Approximate Fixed-Point Elementary Function Accelerator for the SpiNNaker-2 Neuromorphic Chip

Mantas Mikaitis, David R Lester, Delong Shang,
Steve Furber, Gengting Liu, Jim Garside
Advanced Processor Technologies Group
School of Computer Science
The University of Manchester

Stefan Scholze, Sebastian Höppner,
Andreas Dixius
Chair for Highly-Parallel VLSI Systems
and Neuromorphic Circuits
Technische Universität Dresden

***Abstract*—Neuromorphic chips are used to model biologically inspired Spiking-Neural-Networks(SNNs) where most models are based on differential equations. Equations for most SNN algorithms usually contain variables with one or more e^x components. SpiNNaker is a digital neuromorphic chip that has so far been using pre-calculated look-up tables for exponential function. However this approach is limited because the memory requirements grow as more complex neural models are developed. To save already limited memory resources in the next generation SpiNNaker chip, we are including a fast exponential function in the silicon. In this paper we analyse iterative algorithms for elementary functions and show how to build a single hardware accelerator for exp and natural log, for a neuromorphic chip prototype, to be manufactured in a $22\,\mathrm{nm}$ FDSOI process. We present the accelerator that has algorithmic level approximation control, allowing it to trade precision for latency and energy efficiency. As an addition to neuromorphic chip application, we provide analysis of a parameterized elementary function unit that can be tailored for other systems with different power, area, accuracy and latency constraints.**

***Index Terms*—exponential function, logarithm function, hardware accelerators, approximate arithmetic, fixed-point arithmetic, SpiNNaker2, neuromorphic computing, MPSoC**

1. Introduction

SpiNNaker is a digital neuromorphic architecture designed for simulating biologically inspired neural networks called Spiking-Neural-Networks [1]. At the heart of SpiNNaker is a low-power general purpose ARM processor and therefore all the equations of neuronal models are mapped directly onto the machine using basic ARM instructions, contrary to the *analog neuromorphic chips* [2] or *hybrid analog-digital chips* [3] where most models are realised physically. One of the most common functions in SNNs is e^x, used to model exponentially decaying quantities. Most neuron models [4] and biological learning, Spike-Timing-Dependent plasticity, rules (further called STDP) [5] developed on SpiNNaker all use exponentially decaying elements. In software, it is hard to realise a very fast implementation of this function - it requires large look-up tables and multiplication operations and the usual latency is *60-100* clock cycles [6]. Therefore it becomes justifiable to dedicate silicon for the exponential function in a digital neuromorphic chip. However, noting that each new design costs in excess of €250k to fabricate we have a limited number of trials to find the best solution.

In the SpiNNaker-1 system there was no hardware support for transcendental functions, including exponentials, so the models that were developed used pre-computed look-up-tables (LUTs); see [7] for a general analysis of this approach. The SpiNNaker compiler would first take a high level description of the network dynamics specified by the user and pre-calculate a range of values of exponential decay for a specific time constant. Then, the LUTs would be copied into each core's local memory and used while the application is running. However, this approach has two limitations: 1) A limited number of time constants and range of exponential functions can be used due to limited on-chip memory, and 2) in case someone wants to model a learning rule that requires timing constants to be dynamic, the regeneration of look-up-tables on-the-fly or pausing simulation would be required, both major performance bottlenecks. A software exponential is also available in the SpiNNaker software library, but with the latency of 95 clock cycles, it would be a major limit to real-time synaptic plasticity processing, where a single pair of spikes takes approximately 30 cycles as reported in [8]. Indeed when processing most learning rules, we usually need more than one exponential per spike pair processing. Learning rules requiring 3 or more calls to exp already appear in computational neuroscience literature and on SpiNNaker: *e.g* voltage-dependent STDP implemented on SpiNNaker [9], BCPNN [5], [7] and neuromodulated STDP [10] learning rules. Lastly, similar neural network simulations on high performance computers were found to spend $7\,\%$ of total time running soft-exp [14].

The most recent SpiNNaker-2 chip prototype has a fully pipelined exponential built in [11]. However, the implementation is limited to the fixed-point format *s16.15* (Here *s* represents sign bit and the format has 16 integer and 15 fractional bits) and it is not clear how to utilise a fully pipelined

unit in the current plasticity software framework, where multiple other calculations between calls to exponentials are required to be done. Additionally, design in [11] uses identity $e^{a+b+c} = e^a \times e^b \times e^c$ to parallelize computation and therefore requires large multipliers. Here we demonstrate a different implementation based on iterative shift-add algorithms that do not require implementing multiplication. The internal iterative part is done in fixed-point carry-save redundant number representation to reduce the critical path. We have provided two different input and output fixed-point formats which can be mixed to gain more accuracy on some arguments. Furthermore, a useful intrinsic property of the iterative algorithms is that just after a few iterations they already contain the approximate output. We have chosen to use this property to provide accuracy control, following the principles of *approximate computing* [12] (in this case approximation comes not from the errors in the circuit but by not running enough iterations) in order to add options to sacrifice some accuracy to obtain faster and more energy-efficient elementary function. This property will provide a platform for experimenting with concepts arising from the ongoing discussion about the number of bits required for representing weights in STDP [13].

Most of the algorithms for evaluating elementary functions are categorised into two types: *polynomial approximations* or *convergence algorithms* [15], [16]. For this work, we have chosen a well-known convergence algorithm presented in [15] which provides exponential and natural logarithm functions with overlapping hardware components. The unit is included in a prototype neuromorphic chip as an *AHB* slave that can be driven from ARM core by writing and reading the set of specific memory locations, similarly to the implementation demonstrated in [11]. Design synthesis studies are executed on the makeChip hosted design service platform [17] for the GLOBALFOUNDRIES 22FDX technology [18].

2. SpiNNaker neuromorphic chip

In this section we briefly present the SpiNNaker chip - the main building component of a million core computing cluster designed specifically to simulate large scale spiking neural networks. SpiNNaker is a large collection of low power computational nodes that are able to send and receive signals (similar to neuronal spikes in the brain) to/from any other node in the network, and can be programmed to run a piece of code upon receiving each spike in such a way to model biological neurons and the synapses that connect them. The SpiNNaker chip comprises 18 ARM968 processors as well as a block of shared SDRAM of $128\,\mathrm{MB}$. Typically each ARM968 in the SpiNNaker chip will be allocated up to 255 neurons so a single chip can have approximately 4000, where the exact number depends on the complexity of simulation models. An ARM968 contains $64\,\mathrm{kB}$ of data storage memory DTCM (data Tightly-Coupled-Memory) and also $32\,\mathrm{kB}$ of instruction memory, named ITCM (instruction memory). The executable program is constructed according to the high level description provided by user, which takes into account things such as neuron models and learning rule types. The compiled code is then downloaded to ITCM while any data structures, including LUTs for exponential function, that are used while application is running, are stored in DTCM. See [19] for a more detailed review of the architecture and software; Also see [20] for comparison to other neuromorphic systems.

The first prototype chips of the next-generation (SpiNNaker-2) architecture were already manufactured and tested [11], [21]. The core of the SpiNNaker-2 chip is a number of ARM Cortex M4F processing elements (PEs). Each PE has some local memory which will be split for code and data as in SpiNNaker-1 ITCM and DTCM. An off-chip DRAM will be shared among all the cores. Each PE is equipped with a DMA controller to copy data from DRAM to a PE and vice versa. Each SpiNNaker-2 chip has a router used to send spike packets from any core to any other core on the system. It is also used to send data bursts from core-to-DRAM or core-to-core. The accelerator documented in this manuscript is included next to each PE. The first prototype chips with this accelerator included were already designed and are in the process of manufacturing in a $22\,\mathrm{nm}$ FDSOI technology.

3. Algorithm

In this section we will discuss the iterative *shift-add* algorithms presented in Chapter 8 of [15] which can be used to find exponential and natural logarithm functions. To save space, here we only show a variant of the algorithm that computes exponential function, but as shown in [15] logarithm is very similar.

3.1. Iterative algorithm

The main algorithm is based on the convergence principle: the sequences L_n and d_n defined as

$$\begin{aligned} L_0 &= x \\ L_{n+1} &= L_n - ln(1 + d_n 2^{-n}) \end{aligned} \tag{1}$$

$$d_n = \begin{cases} 1 & \text{if } L_n \geq ln(1 + 2^{-n}) \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

Now a sequence E_n is defined such that at any step n of the algorithm,

$$E_{n+1} = E_n + d_n E_n 2^{-n} \tag{3}$$

Then if $L_0 = x$ is in the convergence domain $x \in [0, 1.56202...]$, this gives:

$$\begin{aligned} \lim_{n\to\infty} L_n &= 0 \\ \lim_{n\to\infty} E_n &= E_0 e^{L_0}. \end{aligned} \tag{4}$$

To speed-up this algorithm two additions in equations 1 and 3 are done using carry-save adders. Using carry-save adders requires changing this algorithm in various ways,

TABLE 1. APPROXIMATE MINIMUM AND MAXIMUM RANGES OF VALUES OF *exp* OPERATION WITH DIFFERENT 32 bit 2'S COMPLEMENT FORMATS. * - SATURATES TO 0X0 BELOW THIS RANGE; † - SATURATES TO 0X7FFFFFFF ABOVE THIS RANGE.

Format I/O	Exp IN	Exp OUT
S16.15/S16.15	-10.4^* to $11.1^\dagger$	$0.000031(2^{-15})$ to $65536(2^{16})$
S0.31/S0.31	-1 to $0^\dagger$	0.368 to 1
S16.15/S0.31	-21.488^* to $0^\dagger$	2^{-31} to 1
S0.31/S16.15	-1 to 1	0.368 to e

TABLE 2. APPROXIMATE MINIMUM AND MAXIMUM RANGES OF VALUES OF *ln* OPERATION WITH DIFFERENT 32 bit 2'S COMPLEMENT FORMATS. ‡ - SATURATES TO 0X80000000 BELOW THIS RANGE; † - SATURATES TO 0X7FFFFFFF ABOVE THIS RANGE.

Format	Ln IN	Ln OUT
S16.15/S16.15	$0.000031(2^{-15})$ to $65536(2^{16})$	-10.4 to 11.1
S0.31/S0.31	$0.368^\ddagger$ to 1	-1 to 0
S16.15/S0.31	$0.368^\ddagger$ to $e^\dagger$	-1 to 1
S0.31/S16.15	2^{-31} to 1	-21.488 to 0

including allowing d to have value -1. To save space we skip the detailed explanation and just show our circuits in carry save form. Refer to [15] for more details.

3.2. Range reduction and reconstruction

The algorithm presented in section 3.1 converges only when x is in a limited range. To provide a full range exponential and logarithm function for formats *s16.15* and *s0.31* we must do *range reduction*. This comes in a form of reducing initial x to the convergence range of the iterative algorithm and then reconstructing the result.

3.2.1. Exponential. If x is in the range demonstrated in Table 1, find x' such that:

$$x' = x - n \times ln(2), \tag{5}$$

where

$$n = \lfloor x \times \frac{23}{16} \rfloor. \tag{6}$$

Note that $\frac{23}{16}$ is only around 0.4% smaller than $\frac{1}{ln(2)}$. This gives us, when considering the range of possible arguments in *s16.15* format, $x' \in [\sim -0.0751, \sim 0.7307]$. Then:

$$exp(x) = exp(x' + n \times ln(2)) = 2^n \times exp(x'). \tag{7}$$

Now we can calculate $exp(x')$ using the iterative algorithm and then find the final result in the full range just by shifting n number of places as shown in Equation 7. In total, range reduction for exponential requires only multipliers by constant (Equation 5 and Equation 6) and an adder (Equation 5). Note that for evaluating Equation 7, we will need a shifter with $n \in [-31, 15]$.

3.2.2. Logarithm. If x is in the range demonstrated in Table 2, find $x' \in [\frac{1}{2}, 1]$ such that:

$$x' = \frac{x}{2^k}. \tag{8}$$

If x is represented in fixed-point number system, we can find k by counting leading zeros. When we have x' which is reduced to convergence rate of the algorithm in Section 3.1, we can calculate natural logarithm $ln(x')$. Then we can reconstruct the result in the full range of x:

$$ln(x) = ln(x') + ln(2^k) = ln(x') + k \times ln(2). \tag{9}$$

Therefore, in total range reduction and reconstruction for natural logarithm requires a bidirectional shifter and a *count-leading-zeros* module (Equation 8) as well as multiplier by constant and an adder (Equation 9).

4. Implementation

In this section we demonstrate how to implement the algorithm presented in section 3 in hardware.

4.1. Single iteration

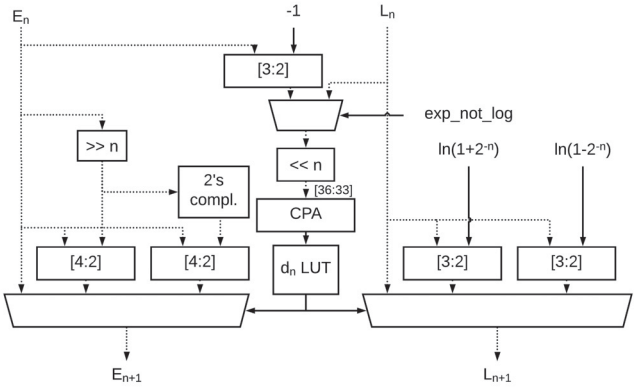

Figure 1. Architecture of a single iteration of the algorithm presented in Section 3. The internal representation is 39 bit carry-save number system. Dashed arrows show carry-save busses that are made out of two 39 bit busses for intermediate carry and intermediate sum. Solid lines are busses for binary numbers in non-redundant representation. Units labelled [3:2] are carry save adders [22] that add a carry save number with a binary number, with the propagation delay of a single full-adder. Units labelled [4:2] function as two [3:2] carry save adders chained in series to allow adding two numbers already in carry-save representation. For [4:2], rather than chaining two [3:2] adders, we have used a fast algorithm presented on page 123 of [16]. Two's complement of E_n is achieved by inverting all bits of the intermediate sum and intermediate carries, feeding these two values into a [4:2] adder, together with the original version of E_n and using a fifth input to [4:2] that is available as carry-in, to add 1 - this gives us subtraction $E_n - E_n 2^{-n}$. Signal *exp_not_log* is used to choose d, which is different for exp and log.

Figure 1 illustrates a hardware unit for a single iteration step of the iterative algorithm. The input to this module is E_n and L_n, values from the previous iteration, and the output is an update of these variables as per equations 1 and 3. At the bottom we have two 3:1 multiplexers that correspond

to different operations on E_n and L_n depending on value $d \in \{0, 1, -1\}$. When d is calculated, the multiplexers at the bottom choose appropriate next iteration value.

It can be seen from the circuit that it has 7 different paths almost independently running in parallel. We have found that the middle part, where we calculate d is a critical path of this unit. To improve this path, we have implemented value d as one-bit-hot pattern $[001_2, 010_2, 100_2]$ instead of the actual values $[0, 1, -1]$, which allowed the bottom multiplexers to be faster. Another slow component on this path is the $4\,\mathrm{bit}$ carry-propagate adder in the middle which is used to calculate non-redundant representation of $L_n^* = 2^n L_n$. We have split this adder into two $2\,\mathrm{bit}$ adders run in parallel, one returning a $2\,\mathrm{bit}$ result and another that is adding lower bits returning a $2\,\mathrm{bit}$ result with carry out. Then we have used this $5\,\mathrm{bit}$ value in the d look-up-table to make appropriate choice. Because two $2\,\mathrm{bit}$ adders can run in parallel, this also gave us a small improvement on the critical path.

4.2. Main architecture

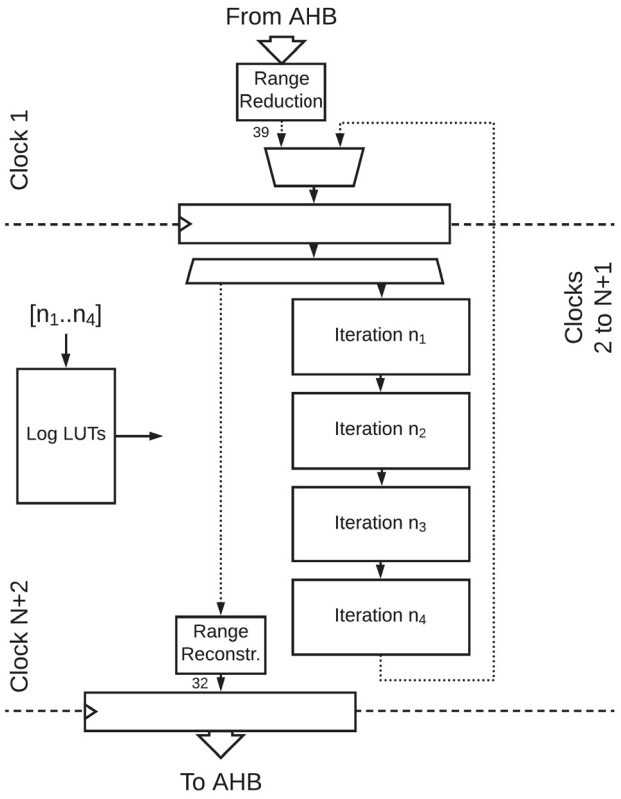

Figure 2. Architecture of the multiple precision exponential and natural logarithm unit. Range reduction and reconstruction stages in the first and the last clock cycles correspond to the operations shown in Section 3.2. Multiplexers are controlled by an FSM. Block "Log LUTs" represents a function which sends a corresponding $ln(1+d_{n_k}2^{-n})$ value (Equation 1) to each iteration depending on $n_{k\in[1,4]}$. The values in these LUTs are stored in the fixed-point format $s4.34$. Because equation 1 contains a subtraction, we store these log table values as 2's complement values in order to be able to do just carry-save additions inside the iteration unit.

Now, we define integer $I \neq 0$ as number of iteration modules instantiated and connected in series. In Figure 2 we demonstrate a top level architecture that has 4 such iterations instantiated and runs them in a loop (Here $I = 4$). First of all, $32\,\mathrm{bit}$ data is taken from the bus when available and sent through range reduction (Section 3.2) module in the first clock cycle. To obtain near full $32\,\mathrm{bit}$ accuracy, range reduction stage is done as $40\,\mathrm{bit}$ operations, i.e. the constants in range reduction equation are hard-coded with 40 bits in the fractional part. Then, the range reduced value is passed into the next clock cycle to start the main calculation using the iterative algorithm. The internal calculation is done using $39\,\mathrm{bit}$ fixed-point values, with $1\,\mathrm{bit}$ for sign, 4 bits for the integer part and 34 bits for the fractional part. The middle datapath is run for N clock cycles which is set-up beforehand. In our implementation, by default $N = 8$, which gives 32 iterations in total. Each iteration sub-unit keeps track of the iteration number that it is running which is used to index the log tables for $ln(1 + d_n 2^{-n})$ when calculating Equation 1. For example, iteration block 1 is running iterations with $n_1 \in \{1, 5, 9, 13, 17, 21, 25, 29\}$. Finally, on the last clock cycle, range reconstruction is done again using 40 bit constants. In the end, result is read out to the bus either in *s16.15* or *s0.31* fixed-point formats, which simply involves a fixed shifter.

5. Results

5.1. Accuracy and monotonicity

To measure accuracy of the unit and verify it over a wide range of available arguments, we have compared the unit to double-precision exponential function of the C standard library math.h. The experiments in this section were run by simulating the Verilog design of the unit. Given an argument x, we calculate absolute error:

$$\Delta exp = |exp'(x) - exp(x)|, \tag{10}$$

where exp' is hardware accelerator function and exp is math.h function exp(). By sweeping through all possible *s16.15* arguments for exponential, we have found that, when the unit is configured to do 8 iterations with $N = 8$, $99.8\,\%$ of values had an absolute error below one LSB ($2^{-15} = 0.000030517578125$, the smallest value representable by the Least Significant Bit in *s16.15* format), meaning that the result from the exponential hardware accelerator is one of two neighbouring values of the double-precision floating point sample. Small number of samples, $0.2\,\%$ had a maximum absolute error of 1.45 LSB. By running a sweep over all values, we have also verified that the exponential function at this accelerator configuration is monotonic.

For the natural logarithm accelerator with $N = 8$ we have also run an exhaustive test of approximately 2 billion samples across the range of possible inputs $x \in [LSB, 65536)$. We have found that $99.999\,\%$ of samples had an absolute error below one LSB and a very small number of tests had a maximum error of 1.01 LSB. By sweeping

over all possible arguments, we have found that logarithm function at this accelerator configuration is also monotonic.

Table 3 lists maximum absolute error and whether function is monotonic or not for all configurations $N \in [1...8]$. Figures 3 and 4 show absolute errors (without the absolute value) of number of arguments spread through full ranges for exp and log in *s16.15*. From these figures it can be seen that most of the values in exponential have absolute error below one LSB, except for the arguments close to the maximum, where function starts to produce errors that are below two LSB values. Analysis that is demonstrated in Figures 3 and 4 correspond to the first row of Table 3.

Table 4 provides the results for when the accelerator is used with $s0.31$ fixed-point format.

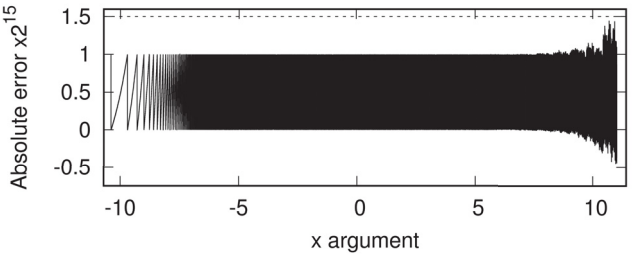

Figure 3. Absolute errors of *s16.15* exponential.

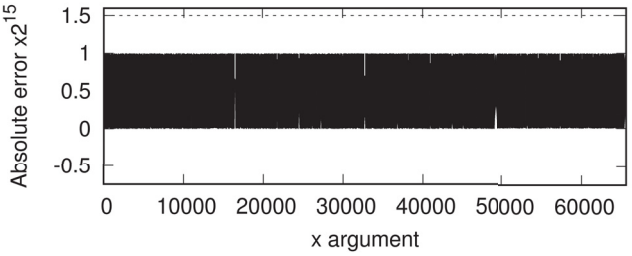

Figure 4. Absolute errors of *s16.15* logarithm.

TABLE 3. ACCURACY AND MONOTONICITY OF EXPONENTIAL AND LOGARITHM FUNCTIONS FOR DIFFERENT NUMBER OF ITERATIONS N, IN THE ACCELERATOR WITH 4 ITERATIONS PER CLOCK CYCLE AND *s16.15* INPUT AND OUTPUT FORMAT. $LSB = 2^{-15} = 0.000030517578125$.

	exp		ln	
N	max Δexp	Monotonic	max Δlog	Monotonic
8	0.00004425	Yes	0.00003082	Yes
7	0.00023559	Yes	0.00003082	Yes
6	0.00387969	Yes	0.00003082	Yes
5	0.06096649	Yes	0.00003112	Yes
4	0.99264343	Yes	0.00004089	No
3	15.3052932	No	0.00019928	No
2	241.053592	No	0.00268463	No
1	3352.69732	No	0.03837280	No

5.2. Synthesis study

We have executed a synthesis study of the presented design using the makeChip hosted design service platform [17]

TABLE 4. ACCURACY AND MONOTONICITY OF EXPONENTIAL AND LOGARITHM FUNCTIONS FOR DIFFERENT NUMBER OF ITERATIONS N, IN THE ACCELERATOR WITH 4 ITERATIONS PER CLOCK CYCLE AND *s0.31* INPUT AND OUTPUT FORMAT. $LSB = 2^{-31} = 0.000000000465661$.

	exp		ln	
N	max Δexp	Monotonic	max Δlog	Monotonic
8	0.000000000722	Yes	0.000000001387	Yes
7	0.000000003744	No	0.000000003613	Yes
6	0.000000059274	No	0.000000040312	Yes
5	0.000000945120	No	0.000000645976	No
4	0.000014910344	No	0.000010420316	No
3	0.000236990545	No	0.000170091129	No
2	0.003536022179	No	0.002655907041	No
1	0.045793333569	No	0.038344341439	No

for the GLOBALFOUNDRIES 22FDX technology [18]. An ultra-low voltage 9t-CNRX Standard-Cell Library with multiple voltage threshold options is used for implementation. Namely two main categories of cells are used: Low-Voltage-Threshold (Further called LVT) and Super-Low-Voltage-Threshold (Further called SLVT) cells - the former with the larger propagation delay but significantly less leakage than the latter, much faster cells. A nominal supply voltage of 0.50V is considered for low power operation. Synthesis is performed in a worst case operating condition at 0.45V and 0C. The power consumption of the circuit is analysed in a typical process condition at worst case power conditions of 0.55V at 85C.

Firstly we synthesised multiple accelerators with different copies of iterations (Fig.1) I placed per clock cycle. In Tables 5 and 6 we show multiple accelerators synthesised with $I \in \{1, 2, 3, 4, 6, 8\}$ for different clock constraints $f_{clk} = 150\,\text{MHz}$ (Table 5) and $f_{clk} = 250\,\text{MHz}$ (Table 6). The tables list the area, which is approximate area before *place and route*, the percentage of SLVT cells used (this gives a hint of major leakage increase and that the synthesizer is struggling to meet the timing constraints) and whether timing was met or not after optimisation stage and using SLVT library. When $f_{clk} = 150\,\text{MHz}$, for $I \in \{1, 2, 3, 4\}$, SLVT cells are not required and for $I > 4$, a lot of SLVT cells are placed on the path with iterations to meet timing constraints. When $f_{clk} = 250\,\text{MHz}$, SLVT cells are required for all the accelerator versions. However, when $f_{clk} = 250\,\text{MHz}, I \in \{1, 2\}$, SLVT cells are used only on the range reduction and reconstruction paths, as the iterative path of the circuit has a smaller propagation delay. When $I > 2$, percentage of SLVT cells rapidly increases to meet the timing constraints on the iterative path. Finally the limitations of this algorithm are reached when $f_{clk} = 250\,\text{MHz}$ and $I = 8$ where timing constraint cannot be met even with SLVT cells on the critical path.

Next, we make a sweep of the clock speed constraint in the range $f_{clk} = \{50, 75, 100, 125, 150, 175, 200, 225, 250\}$ and sample area and leakage (normalised) to compare two accelerators with $I = 1$ and $I = 4$ (Figures 5 and 6). It can be seen in Figure 5 that area is growing, as more larger LVT cells are used, until $150\,\text{MHz}$. After that point, the synthesizer starts using SLVT cells (starting with the smallest

size) to meet the timing constraint. Additionally, notice that the accelerator with $I = 1$, single iteration per clock cycle, does not require a significant amount of SLVT at any of the clock frequencies used and area keeps increasing only on the range reduction and range reconstruction paths.

Figure 6 shows leakage comparison. It can be seen that leakage for the accelerator with $I = 1$ increases at around $f_{clk} = 225\,\mathrm{MHz}$ when the synthesizer starts adding SLVT cells on the range reduction and range reconstruction paths. On the other hand, accelerator with $I = 4$ has a rapid increase in leakage when $f_{clk} > 150\,\mathrm{MHz}$ due to increased usage of SLVT cells both on the iterative path and range reduction and reconstruction paths.

TABLE 5. SYNTHESIS OF ACCELERATORS WITH DIFFERENT NUMBERS OF ITERATIONS PER CLOCK CYCLE. CLOCK WAS CONSTRAINED AT 150 MHz

Iterations per cycle, I	Area (μm^2)	SLVT cells	Timing met	Max latency
1	4507	0%	Y	34
2	6097	0%	Y	18
3	9482	0%	Y	13
4	12342	0%	Y	10
6	15034	29.2%	Y	8
8	19518	47.4%	Y	6

TABLE 6. SYNTHESIS OF ACCELERATORS WITH DIFFERENT NUMBERS OF ITERATIONS PER CLOCK CYCLE. CLOCK WAS CONSTRAINED AT 250 MHz

Iterations per cycle, I	Area (μm^2)	SLVT cells	Timing met	Max latency
1	6108	7.3%	Y	34
2	8755	3.1%	Y	18
3	10361	21.2%	Y	13
4	11524	36%	Y	10
6	17368	59.6%	Y	8
8	21893	68.7%	N	6

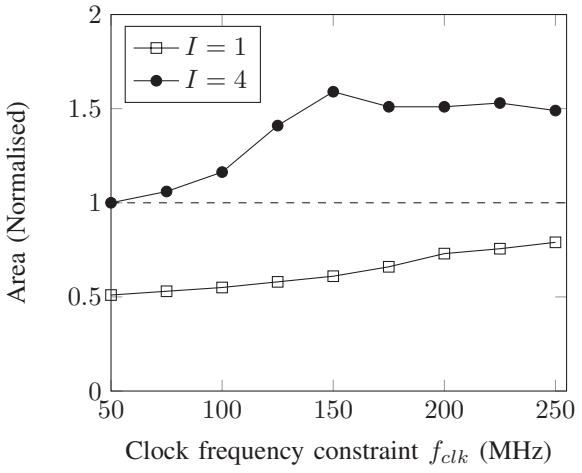

Figure 5. Area of two versions of the accelerator when synthesised with different clock constraints.

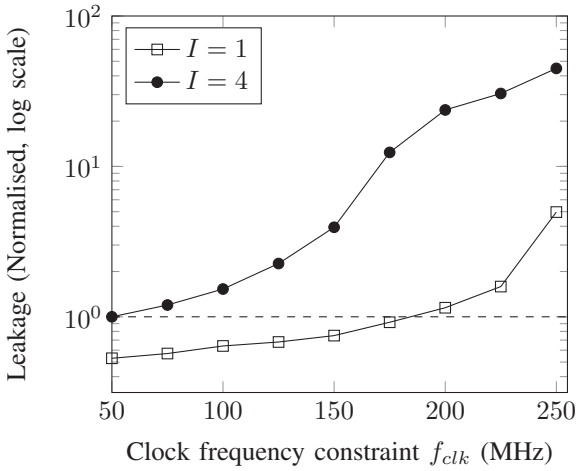

Figure 6. Leakage of two versions of the accelerator when synthesised with different clock constraints.

5.3. Place and route study

TABLE 7. COMPARISON OF THE EXPONENTIAL ACCELERATOR SYNTHESISED WITH A COMPLETE PE AT OPERATING CONDITIONS OF 0.5 V AND 250 MHz CLOCK FREQUENCY, TO SOFTWARE IMPLEMENTATION. * - INCLUDES 2 CYCLES FOR READING AND WRITING OPERATIONS.

	Exp accelerator	Software exp
Throughput	20.8-50M exp/s	2.6M exp/s
Latency	5-12 cycles/exp*	95 cycles/exp
Energy per exp	0.16 nJ/exp-0.39 nJ/exp	2.74 nJ/exp
Total area	5928 μm^2	-

The prototype core (later called PE - Processing Element), which will be later used as a building element in the SpiNNaker-2 massively parallel neuromorphic system, has been implemented with the presented elementary function accelerator (Version with $I = 4$ iterations per clock cycle) included. Figure 7 shows the layout of a single PE after place and route stage, with the accelerator highlighted in red. Figure 8 shows the accelerator layout in more detail.

We have tested the accelerator by running realistic software test cases including arbitrary number of calls to exp and logarithm function, on a netlist of a complete PE. Table 7 provides comparison of the accelerator to a similar software implementation of a fixed-point exponential used in current SpiNNaker systems. It can be seen that at some area we can obtain much higher throughput exponential function with very small energy consumption compared to software version.

Furthermore, in Table 8 we compare the exp accelerator and in Table 9 the natural logarithm accelerator to some other similar systems available in the literature. It can be seen that our solution has advantage by providing options for controlling energy and accuracy of the accelerator as well as providing different input/output formats. Additionally,

TABLE 8. COMPARISON OF THE EXPONENTIAL ACCELERATOR SYNTHESISED WITH A COMPLETE PE AT OPERATING CONDITIONS OF 0.5 V AND 250 MHz CLOCK FREQUENCY, TO SIMILAR SYSTEMS. * - ACCELERATOR ONLY (NO PROCESSOR INVOLVED); † - INCLUDES LOGARITHM FUNCTION.

	Our work†	[11]	[23]	[24]
Technology	22 nm	28 nm	FPGA	65nm
Throughput	20.8M-50M exp/s	83M exp/s	4.4M exp/s	24.8M exp/s
Pipelined	No	Yes	No	No
Energy per exp	0.16-0.39 nJ/exp	0.44 nJ/exp	-	0.002 nJ/exp*
Format	fixed	fixed	float	fixed
Monotonic	Yes	-	-	-
Accuracy control	Yes	No	No	No
Multi-format	Yes	No	No	No
Area	5928 μm^2	10 800 μm^2	-	20 700 μm^2

TABLE 9. COMPARISON OF THE LOGARITHM ACCELERATOR SYNTHESISED WITH A COMPLETE PE AT OPERATING CONDITIONS OF 0.5 V AND 250 MHz CLOCK FREQUENCY, TO SIMILAR SYSTEMS. † - INCLUDES EXPONENTIAL FUNCTION

	Our work†	[23]
Technology	22 nm	FPGA
Throughput	20.8M-50M ln/s	5.5M ln/s
Pipelined	No	No
Energy per exp	0.16-0.39 nJ/ln	-
Format	fixed	float
Monotonic	Yes	-
Accuracy control	Yes	No
Multi-format	Yes	No
Area	5928 μm^2	-

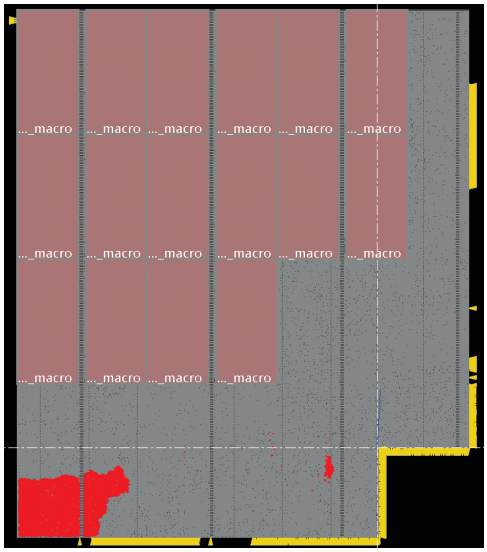

Figure 7. Layout of a processing element (PE) after place and route. Cells marked *...macro* bundled at north-west corner belong to local SRAM. The rest of the cells at the south-east corner belong to an ARM M4F based PE. Out of that, cells highlighted in red belong to the exp/ln accelerator with $N = 4$ iterations together with some AHB bus wrappers.

our implementation achieves strictly increasing, or monotonic function - unfortunately the systems that we compare to did not report about this property. However, by using the iterative algorithm (which makes it hard to introduce pipelined operation) and introducing more control of the unit we pay a small price in throughput compared to the previous SpiNNaker-2 prototype [11].

Figure 8. Layout of a processing element (PE) after place and route, zoomed in to the area at the bottom of the chip.

6. Conclusion

In this paper we have presented a parameterized approximate elementary function accelerator with accuracy control and multiple input/output formats, which can be adapted to a specific system with different constraints. This problem comes down to a 4-dimensional problem of optimising *power(leakage)-area-accuracy-latency*. We have demonstrated results of various accelerators by covering a large space of constraints in this 4-dimensional problem. Furthermore, we have shown how this method is used to design the accelerator for a neuromorphic chip. Our results of benchmarking the design show very high throughput of elementary functions with low energy consumption. Low energy consumption and additional options for reducing it further by controlling the accuracy of the unit will provide a suitable elementary function platform for neuromorphic applications.

For future work, we plan to extend the input/output types with single-precision floating point, which is supported by ARM M4F. This will involve adding a conversion to and from the internal fixed-point representation *s4.34*. For improving accuracy, for example removing errors at the end of the range of *s16.15* exponential function, we are planning to explore various rounding algorithms [25], [26] which would be included into the last clock cycle with range reduction operations. Additionally, we are planning to experiment with higher-radix implementation of the iterative algorithm [27] to speed up the iterative part of the accelerator - as described in [15], a radix-2^k version of the iterative algorithm implementation presented here would converge in n/k iterations, instead of n when radix-2 is used, to give *n-bit* accuracy. The only difficulty with higher radix algorithms would be larger look-up tables for natural logarithm entries in L_n iteration; also a choice of d_n value, which in higher radix can adopt

more values than $d_n \in \{-1, 0, 1\}$, on each iteration would result in more sophisticated d look-up tables and larger bottom multiplexers in the main iteration block. Finally, we plan to do a comprehensive comparison between the two exponential units that were designed for SpiNNaker-2 to understand differences between recurrence algorithms (this work) and polynomial approximation algorithms ([11]) in modern $22\,\mathrm{nm}$ and $28\,\mathrm{nm}$ technologies.

7. Acknowledgements

The authors would like to thank Felix Neumärker for suggestions about Verilog designs; Johannes Partzsch, Dongwei Hu, Michael Hopkins for the useful discussions and reviewers for their valuable feedback. This work was supported by the European Union under Grant Agreements No. 604102 and DLV-720270 (Human Brain Project). Author MM is funded by Kilburn studentship.

References

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

[2] C. Bartolozzi and G. Indiveri, "Synaptic Dynamics in Analog VLSI," *Neural Computation*, vol. 19, no. 10, pp. 2581–2603, 2007, pMID: 17716003. [Online]. Available: https://doi.org/10.1162/neco.2007.19.10.2581

[3] S. Friedmann, J. Schemmel, A. Grübl, A. Hartel, M. Hock, and K. Meier, "Demonstrating Hybrid Learning in a Flexible Neuromorphic Hardware System," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 11, no. 1, pp. 128–142, Feb 2017.

[4] M. Hopkins and S. Furber, "Accuracy and efficiency in fixed-point neural ODE solvers," *Neural Computation*, vol. 27, pp. 2148–2182, 2015.

[5] J. C. Knight, P. J. Tully, B. A. Kaplan, A. Lansner, and S. B. Furber, "Large-scale simulations of plastic neural networks on neuromorphic hardware," *Frontiers in Neuroanatomy*, vol. 10, p. 37, 2016. [Online]. Available: http://journal.frontiersin.org/article/10.3389/fnana.2016.00037

[6] J. Harrison, T. Kubaska, S. Story, M. S. Labs, and I. Corporation, "The computation of transcendental functions on the ia-64 architecture," *Intel Technology Journal*, vol. 4, pp. 234–251, 1999.

[7] B. Vogginger, R. Schüffny, A. Lansner, L. Cederström, J. Partzsch, and S. Höppner, "Reducing the computational footprint for real-time BCPNN learning," *Frontiers in Neuroscience*, vol. 9, p. 2, 2015. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnins.2015.00002

[8] J. C. Knight and S. B. Furber, "Synapse-Centric Mapping of Cortical Models to the SpiNNaker Neuromorphic Architecture," *Frontiers in Neuroscience*, vol. 10, p. 420, 2016. [Online]. Available: http://journal.frontiersin.org/article/10.3389/fnins.2016.00420

[9] F. Galluppi, X. Lagorce, E. Stromatias, M. Pfeiffer, L. A. Plana, S. B. Furber, and R. B. Benosman, "A framework for plasticity implementation on the SpiNNaker neural architecture," *Frontiers in Neuroscience*, vol. 8, p. 429, 2015. [Online]. Available: http://journal.frontiersin.org/article/10.3389/fnins.2014.00429

[10] M. Mikaitis, G. Pineda García, J. C. Knight, and S. B. Furber, "Neuromodulated Synaptic Plasticity on the SpiNNaker Neuromorphic System," *Frontiers in Neuroscience*, vol. 12, p. 105, 2018. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnins.2018.00105

[11] J. Partzsch, S. Höppner, M. Eberlein, R. Schüffny, C. Mayr, D. R. Lester, and S. Furber, "A fixed point exponential function accelerator for a neuromorphic many-core system," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2017, pp. 1–4.

[12] M. D. Ercegovac, "On approximate arithmetic," in *2013 Asilomar Conference on Signals, Systems and Computers*, Nov 2013, pp. 126–130.

[13] T. Pfeil, T. Potjans, S. Schrader, W. Potjans, J. Schemmel, M. Diesmann, and K. Meier, "Is a 4-bit synaptic weight resolution enough? constraints on enabling spike-timing dependent plasticity in neuromorphic hardware," *Frontiers in Neuroscience*, vol. 6, p. 90, 2012. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnins.2012.00090

[14] M. Diesmann, Private Communication.

[15] J.-M. Muller, *Elementary Functions - Algorithms and Implementation*, 3rd ed., 2016.

[16] M. Ercegovac and T. Lang, *Digital Arithmetic*, ser. Morgan Kaufmann Series in Comp. Morgan Kaufmann, 2004. [Online]. Available: https://books.google.co.uk/books?id=uUk_AQAAIAAJ

[17] "Online." [Online]. Available: www.makechip.design

[18] R. Carter, J. Mazurier, L. Pirro, J. U. Sachse, P. Baars, J. Faul, C. Grass, G. Grasshoff, P. Javorka, T. Kammler, A. Preusse, S. Nielsen, T. Heller, J. Schmidt, H. Niebojewski, P. Y. Chou, E. Smith, E. Erben, C. Metze, C. Bao, Y. Andee, I. Aydin, S. Morvan, J. Bernard, E. Bourjot, T. Feudel, D. Harame, R. Nelluri, H. J. Thees, L. M-Meskamp, J. Kluth, R. Mulfinger, M. Rashed, R. Taylor, C. Weintraub, J. Hoentschel, M. Vinet, J. Schaeffer, and B. Rice, "22nm fdsoi technology for emerging mobile, internet-of-things, and rf applications," in *2016 IEEE International Electron Devices Meeting (IEDM)*, Dec 2016, pp. 2.2.1–2.2.4.

[19] E. Painkras, L. Plana, J. Garside, S. Temple, F. Galluppi, C. Patterson, D. Lester, A. Brown, and S. Furber, "SpiNNaker: A 1-W 18-core system-on-chip for massively-parallel neural network simulation," *IEEE Journal of Solid-State Circuits*, vol. 48, no. 8, pp. 1943–1953, 2013.

[20] S. Furber, "Large-scale neuromorphic computing systems," *Journal of Neural Engineering*, vol. 13, no. 5, p. 051001, 2016. [Online]. Available: http://stacks.iop.org/1741-2552/13/i=5/a=051001

[21] S. Höppner, Y. Yan, B. Vogginger, A. Dixius, J. Partzsch, F. Neumärker, S. Hartmann, S. Schiefer, S. Scholze, G. Ellguth, L. Cederstroem, M. Eberlein, C. Mayr, S. Temple, L. Plana, J. Garside, S. Davison, D. R. Lester, and S. Furber, "Dynamic voltage and frequency scaling for neuromorphic many-core systems," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2017, pp. 1–4.

[22] P. Kornerup, "Reviewing 4-to-2 adders for multi-operand addition," in *Proceedings IEEE International Conference on Application- Specific Systems, Architectures, and Processors*, 2002, pp. 218–229.

[23] J. Detrey, F. de Dinechin, and X. Pujol, "Return of the hardware floating-point elementary function," in *18th IEEE Symposium on Computer Arithmetic (ARITH '07)*, June 2007, pp. 161–168.

[24] P. Nilsson, A. U. R. Shaik, R. Gangarajaiah, and E. Hertz, "Hardware implementation of the exponential function using taylor series," in *2014 NORCHIP*, Oct 2014, pp. 1–4.

[25] N. Kavvadias and K. Masselos, "Design of fixed-point rounding operators for the VHDL-2008 standard," in *Proceedings of the 2012 Conference on Design and Architectures for Signal and Image Processing*, Oct 2012, pp. 1–8.

[26] G. Even and P. M. Seidel, "A comparison of three rounding algorithms for IEEE floating-point multiplication," *IEEE Transactions on Computers*, vol. 49, no. 7, pp. 638–650, Jul 2000.

[27] M. D. Ercegovac, "Radix-16 evaluation of certain elementary functions," *IEEE Transactions on Computers*, vol. C-22, no. 6, pp. 561–566, June 1973.