networks.

In our case, the move from RN algorithm to RN model provides efficient conveyance of system details and allows easy extracting of system specifications. The modeling steps pass from necessary improvement through design, implementation, and testing. We obtain an executable model that can be continually developed. After model development, simulation shows whether the model works correctly.

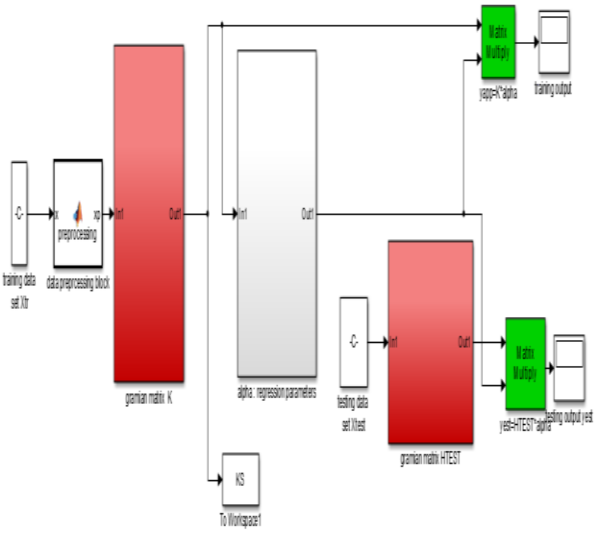

Fig. 2. System generator project for regularization networks co-simulation.

The main goal of the RN modeling is the assigning of complex computation tasks to the hardware. In fact, while software and hardware implementation supplies are integrated with the model, for instance fixed-point and timing behavior, the code could be generated for embedded exploitation and

generate test benches for system verification, saving time and evading manually coded errors. Fig. 2 presents the Simulink Co-simulation model. It contains Simulink and Xilinx system generator components. The red blocks in the figure are executed by the FPGA and the others executed through Simulink.

The adopted approach allows connecting designs straightforwardly to requirements and combining testing with design to constantly identify and correct errors. The reconfigurability of the FPGA and the flexibility of the different blocks of SIMULINK allow the identification of infinity of systems. In the next sections, we will specify the blocks function.

C. Data Preprocessing

The problem of RN method is that the total of parameters is identical to the number of observations. Therefore, to reduce the number of parameters, the number of observations must be reduced and must be a measure of the data spread. Generally, in supervised learning the data are generated by experimental measurement. Whereas, experimentation often makes multiple measurements of the same thing and it is subject to error.

Also, the sampling period of experimental process is small and so the variance of the data sets is small. In this case, the Statistical and mathematical tools [26] as the mean, the median and mode for data quantitative analysis can describe the central tendency of the data set and extract useful information without redundancy. The suitable tool for this work is the mode because it is a statistical term that refers to the most frequently occurring number found in data set of observations. It is found by identifying the most occurring rate in the data set most often representing the data. If the range is big, the central tendency is not as representative of the data as it would be if the range was mall.

Therefore, we can divide the data set in categories as shown in Fig. 3. The mode requires only those values of the data points which can be put into categories. The new chosen data set is composed of the mode of each category.

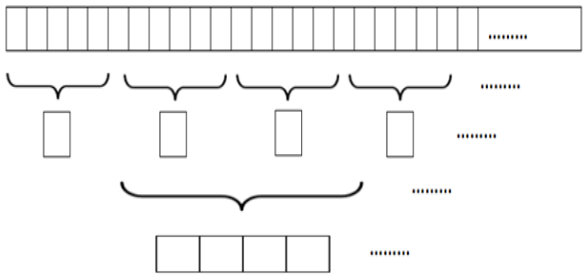

Fig. 3. Scheme of data reduction.

The frequency of each category is the number of data points whose values are in that category, and the mode is the category with the highest frequency. It is possible that more than one category share the highest frequency in which case the data is multimodal. In the training phase, we create a MATLAB function block called Data Preprocessing that receives a large data set and after statistical treatment; produces the data set for Gramian matrix computation. To demonstrate the efficiency of this approach, we executed different testing scenarios to compare between the performances of the real data and the treated data. The PT 326 [27] works as the hair dryer. It heats the air from the atmosphere from 30ºC to 60ºC. In the simulation, we used a single database input/single output (SISO) in the time domain of PT326 process. In previous work [28] a comparative study was established between two Kernel methods; SVM and RN. The consequences demonstrate the competence of the learning algorithms and confirm the excellence of the SVR method in obtaining minimal prediction error and advantage of the RN to gain the calculation time. Approximately, the performances of SVM can be reached by the RN when the data sets are large and with exploitation of a hardware platform for acceleration. Therefore, the choice of implementing the RN method as based on this study. Especially that RN is simpler to implement. In next work, we call the RN method using the reduction method RNR (Reduced). To compare RN to RNR, we employ the same dataset; 100observations for the training phase and 200 new observations for the validation phase. For the RNR, the 100 observations will be reduced to only 10. After obtaining the RKHS model coefficients, the validation data set was chosen randomly and without any treatment to augment the aptitude of generalization of the RKHS model. The Kernel used is polynomial. The optimal parameter λ of the machine learning was obtained by a cross validation technique and it is equal to 0.0001. To evaluate numerically the model performances, we exploit the Normalized Mean Squared Error (NMSE):

$$NMSE = \frac{\sum_{i=1}^{N}(y(i)-\tilde{y}(i))^2}{\sum_{i=1}^{N}(y(i))^2} \quad (15)$$

Where, y(i) is the system output and $\tilde{y}(i)$ is the predicted output. In Table 1 the variation of corresponding Normalized Mean Square Error (NMSE) are cited for each method.

TABLE. I. COMPARING THE RN AND RNR PERFORMANCES

	Kernel type	**EQMN Training**	**EQMN Testing**
RN	Polynomial	8.6768	0.0056
RNR		0.0045	3.9030.10-05

For the same dataset and with the same Kernel and machine learning parameter, the NMSE of RNR is much lower than the NMSE of RN. Thanks to the efficient statistical reduction method.

The following figures (Fig. 4 & 5) show the tough resemblance between the real and expected output for the two methods.

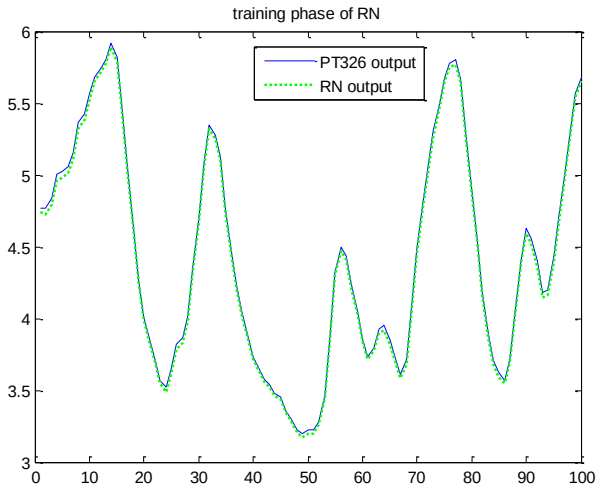

Fig. 4. Training phase of RN and RNR.

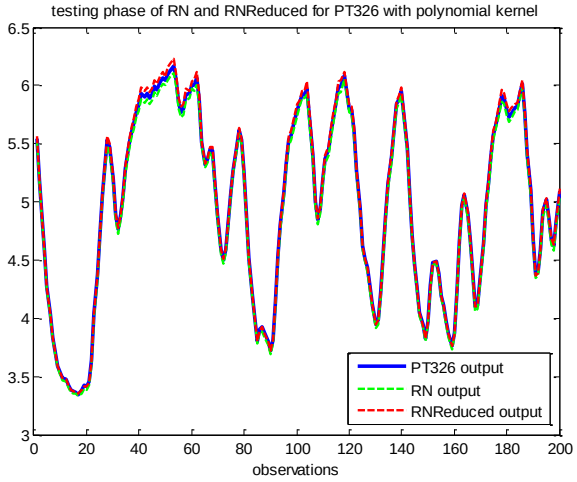

Fig. 5. The testing phase of RN and RNR.

This technique was capable to build a vigorous model for nonlinear system identification. The efficiency of this approach resides in the use of new observations in the testing phase and also facilitates the Gramian matrix implementation in the next section.

D. Accelerating the Gramian matrix computation by a Hardware/Software co-simulation

The Gramian matrix computation is a computationally intensive operation in RN algorithm. Critical speed–up in computation time can be attained by assigning computation tasks to hardware. We present the adopted approach for Gramian matrix computation and its basic principles.

1) *Systolic array architecture for Gramian matrix computation:* The Parallel Matrix Multiplication [29]-[31] has much different identification. In this work, we use the systolic array architecture for the Gramian matrix computation. A systolic array architecture is produced by the interconnection of a set of attached data processing units (DPU) in a regular way [32], [33]. In parallel, each unit or cell receives data from its upstream neighbors to calculate a part of the result. After that it saves the result inside itself and bypasses it downstream neighbors as shown in Fig. 6.

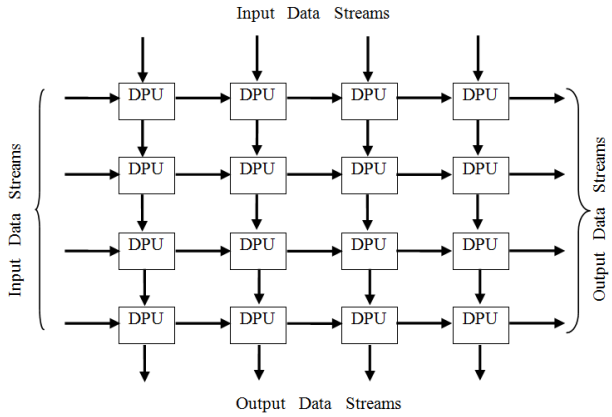

Fig. 6. Principle of systolic array architecture.

The systolic array conception is a mixture between an algorithm and a circuit that implements it. The systolic arrays rely on synchronous data transfers. The individual nodes in systolic array architecture are triggered by the arrival of new data and always treat the data in exactly the same way. We exploit the advantages of this architecture for the implementation of the Gramian matrix on hardware platform. In next paragraph, we explain the proper approach for the matrix computation and the employment of the systolic array method for implementation.

2) Basic principles of serial multiplication: The Gramian matrix $G \in \mathbb{R}^{N\times N}$ *is like that:*

$$G_{ij} = (K(x_i, x_j)), i, j = 1,...,N \quad (16)$$

Where, N is the number of observations and K is the Kernel function that can be chosen either as linear or polynomial Kernel. The Gramian matrix has to be calculated in the training and testing phase. As the input vector **X** can be 1-Dimensional or 2-Dimensional Array, we proposed two Architectures for Gramian matrix computation.

At first, we look at the 1-Dimensional vector multiplication respecting its general constitution. According to the expression of polynomial Kernel with first order:

$$G_{ij} = K(x_i, x_j) = (x_i \times x_j + 1)^{\eta}; i, j = 1,...,N, \eta = 1 \quad (17)$$

It consists to multiply the column vector X(n×1) containing *n* rows and one column with its transpose X^t and add one to the product. In this case, the product of an n dimensional column vector (n×1) by its transpose (row vector (1 × n)) is the Gramian matrix G. That is an (n × n) symmetric and squared matrix. Mathematically, it is presented by the following relationship:

$$G(i, j) = X(i) \times X_t(j) + 1 \quad (18)$$

The key idea here is to calculate the matrix G using the column vector X and its transpose the row vector X^t. The dimension of the given matrices depends on the application.

For efficient implementation and maximum speed-up, the matrix computation is based on systolic array architecture by broadcasting elements of vector X and multiplying it by the corresponding elements of vector X^t. As a simple example, supposing that the vector X is like that: X=[1 2 3] and its transpose X^t =[1 2 3]. The steps of multiplication are shown in Fig. 7.

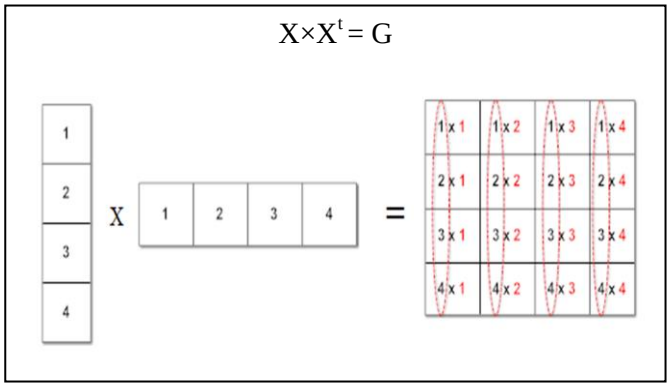

Fig. 7. Example of 1-Dimensional vector multiplication.

From this example, it can be observed that C (i) the i^{th} column of the matrix G is the product of the column vector X by the i^{th} element of this vector:

$$C(i) = X \times X(i)$$

Also for the 2-Dimensional Array, the example of input vector:

X(n×2)

Multiplied by its transpose X^t is presented in Fig. 8:

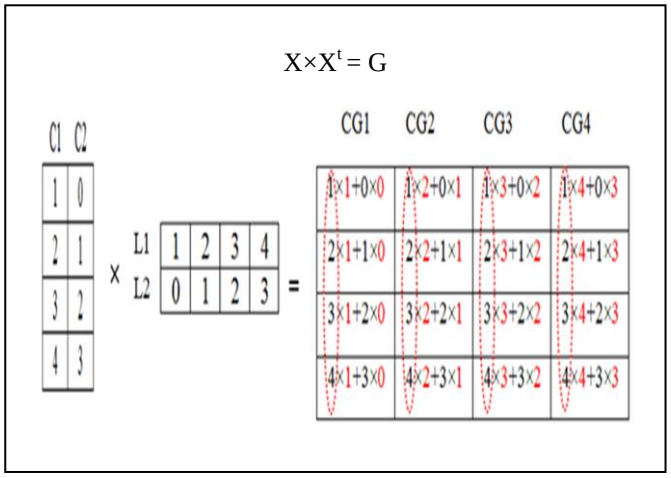

Fig. 8. An example of 2-Dimensional vector multiplication.

Concluding from this example, the expression of Gramian matrix with 2-dimensional vector is a concatenation of column vectors CG.

The i^{th} column of G is the result of sum and product of columns (L1 and L2) and rows (C1 and C2). We can generalize the calculation of each column vector CG of Gramian matrix G as follow:

$$CG(i) = C1 \times L1(i) + C2 \times L2(i) \quad (19)$$

As the input vector X will be streamed the column and rows have no real mean so the previous expression will be modified:

$$CG(i) = C1 \times C1(i) + C2 \times C2(i) \quad (20)$$

The sequence of operations involved in the serial multiplication is as follows:

1) Streaming the elements of column vector X by the input buffer.

2) Calculating the i^{th} column of the matrix G by multiplying each element of the streaming vector by its i^{th} element.

3) Accumulating the multiplier output and writing back the results to the output buffers.

4) Concatenating the n columns to construct the matrix G.

The Fig. 9 represents the system generator blocks for (10×10) Gramian matrix computation using 2-d vector with polynomial Kernel (second order).

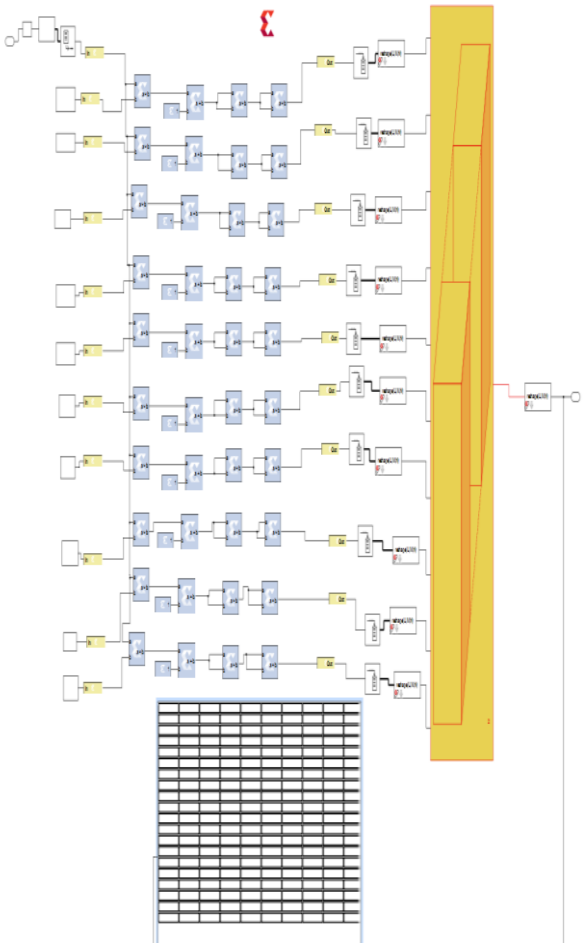

Fig. 9. System generator blocks for Gramian matrix computation.

Before passing to Fig. 10, the type of Kernel function can be selected by a manual switcher. The sigmoid and polynomial Kernels are implemented as shown in Fig. 10:

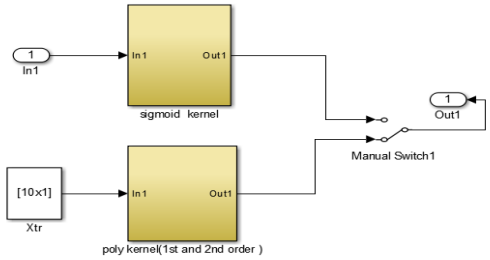

Fig. 10. Selection of Kernel function.

For the polynomial Kernel, it can be chosen for the first or second order by a manual switcher as in Fig. 11:

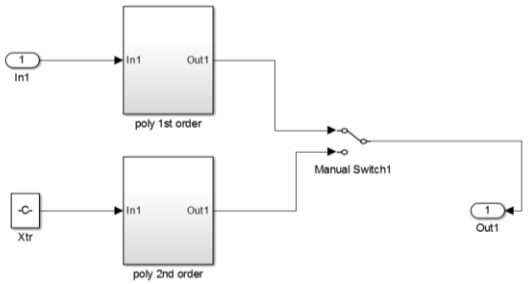

Fig. 11. Selection of the degree of polynomial Kernel function.

The design is able to calculate the Gramian matrix for any system. The user has just to enter the process observations and select the Kernel type. For the regularization parameter, it could be calculated away or in simulation to choose the suitable value. Next, the RN HW/SW Co-design (RN-Cosim) will be tested on a challenging nonlinear system with process noise.

V. Implementation Results and Analysis

A. The Hardware/Software co-simulation steps

In this work, the RN-Cosim co-design was performed using Xilinx System Generator and the Nexys 2 board, which is a complete circuit board and equipped to exploit the circuit development platform based on a Xilinx Spartan 3E FPGA. As shown in Fig. 12, the on-board high-speed USB2 port, jointly with a collection of I/O devices, data ports, and development connectors, enable the conception of a wide range of designs without the demand for any supplementary components.

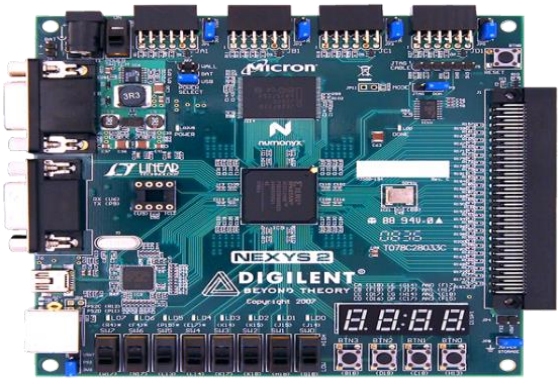

Fig. 12. The Nexys 2-board.

After completing the hardware system, the Simulink environment was exploited to verify functionality of the system. Simulink presents a very supple simulation environment that allows building different testing scenarios. After verifying the functionality of the RN-Cosim model for the different hardware component, the generation of Co-simulation module is executed. While building the hardware system, ISE flow generates a bit-stream that will be later used to configure the FPGA. When the compilation is completed, a new library is created including one block that includes all the functionality required for the system to be executed on the FPGA. The generated library encapsulates the hardware implementation of the RN-Cosim model, which is linked to a bit-stream that will be downloaded into the FPGA during Co-Simulation. The different hardware component is replaced by the new block from the Co-simulation library. Fig. 13 contains the final blocks for the Gramian matrix computation.

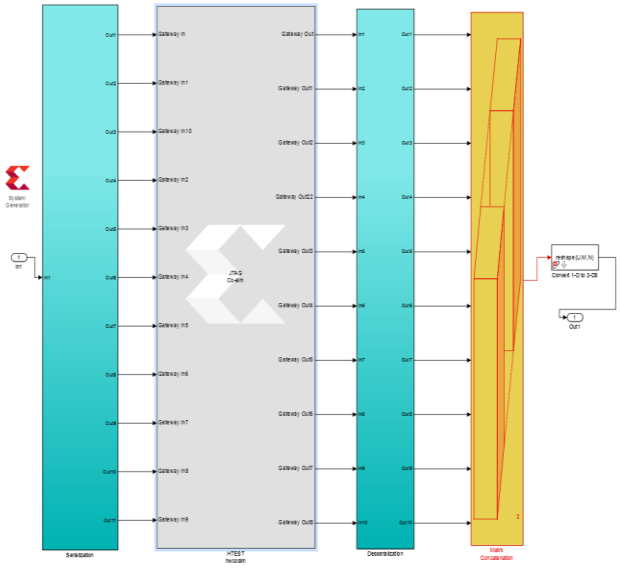

Fig. 13. System generator blocks using JTAG hardware co-simulation block for Gramian matrix computation.

The two blue blocks after and before the hardware co-simulation block, assure the serialization and deserialization of matrix. Then the FPGA is connected to the system generator via the Digilent USB JTAG Cable.

When the design is ready for co-simulation, system generator will first download the bit-stream associated with the block. Once the download completes, system generators reads the inputs from Simulink simulation environment and send them to the design on the board using the JTAG connection. System generator then reads the output back from JTAG and sends it to Simulink for displayed. When simulation is completed, the results should be displayed as shown and the results can be verified by comparing the simulation output to the expected output. The model chosen is a challenging nonlinear system identification; Wiener-Hammerstein benchmark with process noise.

B. *Description of the process and analysis of the implementations results*

The nonlinear system to be modeled is the Wiener-Hammerstein benchmark with process noise [34]. This system is challenging nonlinear system identification due to the process noise present in the system. Moreover, the static nonlinearity is not directly accessible from neither the measured input or output, and the output dynamics are difficult to invert due to the presence of a transmission zero.

The Wiener-Hammerstein benchmark is a well-known block oriented system. As illustrated in Fig. 14, it contains a static nonlinearity f(x) that is sandwiched in between two LTI blocks R(s) and S(s).

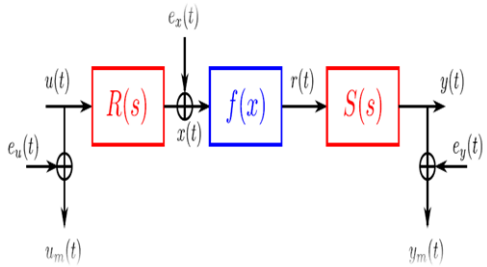

Fig. 14. The Wiener-Hammerstein system with process noise.

The presence of the two LTI blocks results in a problem that is harder to identify. The additive process noise $e_x(t)$ is filtered white Gaussian noise sequence.

The input and output signals of the system are:

1) *r : reference signal, signal loaded into the generator,*
2) *u: measured input signal,*
3) *y: measured output signal,*
4) *f_s: the sample frequency.*

Fig. 15 presents the plot of the measured output signal *y* versus the measured input signal *u* from a thousand of values with sampling time one second.

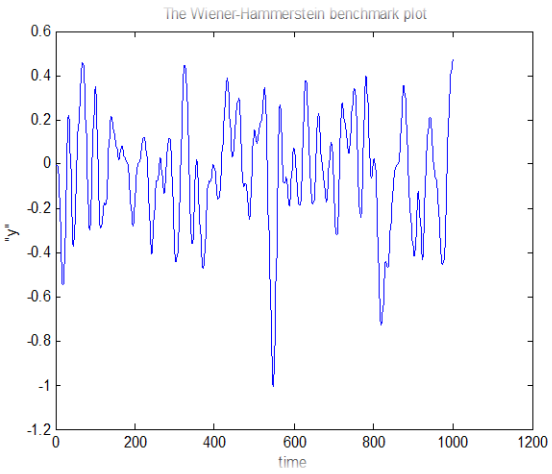

Fig. 15. The plot of Wiener-Hammerstein benchmark with process noise.

To construct the RKHS model, 100 observations are employed for the training phase that will be reduced to only 10 observations. Then, 100 new observations are randomly chosen for the validation phase. The data used for testing phase are not statistically treated. The Kernels used are of type: polynomial (first and second order) and sigmoid. The input vector is a 2-dimensional vector. The optimal parameter λ of the machine learning was obtained by a cross validation technique and it is equal to 0.0001.

By examining the plots in Fig. 16, it can be remarked that the two models outputs (RNR and RN-Cosim) are in concordance with the Wiener benchmark output in the training and testing phase. Comparing to benchmark process, the deviation of the RNR and RN- Cosim is small. This illustrates the excellent performances of the projected identification method.

Table 2 gives the computation time (CT) and NMSE in training and testing phase of RNR algorithm and RN-Cosim with sigmoid and polynomial Kernel.

TABLE. II. COMPARING NMSE AND CT OF RNR AND RN-COSIM

	Kernel type	**NMSE Training**	**NMSE Testing**	**CT(s)**
RNR	Polynomial	6.7980e-09	3.1847e-09	1.3884
RN-Cosim		3.1850e-07	7.1274e-07	0.033066
RNR	Sigmoid	7.9972e-09	3.1847e-08	1.4196
RN-Cosim		3.1850e-07	1.3886e-07	0.034018

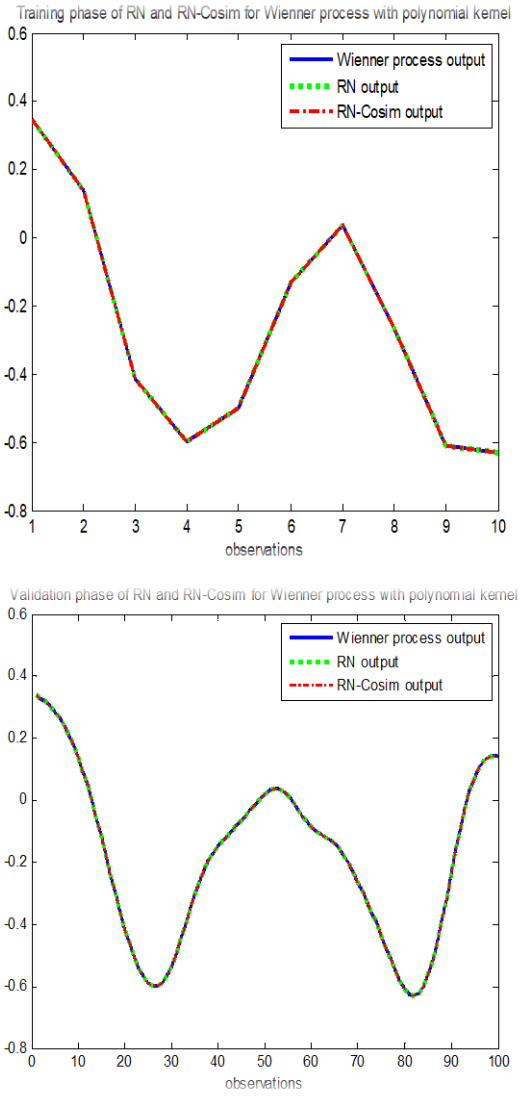

Fig. 16. Training and validation phase of RN and RN-Cosim.

In terms of the model accuracy, the two models are excellent but the RNR gives the lower NMSE comparing to RN-Cosim that uses the fixed point arithmetic. The reduction method based on statistical treatment improves the model accuracy because the reduced data set reflects the central tendency of data that decreases the model error and increases its capacity of generalization.

Considering the simulation speed, compared with RNR algorithm execution, the proposed RN-Cosim co-simulation gives more than 40 times speedup.

Note that, in both cases of sigmoid and polynomial Kernel, RN-Cosim gives significant simulation speedups thanks to the association of Gramian matrix computation to hardware and the immediate execution of the whole model contrary to the sequential algorithm execution. The properties of our co-design are listed in Table 3.

TABLE. III. COMPARING NMSE AND CT OF RNR AND RN-COSIM

	Kernel type	NMSE Training	NMSE Testing	CT(s)
RNR	Polynomial	6.7980e-09	3.1847e-09	1.3884
RN-Cosim		3.1850e-07	7.1274e-07	0.033066
RNR	Sigmoid	7.9972e-09	3.1847e-08	1.4196
RN-Cosim		3.1850e-07	1.3886e-07	0.034018

As seen from Table 4, the resulting architecture requires about 406 slices with 11% utilization from the available resources and about 48 Bonded IOBs with 19% utilization. Whereas the utilization of slice Flip Flop are approximately insignificant and negligible. The proposed architecture has low complexity and low resources consumption that enhanced effectiveness in area and provide a good choice in terms of low-cost hardware. The implemented RN-Cosim co-design reaches 50 MHz as maximum frequency.

TABLE. IV. FPGA RESOURCES UTILIZATION IN THE HW/SW CO-SIMULATION

Logic utilization	Used	Available	Utilization
Number of slices	406	3584	11 %
Number of slice Flip Flops	16	7168	1 %
Number of Bonded IOBs	48	251	19 %
Number of GCLKS	2	24	8 %

Since the current implementation, it is possible to solve various nonlinear system identification tasks in the RKHS space.

VI. CONCLUSION AND FUTURE WORK

This article proposed efficient method of HW/SW Co-simulation using Xilinx system generator. The basic principle of the contribution is to improve the RKHS model performances and to accelerate the computation task by including hardware in the loop. Also we developed a new reduction method to decrease the model errors. The experiments prove that the co-design reach more than 40 times speedup compared with the RN algorithm.

The principal improvement of this advance is the opportunity of modeling and confirming the overall system inside the identical design environment. Moreover, Simulink offers a friendly graphics interface for flexible modeling and simulation. The design was well organized into hierarchical modules including the hardware and software components that require rigorous verification all along the design flow.

Future works will incorporate the use of the Xilinx System Generator development devices for the implementation of another Kernel method like KPCA. As development in our co-

design, new Kernel function will be added in order to increase the simulation accuracy. We will also apply RN-Cosim to systems that are more complex with other FPGA type.

REFERENCES

[1] Bernhard Scholkopf and Alexander J. Smola, "Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond", MIT Press Cambridge, MA, USA 2001

[2] Yingwei Zhang, "Enhanced statistical analysis of nonlinear processes using KPCA, KICA and SVM", Chemical Engineering Science, Volume 64, Issue 5, March 2009, Pages 801-811.

[3] Mikhail Z. Zgurovsky , Yuriy P. Zaychenko, "Neural Networks", Chapter in The Fundamentals of Computational Intelligence: System Approach Volume 652 of the series Studies in Computational intelligence pp 1-37 Date: 02 July 2016.

[4] Alaa F. Sheta, " A Comparison between Regression, Artificial Neural Networks and Support Vector Machines for Predicting Stock Market Index ", communication on (IJARAI) International Journal of Advanced Research in Artificial Intelligence, Vol. 4, No.7, 2015

[5] C.A. Schmidt , S.I. Biagiola, J.E. Cousseau , J.L. Figueroa, "Volterra-type models for nonlinear systems identification" Journal of Applied Mathematical Modelling Volume 38, Issues 9–10, 1 May 2014, Pages 2414–2421.

[6] Cristian Preda, "Regression models for functional data by reproducing kernel Hilbert spaces methods", Journal of Statistical Planning and Inference Volume 137, Issue 3, 1 March 2007, Pages 829–840.

[7] Sergios Theodoridis "Learning in Reproducing Kernel Hilbert Spaces", Machine Learning, A Bayesian and Optimization Perspective 2015, Pages 509–583.

[8] C.BURGES, "A Tutorial on Support Vector Machines for Pattern Recognition." Review 1–43 on Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

[9] John C. Platt. Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines. Technical report, Advances in kernel methods - support vector learning, 1998.

[10] X.Zhang, Y.Zhang: " GPU Implementation of Parallel Support Vector Machine Algorithm with Applications to Intruder Detection" JOURNAL OF COMPUTERS, VOL. 9, NO. 5, MAY 2014

[11] "System Generator for DSP": Getting Started Guide.

[12] "Medical Image Processing on The GPU–Past, Present and Future," Medical Image Analysis, vol. 17, pp. 1073-1094,2013.

[13] S. Asano, T. Maruyama, and Y. Yamaguchi, "Performance Comparison of FPGA, GPU and CPU in Image Processing," in International Conference on Field Programmable Logic and Applications, 2009. FPL 2009, 2009, pp. 126-131.

[14] E. Fykse, "Performance Comparison of GPU, DSP and FPGA Implementations of Image Processing and Computer Vision Algorithms in Embedded Systems," M.Sc. thesis, Department of Electronics and Telecommunications, Norwegian University of Science and Technology, 2013.

[15] J. G. Filho, M. Raffo, M. Strum, and W. J. Chau, "A General-Purpose Dynamically Reconfigurable SVM," in 2010 VI Southern Programmable Logic Conference (SPL),2010, pp. 107-112.

[16] D. Anguita, S. Pischiutta, S. Ridella, and D. Sterpi, "Feed-Forward Support Vector Machine without Multipliers," IEEE Transactions on Neural Networks, vol. 17, pp. 1328-1331, 2006.

[17] L. Bustio-Martínez, R. Cumplido, J. Hernández-Palancar, and C. Feregrino-Uribe, "On the Design of a Hardware-Software Architecture for Acceleration of SVM's Training Phase," in Advances in Pattern Recognition, ed: Springer,2010, pp. 281-290.

[18] A. H. M. Jallad and L. B. Mohammed, "Hardware Support Vector Machine (SVM) for Satellite on-Board Applications," in 2014 NASA/ESA Conference on Adaptive Hardware and Systems (AHS), 2014, pp. 256-261.

[19] K. Nagarajan, B. Holland, A. D. George, K. C. Slatton, and H. Lam, "Accelerating Machine-Learning Algorithms on FPGAs using Pattern-Based Decomposition," Journal of Signal Processing Systems, vol. 62, pp. 43-63, 2011.

[20] X. Pan, H. Yang, L. Li, Z. Liu, and L. Hou, "FPGA Implementation of SVM Decision Function Based on Hardware-friendly Kernel," in International Conference on Computational and Information Sciences , ICCIS 2013 Proceedings, 2013, pp. 133-136.

[21] V.Vapnik, "An Overview of Statistical Learning Theory" , IEEE TRANSACTIONS ON NEURAL NETWORKS, VOL. 10, NO. 5, SEPTEMBER 1999.

[22] Wahba G. "An introduction to model building with Reproducing Kernel Hilbert Spaces". Technical report No. 1020. Department of Statistics, University of Wisconsin-Madison; 2000.

[23] Theodoros Evgeniou, Massimiliano Pontil, Tomaso Poggio, "Regularization Networks and Support Vector Machines" review on journal of Advances in Computational Mathematics (1999).

[24] Inc., T. M.: "Embedded MATLAB User's Guide" The MathWorks Inc, 2007.

[25] "System Generator for DSP Getting Started Guide" UG639 (v 14.3) October 16, 2012

[26] Chris Tsokos and Rebecca Wooten, " Basic Statistics" The Language and Art of Math 2016, Pages 265–327

[27] 'Air Temperature Control', Laboratory Manual .

[28] Intissar Sayehi, Okba Touali, Belgacem Bouallegue,

[29] Rached Tourki. "A comparative study of two kernel methods: Support Vector Regression (SVR) and Regularization Network (RN) and application to a thermal process PT326", 2015 16th International Conference on Sciences and Techniques of Automatic Control and Computer Engineering (STA), 2015

[30] Albert-Jan N. Yzelman, Dirk Roose, Karl Meerbergen, "Sparse Matrix-Vector Multiplication: Parallelization and Vectorization", Multicore and Many-Core Programming Approaches 2015, Pages 457–476.

[31] Urban Borštnika, Joost VandeVondeleb, Valéry Webera, Jürg Huttera, "Sparse matrix multiplication: The distributed block-compressed sparse row library" Parallel Computing Volume 40, Issues 5–6, May 2014, Pages 47–58

[32] Marco Maggioni and Tanya Berger-Wolf, " Optimization techniques for sparse matrix–vector multiplication on GPUs" Journal of Parallel and Distributed Computing Volumes 93–94, July 2016, Pages 66–86.

[33] H. L. P. Arjuna Madanayake, Student Member, IEEE, and Len T. Bruton, Fellow, IEEE, " A Systolic-Array Architecture for First-Order 3-D IIR Frequency-Planar Filters" , IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—I: REGULAR PAPERS, VOL. 55, NO. 6, JULY 2008.

[34] Wei Jin, Chang N. Zhang and Hua Li, " MAPPING MULTIPLE ALGORITHMS INTO A RECONFIGURABLE SYSTOLIC ARRAY" , published in Electrical and Computer Engineering journal from Canadian Conference CCECE 2008 on 4-7 May 2008, pages 001187 – 001192, ISSN - 0840-7789.

[35] M. Schoukens, J.P. Noel, "Wiener-Hammerstein benchmark with process noise", Workshop on Nonlinear System Identification Benchmarks on April 25-27, 2016, Brussels, Belgium.