International Journal of Modelling and Simulation, Vol. 16, No. 3, 1996

A FRACTIONAL POWERS-OF-TWO NUMBER SYSTEM FOR DIGITAL NEURAL NETWORKS

J.J. Cupal, B.M. Wilamowski, R.S. Sandige, and J.J. Miller*

Abstract

A new arithmetic number system based on fractional powers of two is proposed. In this system, the weight of each bit position is a power of $2^{1/n}$, where n = 1,2,3,4 ... Multiplication or division by $2^{1/n}$ can be accomplished by simple shifts of the input data. Although not as simple as in conventional twos complement binary, addition and subtraction can be accomplished using simple ALU structures. Details of how this is accomplished are given, as well as an implementation of hardware structure to perform multiply-accumulate operations commonly found in neural nets and FIR filters. Applications of this arithmetic system in binary neural networks are also given.

Key Words

Neural networks, hardware, arithmetics

1. Introduction

Many recent achievements have been recorded in the relatively new area of research in neural networks. New learning algorithms, new system structures, and many useful applications have been found. Neural network hardware implementations are also a current focus, and it is obvious that better hardware implementation could enhance neural network development. A neuron must perform a series of multiply-accumulate operations, which can be done in analog hardware or in digital computer systems. For many applications, the input signals are in digital format and the analog technique of implementing a neural network is prohibitive. Therefore, an all-digital approach must be taken [1–3]. Because the multiply instructions in a microprocessor require many system clocks, neural networks implemented in a digital computer operate relatively slowly. One can speed up the multiply instruction with the addition of dedicated hardware, but this requires extensive real estate in silicon. One can also use logarithmic arithmetic [4, 5], resulting in fast multiplication (simple addition of the logs), but addition and subtraction are more difficult to do. Another approach is to use the shift operation instead of multiplication [6]. In the case of conventional twos complement binary systems, such multiplication is coarse. Only multiplicands with a value of powers of two are possible, that is, 2, 4, 8, 16... or 0.5, 0.25, 0.125...

The purpose of this paper is to introduce a new number system to perform the arithmetic operations required in sum-of-products algorithms. This system seems ideally suited for applications where fast, but not necessarily accurate, multiplications are required, as is the case for a neuron in a neural network. Instead of binary numbers in base 2, numbers in base $2^{1/n}$ are introduced. Using this approach, fast multiplication and division are possible by simple shifting of the input data. Section 2 introduces the arithmetic system, some neural net applications are given in section 3, and a hardware structure for a $\sqrt{2}$ number system is given in section 4.

2. Concept of the Arithmetic

Consider a neuron that performs arithmetic using a different number system. This number system is to be based upon weights of $2^{1/n}$ (where $n = 2, 3, 4...$) instead of 2^1, as in a conventional binary number system. The use of such a number system has lead to the design of simple neurons with interesting features. For example, if $n = 2$, this new number system is based upon powers of $\sqrt{2}$. In this case, the representation for the number $7.4142_{10} = \sqrt{2} + 6$ or $(10110)_{\sqrt{2}}$ as shown in fig. 1.

In this numbering system, multiplication or division by powers of $\sqrt{2}$ corresponds to the shift left or shift right operation by one position as shown in fig. 2. Multiplication by $(\sqrt{2})^3 = 2\sqrt{2} = 2.8284_{10}$ of the number 7.4142_{10} corresponds to a shift left by three places.

Addition (or subtraction) of two numbers in base $\sqrt{2}$ is similar to binary addition (or subtraction), with the difference that the carry coming out of the addition of any two bits is added in two bit positions to the left as shown in fig. 3. If the base is $2^{1/n}$, the carry is shifted by n. Because of this somewhat unusual feature, it is possible to separate

$\sqrt{2}^7$ or $8\sqrt{2}$	$\sqrt{2}^6$ or 8	$\sqrt{2}^5$ or $4\sqrt{2}$	$\sqrt{2}^4$ or 4	$\sqrt{2}^3$ or $2\sqrt{2}$	$\sqrt{2}^2$ or 2	$\sqrt{2}^1$ or $\sqrt{2}$	$\sqrt{2}^0$ or 1	==	value
0	0	0	1	0	1	1	0		$\sqrt{2}$+6=7.4142

Figure 1. The representation of $7.4142_{10} = \sqrt{2} + 6$ in base $\sqrt{2}$ number system.

* University of Wyoming, Department of Electrical Engineering, Laramie, WY 82071 USA; E-mail: jcupal@uwyo.edu

(paper no. 1993-134)

$8\sqrt{2}$	8	$4\sqrt{2}$	4	$2\sqrt{2}$	2	$\sqrt{2}$	1	==	value
0	0	1	0	1	1	0	0		$6\sqrt{2}+2=10.4853$
0	0	0	0	1	0	1	1		$3\sqrt{2}+1=5.2426$

Figure 2. Multiplication and division by $\sqrt{2}$ as done with shifts in base $\sqrt{2}$ number system. The original value of the number was $7.4142_{10} = \sqrt{2} + 6$, as in fig. 1.

a number in the $\sqrt{2}$ number system in two halves, one representing the square root terms and the other representing the integer terms. Addition (or subtraction) can be done on each half in parallel, making this a faster process than adding the whole value, as in conventional binary adders. Fig. 4 shows an example of the addition operation.

represents the number of left or right shifts. This would reduce the memory requirements to store the weights of any given structure. This, plus the fact that fast multiplication can be performed by simple shifts of the input values, make this new arithmetic an interesting technique for applications in neural networks. The examples above

	$8\sqrt{2}$	8	$4\sqrt{2}$	4	$2\sqrt{2}$	2	$\sqrt{2}$	1	value
A	0	1	1	0	1	0	1	1	$7\sqrt{2}+9$
B	0	0	0	0	0	1	1	1	$\sqrt{2}+3$
carry	1	-	1	1	1	1	-	-	
A+B	1	1	0	1	0	0	0	0	$8\sqrt{2}+12$

Figure 3. Addition of two numbers in base $\sqrt{2}$ number system.

A									+	B									=	A + B								
0		1		1		1				0		0		0		1				1		0		0		0		
	1		0		0		1				0		0		1		1				1		1		0		0	
$7\sqrt{2} + 9 = 18.8995$										$3 + \sqrt{2} = 4.4142$										$8\sqrt{2} + 12 = 23.3137$								

Figure 4. Addition of two numbers in base $\sqrt{2}$ number system. The top row represents the square root terms, the bottom the integer terms. Addition is accomplished by adding the separate representations.

In the example of fig. 5 showing the subtraction of $(00000111)_{\sqrt{2}}$ or $(4.4142)_{10}$ from $(01101011)_{\sqrt{2}}$ or $(18.8995)_{10}$, each 8-bit number can be split into two 4-bit components. Subtraction is accomplished by taking the twos complement of both components of the subtrahend and adding these to the corresponding components of the minuend. Note that the number $(00000111)_{\sqrt{2}}$ has two components: $(0001)_2 \times \sqrt{2}$ and $(0011)_2$ with the respective twos complements $(1111)_2 \times \sqrt{2}$ and $(1101)_2$. Therefore the twos complement in base $\sqrt{2}$ arithmetic of the number $(00000111)_{\sqrt{2}}$ is $(11111011)_{\sqrt{2}}$, which is not the same as the twos complement in conventional binary logic.

Because multiplication or division is done by shifts, the weights of a neuron could be stored as a number that

show arithmetics based upon $2^{1/n}$ with $n = 2$. By using larger values of n, we can reduce the "coarseness" of the weights given in the examples.

3. Applications of the Arithmetic in Neural Networks

To investigate the usefulness of this arithmetic system, several neural nets were implemented and their behaviour was studied. Simulations were run on a 486 PC, using single-precision floating point arithmetic. The applications given here are Boolean in nature (i.e., a parity generator) with bipolar desired output values. In all cases, neurons with the bipolar sigmoid activation function given by equation (1) were used:

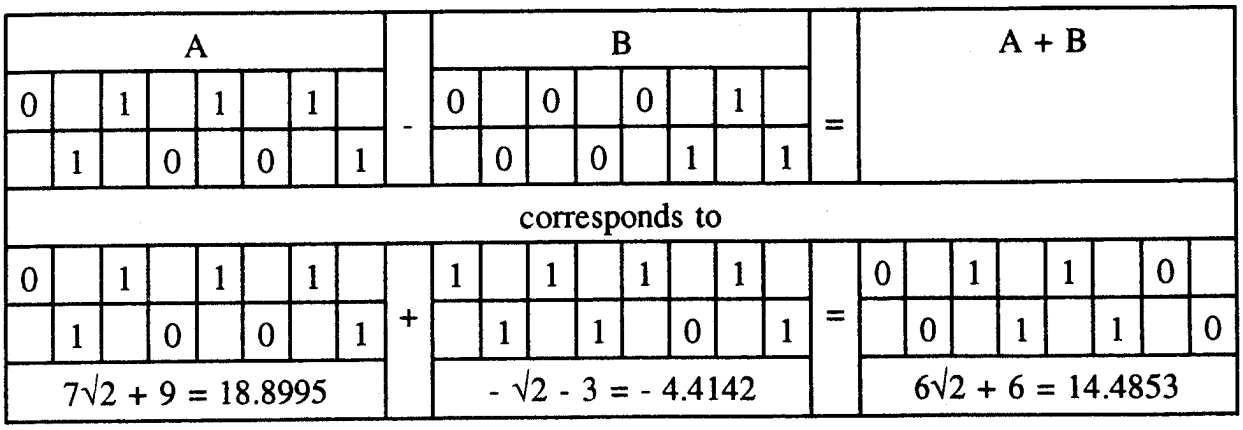


Figure 5. Subtraction of two numbers in base $\sqrt{2}$ number system.

$$f(net) = \frac{2}{1 + \exp(-\lambda net)} \qquad (1)$$

where *net* is the sum of the weighted inputs of the neuron. In this study, the gain factor of this activation function, $\lambda/2$, was chosen to be equal to one.

The networks were trained offline by the conventional back-propagation algorithm using floating point arithmetic. The networks were allowed to converge from randomly selected starting weights to some predetermined final global error. The global error was defined as the RMS error for all of the outputs for any given input vector, averaged for all of the input vectors used to train the neural net.

Once the networks were trained, each neuron was implemented in the $2^{1/n}$ arithmetic system where n could vary from 1 to 16. The desired weights were rounded to the nearest weight in the given number system. The *net* term was then calculated and fed into the activation function. In these simulations, the activation function was implemented in full precision floating point. In actual hardware, this function would be implemented within a ROM based lookup table.

Three different applications were investigated. The first two were Boolean logic functions implemented in a neural net. These had bipolar inputs as well as bipolar desired outputs. The first of these was an odd parity generator, with four inputs, four neurons in the hidden layer, and one output neuron. A training set of 16 input patterns was used. The second was a multiple-output logic function generator. It had four inputs, three neurons in the hidden layer, and three output neurons. In this case, a training set of nine patterns was used to train the net. The functions generated were typical of Boolean equations implemented in hardware.

Fig. 6 shows the global error for neural networks implemented in the $2^{1/n}$ number system for the first two cases. Two plots are shown for each case: one when the networks were trained using floating point arithmetic to an error of 0.1; the other for an error of 0.01. Ten training runs were made, starting with randomly chosen weights. In the process of converting the weights to the values allowed in a particular number system, often fewer than ten distinct sets of weights resulted for a particular implementation. The curves show the data scatter, as well as the average error found for each implementation.

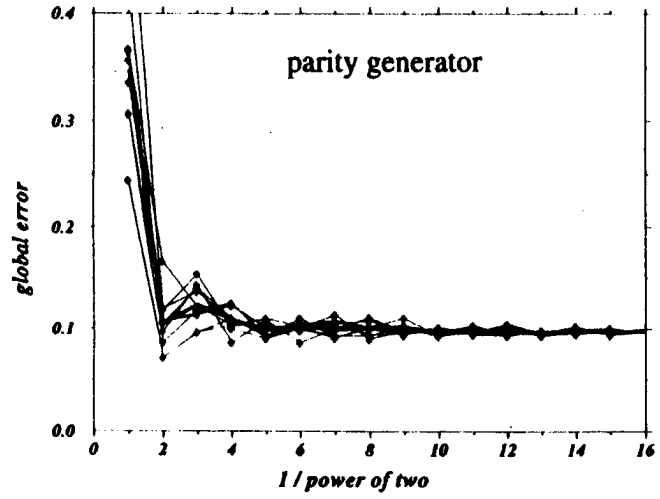
The results shown in fig. 6 indicate that a neural network implemented in a conventional binary number system ($n = 1$) has considerable error. If these networks are implemented in the square root ($n = 2$) number system, this error drops dramatically, and drops even further in the cube root ($n = 3$) system. For higher values of n, the error approaches the error to which the system was trained. Because implementations with higher values of n require additional hardware, it is thought that sufficiently accurate results could be obtained with either the square root or cube root number system.

In the third example, a network was designed to determine if points on the x, y plane are within a unit circle centred at the origin. If a point was within a unit circle, the desired output was +1; otherwise it was −1. The training set of this network consisted of 90 randomly generated points within the range $-2, -2$ and 2,2.

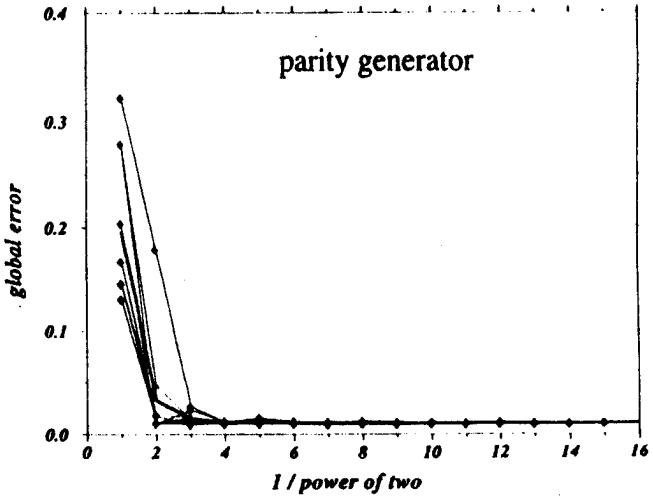
A solution for this network was obtained if the network had two neurons in the input layer, five in the hidden layer, and one in the output layer. As before, the network was trained using floating point arithmetic. Fig. 7(a) shows the decision surface of this neural net when implemented in floating point. Ideally, it should be a cylinder of radius 1 and height 1. Figs. 7(b), (c), and (d) show the same network when it is implemented in the fractional powers of two arithmetic with $n = 1$, $n = 2$, and $n = 3$ respectively. Again the results show that the implementation in the conventional binary number system has considerable error, but the square root and cube root system give a good representation of the floating point system.

4. Hardware Implementation of the Arithmetic

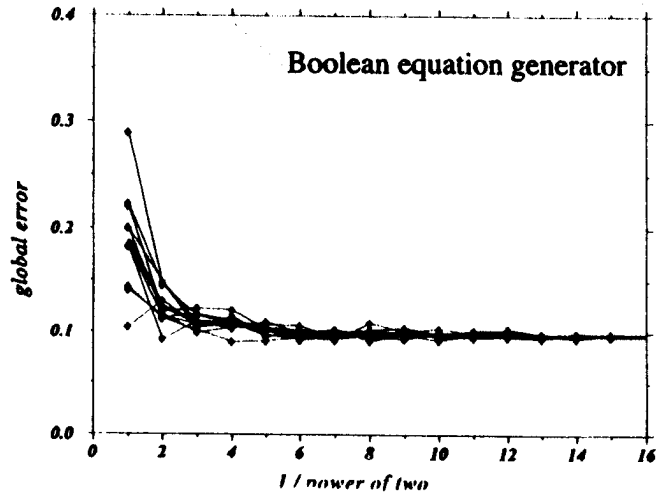
With the proposed arithmetic, multiplication or division can be replaced with shift left or shift right operations. Likewise, addition and subtraction operations can be performed similarly as in binary arithmetic. The only difference is the necessity of independent operations on the n components of the original number. These computations



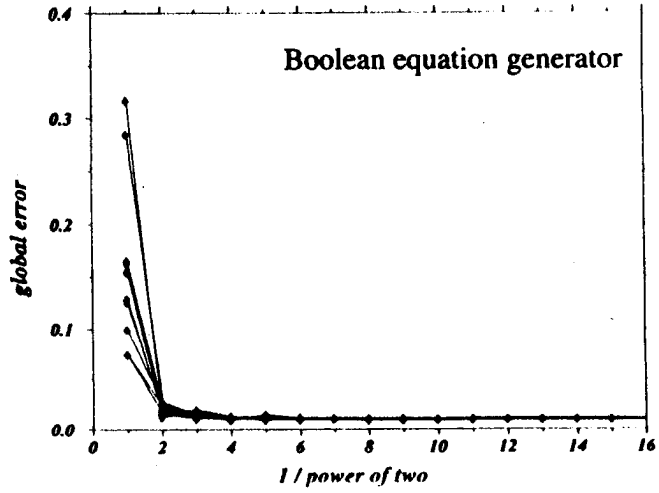
(a)



(b)

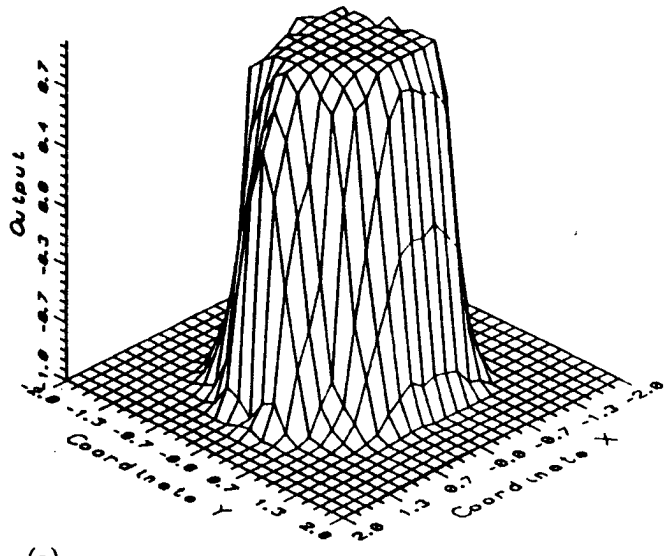


(c)

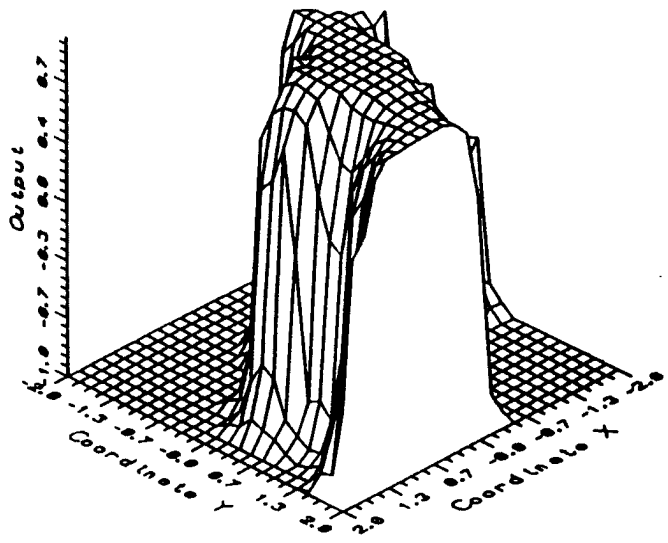


(d)

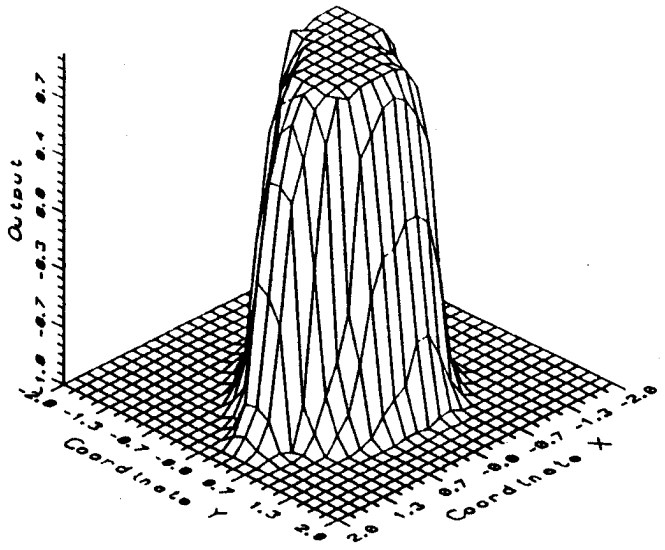
Figure 6. Global error for neural network implemented in the fractional power of two number system versus the reciprocal power of two (n). The figure shows the error for neural networks trained from 10 different randomly chosen initial weights. The heavy line is a plot of the average error. Results are shown for: (a) parity generator trained to an error of 0.1; (b) parity generator trained to an error of 0.01; (c) function generator trained to an error of 0.1; and (d) function generator trained to an error of 0.01.



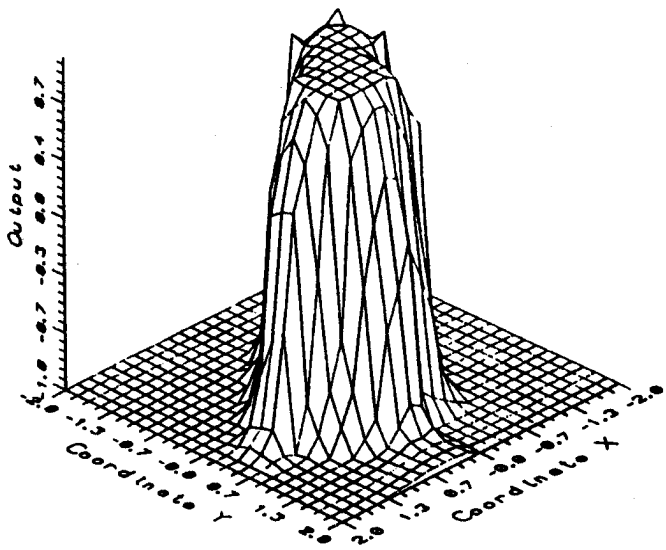
(a)



(b)



(c)



(d)

Figure 7. Decision surface for a neural network that determines if points in the x, y plane are within a unit circle. Solutions are shown for (a) floating-point precision and fractional arithmetic with (b) $n = 1$, (c) $n = 2$, and (d) $n = 3$.

are best performed in parallel using n separate ALU units. Thus, it is desirable to use as low a value for n as possible, yet still produce sufficiently accurate results.

There are several hardware configurations that could implement the $2^{1/n}$ arithmetic discussed above. It is obvious that one shift register and at least two simple ALUs that perform addition or subtraction (using twos complement) are required for the $\sqrt{2}$ arithmetic shown in the examples. One such hardware model is shown in fig. 8. It is intended to perform a multiply-accumulate operation such as would be required in a neural network or FIR filter. The input values are twos complement binary numbers, in this case 16 bits wide. The weights can be represented in a 6-bit number, in a form of sign-magnitude format. Actually, one bit is for the sign, another for the shift direction (multiplication or division), and 4 bits are for the shift count.

The input value is loaded into a 16-bit shift register with the shift direction controlled by the direction bit of the weight and the shift amount controlled by the most significant 3 bits of the shift count. Shifting is done as in normal twos complement data shifts—left shifts fill with zeros and right shifts fill by sign extension. The result of this shift process multiplies the input value by the unsigned fixed weight.

The output of the shift register is fed into one of two add/accumulate structures by the action of two 16-bit multiplexers controlled by the least significant bit of the shift count. If this bit is a one, indicating multiplication/division by $\sqrt{2}$, then the value is added to the accumulator holding the $\sqrt{2}$ terms (represented as *Result_sq_rt* in fig. 8). If this bit is a zero, indicating multiplication/division by integer values, then the value is added to the *Result_int* accumulator. Actually, the ALUs required are somewhat simple, performing either addition or twos complement subtraction as controlled by the sign of the weight.

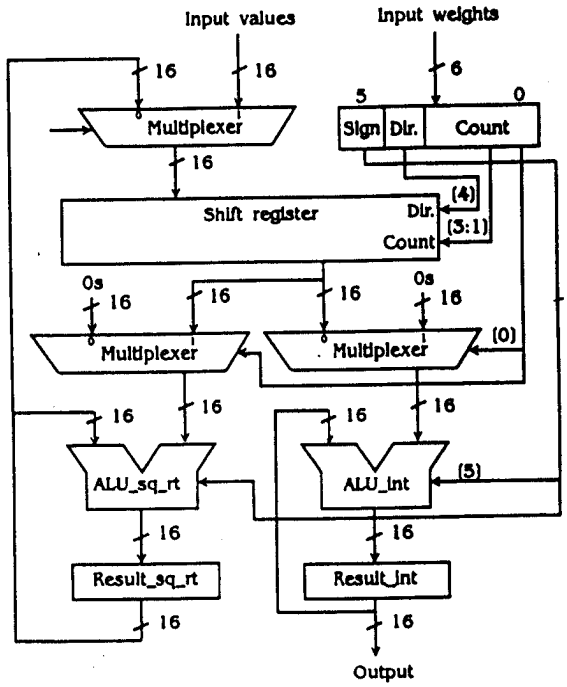


Figure 8. A hardware model of an arithmetic logic circuit to perform multiply-accumulate algorithms in the $\sqrt{2}$ number system.

The hardware controller continues the multiply-accumulate operations until all input values are processed. At the conclusion, the result accumulators hold the square root and integer sums. Because it is desirable to have standard binary outputs, it is necessary to multiply the value in the *Result_sq_rt* accumulator by $\sqrt{2}$ and add this result to the *Result_int* accumulator. This is done by passing the value in the *Result_sq_rt* accumulator back through the multiply-accumulate structure with appropriate weights to approximate multiplication by $\sqrt{2}$ ($\sqrt{2} = 1 + 1/4 + 1/8 + 1/32$).

The hardware structure shown in fig. 8 has been implemented in Verilog HDL. A 20 tap FIR filter structure was demonstrated using all fractional coefficients. That is, the coefficients were not formed into integers as is often the case when using integer multiplications within a microprocessor. As the coefficients are always fractional, the direction bit in each weight is not required, meaning that the weight could be stored in a 5-bit number or, perhaps more appropriately, the extra bit could be used for storage of a 5-bit shift count.

It is possible to speculate as to the speed of the structure shown in fig. 8. Assuming the shift register can do multiple shifts in one clock cycle, it appears that a multiply-accumulate operation could be done in two clocks—one to load the input value and weight, the other to shift the correct amount. The other devices (multiplexers, ALUs) are combinational in nature except for the result registers that are clocked to store the accumulated total as the new input values are loaded. To perform the 20 tap FIR filter mentioned earlier, 48 clocks are required ($20 * 2$ plus 8 for conversion back to integer values). In comparison, a conventional 16-bit by 16-bit multiply requires 16 clock cycles. In this case, a 20 tap FIR filter would require 340 clocks, assuming one clock to load each data pair and 16 clocks to perform the multiply. Of course, the actual times are dependent upon the actual structure inplemented in hardware.

The structure shown in fig. 8 could be used in a FIR filter or in a neural network structure if an activation function generator is added. For faster operation, this activation function would likely be implemented as a lookup table in a ROM. If the application involves binary decisions, this lookup table could be quite simple.

Parallel processing, which is natural for addition or subtraction operations, also can be extended for an entire layer. The structure of a neural network is such that all signals in one neural layer can be processed at the same time, thus allowing for parallel computation of all neurons within a layer.

5. Conclusion

This paper has introduced a new number system based upon fractional powers of two. It was shown that addition and subtraction in this system could be done using simple ALU structures, and multiplication and division by left or right shifts of the input data. By introducing this base $2^{1/n}$ arithmetic, we significantly reduce the coarseness of weights in comparison to normal base 2 arithmetic where multiplication is substituted by shift operations. Simulations show that for simple, binary-type neural networks, a structure based upon the $\sqrt{2}$ number system produces sufficiently accurate results.

The proposed arithmetic has several advantages: (1) Multiple ALU units can provide parallel processing, resulting in fast operation. (2) Because multiplication can be done by data shifting, multiplication of an input value by a constant weight can be accomplished faster than conventional multiply operations within a computer system. In fact, it may be possible to use a serial communication technique between neurons to further enhance multiplication speed and reduce the number of required interconnections. (3) For most practical neuron applications, only 4 to 6 bits of memory are required to store each weight value.

From the hardware implementation given in this paper, it is obvious that an additional ALU and accumulator are required for higher values of n. Our results show that sufficiently accurate results can be obtained in neural network applications with $n = 2$, that is, a $\sqrt{2}$ number system.

References

[1] A.F. Murray & A.W. Smith, Asynchronous VLSI neural networks using pulse-stream arithmetic, *IEEE J. Solid-State Circuits, 23*(3), 1988, 688–697.

[2] M. Mumford, D. Andes, & L. Kern, The Mod 2 neurocomputer system design, *IEEE Trans. on Neural Networks, 3*(3), 1992, 423–430.

[3] G. Pechanek, S. Vassiliadis, & J. Delgago-Frias, Digital neural emulators using tree accumulation and communication structures, *IEEE Trans. on Neural Networks, 3*(6), 1992, 934–940.

[4] M. Arnold, T. Bailey, J. Cowles, & J. Cupal, Implementing back propagation neural nets with logarithmic arithmetic, *Proc. Int. AMSE Conference on Neural Networks,* vol. 1, San Diego, May 29–31, 1991, 75–86.

[5] M. Arnold, T. Bailey, J. Cowles, & J. Cupal, Redundant logarithmic arithmetic, *IEEE Trans. on Computers, 39*(8), 1990, 1077–1086.

[6] M. Marchesi, G. Orlandi, F. Piazza, & A. Uncini, Fast neural networks without multipliers, *IEEE Trans. on Neural Networks, 4*(2), 1993, 53–62.

Biographies

Jerry Cupal is currently teaching in the Electrical Engineering Department at the University of Wyoming. He has been at this university for ten years, and is an Associate Professor. His research interests include applications of digital and microprocessor systems to a variety of problems, including neural networks.

Bodgan Wilamowski is currently on sabbatical leave from the Department of Electrical Engineering at the University of Wyoming. He has been at this university for eight years, and is a Full Professor. He is involved in research of neural and fuzzy systems, as well as solid state electronic devices.

Richard Sandige is a Full Professor in the Department of Electrical Engineering at the University of Wyoming. He has been at the University of Wyoming for six years, previously being employed at Hewlett Packard in Fort Collins, CO. His research interests include the design of digital systems.

Jennifer Miller is currently employed at Hewlett Packard Corp. in Greely, CO. She contributed to this resarch when she was an undergraduate in the Department of Electrical Engineering at the University of Wyoming.