An Area-Efficient and High Throughput Hardware Implementation of Exponent Function

Muhammad Awais Hussain
Department of Electrical Engineering
National Central Unviersity
Taoyuan, Taiwan
awais.hussain@dsp.ee.ncu.edu.tw

Shung-Wei Lin
Department of Electrical Engineering
National Central Unviersity
Taoyuan, Taiwan
sunwait@dsp.ee.ncu.edu.tw

Tsung-Han Tsai
Department of Electrical Engineering
National Central Unviersity
Taoyuan, Taiwan
han@dsp.ee.ncu.edu.tw

2022 IEEE International Symposium on Circuits and Systems (ISCAS) | 978-1-6654-8485-5/22/$31.00 ©2022 IEEE | DOI: 10.1109/ISCAS48785.2022.9937238

***Abstract*— In this paper, an area-efficient and high throughput hardware implementation of the exponent function has been proposed. The proposed exponent calculation method eliminates the memory requirements leading to power and area savings. The pipelined hardware implementation results in a high-frequency design with reduced resources usage. The hardware implementation has been performed for Xilinx Virtex-4 FPGA board and TSMC 90nm process node. The throughput of 411.3 Mbps at 115.7 MHz frequency and 711.11 Mbps at 200 MHz frequency can be achieved for FPGA and ASIC design, respectively. The power consumption is 242mW and 6.1 mW for FPGA and ASIC platforms, respectively.**

Keywords*— *Exponent function, VLSI, FPGA, Digital design, Hardware accelerator.

I. Introduction

The exponential function is an important mathematical function used in digital signal processing applications, such as sine, cosine, logarithm, as well as modern computation algorithms, such as Deep Neural Networks (DNNs) [1], Long Short-Term Memory (LSTM) [2], Graph Neural Networks [3], etc. The exponential function calculation can be executed in general purpose Central Processing Units (CPUs), however, for high-speed calculation of the exponential function, the high-speed special solution is desired. Field Programmable Gate Arrays (FPGA) and Application Specific Integrated Circuit (ASIC) are widely used to perform the acceleration of the mathematical functions which include exponential function also.

There are numerous previous implementations of exponent functions in the hardware domain that are targeted for fixed-point exponent calculation. Zhang et al. [4] utilized a direct function value Look-up Table (LUT) method and Taylor expansion to calculate exponent function. However, the hardware implementation of Taylor series requires the usage of LUTs which results in extra hardware resource usage. Peter [5] and Gao et al. [6] have also utilized Taylor series expansion to perform the hardware implementation of the exponent function. The basic-split calculation method has been used to perform exponent calculation [7]; however, both of their methods are also based on LUT which leads to large memory requirements in the hardware design. For the modern-day complex ASIC designs, it is desired to make a simple exponent function calculation method that can run at high speed with the low area and high throughput. The use of LUT and memory (ROM and RAM) leads to a large chip area as well as high power consumption especially due to SRAM memory usage [8]. The data fetch from SRAM can cause 6x more energy consumption as compared to the normal ALU operations such as multiplication and addition [9].

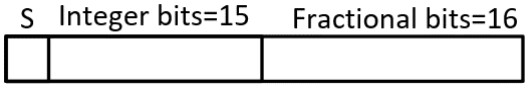

Fig. 1. 32-bit signed number representation. S represents sign-bit, 15-bits are for the integer part and 16-bits are for the fraction part

In this paper, we perform the simplification of the exponent calculation method to eliminate the need of memory while achieving the optimal error in exponent calculation in fixed-point numbers as compared to floating-point numbers. Then, a hardware implementation of the exponent function in ASIC and FPGA platforms is performed such that it can be used in any hardware design to speed up the computation process for the exponent calculation. The proposed design can support a 32-bit fixed number format. Moreover, the aim of designing the proposed exponent accelerator is to eliminate the memory usage in the hardware design as memory usage costs a high area as well as high power consumption in the hardware design.

II. Proposed Method

The proposed method aims to eliminate the need for memory requirements during exponent calculation, so, we do not utilize Taylor series to calculate the exponent value because Taylor series has a high memory requirement, as well as computation complexity is high [4]. So, we simplify the mathematical computation of the exponent function to eliminate memory usage in the hardware design.

The mathematical form of exponent function can be represented as Eq. (1);

$$e^x = 2^{x\log_2(e)} \tag{1}$$

From the above equation, $log_2(e)$ can be presented as a constant number as it represents the simple log_2 value of Euler's number. This eliminates the need of performing log_2 operation in the hardware.

Now, it is supposed that *log* of *e* can be represented as follows;

$$y = \log_2(e) \tag{2}$$

Eq. (3) can be obtained as follows after substituting Eq. (2) in Eq. (1);

$$e^x = 2^{x*y} \tag{3}$$

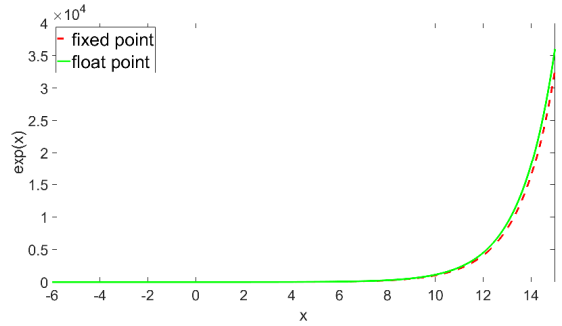

Fig. 2. Error plot showing the difference between fixed- and floating-point exponent values. Green and red lines represent floating- and fixed-point values, respectively. Floating-point values are scaled by multiplication to 1.1 to show the difference between fixed- and floating-point values clearly in the plot

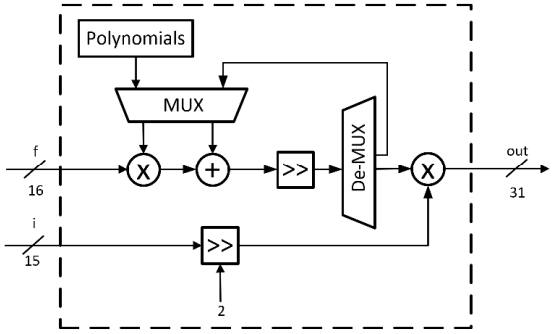

Fig. 3. The proposed hardware design for exponent calculation. The pipelining has been implemented in the hardware design to reduce the critical path during data processing to achieve high design throughput

From Eq. (3), in order to calculate the exponent of the value *x*, only two mathematical operations are required: i) the multiplication of *x* and *y* and ii) one base 2 exponentiation. *z* represents the result of *x*y*. Both x and *y* are fixed-point numbers, so, the product result (*z*) will be also a fixed-point number. The fixed-point value of *z* is made up of integer *i* and fraction *f* part that can be represented as follows;

$$z = i + f \tag{4}$$

Fig. 1 shows the 32-bit fixed-point number format used in the proposed hardware implementation of the exponent function.

Based on Eq. (4), Eq. (3) can be updated as follows;

$$e^x = 2^z \tag{5}$$

Eq. (5) can be further updated as Eq. (6) by using Eq. (4) as follows;

$$e^x = 2^{i+f} \tag{6}$$

Eq. (6) can be rewritten as the following equation;

$$2^z = 2^i * 2^f \tag{7}$$

Based on Eq. (7), it can be seen that the exponent function has been reduced to the two mathematical operations i.e., the calculation of base 2's power for integer and fraction part, and one multiplication operation. The left shift operation can be used to perform efficiently in the hardware design for integer part *i*. Now, the calculation of base 2's power with fraction *f* is the only complex computation part. In general, the complexity of the hardware

Table I. Performance evaluation on FPGA and ASIC

FPGA		ASIC	
LUTs	1311	No. of gates	2691
Registers	188	Frequency	200 MHz
Frequency	115.7 MHz	Power Consumption	6.1 mW
Power Consumption	242 mW	RAM	0
Throughput	411.3 Mbits/s	Throughput	711.11 Mbps
Throughput/LUT	0.31 Mbps/LUT	Area (Whole Chip)	1.9 mm^2

Table II. Comparison of ASIC implementation

	Proposed	Original Implementation [10]	Optimized Implementation [10]
Technology	90nm TSMC	65nm STM	65nm STM
Precision	32-bit fixed point	16-bit fixed point	16-bit fixed point
Area (mm^2)	1.9	0.02*	0.02*
Power Consumption	6.1 mW	60 µW	58 µW
Max Frequency	200 MHz	20.1 MHz	24.8 MHz

* Reported area is the cell area after compilation in Design Compiler. It does not include any results of IC layout.

unit for 2^f depends upon the desired accuracy range. As described earlier, one of the main aims of this hardware implementation is to perform an area-efficient implementation. So, by limiting the accuracy range for 2^f, an area-efficient design can be achieved. In this work, the fraction part *f* is limited in the range of -0.5 to +0.5. We used the minimax approach to calculate the polynomials for the fraction part and third-order polynomials were calculated in Mathematica software. The polynomials have the values of [0.99992807, 0.69326098, 0.24261112, 0.00517166]; which results in the following third order equation;

$$2^x = a + bx + cx^2 + dx^3 \tag{8}$$

where a=0.99992807, b=0.69326098, c=0.24261112, and d=0.00517166.

The fixed-point implementation leads to a maximum relative error of $1e^{-3}$ and maximum absolute error of $6.42e^{-5}$ for values in the range of [-6, 15] as compared to the floating-point implementation of e^x. In Digital Signal Processing (DSP) applications, the third-order polynomials have a small error that can be neglected. The polynomials achieved through the minimax approach are stored in four registers of 32-bits. So, the proposed method does not need to store a large truth table for multiple polynomials. Figure 2 shows the error plot showing the difference between fixed- and floating-point exponent values for the values in the range of [-16, 15]. In order to show the difference of the values between fixed- and floating-point exponent values, floating-point values are scaled by multiplication to 1.1 because without scaling, the value difference is quite small that cannot be displayed in the graph in Fig. 2.

Figure 3 shows the proposed hardware design for the exponent unit based on the proposed simplified exponent

Table III. Comparison of FPGA implementation

	Proposed	[7]	[5]	[11]	[12]
Method	2's Exponent	Basic-Split	Power Series	CORDIC	Taylor Series
Input Range	[-6,15]	[-10, 10]	[-2, 2]	[-9.5, 9.5]	NA
Max. Abs. Error	$6.42e^{-5}$	NA	$8.9e^{-7}$	NA	NA
Max. Rel. Error	0.001	NA	NA	NA	NA
RMSE*	0.33	NA	NA	NA	0.61
FPGA	Virtex4	Virtex4	Virtex4	Spartan6	Virtex4
Precision	32-bit fixed point	32-bit fixed point	32-bit floating point	32-bit fixed point	64-bit fixed point
Area	LUT: 1311 FF: 188	LUT: 21526 FF: 13755	LUT: 2615 Slices: 1350	LUT: 5507 FF: 449	LUT: 13,614 FF: 18,704
Max Frequency	115.7 MHz	100.7 MHz	66.82 MHz	177.78 MHz	200 MHz
Throughput	411.3 Mbits/s	NA	142.6 Mbits/s	437.61 Mbits/s	426.66 Mbits/s

*: Root Mean Square Error

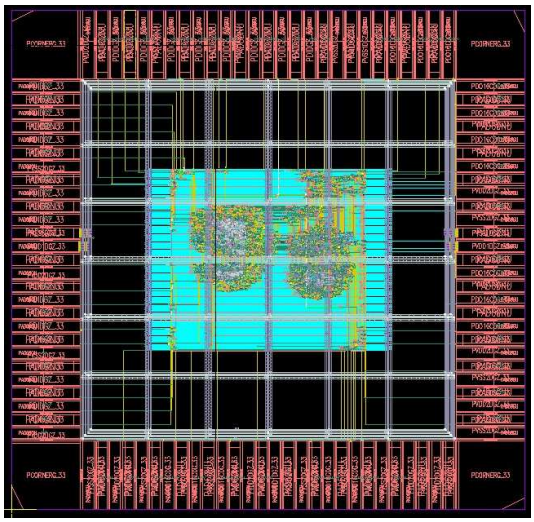

Fig. 4. IC layout of the proposed exponent design

calculation method. The main aim of the exponent calculation method is to eliminate the need of the memory requirement in the hardware design, so, the proposed hardware design does not utilize any memory during computation of the exponent function. In order to achieve the high-speed design, pipelining has been applied in the hardware design to reduce the critical path. The proposed design utilizes only shift operations, one 32-bit adder, and one 32-bit multiplier for exponent calculation.

III. Implementation Results

The proposed method has been implemented in Verilog HDL and for evaluation purposes, the design has been synthesized and evaluated for two different hardware platforms: FPGA and ASIC. For FPGA implementation, the proposed design has been synthesized for Xilinx Virtex-4 FPGA and it achieves 115.7 MHz of working frequency while the design utilizes only 1311 LUTs. Table I shows the implementation evaluation of the proposed hardware design on Virtex-4 (12SF363) FPGA. It should be noted here that the proposed design does not utilize any storage memory during performing multiple calculations, so, no BRAM in FPGA has been utilized. The proposed hardware implementation in FPGA is able to achieve a throughput of 411.3 Mbps. Moreover, the design consumes only 188 registers and 242 mW of power consumption.

The throughput is calculated as following in this paper;

$$Throughput = \frac{B}{C} \times F \tag{9}$$

where B represents the number of bits of the input number, C is the number of clock cycles required to process one input number, and F is the maximum frequency achieved by the hardware design.

The proposed hardware design of the exponent unit has been implemented in ASIC also. The design has been synthesized for TSMC 90nm technology process node. The results reported in Table I for ASIC design are achieved after the layout design in Synopsys IC Compiler. Moreover, the verification of the design has been performed by performing the gate-level simulation. The number of gates reported in Table I are calculated using Nand-2 cell area from TSMC 90nm technology process. Figure 4 shows the ASIC layout of the proposed hardware design. The proposed design can achieve a working frequency of 200 MHz while the power consumption is only 6.1 mW. The power consumption results reported in this paper for ASIC design include the power consumption of the whole chip i.e., Input/Output (I/O) pads and core. For power consumption results calculation, standard input is used i.e., multiple values of input data are passed to the hardware design, and the switching activity file is recorded and analyzed for accurate power consumption results.

To compare with other exponent function hardware implementations, we have selected the work of Nilsson *et al.* [10] for ASIC implementation and Sun *et al.* [7], Rekha and Menon [11], Wielgosz *et al*. [12], and Peter [5] for FPGA implementation. Table II shows the comparison of the proposed ASIC implementation with Nilsson *et al.* [10]. The work done in [10] utilizes Taylor series expansion to calculate exponent value which requires a large storage memory as well computation resources. The proposed implementation has double precision and the proposed design can achieve 8x-10x higher working frequency as compared to the work of [10]. However, the proposed design has higher power consumption and a larger area as compared to [10]. The high power consumption is because the proposed design has a high working frequency and power consumption results include the power consumption of the whole chip including core and I/O pads. The power results reported in [10] are after Design Compiler synthesis and do not include power consumption results of I/O pads, clock network, etc. The area of the proposed design is larger as compared to the hardware design proposed by Nilsson *et al.* [10] because the proposed design includes the area of I/O pads along with the core area. Moreover, in Synopsys IC

compiler, the core area cannot become smaller than 0.27 mm^2, so, it is not possible to achieve 0.02 mm^2 (reported in [10]) of the core area during the layout process in Synopsys IC Compiler. I/O pads also increase chip area which is also not reported in the hardware implementation by Nilsson *et al.* [10].

In order to compare FPGA implementation results, post-synthesis results are compared with the multiple previous works. For a fair comparison with other works, the design is synthesized and routed for Virtex-4 (12SF363) FPGA. Table III includes details of different performance parameters comparison with other works. Moreover, the input range supported by each work is also included in Table III. There is no comparison of absolute and relative error with the work of [7] because the absolute error provided in [7] is for softmax function and no error results are included for exponent functions.

A hardware implementation of softmax function is proposed in [7]. Softmax function is made up of exponent function along with other computational operations, so, the authors in [7] have included the implementation results of only exponent unit implementation as well in the paper. So, in this paper, we compare the results of only the exponent module from [7]. Softmax architecture results are not reported in this paper. It is evident from Table III that the proposed hardware design uses 16.4x fewer LUTs and 73.1x fewer Flip Flops (FF) as compared to [7]. The high LUT count in [7] can be attributed to two reasons: i) high memory usage as Basic-Split method needs to store 320 bytes of data in Read-Only Memory (ROM), and ii) a large number of arithmetic functions i.e., four multipliers, one division, and adder is used. Moreover, we have compared the proposed implementation with a floating-point hardware implementation based on power series [5]. Power series-based hardware implementation [5] used only one floating-point adder and two floating-point multipliers without storing any data in ROM or Random Access Memory (RAM). However, the design of [5] uses more LUTs and FFs than the proposed hardware design. Moreover, the proposed hardware design can achieve a higher working frequency than [5] leading to higher throughput as well. It can be noted from Table III that Basic-Split method (fixed-point) [7] uses almost 9x-10x more resources than Power Series method (floating-point) [5] which can be attributed to the usage of ROM and more arithmetic units as compared to the hardware implementation of [5]. CORDIC based-implementation [11] achieves higher throughput than the proposed design, however, CORDIC algorithm uses more LUTs and FFs. The proposed design has 4.2x smaller LUT count than [11] where throughput only lags by 6% as compared to CORDIC implementation. The look-up table based hardware implementation for Taylor series [12] uses 10.3x and 99.4x more LUT and FF resources, respectively, as compared to the proposed design.

IV. Conclusion

The exponent function is an important mathematical function used in digital signal processing applications as well as many AI algorithms. It is desired to achieve a fast and area-efficient implementation of exponent function due to the computational complexity of exponent function. In this paper, we propose an area-efficient and fast implementation of the exponent function and it has been synthesized for FPGA and ASIC platforms. The proposed hardware design eliminates the memory requirements due to the simple computation algorithm. The hardware design can achieve 200 MHz and 115.7 MHz of the working frequency for ASIC and FPGA implementation, respectively. The pipelined design reduces the critical path of hardware design which helps to achieve the throughput of 411.3 Mbps for FPGA platform and 711.11 Mbps for ASIC platform. The power consumption of the proposed hardware design is 242mW and 6.1 mW for FPGA and ASIC platforms, respectively. The proposed hardware implementation has low area while working at high frequency and low power consumption as compared to multiple recent works. The proposed exponent function hardware design can be combined with other systems where exponent computations are required especially in the field of AI and digital signal processing.

References

[1] J. Qiu *et al.*, "Going Deeper with Embedded FPGA Platform for Convolutional Neural Network," pp. 26–35, 2016, doi: 10.1145/2847263.2847265.

[2] R. C. Staudemeyer and E. R. Morris, "Understanding LSTM -- a tutorial into Long Short-Term Memory Recurrent Neural Networks," *arXiv*, Sep. 2019, Accessed: Dec. 22, 2020. [Online]. Available: http://arxiv.org/abs/1909.09586.

[3] J. Zhou *et al.*, "Graph Neural Networks: A Review of Methods and Applications," *AI Open*, vol. 1, pp. 57–81, Dec. 2020, Accessed: Jun. 06, 2021. [Online]. Available: http://arxiv.org/abs/1812.08434.

[4] M. Zhang, S. Vassiliadis, and J. G. Delgado-Frias, "Sigmoid generators for neural computing using piecewise approximations," *IEEE Trans. Comput.*, vol. 45, no. 9, pp. 1045–1049, 1996, doi: 10.1109/12.537127.

[5] M. Peter, "High throughput floating point exponential function implemented in FPGA | IEEE Conference Publication | IEEE Xplore," Jul. 2015, Accessed: Jun. 05, 2021. [Online]. Available: https://ieeexplore.ieee.org/document/7309545.

[6] Y. Gao, W. Liu, and F. Lombardi, "Design and Implementation of an Approximate Softmax Layer for Deep Neural Networks," Sep. 2020, pp. 1–5, doi: 10.1109/iscas45731.2020.9180870.

[7] Q. Sun *et al.*, "A High Speed SoftMax VLSI Architecture Based on Basic-Split," Dec. 2018, doi: 10.1109/ICSICT.2018.8565706.

[8] M. A. Hussain and T. H. Tsai, "Memory Access Optimization for On-Chip Transfer Learning," *IEEE Trans. Circuits Syst. I Regul. Pap.*, vol. 68, no. 4, pp. 1507–1519, Feb. 2021, doi: 10.1109/TCSI.2021.3055281.

[9] V. Sze, "Designing Hardware for Machine Learning: The Important Role Played by Circuit Designers," *IEEE Solid-State Circuits Mag.*, vol. 9, no. 4, pp. 46–54, 2017, doi: 10.1109/MSSC.2017.2745798.

[10] P. Nilsson, A. U. R. Shaik, R. Gangarajaiah, and E. Hertz, "Hardware implementation of the exponential function using Taylor series," Jan. 2015, doi: 10.1109/NORCHIP.2014.7004740.

[11] R. Rekha and K. P. Menon, "FPGA implementation of exponential function using cordic IP core for extended input range," in *2018 3rd IEEE International Conference on Recent Trends in Electronics, Information and Communication Technology, RTEICT 2018 - Proceedings*, May 2018, pp. 597–600, doi: 10.1109/RTEICT42901.2018.9012611.

[12] M. Wielgosz, E. Jamro, and K. Wiatr, "Highly Efficient Structure of 64-Bit Exponential Function Implemented in FPGAs," in *International Workshop on Applied Reconfigurable Computing*, 2008, vol. 4943 LNCS, pp. 274–279, doi: 10.1007/978-3-540-78610-8_28.