A fixed point exponential function accelerator for a neuromorphic many-core system

DOI:
10.1109/ISCAS.2017.8050528

Document Version
Accepted author manuscript

A Fixed Point Exponential Function Accelerator for a Neuromorphic Many-Core System

Johannes Partzsch, Sebastian Höppner, Matthias Eberlein,
Rene Schüffny, Christian Mayr
Chair for Highly-Parallel VLSI Systems
and Neuromorphic Circuits,
Technische Universität Dresden, 01062 Dresden, Germany.
Email: johannes.partzsch@tu-dresden.de

David R. Lester, Steve Furber
School of Computer Science,
University of Manchester,
Oxford Road, Manchester M13 9PL, UK.
Email: david.r.lester@manchester.ac.uk

***Abstract*—Many models of spiking neural networks heavily rely on exponential waveforms. On neuromorphic multiprocessor systems like SpiNNaker, they have to be approximated by dedicated algorithms, often dominating the processing load. Here we present a processor extension for fast calculation of exponentials, aimed at integration in the next-generation SpiNNaker system. Our implementation achieves single-LSB precision in a 32bit fixed-point format and 250Mexp/s throughput at 0.44nJ/exp for nominal supply (1.0V), or 0.21nJ/exp at 0.7V supply and 77Mexp/s, demonstrating a throughput multiplication of almost 50 and 98% energy reduction at 2% area overhead per processor on a 28nm CMOS chip.**

***Keywords*—MPSoC, neuromorphic computing, SpiNNaker, exponential function**

I. Introduction

In models of neural processing, one of the main mathematical primitives encountered is the exponential function. Examples include the shape of the postsynaptic current, activation function of ion channels, time courses of short term depression, or spike-time-dependent plasticity (STDP) [1]. When emulating these models in neuromorphic circuits, a circuit implementation of the exponential function has to be found. Traditionally, the subthreshold voltage-current dependence of CMOS transistors has been used to generate an exponential current waveform based on a linear (saw-tooth) voltage [2]. The OTA-C approach uses a transconductance to emulate an RC decay [3], [4]. In deep submicron technologies, solutions have to be found that rely less on analog performance, such as a switched capacitor charge sharing emulating an exponential decay [5]. However, the above approaches have four main drawbacks. First, they suffer to varying degrees from deviations due to the inevitable fixed-pattern noise [6]. Second, due to the analog, non-programmable nature of the implementation, the model behaviour can usually only be configured in a narrow range. Third, they are limited by leakage to maximum time constants on the order of seconds [7], insufficient for various plasticity models [1]. Fourth, as analog circuits, they do not scale to ultra-deep submicron technologies and therefore cannot take advantage of the high packing densities and low power operation offered by these technologies. In contrast, the SpiNNaker digital neuromorphic approach [8] uses many-core processor systems to execute software programs that emulate neuroscience models. This achieves unlimited time constants plus flexibility and reliability. However, the exponential calculation on the current SpiNNaker system is carried out in software, making it several orders of magnitude less energy efficient than analog realizations. In this paper, we present a digital exponential function accelerator that is integrated in a prototype chip of the 2nd generation SpiNNaker system. It combines scalability to ultra-deep submicron, unlimited time constants, flexible programmable operation with an energy consumption competitive to analog approaches.

II. Background

In neuromorphic MPSoCs, spiking neural networks are emulated by calculating on each processor a set of model equations. SpiNNaker [8] is the first system to exploit this approach, modifying and extending the MPSoC infrastructure (for details, see [9]). One SpiNNaker chip includes 18 ARM cores, and together with 128MByte external DRAM consumes 1W, being significantly more energy efficient than standard high-performance computing. Exponentials are expensive operations on standard processors, as they have to be approximated via tens of basic mathematical operations, and often dominate processing load. Thus, strategies have been developed for avoiding or coarsely approximating exponentials in spiking neural network simulation and hardware [10], [11], [12]. In fixed time step simulations, exponential decays can be realized by a multiplication per time step. However, state variables have to be accessed in each time step, which can become a bottleneck when fetching them from external DRAM. Alternatively, exponentials can be stored in look-up tables [10]. However, this is efficient only if a limited number of different values has to be evaluated repeatedly. The above solutions are restricted to special cases or compromise accuracy. Thus, a hardware accelerator for exponential calculation is an attractive option for boosting performance in the 2nd generation SpiNNaker system. A prototype chip for this purpose has been implemented, following the structure shown in Fig. 1. Each of the four processing elements (PEs) includes one exponential accelerator, whose design is described next.

III. Exponential Algorithm

The exponential accelerator should be easy to integrate in existing software implementations. Therefore, we use the signed fixed-point s16.15 number format, which has been widely used in the SpiNNaker software framework.

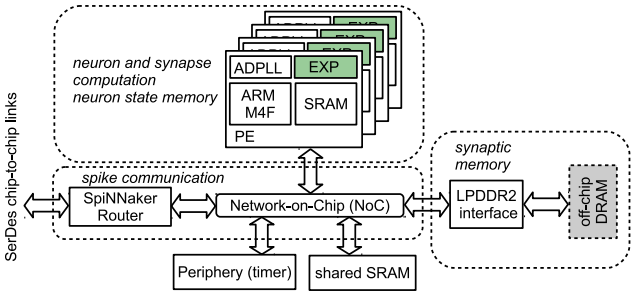

Fig. 1. Integration of the exponential accelerator

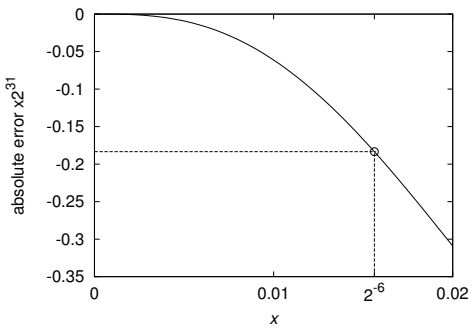

Fig. 2. Absolute error of employed polynomial approximation

From the wide range of available algorithms for exponential calculation, we chose a hybrid algorithm, combining two LUTs with a polynomial approximation [13], utilizing the self-similarity of the exponential function:

$$y = \exp(x) = \underbrace{\exp(n)}_{f_{\mathrm{int}}(n)} \cdot \underbrace{\exp(p)}_{f_{\mathrm{frac}}(p)} \cdot \underbrace{\exp(q)}_{f_{\mathrm{poly}}(q)} \text{ with } x = n + p + q \tag{1}$$

The integer part of the operand is represented by n, whereas the fractional digits are split into an upper part p and a lower part q. The two functions $f_{\mathrm{int}}(n)$ and $f_{\mathrm{frac}}(p)$ are evaluated via two separate LUTs, whereas the function $f_{\mathrm{poly}}(q)$ represents a polynomial.

There are several arguments for this hybrid approach. First, the integer LUT utilizes the amplifying effect of the exponential function in conjunction with the limited dynamic range of the employed fixed-point s16.15 format: Only a limited range of operands x produce a valid result y in the supported range. The maximum representable value $2^{16} - 1 \approx \exp(11.1)$ limits meaningful x at positive numbers, whereas the minimum representable value $2^{-15} \approx \exp(-10.4)$ restricts x at negative numbers. All operands outside this range evaluate either in an overflow or a result of $y = 0$, and can be filtered out beforehand. In effect, the integer LUT needs only 23 entries. The fractional look-up is introduced to restrict the operands range of the polynomial approximation. As a result, the required order of the polynomial is reduced compared to a full polynomial approximation of the fractional part at the same accuracy level. The trade-off here is to make the fractional LUT small enough, so that the savings due to the reduced polynomial order dominate over the effort for the LUT and the additional multiplication. The resulting split into two LUTs significantly reduces the total memory size compared to a combined LUT.

Overall, the look-up and polynomial evaluation can be well parallelized in hardware, which results in a short latency, especially compared to iterative algorithms. This advantage comes at the price of several multiplications for polynomial evaluation and calculation of the final result. However, due to the limited operand range of the polynomial, most multipliers can be designed with relatively low bit widths, which reduces the overall hardware effort. This will be discussed next.

IV. Implementation

The exponential accelerator should be accurate to one LSB for all operands to not compromise on accuracy compared to a software implementation. In worst-case (high operand values), all 32 digits except the sign bit are filled in the result. Thus, all components have to have at most an absolute error of 2^{-31}. To fulfill this requirement, we chose a 6bit fractional LUT for f_{frac}, which restricts f_{poly} to a 4th-order polynomial on the interval $0 \leq q < 2^{-6}$. Lower LUT bit widths would require higher-order polynomials with correspondingly higher effort. Starting from a Taylor expansion, the polynomial coefficients were optimized for low number of valid digits in binary notation, to later simplify multiplication with these coefficients. This resulted in:

$$\begin{aligned} f_{\mathrm{poly}}(q) &= 1 + q + \frac{1}{2} \cdot q^2 + c_3 \cdot q^3 + c_4 \cdot q^4 \text{ , with} \\ c_3 &= 2^{-3} + 2^{-5} + 2^{-7} + 2^{-9} + 2^{-11} + 2^{-13} \\ c_4 &= 2^{-5} + 2^{-7} + 2^{-8} \end{aligned} \tag{2}$$

The coefficient $1/2$ is realized by a constant shift, while c_3 and c_4 have only 6 and 3 valid digits, respectively, effectively replacing two multiplications by a few additions. Figure 2 shows the absolute error of the polynomial. For the envisaged interval, it stays well below the required accuracy of 2^{-31}, which leaves headroom for rounding errors and quantization of intermediate results.

The structure of the complete implementation is shown in Fig. 3. Multiplication of the fractional LUT result was interleaved with the polynomial evaluation. This results in three more multipliers. However, their bit widths are greatly reduced, which relaxes timing and makes the implementation more suited for tight timing requirements. Only one multiplication remains with two wide operands of 32bit or more, whose operation could easily be split over more than one clock cycle to meet increased clock speed requirements.

Due to its structure, the implementation can be operated as a pipeline, which is made accessible from outside by an output FIFO with four entries. Integration with the ARM-M4F processor is realized via the AMBA high-performance bus (AHB), see Fig. 4. Calculation of one exponential requires one bus write and read operation, resulting in a minimum of 2 clock cycles per exponential in pipelined operation, and a total latency of 6 clock cycles for single operation. The bus interface makes the exponential accelerator easily portable to other processors and also simplifies software integration via memory-mapped access, i.e. write and subsequent read of one memory address. The read operation is stalled if the output FIFO is empty and a value is still in the calculation pipeline, making timing of access completely transparent to the user.

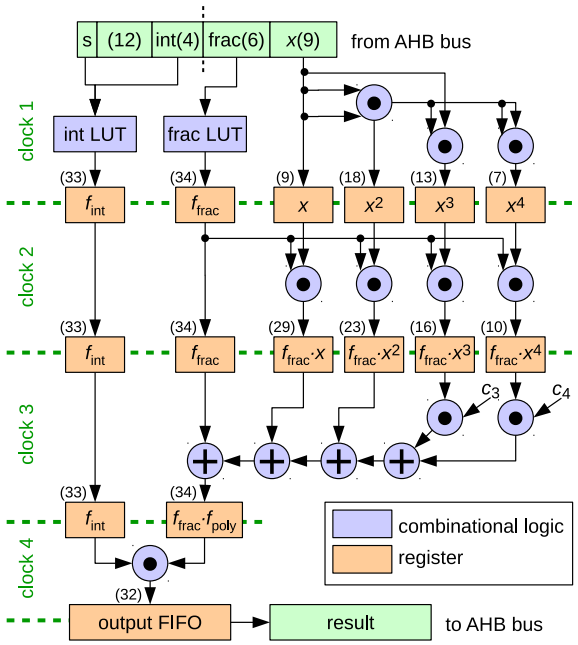

Fig. 3. Overview of exponential accelerator implementation. Numbers in brackets denote bit widths.

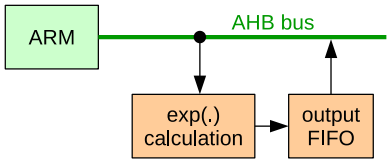

Fig. 4. Integration of exponential accelerator unit

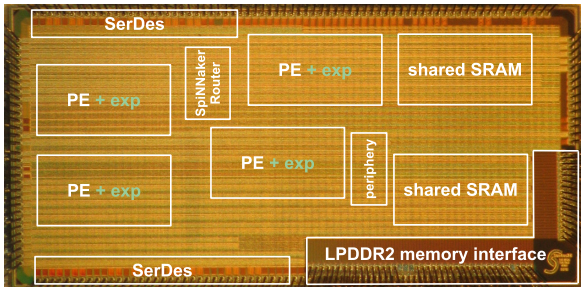

Fig. 5. Photograph of the testchip

V. Results

The prototype chip of Fig. 1 has been implemented and manufactured in GLOBALFOUNDRIES 28nm SLP CMOS technology. Its photograph is shown in Fig. 5. The exponential accelerator is included in each of the four PEs.

We evaluated the performance of the exponential accelerator with respect to accuracy, speed and energy per operation. Accuracy was assessed via comparison to the double-precision floating point exponential of the C standard libary. The resulting sweep over all operands is shown in Fig. 6. The absolute error is always below one LSB, meaning that the result of the exponential accelerator is always one of the two neighbouring fixed-point values of the double-precision reference.

Figure 7 shows measurement results for the energy per

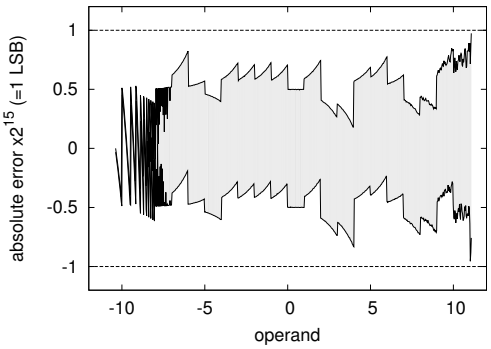

Fig. 6. Absolute error of exponential accelerator; sweep over all operands inside dynamic output range (operands ranging from -10.4 to 11.1)

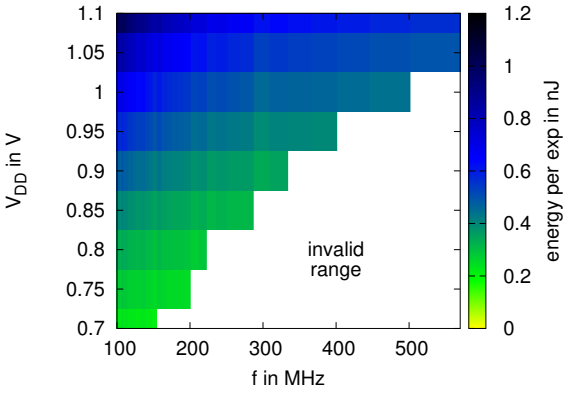

Fig. 7. Measured total energy per exponential with the accelerator. The white area denotes non-operational voltage/frequency combinations.

exponential for different operating voltages and frequencies. The values were extracted from measurements of total PE power when running the exponential accelerator at maximum throughput in pipelined operation (2 clock cycles per exponential, see Sec. IV), so power consumption of the processor and its local memory is included like in a real application scenario. At nominal operating conditions of 1.0V and 500MHz clock frequency, a value of 0.44nJ/exp at a throughput of 250Mexp/s is achieved. For a reduced supply voltage of 0.7V and 154MHz clock frequency (maximally achievable at this voltage level), energy goes down to 0.21nJ/exp at 77Mexp/s.

We compared the exponential accelerator with a software version of the same algorithm, running on the ARM-M4F on the testchip. It typically requires 95 clock cycles per exponential. Correspondingly, measured energy per exponential is significantly higher, as shown by the comparison in Fig. 8. The exponential accelerator requires only 1.8% of the software implementation for both nominal operating conditions (1.0V/500MHz) and reduced supply voltage (0.7V/154MHz).

The main characteristics of the exponential accelerator and its corresponding software version are summarized in Table I. The additional area for the accelerator block makes up for approx. 2% of the area per processing element.

Table II compares our exponential accelerator to other hardware implementations. The work in [14] achieves higher energy efficiency at lower resolution and speed. However, their

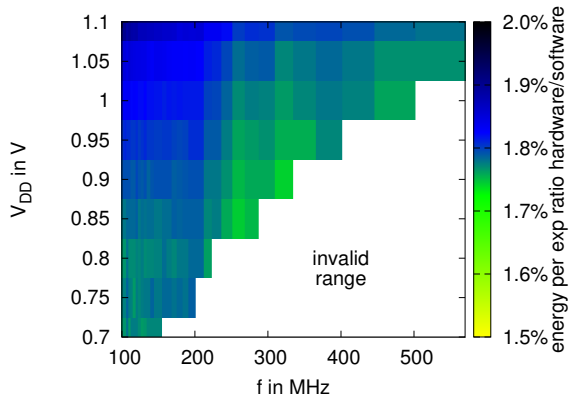

Fig. 8. Ratio of measured energies per exponential, accelerator implementation to software implementation

Measure	exp accelerator	software exp
Throughput @500MHz	250Mexp/s	5.3Mexp/s
Time per exp, pipelined	2clks/exp	95clks/exp
Latency	6clks/exp	95clks/exp
Energy per exp (nominal)	0.44nJ/exp	25nJ/exp
Energy per exp (0.7V/154MHz)	0.21nJ/exp	12nJ/exp
Total area	10800μm^2	-

TABLE I. SUMMARY OF EXPONENTIAL ACCELERATOR IMPLEMENTATION AND SOFTWARE IMPLEMENTATION

Measure	This work	[14]	[15]	[13]
Throughput, pipeline	250Mexp/s	24.8Mexp/s	5.9Mexp/s	230Mexp/s
Throughput, single	83Mexp/s	24.8Mexp/s	5.9Mexp/s	10Mexp/s
Technology/Platform	28nm	65nm	FPGA	FPGA
Total area	10800μm^2	20700μm^{2**}	-	-
Data format	fixed	fixed	float	float
Rel. accuracy	31bit	15bit	21bit***	55bit
Energy per exp	0.44nJ/exp*	0.096nJ/exp**	-	-

*total system power **only Synthesis results available ***checked only on range -2..2

TABLE II. COMPARISON WITH OTHER HARDWARE IMPLEMENTATIONS

power figure is only estimated after Synthesis and done for the exponential block exclusively, whereas our value measures power spent for the whole processing element. In contrast to our work, the FPGA implementations of [13] and [15] both target floating point values. The implementation of [13] achieves almost the same throughput and higher accuracy. However, this comes at the price of a significantly higher pipeline latency and a big total LUT size of 144kByte (compared to 404Byte in our work). The floating-point representation significantly extends the numerical range compared to our fixed-point implementation, which may be a necessity in certain use cases or result in less effort to be put into normalization. Our fixed-point implementation could still be utilized in this case for calculating the mantissa, while the exponent would require a few additional operations in the processor. This would significantly speed up calculation of floating-point exponentials as well.

VI. CONCLUSION

This work presents a hardware accelerator for exponential function calculation, targeted for integration in the next-generation SpiNNaker neuromorphic MPSoC. It achieves single-LSB precision for the employed s16.15 fixed-point format. Measurements on a prototype chip demonstrate a maximum throughput of 250Mexp/s at a total energy of 0.44nJ per exponential. Scaling of supply voltage to 0.7V reduces this value to 0.21nJ/exp, outperforming a software implementation by a factor of 55 at a speed-up of almost 50. With these characteristics, the exponential accelerator will help in speeding up event-based computations of a wide range of neuron and synapse models.

ACKNOWLEDGMENT

This work was supported by the European Union under Grant Agreements No. 604102 and DLV-720270 (Human Brain Project), No. 692519 (PRIME), ERC Advanced Grant BIMPC (320689), EPSRC grant BIMPA (EP/G015740) and the Center for Advancing Electronics Dresden (cfaed). The authors thank ARM and Synopsis for IP and the Vodafone Chair at Technische Universität Dresden for contributions to RTL design.

REFERENCES

[1] C. Koch, *Biophysics of computation: information processing in single neurons*. Oxford university press, 2004.

[2] C. Bartolozzi and G. Indiveri, "Synaptic Dynamics in Analog VLSI," *Neural Computation*, vol. 19, no. 10, pp. 2581–2603, 2007.

[3] M. Noack, J. Partzsch, C. Mayr, and R. Schüffny, "Biology-derived synaptic dynamics and optimized system architecture for neuromorphic hardware," in *International Conference on Mixed Design of Integrated Circuits and Systems (MIXDES)*, 2010, pp. 219–224.

[4] T. Koickal, A. Hamilton, S. Tan, J. Covington, J. Gardner, and T. Pearce, "Analog VLSI circuit implementation of an adaptive neuromorphic olfaction chip," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 54, no. 1, pp. 60–73, 2007.

[5] M. Noack, J. Partzsch, C. Mayr, S. Hänzsche, S. Scholze, S. Höppner, G. Ellguth, and R. Schüffny, "Switched-capacitor realization of presynaptic short-term-plasticity and stop-learning synapses in 28 nm CMOS," *Frontiers in Neuroscience*, vol. 9, 2015.

[6] S. Henker, C. Mayr, J.-U. Schlüßler, R. Schüffny, U. Ramacher, and A. Heittmann, "Active pixel sensor arrays in 90/65nm CMOS-technologies with vertically stacked photodiodes," in *IEEE International Image Sensor Workshop (IIS)*, 2007, pp. 16–19.

[7] C. Mayr, J. Partzsch, M. Noack, S. Hänzsche, S. Scholze, S. Höppner, G. Ellguth, and R. Schüffny, "A biological real time neuromorphic system in 28nm CMOS using low leakage switched capacitor circuits," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 10, pp. 243–254, 2015.

[8] S. Furber, F. Galluppi, S. Temple, and L. Plana, "The SpiNNaker project," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 652–665, 2014.

[9] S. Furber, D. Lester, L. Plana, J. Garside, E. Painkras, S. Temple, and A. Brown, "Overview of the SpiNNaker system architecture," *IEEE Transactions on Computers*, vol. 62, no. 12, pp. 2454–2467, 2013.

[10] B. Vogginger, R. Schüffny, A. Lansner, L. Cederstroem, J. Partzsch, and S. Höppner, "Reducing the computational footprint for real-time BCPNN learning," *Frontiers in Neuroscience*, vol. 9, p. 2, 2015.

[11] A. Cassidy, A. G. Andreou, and J. Georgiou, "A combinational digital logic approach to STDP," in *IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2011, pp. 673–676.

[12] N. Schraudolph, "A fast, compact approximation of the exponential function," *Neural Computation*, vol. 11, pp. 853–862, 1999.

[13] W. Yuan and Z. Xu, "FPGA based implementation of low-latency floating-point exponential function," in *International Conference on Smart and Sustainable City (ICSSC)*, 2013.

[14] P. Nilsson, A. Shaik, R. Gangarajaiah, and E. Hertz, "Hardware implementation of the exponential function using taylor series," in *Norchip*, 2014.

[15] P. Malik, "High throughput floating point exponential function implemented in FPGA," in *IEEE Computer Society Annual Symposium on VLSI*, 2015.