VLSI Architecture of Exponential Block for Non-Linear SVM Classification

Sumeet Saurav, Sanjay Singh, Anil K Saini, Ravi Saini
IC Design Group
CSIR-Central Electronics Engineering Research Institute
Pilani, Rajasthan, India
sumeetssaurav@gmail.com

Shradha Gupta
AIM & ACT
Banasthali Vidhyapeeth
Tonk, Rajasthan, India

***Abstract*—In this work, we present a dedicated hardware implementation of exponential function computation unit using CORDIC (Coordinate Rotation Digital Computer) algorithm for extended range of input arguments. Hardware architecture design is done keeping in view its possible integration in the hardware implementation of the Radial Basis Function (RBF) based Support Vector Machine (SVM) classifier. The designed architecture is prototyped on a field programmable gate array (FPGA) to meet the specific requirement of performance. The proposed design is operating at a maximum clock frequency of 249 MHz. This shows good performance of our proposed architecture in terms of speed. Synthesis result also reveals that the proposed architecture is resource efficient.**

Keywords—CORDIC; FPGA; Elementary Functions; Non-Linear SVM Classification; VLSI Architecture.

I. INTRODUCTION

Exponential value computation is an important operation found in many applications related with wireless systems, computer graphics, digital signal processing (DSP), scientific computing, artificial neural networks and multimedia [1]. These applications require dedicated hardware blocks for the computation of elementary functions like exponential to meet the timing constraints.

A number of works has been reported in literature dealing with the hardware implementation of the exponential block using CORDIC algorithm. In [2], the authors have discussed that the convergence range of the CORDIC algorithm can be expanded by introducing extra iterations. However, these extra iterations increases the computation time of the algorithm. In [3], a parallel approach has been used to implement the algorithm by using arrays and trees. However, the major limitation of the algorithm is that it extensively makes use of Dadda and Wallace tree multipliers which increases the resource utilization. On the other hand, in CORDIC algorithm any elementary function can be implemented by incorporating only shift and add operations which is more resource efficient. In [4], the author have discussed two different number formats for the implementation of CORDIC algorithm and concluded that the IQ-Math number format covers less space on FPGA as compared to the floating-point number format. In [5], the authors have proposed BKM algorithm based on redundant number system for complex elementary functions. However, the major limitation of the algorithm is that, it requires 8n constant storage for n bit accuracy and also has complex selection rules. Some more work can also be found in literature dealing with CORDIC algorithm modification based on the specific requirement of application in [6]-[8].

In this paper, we have discussed the algorithmic description and hardware implementation of an exponential block required for kernel computation in the hardware implementation of Radial Basis Function (RBF) based SVM classification algorithm. Requirement of large input range of values for kernel computation has led to the development of an architecture of exponential function computation for the extended range of input argument. In order to have less resource utilization on our target FPGA, we have used fixed-point number format in our implementation.

The rest of the paper is organized as follows: In section II we have discussed the algorithm for exponential function computation and its extended range version. Hardware implementation of the exponential function algorithm has been described in section III. The architecture design of the exponential function algorithm for the extended range of inputs forms the content of the section IV. The synthesis results have been presented in section V followed by conclusion in section VI.

II. DESCRIPTION OF EXPONENTIAL COMPUTATION ALGORITHM

The standard algorithm for the computation of exponential function $y = \exp(x_o)$ for an input x_o uses two recursive equations shown by (1) and (2) [9].

$$x_{i+1} = x_i - \ln b_i \tag{1}$$

$$y_{i+1} = y_i b_i \tag{2}$$

These two recursive equations are dependent to each other in such a way that if one equation converged to a constant value, the other equation produces the desired result. All b_i values are chosen in such a way that the value at order m of the sequence $x_o, x_1, \ldots, x_m$ converges to 0 (i.e. $x_m = 0$). In this algorithm, m defines the required number of iterations to provide desired result. The value of m taken is 25 in our case.

In order to simplify the multiplication as given in (1) and (2), b_i's is selected as shown in (3).

$$b_i = (1 + s_i 2^{-i}) \tag{3}$$

$$\text{where } s_i \in \{-1, 0, 1\}$$

So that the multiplication is converted into shift and add operation. In this basic algorithm two sided selection rule $s_i \in \{-1, 1\}$ is used which depends on the positive or negative value of x_{i+1} defined as shown below in (4).

$$s_i = \begin{cases} +1 & if \quad x_{i+1} \geq 0 \\ -1 & if \quad x_{i+1} \prec 0 \end{cases} \tag{4}$$

All possible values of b_i can be pre-calculated and stored in Look-up table (i.e. a ROM of size (m x input size)), since their online computation will increase the computation time of the algorithm [10].

Now, substituting the value of b_i in (1) and (2), we get

$$x_{i+1} = x_i - \ln(1 + s_i 2^{-i}) \tag{5}$$

$$y_{i+1} = y_i + (y_i s_i) 2^{-i} \tag{6}$$

However, the major limitation of this algorithm is that it can be used only for the input coming under the range given by (7).

$$-1.24 \leq x_0 \leq 1.56 \tag{7}$$

The algorithmic description in the flowchart form has been shown in the fig.1.

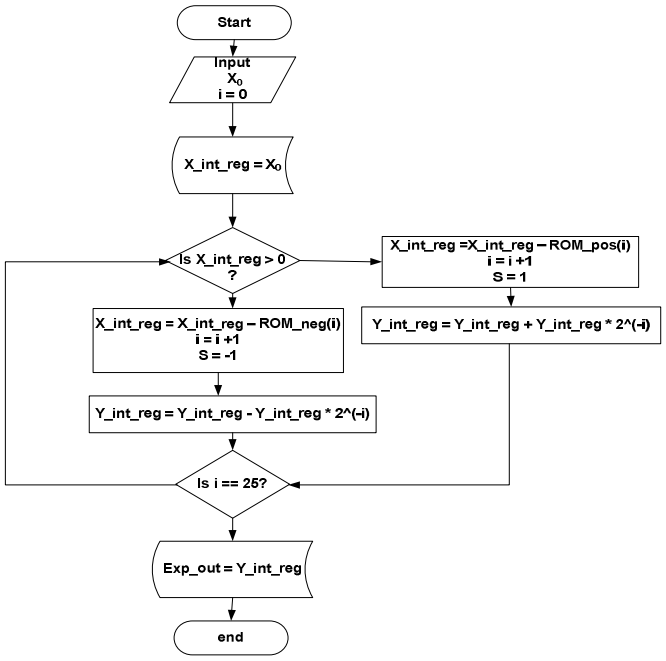

Fig.1. Flowchart of the Exponential Function Algorithm

For our application, we require exponential value computation beyond the range given in (7), therefore we extended this standard algorithm in order to fulfill our requirement.

A. Exponential Function with Extended Range

The systematic flowchart representation of the algorithm for the exponential function computation with extended range has been shown in the fig. 2.

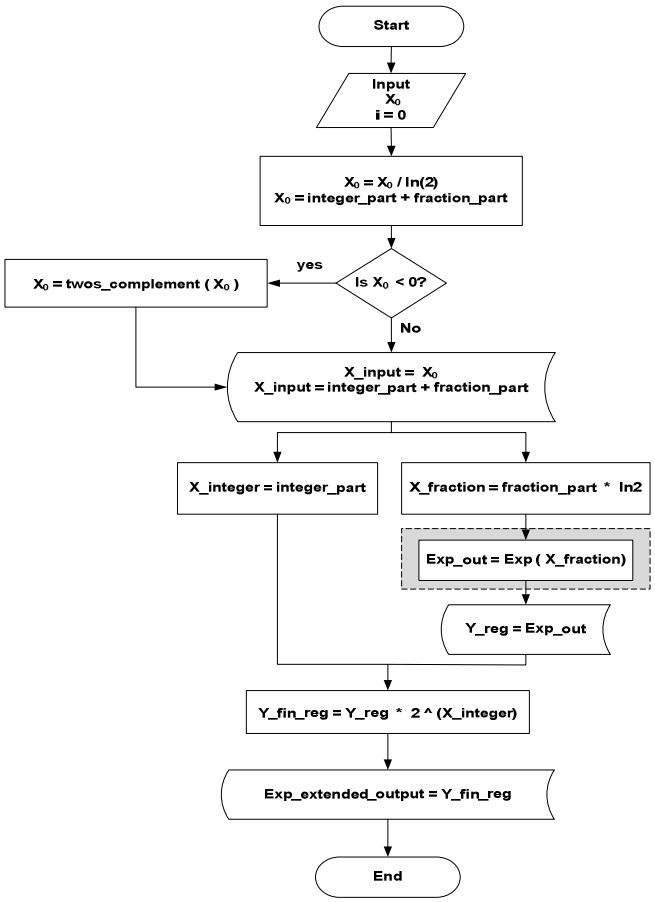

Fig. 2. Flowchart of Exponential Function with Extended Range

The steps involved in the calculation of exponential function for extended range of inputs are as follows:

- To handle inputs beyond the range given by (7), the following equivalent representation is used

$$x_o = x_o \log_2 e . \ln 2 \tag{8}$$

- Now $x \log_2 e$ can be expressed as I + f, where I is the integer part and f is the fractional part. We can now write

$$y = e^x = e^{(I+f)\ln 2} = e^{I \ln 2} . e^{f \ln 2} = 2^I . e^{f \ln 2} \tag{9}$$

- Therefore, to compute e^x we need to compute $e^{f \ln 2}$ and shift the result to the left by I bits if x is positive and to the right by I bits if x is negative.

- To compute $e^{f \ln 2}$ (i.e. exponential of fractional part), we use previously described standard exponential function algorithm. The computed result is then shifted either left or right depending on the sign of x and by amount equal to I number of bits.

III. Hardware Implementation of Exponential Function Algorithm

Our proposed architecture for exponential function algorithm is shown in fig. 3. Different blocks used in the architecture are: 2:1 Multiplexer, Registers, 25-bit Adder/subtractor, Shifter, Counter, and FSM controller. The size of fixed point input data is taken as 25 bits and 2.23 format is used (where left most 2 bits are used for integer part and remaining 23 bits are used for fraction part).

Detailed description of the blocks used in our proposed architecture has been discussed below.

A. Multiplexer

A total of two 2:1 multiplexers are used in this architecture. Mux1 is used to preset the value of x_o for 0^{th} iteration. Mux2 is used to select the values either from ROM1 (stored with pre calculated values of $\ln(1+2^{-i})$) or from ROM2 (stored with pre calculated values of $\ln(1-2^{-i})$).

B. Registers

Registers are used in this architecture to store the input operand, intermediate results, and the final output.

C. 25-Bits Adder/Subtractor

Two 25-bits adder and subtractor are used in the architecture to perform the required operation included in recursive formula defined in (5) and (6).

D. Shifter

A Barrel shifter is used to shift values in Y_reg by a specified amount given by the value of counter (i^{th} iteration).

Implementation of such barrel shifter is done by using sequence of multiplexers. Bits shifting in Y_reg depend on the value of counter in our architecture.

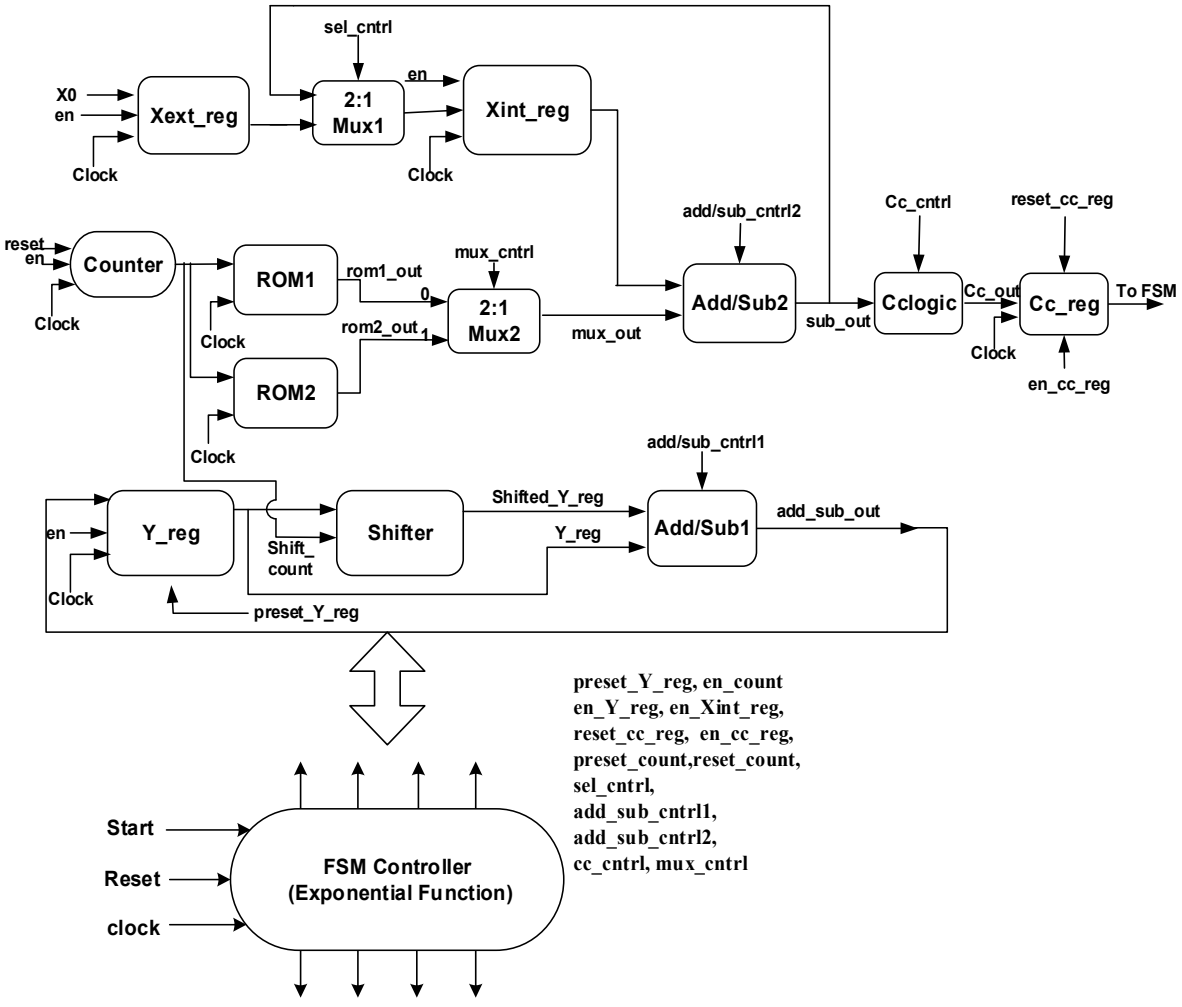

Fig. 3. Architecture of Exponential Function Algorithm

E. Counter

Counter is used to give address to all b_i values stored in ROM1 and ROM2. Counter value is also used to decide the amount by which the values in Y_reg will be shifted. When the value of the counter becomes equal to 25, it indicates that all iterations required for the convergence of the proposed algorithm have been processed.

F. FSM Controller

A controller, as shown in fig. 3 is designed and used to provide appropriate control signals to different blocks used in the proposed architecture.

Controls are required at each step to control the data movement to/from the functional unit. Triggering of these functional units depends on controller. All the steps that are performed by the controller are given below:

1) Step1: Reset the counter and store the input value in Xext_reg and after that controller goes to next step.

2) Step2: Store the value of Xext_reg into Xint_reg and preset the value of Y_reg to 1. After this the controller goes to the next step.

3) Step3: Contents of ROM (25 bits) whose address is given by counter is subtracted from Xint_reg and the subtracted result is again stored in the Xint_reg. Also Addition/subtraction of Shifted_Y_reg and Y_reg is done in this step and the result is stored in the Y_reg. Counter is incremented to address the next iteration and the controller goes to next step.

4) Step4: Value of counter is checked to see whether its count becomes 25. If the counter value is less than 25, controller goes to step3 otherwise it goes to the next step.

*5) Step5: A*fter completion of the 25 iterations, the result is available in the Y_reg and the controller goes back to the initial state i.e. step1.

IV. Hardware Implementation of Exponential Function Algorithm with Extended Range

In this section, an architecture is proposed for extended range of inputs as shown is fig. 4. Blocks of this architecture is arranged in such a way that integer and fraction parts can be separated and on which operations can be performed easily as described in section II. The size of fixed point input data is taken as 24 bits and 8.16 format is used (where left most 8 bits are used for integer part and rest of the 16 bits are used for fraction part).

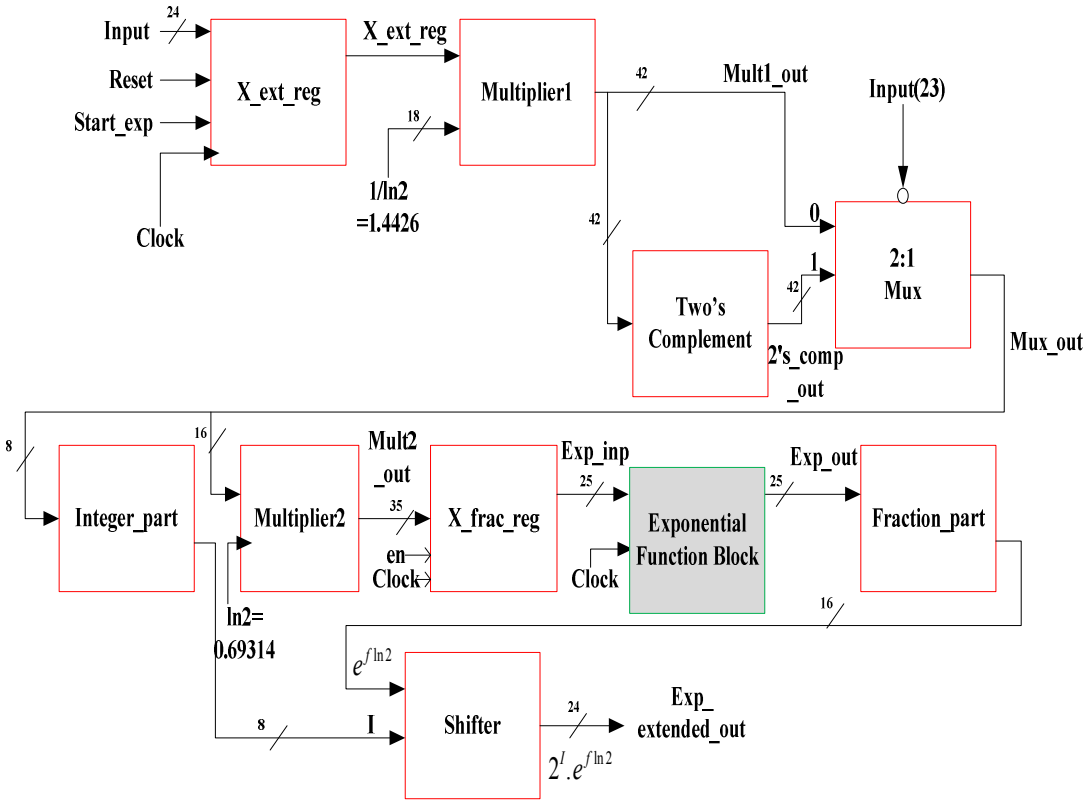

Fig.4. Architecture of Exponential Function Algorithm with Extended Range

In this architecture, Multiplier1 is used to perform the multiplication operation between constant (1/ln2) and input data and result is stored in mult1_out. If the input data is negative, we need to perform two's complement of mult1_out for the accurate separation of integer and fraction. On the other hand, if the input data is positive, there is no need to

perform two's complement and we can separate integer and fractional parts directly. Therefore, a 2:1 mux is used to select either two's complement of mult1_out or directly the mult1_out. Currently, the size of fixed point Mux_out is taken as 42 bits in 10.32 format (where 10 bits are used for integer part and 32 bits are used for fraction part).

Integer value (8-bit) is separated from the Mux_out and stored in Integer_part register. Similarly, 16-bit fractional value is separated and multiplied with constant (ln2) and result is stored in X_Frac_reg. The value of X_Frac_reg becomes the input of Exponential Function Block. At this point, we can say that all of the operations associated with fraction part (i.e. computation of $e^{f \ln 2}$) have been performed successfully and result of this part is stored in Exp_out. The value of Exp_out is shifted according to the value stored in Integer_part register. Finally, result of the exponential function with extended range is stored in Exp_extended_out. Since we have used 8 bits for the integer, this extended version of the exponential function can be used for the input values in the range given by (10).

$$-128 \leq x_0 \leq 127 \qquad (10)$$

V. RESULTS

The synthesis of the designed architectures is carried out using Xilinx ISE 14.2 by utilizing Xilinx Vertex5 5vfx130t-2ff1738 FPGA board as target device. Simulation of the design has been performed on Modelsim 10.1 C simulator. Synthesis result of the exponential function algorithm and its extended version is shown in table I and table II respectively. The synthesized architectures operates at a maximum clock frequency of 249 MHz. Synthesis result shows good performance of our proposed architectures in terms of both speed and area.

TABLE I. HARDWARE RESOURCE UTILIZATION OF EXPONENTIAL FUNCTION ALGORITHM

Logic Utilization	Available	Used	Utilization Rate
Slice LUTs	81920	204	1%
Occupied Slice	20480	80	1%
Slice Registers	81920	88	1%
Bonded IOB	840	53	6%
DSP48Es	-	-	-

TABLE II. HARDWARE RESOURCE UTILIZATION OF EXPONENTIAL FUNCTION ALGORITHM WITH EXTENDED RANGE

Logic Utilization	Available	Used	Utilization Rate
Slice LUTs	81920	473	1%
Occupied Slice	20480	181	1%
Slice Registers	81920	120	1%
Bonded IOB	840	51	6%
DSP48Es	320	2	1%

VI. CONCLUSION

In this paper, we have proposed dedicated hardware architecture for the computation of exponential function for extended range of values. Proposed architecture is suitable for integration with the hardware implementation of the Radial Basis Function (RBF) support vector machine (SVM) classification algorithm as it provides optimum result which was not possible with the standard CORDIC based exponential function computation unit.

ACKNOWLEDGMENT

The authors express their deep sense of gratitude to Dr. Chandra Shekhar, Director CSIR-CEERI for encouraging research and development activities. Authors, would like to thanks Dr. AS Mandal, Chief-Scientist and Prof. Raj Singh, Chief Scientist and Group Leader, IC Design Group, CSIR-CEERI, for their constructive suggestions. The authors also like to thanks Mr. Sanjeev Kumar, Technical Officer, CSIR-CEERI for providing technical support during the work. The financial support of DeitY/MCIT is gratefully acknowledged.

REFERENCES

[1] J.-A. Pineiro, M. D. Ercegovac, and J.D. Bruguera,"Algorithm and Architecture for Logarithm, Exponential, and Powering computation," IEEE Transactions on Computers, vol. 53, no. 9, pp. 1085-1096, 2004.

[2] X. Hu, R. G. Harber, and Steven C. Bass,"Expanding the range of convergence of the CORDIC algorithm," IEEE Transactions on Computers, vol. 40, no. 1, pp. 13-21, 1991.

[3] D. M. Mandelbaum, and S. G. Mandelbaum, "A fast, efficient parallel-acting method of generating functions defined by Power series, including Logarithm, Exponential, and Sine, Cosine," IEEE Transactions on Parallel and Distributed Systems, vol. 7, no. 1, pp. 33-45, 1996.

[4] V. Chandra S,"A Survey on CORDIC Algorithm Implementations Using Different Number Format," International Journal of Innovative Research in Science, Engineering and Technolongy, vol. 3, no. 6, pp. 13452-13458, 2014.

[5] J.-C. Bajard, S. Kla, and J.-M. Muller,"BKM: A New Hardware Algorithm for Complex Elementary Functions," in Proceedings of the 11th IEEE Symposium on Computer Arithmetic, pp. 146-153, 1993.

[6] V. Kantabutra,"On Hardware for Computing Exponential and Trigonometric Function," IEEE Transactions on Computers, vol. 45, no. 3, pp. 328-339,1996.

[7] M. Arora, R. S. Chauhan, and L. Bagga,"FPGA Prototyping of Hardware Implementation of CORDIC Algorithm," International Journal of Scientific and Engineering Research, vol. 3, no. 1, pp.1, 2012.

[8] A. Naseem, and P. David Fisher,"The Modified CORDIC Algorithm," in Proceedings of the 7th IEEE Symposium on Computer Arithmetic, pp. 144-152, 1985.

[9] I. Koren, "Computer arithmetic algorithms", 2nd ed.,Universities Press, 2002.

[10] J.-M. Muller,"Elementary Functions Algorithms and Implementation", 2nd ed. Birkhauser: Boston, 2006.