Blending Methodologies for Optimizing Fuzzy Inference Engine Designs

Rob Chapman, Adam F. Gobi and Witold Pedrycz
Department of Electrical and Computer Engineering
University of Alberta
Edmonton, AB T6G 2V4
{rchapman,gobia,pedrycz}@ece.ualberta.ca

***Abstract* -- The software paradigm of writing sequential programming tasks executed on a single processor has pervaded computers since their dawning. In spite of progress, sequential execution of certain algorithms remain limited by this paradigm. In particular, fuzzy control systems involve fuzzy set operations which require significant amounts of vector and matrix computation. This computation can be considered an inherently parallel task, and performing these operations in software results in inefficient execution, severely limiting the use of fuzzy operations in real-time systems where fast responses are required. This paper will explore solutions to this problem using software, hardware and finally a hybrid approach with a proposed computing architecture and platform known as a granular computer (GC).**

I. Introduction

In the last decade there have been many proposed solutions to speed up fuzzy set operations. These include:

- algorithmic tuning [3][4] such as data driven execution to minimize the number of void computations
- enhanced microcontrollers [1][2][3] with fuzzy hardware and fuzzy instructions or fuzzy co-processors
- parameterized hardware [4][5][6]

While each of these proposed solutions offer benefits, they all have limitations in some way [3]. These include disadvantages in speed, flexibility, support for other system tasks, or the number of chips and support hardware required. In this paper, we propose a different approach by blending software and hardware paradigms.

Our novel approach attempts to address these limitations by modeling an FPGA development platform, which offers an excellent environment for exploration of fuzzy operations, as a familiar software programming environment. The FPGA platform serves as a fast, compact single chip solution, allowing very powerful integration of hardware with software with the sizes commercially available today and with a big nod to the future. The microprocessor has had its peak and the next rise in computational power will be from FPGA-like chips.

Conventionally programmed in a hardware language, FPGA programming environments have started to offer embedded processors. Xilinix offers up to four 400MHz, embedded IBM PowerPC 405 Processor hard cores or a soft processing solution with the MicroBlaze™ core. Altera offers the equivalent with an ARM922T™ processor, and the Nios® soft RISC processor. While this integrates the software programming environment with an FPGA, it does little to take advantage of it. It is not unlike having a prebuilt castle which you can decorate the rooms, but you can not easily change the number of rooms, turrets or add other novel structures.

Our development platform is a granular computer (GC): a flexible architecture and methodology for combining software programs and hardware engines in programmable hardware. This ability to blend the domains of software and hardware, as well as program with data, will be exploited later in the paper.

The GC model for programming is a simplified, parameterized processor with no instruction set. As part of a design, instructions and instruction sequences, known as primitives, are either created or included from a library set, and are used to construct sequential programs. For speeding up algorithms which have parts that can be parallelized or would benefit from specific hardware support, new parts can be added to the processor components and then controlled through new operations. Even though there are no math operations in hardware, by using specific logic operations in parallel and in sequence, addition, subtraction, multiplication and division can be simulated but they do cost more in cycles than on a microcontroller with dedicated math hardware.

This paper explores architectures and ideas encompassed in implementing a fuzzy interface and fuzzy logic operators with the application being fuzzy control. Using these concepts, we demonstrate how one would use the GC to go about building a fuzzy inference engine. Results from experiments with our fast fuzzy inference engine are presented, with comparisons to industrial platforms offering similar functionality. Our aim is to have a fuzzy inference engine that offers an unparalleled combination of optimal speed, flexibility, and size.

II. Fuzzy Inference Realization Options

A fuzzy inference engine consists of three fundamental parts[11]:

1. fuzzification - a single crisp input value is characterized by expressing membership in a family of fuzzy sets
2. rule evaluation - logical inferences are made from the fuzzified inputs using t-norms and t-conorms to create fuzzy outputs
3. defuzzification - fuzzy outputs are coalesced into singular crisp outputs

A. Realization Strategies

Given software and hardware designs, there are three basic ways to realize the fuzzy inference components:

- Software - One approach is to define the entire fuzzy engine in software. This is appealing in that the design is flexible and quick to turn around and test. It also makes use of the powerful software programming paradigm.
- Hardware - Alternatively, a hardware solution [5][6][8] can be used where the engine is described using hardware components and the parameters are used to adapt it to the specific problem. This, however, is usually supplemented with a microprocessor to take care of all the other tasks required. This approach allows algorithms which operate in parallel to be implemented in parallel, creating a significant speed advantage over the software approach.
- Hybrid - The best approach is a hybrid approach which involves integration of hardware for speed and software for complexity management. This means mixing a processor with support hardware.

1) Fuzzy Enhanced Microcontroller

Eight years ago, Motorola introduced an augmented microprocessor solution with the production of the 68HC12 [1][2]. Their projections indicate they will have sold a billion of these processors in the next five years.

With the HC12 bringing fuzzy logic into the mainstream embedded market through the use of dedicated on-board logic and an augmented instruction set, it makes a good reference point for alternatives when parameterization is insufficient and a change in structure or embedded algorithm is necessary to solve more complex problems, increase speed, increase integration or change precision.

2) Granular Computer

Software can offer precision, integration and complexity management but at a sacrifice of speed [3]. By moving one level deeper in layers of programmability, from software parameterizing a fixed microprocessor architecture to a matrix of programmable hardware, we gain this increase in all aspects but complexity. Complexity comes at a cost of programmability. The software model supported with the microprocessor architecture is a very successful and powerful one that has grown out of a need to solve problems of a greater degree of complexity that would otherwise be impractical with hardware [3]

In this moving down one level of programmability, we have lost the ability to solve complex problems, assuming we program hardware. Instead, if we abstract the microprocessor architecture, granulate it by component, parameter and operation, then we create a platform with the blending of solutions that gives us the programmability of software but the speed of hardware. This proposed model for hardware computing is referred to as a granular computer (GC).

To illustrate how we may adapt to this new platform, the HC12 model of fuzzy control will be examined and then copied onto the GC. Different algorithms, will be presented to explore the flexibility obtained by this proposed platform and some measurements will be made.

B. Granular Computer Platform

A granular computer is similar to a conventional processor but all aspects are parametrized or optioned. The width of the data can be set as needed from 1 bit to as wide as required. The length of the memory space is definable as well and can include ROM, RAM and external IO (EIO) sections. EIO is where all external control spaces are made to other hardware components such as serial peripherals, special execution units, attached devices and other GCs. Working data and program context are kept on stacks whose depth is also definable. The initial instruction set is empty with instructions added as needed for the computing task at hand. This granularization of the processor is where we gain the power of hardware without sacrificing the power of software.

While software and hardware design are well understood, GC design is new and will be explored here with detail enough for the reader to understand the fundamentals.

1) Multi-Layer Programming

A GC is programmed at up to four different levels depending on the requirements of the task:

1. Operations - VHDL is used to define the default components and any extensions to the basic processor architecture illustrated in Fig. 1.
2. Primitives - The components are controlled by a sequence of instructions which are composed of parallel operations for the components as illustrated in Fig. 2.
3. Program - At the highest level, programs are built in a conventional manner from primitives and other programs.
4. Compiler - The compiler consists of a Timbre [10] script and can be modified to support different GC language con-

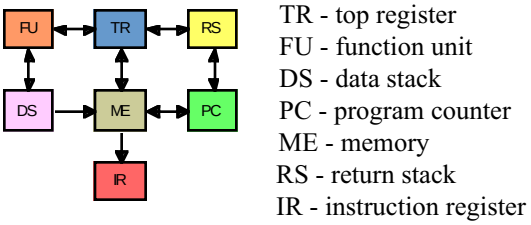

Figure 1: GC components and data flows. Each component executes an operation every clock cycle.

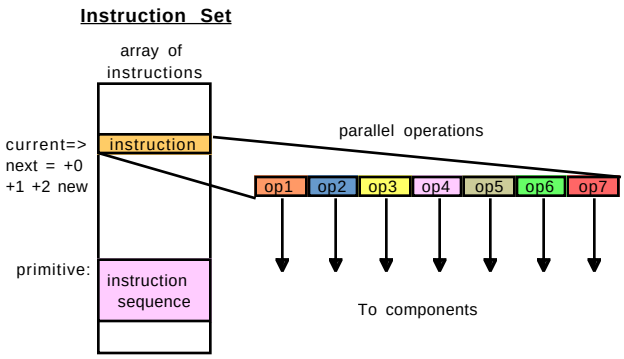

Figure 2: Parallel operations are combined to form an instruction. Several sequential instructions form a primitive.

structs. This would be used to define the executable data algorithm in the next section.

2) Only What is Needed

The programming philosophy for a GC is to only use what is needed. Depending on the speed, complexity, or size requirements, different levels of the granular computer can be used for a design solution. For instance, the compiler can be modified to produce executable data structures from the same linguistic data descriptions that can be used to create simple data structures which are smaller but more computationally intensive. And if we only require 10 bits of precision, then we only need to use 10-bit memory cells whereas a fixed processor would be 8, 16 or 32 bits.

3) Hardware, Software Ballet

Within the GC platform, there is a continuous blend of software and hardware. VHDL parts can be described in a behavioural manner much like software or they can be described like hardware components using RTL constructs. Hardware components are controlled by software algorithms either at the instruction level or at the program level. And finally, all the hardware and software parts are really just bits inside of a programmable FPGA.

4) Speed up the Quarks

The GC is RISC-like in that it executes one instruction every clock cycle and the instruction set is minimal. The one instruction can be executing up to 7 parallel operations in the different components. In the case where an algorithm needs to be sped up, the fundamental parts or the quarks can be factored out and defined in hardware. For the fuzzy logic operations, we added a max-min operation to the FU which calculated the maximum and minimum of two values in one clock cycle. This not only saved several cycles of instructions, but it also reduced the amount of code to describe the fuzzy operations. In another example, the addition and multiplication operations were augmented to provide upper bounds on the computations without taking any extra cycles and simplifying error handling within the higher level algorithms.

III. Granular Computer Implementation

For our purposes, we will examine the fuzzification algorithm. Of the three steps, this one is the most complex to code.

A. Fuzzy Inference Engine in Software on a Microcontroller

The first fuzzify algorithm is the HC12 algorithm which works for triangular, trapezoidal fuzzy sets and boolean sets by using a brute force method where both slopes of a fuzzy set are calculated. The algorithm can be described in software, in C, and operate on the exact same data structure. This serves as a high level view of the algorithm, and gives a method of simulation for platforms other than the68HC12. This can be an expensive algorithm as both slopes in a fuzzy set are calculated. This is fuzzify algorithm 1 (FA-1).

B. Fuzzy Inference Engine on an Enhanced Microcontroller

Using the same C code interface in the previous section the body of the function can be replaced with assembler instructions to take advantage of fuzzy assembler instructions like on the 68HC12. Unfortunately, while this makes fuzzification faster, it makes the code specific to a microprocessor and usually a compiler as well due to the nonstandard use of embedded assembler. Portability and maintainability are sacrificed for speed, which unfortunately is usually a paramount requirement.

C. Improve the Algorithm on a GC

The algorithm for fuzzification can be modified to run more efficiently on a GC by avoiding temporally expensive computation such as multiplication. In this algorithm, only the slope which is intercepted is calculated. This requires the addition of two parameters to the fuzzy set descriptions for the end points of the slopes. The decision making is also simpler in that only one limit is checked at a time. By breaking up a fuzzy set into intervals, only the necessary calculations for each fuzzy set are done. This is fuzzify algorithm 2 (FA-2).

In Fig. 3, the algorithm has been simplified even more with the data structure now becoming an active component of execution. The previous algorithms kept program and data separate while this one merges the two together so that the data is executed. As well, the interval scheme has been optimized by partitioning all the fuzzy sets on the universe of discourse into N discrete intervals which contain 1 or more subsets of the original fuzzy sets. This reduces computation wasted in the redundant checking when fuzzy sets overlap. Now only the subsets of the fuzzy sets which are on the interval containing the crisp value are calculated. This is fuzzify algorithm 3 (FA-3).

1) From Fuzzy Sets to Fuzzy Execution

The family of fuzzy sets can be described by (1) and the fuzzy set parameters and subsets by (2). This is modified to become the family of intervals of fuzzy subsets F_I as in (3).

$$F = \{fs_k\}_n \quad (1)$$

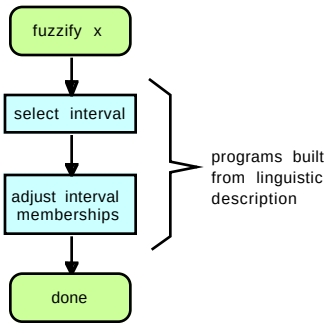

Figure 3: By integrating the parameters from the fuzzy sets with custom instructions, we can turn linguistic descriptions directly into executable programs.

$$fs = \{p0, \{s_i, p_i\}_m\} \quad (2)$$

$$F_I = \{p_k \{s_j, p_j\}_n\}_p \quad (3)$$

By combining the fuzzy subset parameters with GC primitives, the GC compiler creates executable code where the parameters are now part of the program. This differs from other platforms in that the instructions and parameters remain in separate domains.

2) Example

In Fig. 4 we have 4 fuzzy sets with differing degrees of over-

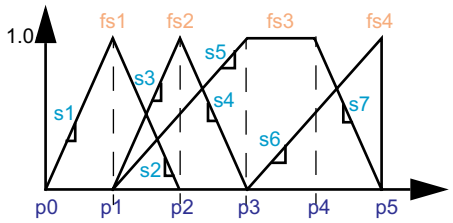

Figure 4: An example of a family of fuzzy sets where the points and slopes are marked. This can be represented as a family of fuzzy sets as in (Table 1) or as a set of intervals of fuzzy subsets (Table 2, Table 3).

lap and one set is trapezoidal while the other three are triangular. All the points where each fuzzy set is segmented into subsets, are marked with p*n*. These values are used to find out which interval an input x belongs to. Once that has been determined, then the fuzzy subsets in that interval are used to calculate membership values for the input. Since the boundaries are straight, the fuzzy sets can be defined by the parameters in Table 1. The format for describing a fuzzy subset is point fol-

TABLE 1
Parameters for fuzzy sets graphed in Fig. 4.

fuzzy set	parameters
fs1	p0, s1, p1, s2, p2
fs2	p1, s3, p2, s4, p3
fs3	p1, s5, p3, 1, p4, s7, p5
fs4	p3, s6, p5

lowed by slope and then point. In the case of the trapezoidal interval, fs3, where the membership value is 1, the slope is 1.

In Table 2, the intervals have been encoded with increasing

TABLE 2
Executable interval selector.

label	instruction	limit	vector
X:	interval	p1	p1do
	interval	p2	p2do
	interval	p3	p3do
	interval	p4	p4do
	interval	p5	p5do

points and integrated with a GC primitive. This primitive called interval uses the parameter following it, p_k to determine if the input value is in the current interval. The vector following the parameter is a pointer to a sequence of executable code which calculates all the membership values for the appropriate fuzzy sets. In this case the vectors would be coded as in Table 3. Each interval has one or more subsets of the fuzzy

TABLE 3
Execution vectors for intervals.

label	subset 1	subset 2	subset 3	end
p1do:	rise s3,p1,fs1	done		
p2do:	fall s2,p2,fs1	rise s3,p1,fs2	rise s5,p1,fs3	done
p3do:	fall s4,p3,fs2	rise s5,p1,fs3	done	
p4do:	set fs3	rise s6,p3, fs4	done	
p5do:	fall s7,p5,fs3	rise s6,p3, fs4	done	

sets and the subsets fall into three categories: rise, fall and set. This corresponds to rising slope, falling slope and a fuzzy subset where the membership is 1. The rise and fall primitives require the slope and the point. All require the fuzzy set variable to store the calculated membership value.

This method, although it is sequential, should be quite fast as it minimizes decisions and calculations. It also removes the need for the designer to write a program since the compiler will create the program from the fuzzy set specification. Thus, it simplifies programming as well.

D. Rule Evaluation

Rule evaluation is the central element of a fuzzy inference engine. The algorithm is illustrated in Fig. 5. With resultant fuzzy input values from fuzzification, we process a list of rules from the knowledge base, producing fuzzy output. These outputs can be thought of as raw suggestions for what the system output should be in response to the current input conditions. We currently have two variations of rule evaluation implemented [7][9]. They are different in the triangular norm pair used for inference, either Minimum and Maximum (IA-1), or Algebraic Product and Probabilistic Sum (IA-2).

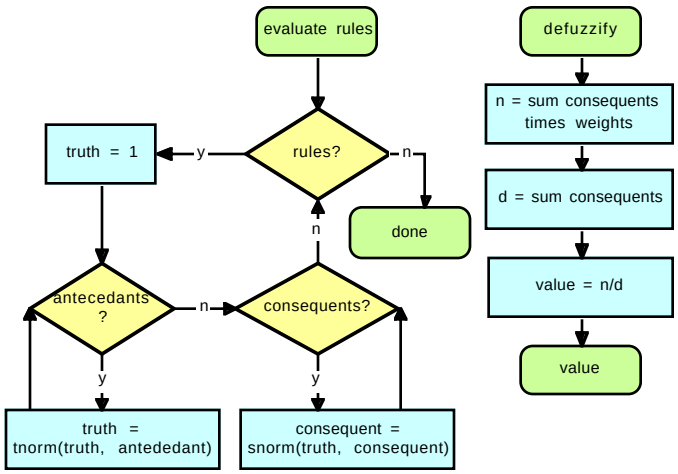

Figure 5: Rule evaluation and defuzzification algorithms. These are much simpler than fuzzification but the rule evaluation data structure can be quite large.

1) From Syntax to Structures

An understanding of the structure and syntax of rules is needed to understand how we perform the rule evaluation task. The following is an example of a typical rule:

If temperature is warm and humidity is high then heat is (should be) off.

It is quite simple to reduce such a rule to a small list of memory pointers. The left portion of the rule is a statement of input conditions and the right portion of the rule is a statement of an output action.

The left portion of a rule is made up of one or more (in this case two) antecedents connected by an AND operator which represents a t-norm. Each antecedent expression consists of the name of a system input, followed by a label name, which is defined by a membership function in the knowledge base. Each antecedent expression corresponds to one of the fuzzy inputs in RAM, so therefore it can be represented simply as a pointer to the particular fuzzy input.

The right portion of a rule is made up of one or more consequents. A consequent expression consists of the name of a system output, followed by a label name. The expression corresponds to a specific fuzzy output in RAM, and, like the antecedents, can be represented by a pointer to the fuzzy output.

AND is the only operator allowed to connect antecedent expressions, and so there is no need to include this in the encoded rule. Note that the OR operator, using a t-conorm, is inherent in the rule list structure. By defining several rules for the same fuzzy output, the consequent is calculated by ORing the result of the antecedent calculation with the current value of the fuzzy output, which may have been changed by a previous rule in the knowledge base.

An entire rule is encoded as a simple list of pointers to the addresses of desired fuzzy inputs (representing antecedents), followed by a negative marker value, followed by pointers to addresses of fuzzy outputs (consequents). A second marker value is placed after the consequents to indicate the beginning of a new rule.

One can see that if we always traverse through the entire rule list upon receiving new crisp input values, there is a lot of wasted computation. This is because inputs may be fuzzified into only a few nonzero fuzzy inputs, which would then correspond to only a sub-portion of the rule base. For supporting all possible control applications, however, it is difficult to predict what type of membership functions or rules will be defined, making it hard to devise a more efficient rule evaluation strategy instead of simply running through all rules in the knowledge base.

2) Rule Optimization

A general control system presents an excellent opportunity for optimizing rule evaluation. We define five membership functions for each of the two inputs, as well as the output. If we were to create a standard rule base, we would have 25 rules total, covering 25 possible combinations of fuzzy input labels. In addition to this, it is also standard to design the input membership functions in such a way that each fuzzification results in two or less nonzero membership values. Therefore, with up to two fuzzy inputs for each input, we would need to evaluate up to a maximum of 4 rules, comprising only 16% of the total rule base. This is inference algorithm 3 (IA-3).

With these 2 inputs, we can express the 25-rule knowledge base as 5x5 table, indexed by fuzzy input labels. Upon completion of fuzzification, we can use the nonzero fuzzy inputs to form up to 4 indexes into this table, effectively forming the antecedent portions of each rule. The values in the table would simply contain pointers to the fuzzy outputs, representing the rule consequents.

This method has the potential to speed up rule evaluation significantly. While seemingly limited in application, this particular fuzzy control system architecture is very widely used in industry [2].

E. Defuzzification

Regardless of implementation method, the end result of the rule evaluation step is a list of suggested outputs. However, these raw results cannot be supplied directly to the system outputs because they may be ambiguous. For instance, one raw output can indicate that the system output should be medium with a degree of truth of 50% while, at the same time, another indicates that the system output should be low with a degree of truth of 25%. The defuzzification step resolves these ambiguities.

Defuzzification is the final step, combining the raw fuzzy outputs into a composite system output. Unlike the trapezoidal shapes used for inputs, singletons are used for output membership functions. As with the inputs, the x-axis represents the range of possible values for a system output. Singleton membership functions consist of the x-axis position for a label of the system output, and fuzzy outputs correspond to the y-axis height of the corresponding output membership function. The defuzzify routine calculates the numerator and denominator sums for a weighted average of the fuzzy outputs according to Fig. 5.

IV. RESULTS

The results for different platforms are listed in Table 4. For all the tests:

- fuzzify was with 5 triangular 1/2 overlapping fuzzy sets
- inference was with 25 rules with two antecedents and one consequence
- defuzzify used 5 outputs and weights
- precision is 8 bits

From these results we can see that the closest in speed to the GC is the HC12. The HC12 does this by having dedicated hardware units for multiplication, fuzzification, rule evaluation and defuzzification. In comparison, the GC has no dedicated hardware for any of these processes,.only a few quarks for all of them. This is a double saving in that there is much less hardware used and the same algorithm runs faster. The bonus

TABLE 4
Execution time in microseconds of a few algorithms on different platforms.

Platform	fuzzify			inference			defuzzify
	FA-1	FA-2	FA-3	IA-1	IA-2	IA-3[a]	DA-1
HC12@50Mhz in C	30	n/a	n/a	142	190	23	22
MC2114@32Mhz in C	14	n/a	n/a	65	274	10.4	12.5
HC12@50Mhz in asm	9	n/a	n/a	35	n/a	5.6	10
GC@50Mhz	6.5	5.5[b]	1.9[c]	18	36	2.9	9.8

a. Estimated by assuming 16% of execution time of IA-1.
b. Estimated by doing two less multiplies than FA-1.
c. Estimated by counting instructions executed.

with using the GC platform is that we can go beyond the algorithms and implement newer ones as indicated in the other columns.

IA-1's execution time on the GC is close to FA-1 and DA-1 opening up the potential for pipelining GCs on the FPGA to increase throughput. A more computationally intensive inference algorithm, IA-2 allows a different description of a fuzzy relationship that is still quite efficient on the GC platform, suggesting more complex relationships could be implemented. By applying IA-3 we reduce the inference time down to that of fuzzification and defuzzification instead of dominating the execution time. With larger families of fuzzy sets on the inputs, the number of rules increases quadratically so saving time in the rules is even more important.

The first two inference algorithm curves are shown in Fig. 6 with the same input value used on both inputs of the fuzzy inference engine over the range of the 8-bit input.

In Table 4 we used the HC12 with both C and fuzzy assembler implementations. The MC2114, which is a 32-bit microcontroller from Motorola, is in a similar market to the 16-bit microcontroller, the HC12.

V. CONCLUSION

We've ported algorithms for running a fuzzy inference engine on a fuzzy-enhanced microcontroller to C and to the GC to see the comparison. The GC port runs quite favourably, and is indeed faster than all the other platforms even though it is using multi-step algorithms for addition and multiplication. Due to the flexibility of the GC platform, we are not limited to using this implementation. Using better algorithms and more efficient data structures has been discussed, which could result in even higher speeds. As well, the algorithms may be varied or altered without much difficulty, making the GC platform a excellent platform for exploratory soft computing. Finally, if speed was the top requirement, then we could wire in a hardware engine[8] that would operate in parallel.

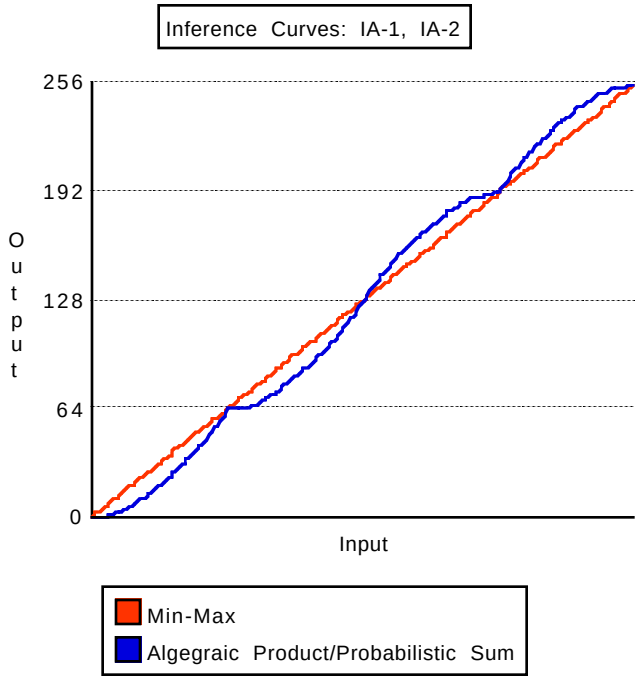

Figure 6: Inference transfer curves over the full range of the input and output. Two inputs are fed the same value.

REFERENCES

[1] G. Viot, "Meet the 68HC12", *Northcon/96*, vol., iss., 4-6 Nov 1996, pp.180-185

[2] G. Viot, Third generation fuzzy processors", *Northcon/96*, vol., iss., 4-6 Nov 1996, pp.203-209

[3] A. Costa, A. De Gloria, F. Giudici, M. Olivieri, "Fuzzy logic microcontroller", *IEEE Micro,* vol.17, iss.1, Jan/Feb 1997, pp.66-74

[4] A. Costa, A. De Gloria, M. Olivieri,"Hardware design of asynchronous fuzzy controllers", *IEEE Transactions on Fuzzy Systems*, vol.4, iss.3, Aug 1996, pp.328-338

[5] A. Nelson , T. Marcelo, "Custom architectures for fuzzy and neural networks controllers", *Journal of Computer Science & Technology*, vol. 2 - No. 7.

[6] G. Louverdis, I. Andreadis, "Design and implementation of a fuzzy hardware structure for morphological color image processing", *IEEE Transactions onCircuits and Systems for Video Technology*, vol.13, iss.3, Mar 2003, pp. 277- 288

[7] O. Cordon, F. Herrera, A. Peregrin, "A practical study on the implementation of fuzzy logic controllers", *Technical Report #DECSAI-98107*, DECSAI University of Granada, Spain, July 1998.

[8] Z. Salcic, "High-speed customizable fuzzy-logic processor: architecture and implementation", *IEEE Transactions on Systems, Man and Cybernetics*, Part A, vol.31, iss.6, Nov 2001, pp.731-737

[9] T. Hollstein, S.K. Halgamuge, M. Glesner, "Computer-aided design of fuzzy systems based on generic VHDL specifications", IEEE Transactions on Fuzzy Systems, vol.4, iss.4, Nov 1996, pp.403-417

[10] R. Chapman, "It has Timbre", *Proceedings of the 1994 Rochester Forth Conference*, June 1994, pp. 16-19

[11] W. Pedrycz, F. Gomide, *An Introduction to Fuzzy Sets*, MIT Press, 1998