Efficient digital implementation of the sigmoid function for reprogrammable logic

M.T. Tommiska

Abstract: Special attention must be paid to an efficient approximation of the sigmoid function in implementing FPGA-based reprogrammable hardware-based artificial neural networks. Four previously published piecewise linear and one piecewise second-order approximation of the sigmoid function are compared with SIG-sigmoid, a purely combinational approximation. The approximations are compared in terms of speed, required area resources and accuracy measured by average and maximum error. It is concluded that the best performance is achieved by SIG-sigmoid.

1 Introduction

Artificial neural networks (ANNs) have been mostly implemented in software. This has benefits, since the designer does not need to know the inner workings of neural network elements, but can concentrate on the application of the neural network. However, a disadvantage in real-time applications of software-based ANNs is slower execution compared with hardware-based ANNs.

Hardware-based ANNs have been implemented as both analogue and digital circuits. The analogue implementations exploit the nonlinear characteristics of CMOS (complementary metal-oxide semiconductor) devices, but they suffer from thermal drift, inexact computation results and lack of reprogrammability.

Digital hardware-based implementations of ANNs have been relatively scarce, representive examples of recent research can be found in [1–3]. Recent advances in reprogrammable logic enable implementing large ANNs on a single field-programmable gate array (FPGA) device. The main reason for this is the miniaturisation of component manufacturing technology, where the data density of electronic components doubles every 18 months [4].

Special attention must be paid to an area-efficient implementation of every computational element when implementing large ANNs on digital hardware. This holds true for the nonlinear activation function used at the output of neurons [5].

A common activation function is the sigmoid function (Fig. 1)

$$y = \frac{1}{1 + \varepsilon^{-x}} \tag{1}$$

An advantage of the sigmoid function is its derivative (see Fig. 1)

$$\frac{dy}{dx} = y(1 - y) \tag{2}$$

whose existence is essential in neural network training algorithms. Since the sigmoid function has a symmetry point at (0, 0.5), only half of the x–y pairs have to be computed

$$y_{x>0} = 1 - y_{x\leq 0} \quad \text{or} \tag{3}$$

$$y_{x<0} = 1 - y_{x\geq 0} \tag{4}$$

A straightforward sigmoid implementation requires a lot of area, and an approximation is the only practical solution in digital ANNs.

2 Implementations of sigmoid function

Digital hardware implementations of the sigmoid function are divided into three main categories: piecewise linear (PWL) approximations, piecewise second-order approximations and combinational approximations. The efficiency criteria for a successful approximation are the achieved accuracy, speed and area resources.

The maximum and average error are used to evaluate the accuracy of an approximation. Following the methodology in [6], if a function $f(x)$ is approximated by a function $\hat{f}(x)$ in the interval $x \in (\alpha_0, \alpha_1)$, the average E_{ave} and maximum $E_{\max}$ errors are obtained by uniformly sampling x on 10^6 equally spaced points in the domain of (α_0, α_1)

$$\begin{cases} E_{ave} = \dfrac{\sum_{i=0}^{10^6-1} |\hat{f}(x_i) - f(x_i)|}{10^6} \\ E_{max} = \max |\hat{f}(x_i) - f(x_i)| \end{cases} \tag{5}$$

Evaluating speed is straightforward, provided that all the implementations under comparison compute the sigmoid function in a single clock cycle. In this case the speed metric is the maximum clock rate, typically denoted in megahertz (MHz).

When evaluating the area resources of FPGA-based implementations, the basic unit is a logic element (LE) discussed in Section 3.

A straightforward implementation of the sigmoid function is not feasible, since both division and exponentiation are very demanding operations, as they require a lot of area resources and converge slowly.

In this paper, only fixed-point notation is used. Floating-point arithmetic does not suit a digital approximation for two main reasons: the area requirements of floating-point

IEE Proceedings online no. 20030965
doi: 10.1049/ip-cdt:20030965
Paper first received 12th June 2003 and in revised form 20th August 2003

The author is with Signal Processing Laboratory, Helsinki University of Technology, Otakaari 5 A, FIN-02150, Finland

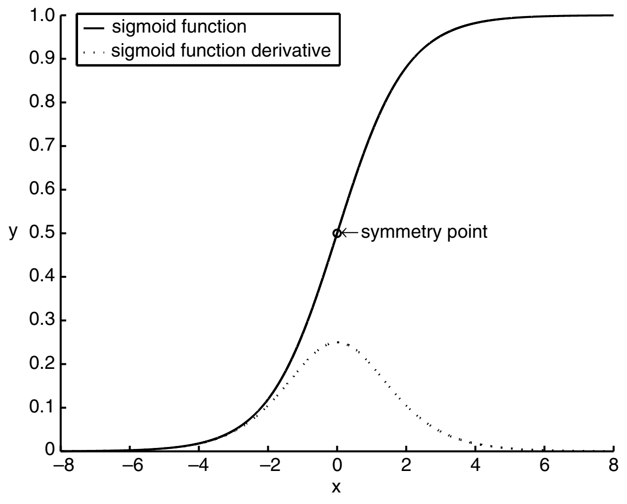

Fig. 1 *Sigmoid function and its derivative*

arithmetic are extensive compared with fixed-point arithmetic [7]; and the sigmoid function has a limited range in both its input and output. The fixed-point format is defined as follows:

$$[s]\ a{\cdot}b \tag{6}$$

where the optional s denotes a sign bit with 0 for positive and 1 for negative numbers, a is the number of integer bits and b is the number of fractional bits.

If the sign bit is present, two's complement notation is used, otherwise unsigned notation is used. The examples in Table 1 clarify the notation.

If the sign bit is used, the minimum x_{min} and maximum x_{max} numbers in $sa.b$ notation are

$$\begin{cases} x_{min} = 2^{-a} \\ x_{max} = 2^{a} - 2^{-b} \end{cases} \tag{7}$$

If the sign bit is not used, the minimum x_{min} and maximum x_{max} numbers in $a.b$ notation are

$$\begin{cases} x_{min} = 0 \\ x_{max} = 2^{a} - 2^{-b} \end{cases} \tag{8}$$

The maximum truncation error E_{trun} is the absolute difference between the real number x and its truncated binary representation $\hat{x}$, that is $|x - \hat{x}|$. Excluding overflows and underflows, the maximum truncation error in $[s]a.b$ notation is

$$E_{trunmax} = 2^{-(b+1)} \tag{9}$$

Table 1: Examples of binary notation

Format	Example binary number	Decimal number
*s*3.5	011001100	6.375
*s*2.5	11001001	−1.71875
0.8	11001001	0.78515625

2.1 *Piecewise linear approximations of sigmoid function*

Piecewise linear (PWL) approximation is a method to obtain low values for both maximum and average error with low computational complexity. In the following subsections, four PWL schemes are presented. They differ in the number and location of start and end points of the approximating lines and the selection criteria and algorithms. None of the presented PWL approximations require multipliers, which is positive in hardware implementability.

2.1.1 A-law based approximation: Myers and Hutchinson designed an approximation based on the A-law companding technique [8]. A modified curve was developed so that the gradient of each linear segment is expressed as a power of two. This enables replacing multipliers with shifters. The curve has seven segments and its breakpoints are presented in Table 2.

2.1.2 Approximation of Alippi and Storti–Gajani: Alippi and Storti–Gajani based their approximation on selecting an integer set of breakpoints, and setting the y-values as power of two numbers [9]. If only the negative x-axis is considered, see (3), and (x) is defined as the integral part of x, the decimal part of x with its own sign is denoted $\hat{x}$ and defined as follows:

$$\hat{x} = x + |(x)| \tag{10}$$

The general expression for approximating the sigmoid function becomes

$$y = \frac{\frac{1}{2} + \frac{\hat{x}}{4}}{2^{|x|}} \tag{11}$$

Since the formula involves only additions and shift operations, it is well suited for digital implementation.

2.1.3 PLAN approximation: The PLAN approximation (piecewise linear approximation of a nonlinear function) was proposed by Amin, Curtis and Hayes–Gill [10]. The PLAN approximation uses digital gates to directly transform from x to y. The approximation of the sigmoid function is presented in Table 3. The calculations need only be performed on the absolute value of the input x, see (3). After simplifying the shift and addition operations implicit in Table 3, the bit-level logic equations become effective to implement.

2.1.4 Centred recursive interpolation of sigmoid function: Basterrextea, Tarela and del Campo presented a recursive algorithm for approximating the sigmoid function [11]. The algorithm is based on the centred recursive interpolation (CRI) method, which attempts to improve the accuracy recursively, as the number of linear segments increases exponentially with every round.

The initial three straight lines are

$$\begin{aligned} y_1(x) &= 0 \\ y_2(x) &= \frac{1}{2} * \left(1 + \frac{x}{2}\right) \\ y_3(x) &= 1 \end{aligned} \tag{12}$$

Table 2: Breakpoints of A-law based sigmoid approximation

Break-points	*x*	−8.0	−4.0	−2.0	−1.0	1.0	2.0	4.0	8.0
	y	0.0	0.0625	0.125	0.25	0.75	0.875	0.9375	1.0

Table 3: PLAN approximation equations

Operation	Condition
$Y = 1$	$\|X\| \geq 5$
$Y = 0.03125 \cdot \|X\| + 0.84375$	$2.375 \leq \|X\| < 5$
$Y = 0.125 \cdot \|X\| + 0.625$	$1 \leq \|X\| < 2.375$
$Y = 0.25 \cdot \|X\| + 0.5$	$0 \leq \|X\| < 1$

Since only the positive x-axis needs to be considered, see (4), the CRI algorithm is as follows:

```
$g(x) = y_2(x);\; h(x) = y_3(x);$
for $(i = 0;\; i = q; i++)\{$
    $g\prime(x) = \mathrm{Min}[g(x), h(x)];$
    $h(x) = \frac{1}{2}(g(x) + h(x) - \Delta);$
    $g(x) = g'(x);$
    $\Delta = \frac{\Delta}{4};\}$
$g(x) = \mathrm{Min}[g(x), h(x)];$        (13)
```

where q is the interpolation level, Δ is the depth parameter dependent on q, $h(x)$ is the linear interpolation function, and $g(x)$ is the resulting approximation. Neither multiplications nor divisions are needed, since they are reduced to shiftings.

The optimum values for Δ have been calculated in [11] for $q = 1\ldots3$ and are denoted as $\Delta_{q,opt}$. They are presented in Table 4 with the number of linear segments in the approximating curve. Increasing q above 3 does not provide additional accuracy, as the approximation saturates [11].

The CRI method requires iterative calculation for $q + 1$ clock cycles. This makes the CRI method a slow approximation algorithm, since typically more than one clock cycle is required to reach satisfactory accuracy.

2.2 Piecewise second-order approximation of sigmoid function

The sigmoid function has also been implemented as a piecewise second-order approximation. In general, this implies that the sigmoid function is approximated by

$$y(x) = c_0 + c_1 * x + c_2 * x^2. \tag{14}$$

The obvious disadvantage is the need for multiplications.

Zhang, Vassiliadis and Delgado–Frias have presented a second-order approximation scheme requiring one multiplier [6]. In the interval $]-4, 4[$, the sigmoid function is computed as follows:

$$y = \begin{cases} 2^{-1} * (1 - |2^{-2} * x|^2) & -4 < x < 0 \\ 1 - 2^{-1} * (1 - |2^{-2} * x|^2) & 0 \leq x < 4 \end{cases} \tag{15}$$

After simplifications (15) can be implemented with one multiplier, two shifters and two XORs.

Table 4: CRI approximation

Interpolation level	Optimum depth parameter	Number of segments
$q = 0$		3
$q = 1$	$\Delta_{1,opt} = 0.30895$	5
$q = 2$	$\Delta_{2,opt} = 0.28094$	9
$q = 3$	$\Delta_{3,opt} = 0.26588$	17

2.3 Combinational approximation of sigmoid function

When both the input and output of an approximation contain only a few bits, an alternative is to use a direct bit-level mapping. No arithmetic operators are needed and area requirements remain low.

Every Boolean function can be expressed in canonical form as a sum of its 1-minterms [12]. This is called a sum-of-products (SOP) representation and it corresponds to a simple digital implementation, where the products or 1-minterms formed by ANDing the required inputs are summed by a multiple-input OR gate to produce the output.

Quine [13] and McCluskey [14] presented a procedure leading to a minimised SOP representation for a given function. Based on their procedure, researchers have implemented logic minimisation programs. In this paper, the McBoole logic minimiser [15] was used to compute minimised functions for all output bits of a sigmoid function approximation. The purpose was to find out whether an entirely combinational approximation, named SIG-sigmoid, would outperform previously published methods.

2.3.1 SIG-sigmoid:

Implementations are named as follows:

`sig_xyzo`

where x is the number of input integer bits, y is the number of input fractional bits, z is the number of output fractional bits (output integer bits are not needed), and o is either a, n or p:

a or *all*, when all bits in all input values (both positive and negative) are combinationally mapped to bits in the output values
n or *negative*, when bits in only negative input values are combinationally mapped to output bits. The positive input values are handled according to (3). A z-bit-wide adder/subtractor is needed.
p or *positive*, when bits in only positive input values are combinationally mapped to output bits. The negative input values are handled according to (4). A z-bit-wide adder/subtractor is needed.

In addition to x integer bits and y fractional bits, the input has also a sign bit. The naming convention is clarified in Table 5. The fixed-point format has been defined in (6).

The E_{max} (5) of SIG-sigmoid implementations is defined directly by z (9), because truncation error is the only error source in direct bit-level combinational mapping

$$E_{max} = 2^{-(z+1)} \tag{16}$$

Equation 16 represents the worst-case E_{max} of SIG-sigmoid implementations. Often the E_{max} of an implementation is

Table 5: SIG-sigmoid naming convention

SIG-Sigmoid name	Input format	Output format	Explanation
`sig_236a`	*s*2.3	0.6	Bit-level mapping for all inputs
`sig_235n`	*s*2.3	0.5	Bit-level mapping for negative inputs only
`sig_346p`	*s*3.4	0.6	Bit-level mapping for positive inputs only

Table 6: Average E_{ave} and maximum E_{max} errors of sigmoid function approximations

Approximation	Input range	E_{ave}	E_{max}
A-law based approximation	[–8,8[	2.47%	4.90%
Approximation of Alippi and Storti–Gajani	[–8,8[	0.87%	1.89%
PLAN approximation	[–8,8[	0.59%	1.89%
CRI approximation, $q = 0$	[–8,8[	2.41%	11.9%
CRI approximation, $q = 1$	[–8,8[	1.20%	3.78%
CRI approximation, $q = 2$	[–8,8[	0.92%	2.45%
CRI approximation, $q = 3$*	[–8,8[	0.85%	2.06%
Approximation of Zhang *et al.*	]–4,4[	0.77%	2.16%
`sig_235p`†	[–4,4[	0.69%	1.51%
`sig_236p`	[–4,4[	0.40%	0.77%
`sig_336p`	[–8,8[	0.33%	0.77%
`sig_337p`	[–8,8[	0.17%	0.39%

* As mentioned in Section 2.1.4 [11], increasing the interpolation level q above 3 does not provide additional accuracy.
† The naming convention is explained in Section 2.3 and in Table 5.

less than defined by (16), which is also evident later in Table 6.

The input range is defined by x, the number of input integer bits. If $x = 3$, the input range is $[-8,8[$, and if $x = 2$, the input range is $[-4,4[$, which suffices in many applications.

MATLAB was first used to calculate the real sigmoid values in the input range. Both the input and output values were truncated according to the `sig_xyzo` naming convention. The MATLAB output file was modified to meet the requirements of the McBoole logic minimiser [15], whose output was subsequently modified (for example, ANDs and ORs were added) and renamed as a VHDL [16] file `sig_xyzo.vhd`. The VHDL file was synthesised with Synplicity's Synplify Pro 7.1 logic synthesis software [17]. The report file of Synplify Pro 7.1 indicated the required area resources and timing characteristics of a SIG-sigmoid implementation. These are collected in Tables 8 and 9 in Section 4.

Since SIG-sigmoid implementations are represented as two-level logic, they fit well into the inner architecture of FPGAs and are fast compared with other approximations.

When comparing representative SIG-sigmoid implementations with other approximations, it turned out that the best performance was among `sig_xyzp` implementations, i.e. designs where only bits in positive input values were combinationally mapped to output bits. `Sig_xyza` implementations do not require a z-bit-wide adder/subtractor, but the twice larger number of inputs requires more area resources than `sig_xyzp` implementations. A block diagram of `sig_xyzp` implementations is presented in Fig. 2.

2.4 Other approximations of sigmoid function

In addition to the six approximations of the sigmoid function presented in Sections 2.1–2.3, other approximations have also been proposed. This Section describes briefly other approximations of the sigmoid function, and motivates their exclusion from the comparison in Section 4.

Implementing a sigmoid function as a lookup table is straightforward. As the approximated values can be calculated in advance, the E_{max} is defined by (9), where b is the number of output fractional bits.

However, the lookup-table implementation is a limiting factor, as the memory requirements of a pipelined neural network implementation grow. Since internal memory is limited in FPGAs, it has other purposes than serving only as a storage for values of a lookup-table approximation. Self-contained neurons are desirable, and sharing a lookup-table approximation between all neurons decreases the execution speed by orders of magnitude. As FPGA technologies mature, both the amount and speed of available internal memory increase, and this may make lookup tables an attractive implementation option in the future.

Other approximations of the sigmoid function include a polynomial floating-point approximation [19]. Since both a floating-point multiplier and a floating-point adder are required, the area requirements are extensive compared with fixed-point implementations.

Both a second-order approximation [20] and a generic nonlinear activation function operator [21] require a multiplier, which does not compare well with simpler multiplierless approximations.

If the main neural network uses a generic parameterisable multiplier for multiplying the neuron inputs with weights, the multiplier could also be used in a time-sharing manner for computing an approximation of the sigmoid function. However, an area-efficient implementation of a neural network is most likely to use constant coefficient multipliers when the weights are known, and therefore assuming the availability of a generic multiplier is not realistic.

3 Field-programmable gate arrays

The traditional measurement unit of area usage is *gate*, or a two-input NAND (NOT AND) gate requiring four transistors [22]. This metric suits full-custom application-specific integrated circuits (ASICs), where the entire architectural structure is specified in the design process. ASICs are suitable for high-volume designs, which amortises the startup costs of ASIC production. On the contrary, field-programmable gate arrays (FPGAs) require no upfront costs, since the unit cost of a single device is more or less constant irrespective of the number of devices required [23].

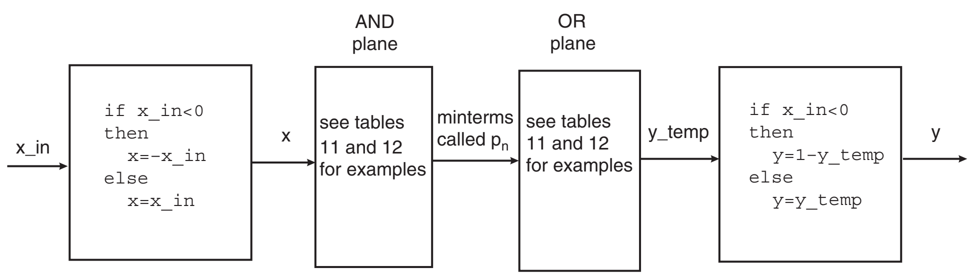

Fig. 2 *Block diagram of `sig_xyzp` implementations*

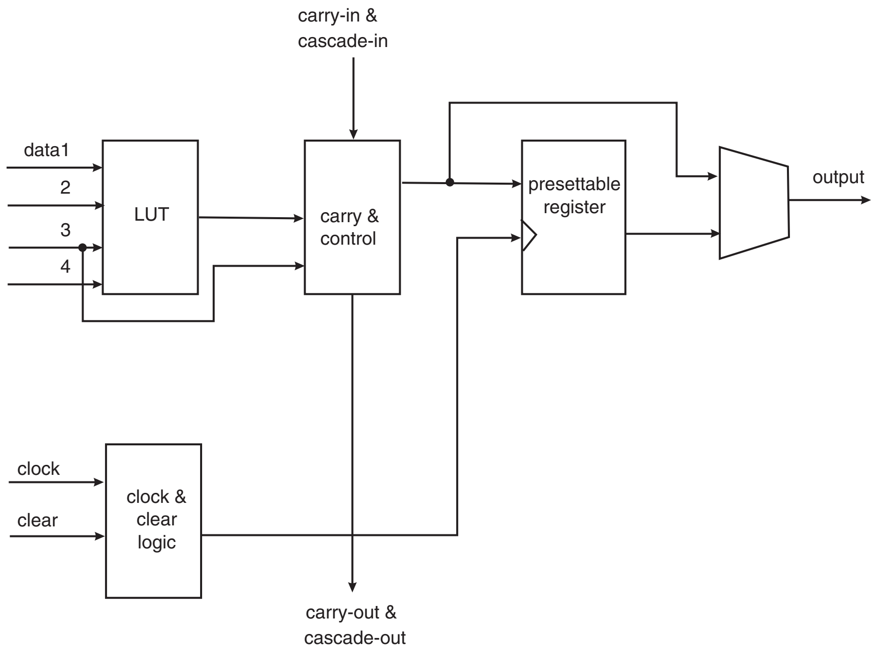

Fig. 3 *Block diagram of logic element (LE) in APEX II devices*

Another advantage of FPGAs is reprogrammability, which makes them ideal candidates in prototyping applications, where dynamic updating is required.

Implementing neural networks in FPGAs is attractive, because this is usually done in low-volume experimental projects. Furthermore, numerous updating cycles are performed in searching for an optimum neural network configuration.

When the area requirements of FPGA-based designs are compared, the gate-count figures of ASICs do not serve their purpose. This is because FPGAs have a pre-existing structure of reprogrammable functional elements, which cannot be effectively described as equivalent gate counts.

FPGAs consist of a regular and hierarchical structure of reprogrammable basic functional elements, which are most often called *logic elements* (LE), although the terminology varies somewhat among manufacturers. LEs are connected by hierarchical routing, where connections to neighbouring elements are fastest. In this paper the LEs are regarded as the area measurement unit in FPGAs.

The internal structure of a generic LE typically consists of a four-input lookup table (LUT) for combinational functions of up to four arguments, a presettable register for state machines and sequential logic, fast carry and cascade chains for high-speed arithmetic and additional clock control logic. As a representative example of a modern LE, the high-level structure of an LE in Altera's APEX II devices is described in Fig. 3.

4 Comparison of sigmoid function approximations

To compare the sigmoid function approximations, all algorithms presented in Sections 2.1–2.3 were implemented both as MATLAB .m-files and as VHDL descriptions. The MATLAB m-files were used to calculate E_{ave} and E_{max} (5). The error figures of the A-law based approximation (Section 2.1.1), approximation of Alippi and Storti–Gajani (Section 2.1.2), PLAN approximation (Section 2.1.3), CRI approximation (Section 2.1.4), approximation of Zhang *et al.* (Section 2.2) and representative SIG-sigmoid approximations (Section 2.3) are presented in Table 6.

Although the E_{max} of both `sig_236p` and `sig_336p` is the same, the E_{ave} is smaller for `sig_336p` than for `sig_236p`. This is due to the larger input range of `sig_336p`, as the output of the sigmoid function saturates at 0 or 1 and the average accuracy for $|x| > 4$ increases asymptotically.

The approximation errors in Table 6 are presented in Figs. 4 and 5.

To conform with the original representations of the sigmoid function approximations, the inputs and outputs were represented in VHDL descriptions in fixed-point format as suggested in or as could be concluded from the original publications. This is summarised in Table 7.

The VHDL descriptions were synthesised with Synplicity's Synplify Pro 7.1 logic synthesis software, whose report file informed the area resources and timing characteristics of a particular approximation. The number of required logic elements was regarded as the area usage metric. The targeted FPGA device family was Altera's APEX II (see also Fig. 3) [18], a representative modern programmable logic device family both by its architecture and performance. The CRI approximation was not coded in VHDL, as the results reported in [24] were available. The area requirements of the A-law based approximation (Section 2.1.1), approximation of Alippi and Storti–Gajani (Section 2.1.2), PLAN approximation (Section 2.1.3), CRI approximation (Section 2.1.4), approximation of Zhang *et al.* (Section 2.2) and several representative SIG-sigmoid approximations (Section 2.3) are presented in Table 8 and the maximum clock rate (in MHz) is presented in Table 9.

It is suggested, that a quality factor Q be defined to represent the accuracy and usability of a sigmoid function approximation

$$Q = \frac{f_{max}}{LEs * E_{ave} * E_{max}} \tag{17}$$

where $Q =$ quality factor, $f_{max} =$ clock rate (Table 9), $LEs =$ Number of logic elements (Table 8), $E_{ave} =$ average error in per cent (Table 6), and $E_{max} =$ maximum error in per cent (Table 6). Based on (17) and Tables 6, 8 and 9, the approximations are arranged into an order of superiority. This is presented in Table 10.

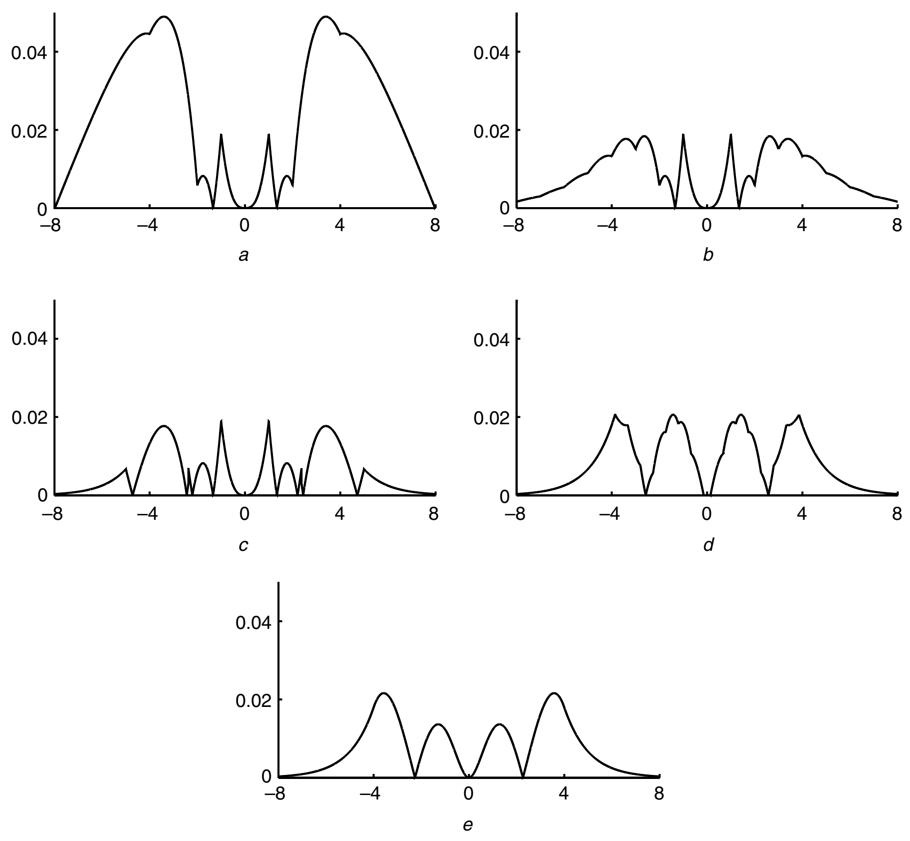

Fig. 4 *Error function in previously published sigmoid function approximations*

a A-law based approximation
b Approximation of Alippi *et al.*
c PLAN approximation
d CRI approximation, $q = 3$
e Approximation of Zhang *et al.*

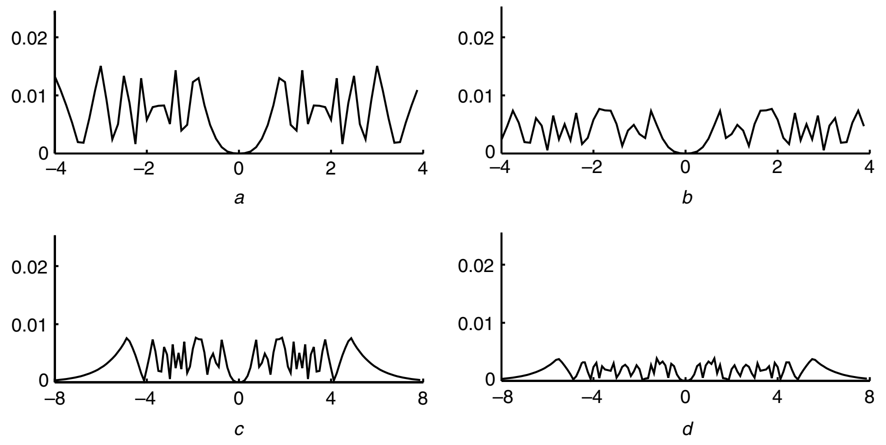

Fig. 5 *Error function in representative SIG-sigmoid approximations*

a `sig_235p`
b `sig_236p`
c `sig_336p`
d `sig_337p`

4.1 Required approximation accuracy

The precision requirements in multilayer perceptron networks are stricter in the training phase than in feedforward operation of the network. The minimum bit widths for both network weights and the activation function (e.g. sigmoid) were analysed in [25]. It was concluded by statistical analysis, that the minimum bit width for the activation function in the back-propagation learning algorithm is 8–10 bits, whereas the feedforward operation requires a precision of 7 or 8 bits.

An extensive review of the quantisation errors in hardware implementations of neural networks is presented in [26], where hardware-friendly learning algorithms, for example perturbation algorithms and local-learning algorithms, are proposed. The discussion concentrates on the required accuracy of weight representation, and the precision of the activation function is not specifically researched. The robustness of the activation function with the proposed hardware-friendly learning algorithms remains a subject of interesting further studies.

Table 7: Fixed-point format (6) used in VHDL descriptions

Approximation	Input	Output
A-law based approximation	s3.6	0.7
Approximation of Alippi and Storti–Gajani	s3.6	0.7
PLAN approximation	s4.5	1.7
CRI approximation, $q = 1 \ldots 3$*		
approximation of Zhang *et al.*	s3.10	3.10
sig_235p	s2.3	0.5
sig_236p	s2.3	0.6
sig_336p	s3.3	0.6
sig_337p	s3.3	0.7

*Authors' results reported [24].

Table 8: Area requirements in logic elements (LE) of sigmoid function approximations

Approximation	LEs*
A-law based approximation	36
Approximation of Alippi and Storti–Gajani	36
PLAN approximation	39
CRI approximation, $q = 0 \ldots 3$†	65
Approximation of Zhang *et al.*	176
sig_235p	22
sig_236p	25
sig_336p	32
sig_337p	45

*Target device is EP2A15F672C7, the smallest member of Altera's APEX II family.
† Target device was Altera's EPC10K20RC240–4 [24], but since its internal architecture is almost equivalent to the APEX II architecture the results are comparable.

Table 9: Clock rate of sigmoid function approximations

Approximation	Clock rate* MHz
A-law based approximation	58.6
Approximation of Alippi and Storti–Gajani	64.2
PLAN approximation	75.8
CRI approximation, $q = 0$	34.7
CRI approximation, $q = 1$†	16.3
CRI approximation, $q = 2$	11.6
CRI approximation, $q = 3$	8.7
Approximation of Zhang *et al.*	66.4
sig_235p	89.5
sig_236p	94.7
sig_336p	85.7
sig_337p	76.4

*Target device is EP2A15F672C7.
† Since iterative CRI approximation requires $q + 1$ cycles to complete, normalised clock rate as defined by $f_{nom}/q + 1$, where $f_{nom} = 34.72$ MHz [24] is reported.

Table 10: Quality factor sigmoid function approximations

Approximation	Quality factor
sig_337p	25,608
sig_236p	12,299
sig_336p	10,540
sig_235p	3,905
PLAN approximation	1,743
Approximation of Alippi and Storti–Gajani	1,085
approximation of Zhang *et al.*	0,227
A-law based approximation	0,134
CRI approximation $q = 2$	0,079
CRI approximation $q = 3$	0,076
CRI approximation $q = 1$	0,055
CRI approximation $q = 0$	0,019

The empirical experiments reviewed in [26] exhibit a more robust performance with smaller bit widths in weight values than could be inferred from [25], since the statistically simulated minimum bit widths tend to represent worst-case scenarios. Taking into account the well-known robustness of neural networks, it has been assumed in this paper that satisfactory operation of a neural network can be achieved with one bit less than proposed in [25]. A precision of 7 bits corresponds to a E_{max} of 0.39%, see (9), which is regarded as a functional limit on the accuracy of the activation function in network training. With regards to network forward operation, it has been assumed that a precision of 6 bits corresponding to a E_{max} of 0.78%, see (9) is tolerable.

The tentative assumptions in the previous paragraph have to be verified with extensive simulations and real world experiments. The number of hidden layers and connections of a neural network are also significant factors in neural network performance.

4.2 Recommendations

Based on the discussion in the previous Section, the following recommendations can be made:

If the input number range is $[-8,8[$, the best choice is the sig_337p SIG-sigmoid implementation. It has the smallest E_{ave} and E_{max} (Table 6), and requires only 45 logic elements. If area is the most important criterion and training is not required, the sig_336p SIG-sigmoid implementation might the best alternative, as its E_{max} is still tolerable at 0.77% and only 32 LEs are required.

If the input number range is $[-4,4[$, network training probably cannot be performed, as all approximations under comparison have a too high E_{max}. When it comes to network operation, the best alternative is the sig_236p SIG-sigmoid implementation, since it requires only 25 LEs and its E_{max} is still tolerable at 0.77%. The smaller sig_235p SIG-sigmoid implementation with 22 LEs increases E_{max} to 1.51%, which is probably too high.

The logical equations for the sig_337p and the sig_236p SIG-sigmoid implementations are presented in Tables 11 and 12, which can be used with Fig. 2 to deduce the complete operation of both sig_337p and sig_236p.

When other approximations besides SIG-sigmoid are compared, the best choice is the PLAN approximation, as it requires only 39 LEs and has the lowest E_{ave} of 0.59% among the other approximations. This is not surprising, as the internal structure of the PLAN approximation implementation resembles the SIG-sigmoid implementations. The second best choice is the approximation of Alippi and Storti–Gajani, whose E_{ave} of 0.89% underperforms the PLAN approximation, but otherwise these two are quite similar in characteristics.

Table 11: Logical equations for `sig_337p`

Input	$x_5x_4x_3x_2x_1x_0$[1]
AND plane[2]	
p_1	AND$\{x_5, x_2\}$
p_2	AND$\{x_5, x_4\}$
p_3	AND$\{x_5\}$
p_4	AND$\{x_5, x_3\}$
p_5	AND$\{x_4, \overline{x_3}, \overline{x_2}, \overline{x_1}, \overline{x_0}\}$
p_6	AND$\{\overline{x_4}, x_3, \overline{x_2}, \overline{x_1}, \overline{x_0}\}$
p_7	AND$\{\overline{x_4}, \overline{x_3}, x_2, x_1, \overline{x_0}\}$
p_8	AND$\{x_3, \overline{x_2}, x_1, \overline{x_0}\}$
p_9	AND$\{x_4, x_3, x_1, x_0\}$
p_{10}	AND$\{x_4, \overline{x_3}, x_1, x_0\}$
p_{11}	AND$\{x_4, x_2, x_1\}$
p_{12}	AND$\{\overline{x_4}, x_3, x_1, x_0\}$
p_{13}	AND$\{x_3, x_2, x_1\}$
p_{14}	AND$\{x_3, \overline{x_1}, x_0\}$
p_{15}	AND$\{x_4, x_2, x_0\}$
p_{16}	AND$\{x_4, \overline{x_3}, \overline{x_2}, x_1\}$
p_{17}	AND$\{\overline{x_4}, \overline{x_3}, \overline{x_2}, x_1\}$
p_{18}	AND$\{\overline{x_4}, x_3, x_2\}$
p_{19}	AND$\{x_4, x_3, x_2\}$
p_{20}	AND$\{\overline{x_3}, x_2\}$
p_{21}	AND$\{\overline{x_4}, x_2, x_1, x_0\}$
p_{22}	AND$\{\overline{x_4}, x_2, \overline{x_1}, x_0\}$
p_{23}	AND$\{x_4, \overline{x_3}, x_2, \overline{x_1}\}$
p_{24}	AND$\{x_4, \overline{x_2}, \overline{x_1}, x_0\}$
p_{25}	AND$\{\overline{x_4}, \overline{x_3}, \overline{x_2}, x_0\}$
p_{26}	AND$\{x_4, x_3\}$
p_{27}	AND$\{x_4, x_3, \overline{x_2}, \overline{x_0}\}$
Output	$y_6y_5y_4y_3y_2y_1y_0$[3]
OR plane[4]	
y_6	'1'[5]
y_5	OR$\{p_3, p_5, p_8, p_{10}, p_{11}, p_{12}, p_{13}, p_{14}, p_{15}, p_{16}, p_{18}, p_{23}, p_{24}, p_{26}\}$
y_4	OR$\{p_3, p_5, p_6, p_{10}, p_{11}, p_{15}, p_{16}, p_{20}, p_{24}, p_{26}\}$
y_3	OR$\{p_3, p_6, p_{11}, p_{13}, p_{17}, p_{18}, p_{21}, p_{26}\}$
y_2	OR$\{p_3, p_6, p_7, p_9, p_{12}, p_{13}, p_{16}, p_{19}, p_{23}, p_{25}\}$
y_1	OR$\{p_3, p_6, p_7, p_8, p_{12}, p_{21}, p_{22}, p_{23}, p_{24}, p_{27}\}$
y_0	OR$\{p_1, p_2, p_4, p_5, p_7, p_8, p_{10}, p_{13}, p_{14}, p_{15}, p_{18}, p_{22}\}$

[1] Input is positive, x_6 would be redundant sign bit
[2] See also Fig. 2
[3] There is no sign bit
[4] See also Fig. 2
[5] $y \geq \frac{1}{2}$

The approximation of Zhang *et al.* needs too many LEs as it requires a multiplier. This is also corroboration for excluding other approximations that require a multiplier (Section 2.4). The CRI approximation is slow and not very precise and the A-law based approximation is too inaccurate, especially when $x \approx \pm 4$.

5 Conclusions and future work

Comparing published digital sigmoid function approximations, it was concluded that a novel purely combinational implementation called SIG-sigmoid developed at the Signal Processing Laboratory of the Helsinki University of Technology is fastest, smallest and most accurate. This assessment was based on implementing several published sigmoid function approximations in both MATLAB and VHDL and comparing their quality factor defined by (17)

If the input number range is $[-4,4[$, the best choice is the `sig_236p` SIG-sigmoid implementation, and if the input number range is $[-8,8[$, the best choice is the `sig_337p` SIG-sigmoid implementation.

FPGAs open new possibilities for implementing ANNs, since the number of neurons, layers and interconnections can be varied dynamically. Also the time-consuming ANN training benefits from a reprogrammable implementation.

Future work involves estimating the maximum size of ANNs in modern FPGAs. The main points are the size and parameterisability of multipliers and the number of inter-layer interconnections. The first defines mainly the required area resources and the second defines the required routing resources.

Table 12: Logical equations for `sig_236p`

Input	$x_4x_3x_2x_1x_0$[6]
AND plane[7]	
p_1	AND$\{x_4, x_3, x_2\}$
p_2	AND$\{x_4\}$
p_3	AND$\{\overline{x_4}, x_3, \overline{x_2}, \overline{x_1}, \overline{x_0}\}$
p_4	AND$\{x_4, \overline{x_3}, \overline{x_2}, x_0\}$
p_5	AND$\{x_4, x_2, x_0\}$
p_6	AND$\{x_4, x_3, x_1\}$
p_7	AND$\{\overline{x_4}, x_2, x_1, \overline{x_0}\}$
p_6	AND$\{x_4, x_2, x_1\}$
p_9	AND$\{\overline{x_4}, \overline{x_3}, \overline{x_2}, x_1\}$
p_{10}	AND$\{x_4, x_3, x_2, x_1\}$
p_{11}	AND$\{x_4, \overline{x_2}, x_1\}$
p_{12}	AND$\{\overline{x_4}, x_3, x_2, x_0\}$
p_{13}	AND$\{x_4, x_3, \overline{x_2}, \overline{x_1}\}$
p_{14}	AND$\{\overline{x_4}, x_3, x_1, x_0\}$
p_{15}	AND$\{\overline{x_4}, \overline{x_3}, \overline{x_1}, x_0\}$
p_{16}	AND$\{x_4, \overline{x_3}, x_2, \overline{x_1}, \overline{x_0}\}$
p_{17}	AND$\{x_3, x_0\}$
p_{18}	AND$\{x_2, x_1, x_0\}$
p_{19}	AND$\{x_3, x_1\}$
p_{20}	AND$\{\overline{x_2}, x_1, x_0\}$
p_{21}	AND$\{\overline{x_3}, x_2\}$
p_{22}	AND$\{x_3, x_2\}$
Output	$y_5y_5y_4y_3y_2y_1y_0$[8]
OR plane[9]	
y_5	'1'[10]
y_4	OR$\{p_2, p_4, p_{17}, p_{19}, p_{22}\}$
y_3	OR$\{p_2, p_3, p_4, p_{10}, p_{21}\}$
y_2	OR$\{p_3, p_5, p_6, p_8, p_9, p_{13}, p_{18}, p_{22}\}$
y_1	OR$\{p_1, p_3, p_7, p_{11}, p_{15}, p_{16}, p_{19}, p_{20}\}$
y_0	OR$\{p_3, p_4, p_7, p_{10}, p_{12}, p_{13}, p_{14}, p_{16}, p_{18}\}$

[6] Input is positive, x_5 would be redundant sign bit
[7] See also Fig. 2
[8] There is no sign bit
[9] See also Fig. 2
[10] $y \geq \frac{1}{2}$

6 References

1 Krips, M., Lammert, T., and Kummert, A.: 'FPGA implementation of a neural network for a real-time hand tracking system'. Proc. 1st IEEE Int. Workshop on Electronic Design, Test and Applications (DELTA), Christchurch, New Zealand, 29–31 January 2002, pp. 313–317

2 Omondi, A.R., and Rajapakse, J.C.: 'Neural networks in FPGAs'. Proc. 9th Int. Conf. on Neural Information Processing (ICONIP), Singapore, 18–22 November 2002, vol. 2, pp. 954–959

3 Maya, S., Reynoso, R., Torres, C., and Arias–Estrada, M.: 'Compact spiking neural network implementation in FPGA'. Proc. 10th Int. Conf. on Field Programmable Logic and Applications (FPL), Villach, Austria, 27–30 August 2000, pp. 270–276

4 Moore, G.E.: 'Cramming more components onto integrated circuits', *Electron.*, 1965, **38**, (8), pp. 114–117

5 Zurada, J.M.: 'Introduction to artificial neural systems' (West Publishing Co., St. Paul, MN, 1992), pp. 25–52

6 Zhang, M., Vassiliadis, S., and Delgago–Frias, J.G.: 'Sigmoid generators for neural computing using piecewise approximations', *IEEE Trans. Comput.*, 1996, **45**, (9), pp. 1045–1049

7 Parhami, B.: 'Computer arithmetic: Algorithms and hardware designs', (Oxford University Press, New York, 2000), p. 282

8 Myers, D.J., and Hutchinson, R.A.: 'Efficient implementation of piecewise linear activation function for digital VLSI neural networks', *Electron. Lett.*, 1989, **25**, (24), pp. 1662–1663

9 Alippi, C., and Storti–Gajani, G.: 'Simple approximation of sigmoidal functions: realistic design of digital neural networks capable of learning'. Proc. IEEE Int. Symp. on Circuits and Systems, Singapore, 11–14 June 1991, pp. 1505–1508

10 Amin, H., Curtis, K.M., and Hayes–Gill, B.R.: 'Piecewise linear approximation applied to nonlinear function of a neural network', *IEE Proc. Circuits, Devices Sys.*, 1997, **144**, (6), pp. 313–317

11 Basterretxea, K., and Tarela, J.M.: 'Approximation of sigmoid function and the derivative for artificial neurons', in Mastorakis, N. (Ed.): 'Advances in neural networks and applications' (WSES Press, Athens, 2001), pp. 397–401

12 Gajski, D.D.: 'Principles of digital design' (Prentice Hall, Englewood Cliffs, NJ, 1997), pp. 62–162

13 Quine, W.V.: 'A way to simplify truth functions', *Am. Math. Mon.*, 1955, **62**, pp. 627–631

14 McCluskey, E.J.: 'Minimization of boolean functions', *Bell Sys. Tech. J.*, 1956, **35**, (5), pp. 1417–1444

15 Dagenais, M.R., Agarwal, V.K., and Rumin, N.C.: 'McBoole, a new procedure for exact logic minimization', *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, 1986, **5**, (1), pp. 229–238

16 Ashenden, P.J.: 'The designer's guide to VHDL' (Morgan Kaufmann, San Francisco, CA, 2001, 2nd edn.)

17 Synplify Pro®, User Guide and Tutorial, Synplicity Inc., 2002

18 APEX II programmable logic device family datasheet, www.altera.com/literature/ds/ds_ap2.pdf, accessed on 20th August 2003

19 Faiedh, H., Gafsi, Z., and Besbes, K.: 'Digital hardware implementation of sigmoid function and its derivative for artificial neural networks'. Proc. 13th Int. Conf. on Microelectronics, Rabat, Morocco, 29–31 Oct. 2001, pp. 189–192

20 Kwan, H.K.: 'Simple sigmoid-like activation function suitable for digital hardware implementation', *Electron. Lett.*, 1992, **28**, (15), pp. 1379–1380

21 Sammut, K.M., and Jones, S.R.: 'Implementing nonlinear activation functions in neural network emulators', *Electron. Lett.*, 1991, **27**, (12), pp. 1037–1038

22 Smith, M.J.S.: 'Application-specific integrated circuits' (Addison–Wesley, Boston, MA, 1997), p. 2

23 Meyer–Baese, U.: 'Digital signal processing with field-programmable gate arrays' (Springer-Verlag, Berlin, 2001), pp. 6–11

24 Basterretxea, K., Tarela, J.M., and del Campo, I.: 'Digital design of sigmoid approximator for artificial neural networks', *Electron. Lett.*, 2002, **38**, (1), pp. 35–37

25 Holt, J.L., and Hwang, J-N.: 'Finite precision error analysis of neural network hardware implementations', *IEEE Trans. Comput.*, 1993, **42**, (3), pp. 281–290

26 Moerland, P.D., and Fiesler, E.: 'Neural network adaptations to hardware implementations', Chapter E1.2 in 'Handbook of neural computation' (IOP Publishing Ltd. and Oxford University Press, New York, 1997)