Implementing a Fuzzy System on a Field Programmable Gate Array

Michael McKenna and Bogdan M. Wilamowski
University of Wyoming University of Idaho

Abstract

Fuzzy controllers are traditionally implemented in a microprocessor and they produce relatively raw surfaces. The purpose of this work is to implement a fuzzy control system in a FPGA and to have resulted control surface as smooth as possible. The FPGA has allowed designers to create large designs test them and make modifications very easily and quickly. This approach uses a new weighted average concept to keep the fuzzy lookup table small, yet the input sizes can be large. This is implemented by using three or four most significant bits of each input to determine the address for the lookup table. A weighted average is performed using the remaining bits to eliminate rawness.

1 Introduction

In recent years, fuzzy systems have become very popular and are being used in many applications [1][2][3][4]. Fuzzy systems give the digital designer the ability to use non-linear controllers for their application. Fuzzy controllers are traditionally implemented in a microprocessor [5] but also dedicated VLSI chips have been developed [6][7][8][9][10]. The purpose of this work is to implement a fuzzy control system in a Field Programmable Gate Array type chip. The project is being done on this type of chip so if design changes are needed the chip can be reprogrammed for the new design in a matter of seconds. The traditional fuzzy systems work well for inputs that have 8 bits or less. The number of inputs is also a factor in these designs since as the number of inputs increases the size of the lookup table increases exponentially. The design will use a high-level programming language for the simulation of the weighted averaging technique and Hardware Description Language (HDL) will be used for the implementation. The FPGA has allowed designers to create large designs test them and make modifications very easily and quickly. This project is aimed for lab experiments and will not be tested or used on any actual systems.

Traditional fuzzy controllers use a minimum, a maximum, and inversion operators. A minimum operator does exactly what the name says; find the minimum of the inputs. In a Boolean system there are two states that a signal can be in: a high state and a low state. A "1" for high and a "0" for low represent these two states. In a fuzzy system there can be several states between "0" and "1." The digital AND gate works as a minimum function by finding the minimum of the two inputs. The digital OR gate works for the maximum function. In a binary system, taking the input value and bitwise inverting (each bit individually) them can accomplish the inversion function.

There are four main steps to a fuzzy controller (Fig. 1). The first step is converting an analog input into fuzzy variables, which is done in the block named fuzzifier. There are usually three to nine fuzzy variables that are produced for each analog input. The general block diagram of a fuzzy controller can be seen below.

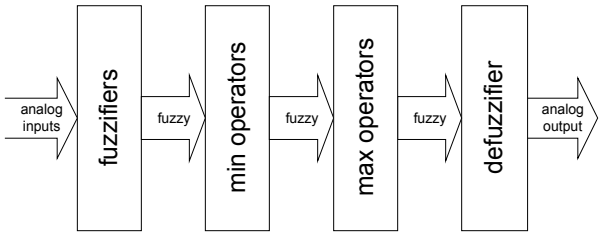

Fig. 1 Block diagram for a fuzzy system.

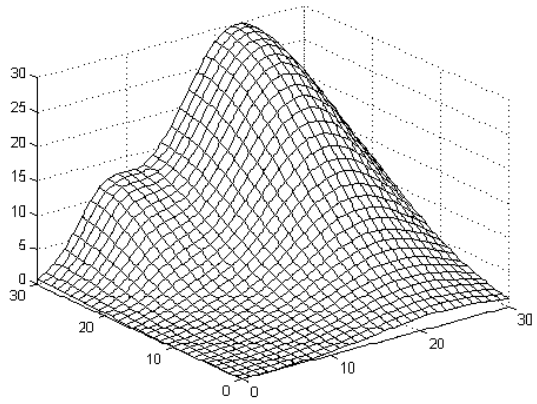

Fig 2 Desired control surface.

The fuzzifiers can apply several different membership functions to the analog inputs. The most common are the trapezoidal, triangular and the gaussian functions. If the surface illustrated in Fig 2 is the desired output surface.

As seen in Fig 3 the output of the trapezoidal membership function is very choppy as compared to the desired surface in Fig 2. Traditional fuzzy systems that are implemented in hardware are often done in Hardware Description Languages.

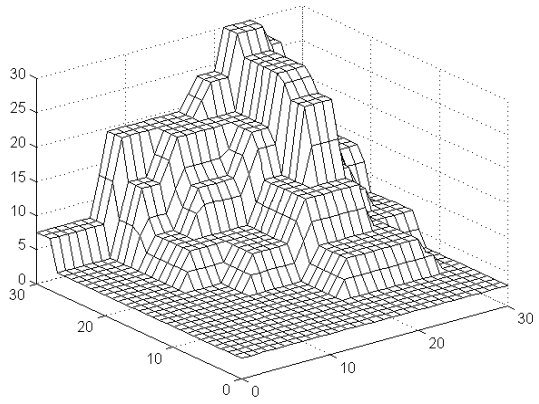

Fig 3 Control surface using the trapezoidal membership functions.

2. Hardware Description Languages (HDL)

Hardware Description Languages (HDL) were developed to describe how hardware behaves. There are two main differences between traditional programming languages and HDL:

1) Traditional languages are a sequential process whereas HDL is a parallel process,
2) HDL runs forever whereas traditional programming languages will only run forever if directed.

This leads to confusion as to why some things may be implemented in HDL and why other things are not. For example, if the following two lines of code:

C=A+B; (1)

A=C+D; (2)

were analyzed using a software approach and a hardware approach one will find both of the differences and the different mindset one needs to write HDL. In traditional programming languages, the first line of code in equation (1) would execute and result in the addition of A and B and storing it into C, and then taking C and D and adding them together and storing it back in A. In software this is possible and is used very often. In HDL the two lines would execute at the same time causing combinational logic feedback. The most popular style of HDL is high level behavioral. This is the highest level and can model large designs. Behavioral style uses many of the same syntax that traditional programming languages use. This style of HDL is the most powerful level of HDL and can be written much faster than a schematic can be drawn. Schematic capture is often thought of as the way an electrical engineer creates digital hardware but this approach is only used in very limited cases.

The advantages of a custom VLSI chip are that it will run faster and (once the prototypes are tested) be very cheap to produce. The reason they are cheap to produce is once the mask is created, millions of chips can be made very inexpensively. The disadvantages of a custom VLSI design are that the time to create the first chip and receive it back is often months and that if the design is incorrect the chip is no good and must be thrown away. The advantages of the FPGA are that the time it takes to program one is on the order of seconds, so if the design is wrong they can be reprogrammed with the fixes and they are relatively cheap if only a few chips are required. An FPGA will also run slower than the custom VLSI chip. This project is being implemented in an FPGA so it could be reprogrammed and tested on site.

3 Field Programmable Gate Array (FPGA)

A Field Programmable Gate Array (FPGA) is a digital integrated circuit that can be programmed to do any type of digital function. There are three main advantages of an FPGA over a microprocessor chip for fuzzy systems:

(1) An FPGA has the ability to be reprogrammed on the fly,
(2) The new FPGA's that are on the market will support hardware that is upwards of 1 million gates,
(3) An FPGA used as a fuzzy controller will be semi-custom hardware, and
(4) The FPGA will operate faster than a microprocessor chip.

FPGA's are programmed using support software and a download cable connected to a host computer. Once they are programmed, they can be disconnected from the computer and will retain their functionality until the power is removed from the chip. A Read Only Memory (ROM) type of a chip that is connected to the FPGA's programmable inputs can also program the FPGA upon power-up. This means that when a board is in place in a remote location, the chip can keep running while the designer updates the design back at a lab. Once the designer updates the design he or she can program another ROM chip and take it to the site and replace the old ROM chip; upon the next power-up the chip will be reprogrammed to the new design. The other aspect of being able to be reprogrammed on the fly means that there does not need to be any down time for the controller. Down time is when the entire system has to be shut down. If a microprocessor needs to be reprogrammed then the entire system must be taken down and the microprocessor will be reprogrammed and then the system can be brought back up on line. The FPGA's can be programmed while they are running, because they have reprogram times on the order of microseconds. This short time means that the system will not even know that the chip was

reprogrammed and there may be a small waiting period but the system will not have to be shut down.

The fact that an FPGA is a programmable chip means that the controller will be running as an Application Specific Integrated Circuit (ASIC). When a piece of hardware is custom made for an application the design will be able to run much faster than a general purpose microprocessor that is running from software that has been downloaded on it. Part of the FPGA is made up of Combinational Logic Blocks (CLB). These blocks are made up of an array of digital AND, OR and INVERT gates. The CLB's are implemented in all FPGA chips and are used to implement asynchronous Boolean equations inside the chip. Combining the FPGA and the HDL with fuzzy system concept the design process can begin. The goal of this project is to implement a fuzzy system on an FPGA using HDL and a weighted averaging technique.

4 Implementing a Fuzzy System on a FPGA

The design of an FPGA-based fuzzy controller can be very simple. It consists of an FPGA, analog-to-digital (A/D) converters for the inputs, a digital-to-analog (D/A) converter for the output and a ROM chip. The ROM chip is used as a fuzzy lookup table (LUT). The reason the LUT is on an external chip is so the entire FPGA doesn't have to be reprogrammed if the control surface changes. Only the external ROM chip needs to be changed. A block diagram of an FPGA-based fuzzy controller is illustrated in the Fig 4.

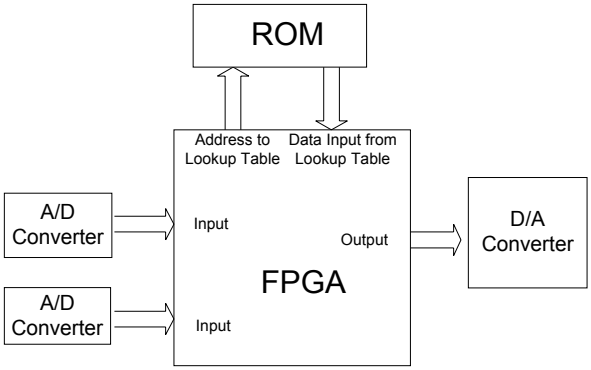

Fig 4 Block diagram of fuzzy control board with FPGA

The traditional fuzzy system implementation is easily programmed into an FPGA and works well as long as there are only two inputs. However if the inputs are increased to three then the LUT using traditional implementation methods becomes very difficult to handle. The size of the LUT also grows exponentially as inputs are added. This is the downfall of a fuzzy controller. There have been attempts to correct this problem by combining two or more inputs together before the fuzzy controller so the number of inputs would remain around two.

The first design step was to select a FPGA. The Xilinx 4000 series FPGA chip was chosen to implement this design on. Xilinx recommends the 4000 series when the primary functionality of the design is computing arithmetic functions. Since the fuzzy systems main functionality is to do multiplication and addition then this chip is the natural choice. The drawback of this chip is that the designer must remove chip power for reprogramming. This means it cannot be programmed on the fly. This is acceptable because the scope of this project does not require system reprogramming on the fly. The presented approach uses a new weighted average concept to keep the fuzzy lookup table small, yet the input sizes can be large. This is implemented by using three or four most significant bits of each input to determine the address for the lookup table. A weighted average is performed using the remaining bits to eliminate rawness. The block diagram of the system is shown in Fig. 1. This new method may cause a decrease in speed (which is not crucial in FPGA implementation) but could result in a smoother surface, and has the ability for designs with a larger number of inputs.

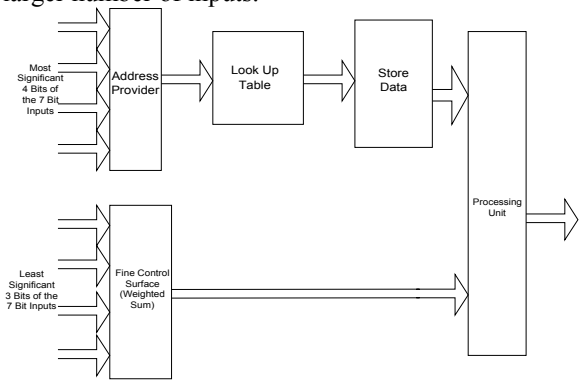

Fig 5. Block diagram of the proposed approach

As seen in this Fig, there are four inputs. This is a good test to determine if the weighted average approach works with more than two inputs. The difficult part of the project was implementation of the novel weighted sum approach.

The concept of weighted averaging was verified using a high level programming language. In this case MATLAB was used for the simulations. The first simulation was done with only two inputs. With two inputs there are four random discrete values from the lookup table that are needed when the weighted averaging is used. The following MATLAB code was used to perform the simulation.

```
w(1,1)=(1-w1) * (1-w2);
w(1,2)= w1 * (1-w2);
w(2,1)=(1-w1) * w2;
w(2,2)= w1 * w2;
z=y11*w(1,1)+y12*w(1,2) +y21*w(2,1)+y22*w(2,2);
```

where:

z = the output

y = the look up table values
w = least significant bits of the inputs
(1-w) = the inversion of the least significant bits of the inputs

This code was found to be correct by looking at the values that are created between the poles. Once it was found that this approach of weighted values would work the code was then placed into two nested for loops to generate a control surface.

With the weighted average concept complete, the next concern was to determine if the edges of this control surface would match that of the neighboring control surface. The address provider block in Fig 5 was broken up into a more detailed schematic and Verilog code was then written for the entire diagram. The new schematic can be seen in Fig 6.

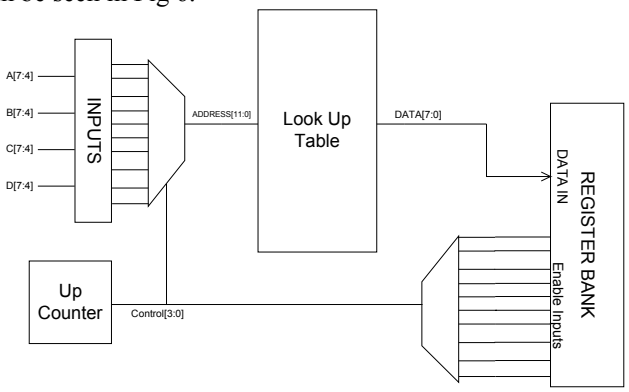

Fig. 6. Block Diagram of the Address Provider.

This block diagram has a large multiplexer for the inputs. It had to be designed so the Xilinx Software would be able to efficiently place and route it. The design in the Fig 7 was created to efficiently implement the multiplexer by breaking it up into four smaller multiplexers.

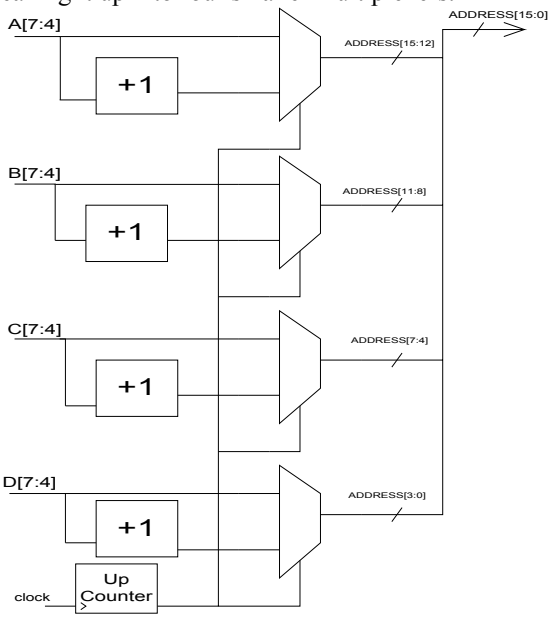

Fig. 7. Block Diagram for the Input Multiplexer Structure

With this part of the design complete, the next task was to design the weighted average part so those values could be multiplied by the lookup table values and then accumulated for the final answer. The weighted average block diagram can be seen in Fig 8.

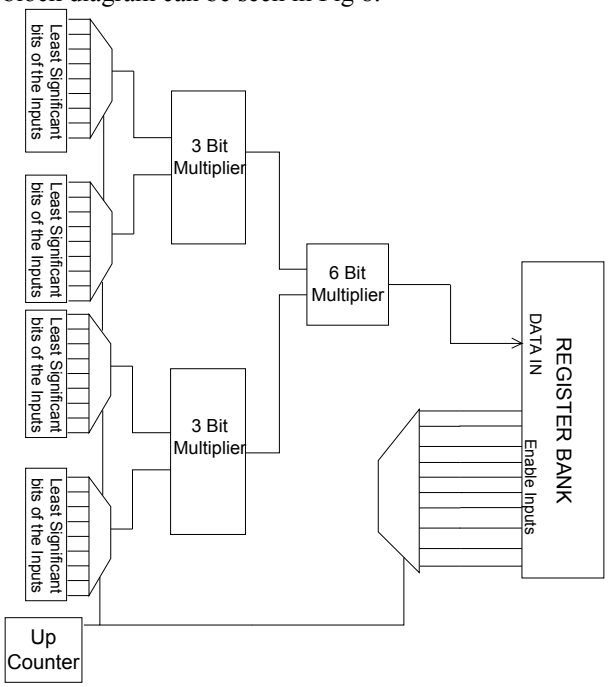

Fig. 8 Block Diagram of the Weighted Average Block

In Fig 5.8 the input blocks labeled “Least Significant bits of the Input” are the least significant bits that are inverted or passed through in a specific pattern. The Verilog code for this pattern can be seen below.

assign z11=~A[3:1]; assign z12=~B[3:1];
assign z21=~A[3:1]; assign z22=B[3:1];
assign z31=A[3:1]; assign z32=~B[3:1];
assign z41=A[3:1]; assign z42=B[3:1];

The purpose of the processing block is to multiply the proper weighted average values by the LUT values. The process then sums all of the values to create the output. The first design attempted of the processing block, resulted in 16 parallel multipliers followed by 15 adders. This caused the design to be very large and thus unable to fit on the targeted chip. This was 183% larger than the capacity of the chip, which was selected for this project. The second design approach to the processing block was done in a lookup table fashion. Taking the two input values and concatenating them together to create an address constructed a lookup table. This design approach took 175% of the targeted chip and therefore could not be used either. The third design approach was to replace the

parallel multipliers with one single multiplier and use it sequentially. The block diagram for this approach to the processing unit can be seen in Fig 9.

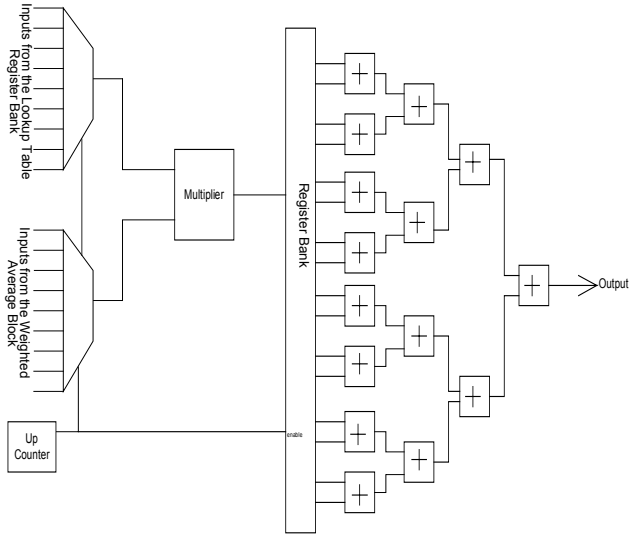

Fig. 9 Block Diagram for the Processing Unit

The sharing of resources eliminates a lot of the gate count. The entire design with this approach used 97% of the targeted chip. Another approach that could be taken is to share some of the adders along with the multiplier if space was still an issue.

The PC84 pin package is the standard package normally used at the University of Wyoming. The design could not fit in this package. This is because the PC84 package has 61 user available input/output pins. This design needs 68 input/output pins. There are four 7 bit inputs, two 16 bit output lines and one more 8 bit input. The four 7 bit inputs are the inputs to the design. The first 16 bit output is the addressing lines for the lookup table and the 8 bit input is the data lines from the lookup table. The final 16 bit output is the output of the design. A block diagram of the input/output structure can be seen in Fig 10.

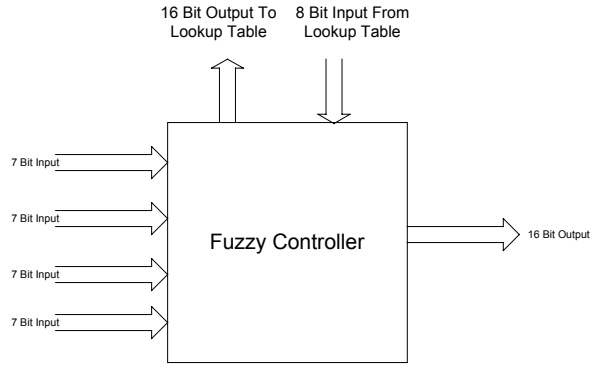

Fig.10 Final Block Diagram of the Input/Output of the FPGA

With the number of in/out pins needed, a PC84 pin package could not be used. The package selected for the design is the PQ160. This pin package is a surface mount package that has 129 available user input/output pins.

The nonlinear function of four variables used to verify the concept of this project is given by the equation (3). The graph of this equation is shown in Fig. 11.

$$z = \left(5 * \frac{\sin(\sqrt{(x+10)^2 + (y-3.3)^2})}{\sqrt{(x+10)^2 + (y-3.3)^2} + 2} \right) * \sqrt{\sqrt{1 + 0.5w^3 (z-9)^2}} \qquad (3)$$

An example this surface using a lookup table of 65K in size can be seen in Fig 12.

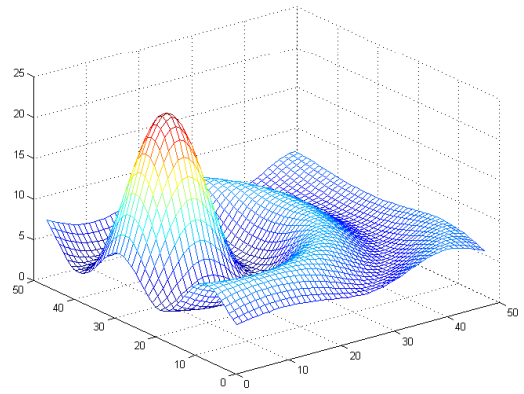

Fig. 11 Desired Surface.

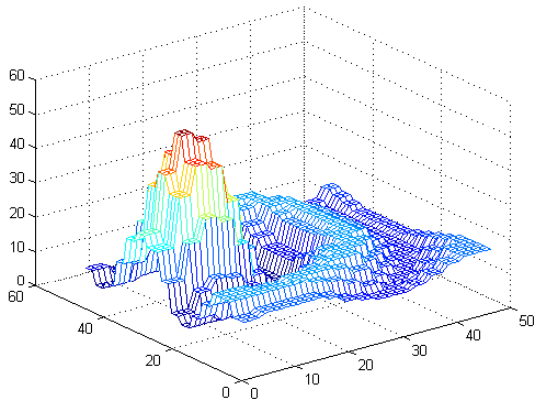

Fig. 12 Control Surface Rendered using the Traditional Method with a 64K Lookup Table with MATLAB.

The Fig 12 clearly shows extreme roughness on the control surface; therefore, the traditional method would not work for many applications. When the output surface is very choppy, a very unstable system may result if this surface was used in a control application.

5 HDL Design

The design was then placed into Veribest using Verilog type HDL. The code was written to send each input address, one at a time, to the lookup table, get a value back and store it in the register bank, this sharing resource code structure is used two more times for each multiplier. The reason for sharing resources is so the hardware will not become too large to fit in the chip. The code was written for two inputs to correct any coding mistakes and then adapted to four inputs for the finished design. The design was then synthesized using FPGA Express. This program does the fitting, place and route, timing, and generates the bitstream that is used to program the chip. This file has a log of the inputs and the output results may be plotted. The design fit on the 4010 FPGA using 97% of the chips resources. The maximum delay between flip-flops determines the clock frequency. FPGA Express provides this number for the designer based on the chip selected and the way the software did the place and route part of the design. The maximum clock frequency for this design is 9.6MHz. This means that the design can easily be run in the order of microseconds, which is very fast when controlling a mechanical system. The plot from actual hardware results can be seen in the Fig below.

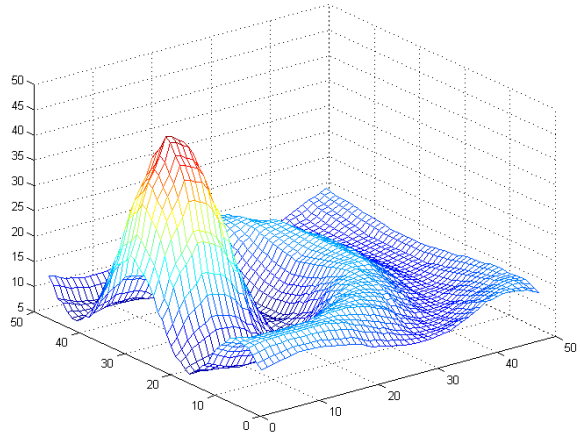

Fig. 13 Actual Output from FPGA

As seen in Fig 12 the actual results are very close to the desired results in Fig 11. Clearly, the weighted approach to the traditional fuzzy system works very well when it is simulated using actual hardware. This plot is very smooth and would offer a very good control surface.

6 Conclusion

In conclusion the design works very well and the outputs are very smooth. The new method of weighted averaging is far superior to the traditional ways of doing fuzzy control. The traditional fuzzy systems provide a very rough control surface, which can cause the system being controlled to become unstable. With smoother surfaces the system being controlled by the new weighted averaging fuzzy controller will be more stable in comparison with traditional methods. In traditional fuzzy systems number of inputs are usually limited to 2 or 3. This method will allow much larger number of inputs. The weighted average method helps solve the exponential growth problem and complexity of the LUT by allowing very smooth surfaces with small a simple LUT.

7 References

[1] Wilamowski B. M. "Neuro-Fuzzy Systems and its applications" tutorial at 24th IEEE International Industrial Electronics Conference (IECON'98) August 31 - September 4, 1998, Aachen, Germany, vol. 1, pp. t35-t49.

[2] Kosko B., (1993) *Fuzzy Thinking, The New Science of Fuzzy Logic*. Hyperion, New York.

[3] Passino, K. M., S. Yurkovich, Fuzzy Control, Addison-Wesley, 1998.

[4] Wang, L., Adaptive Fuzzy Systems and Control, Design and Stability Analysis, PTR Prentice Hall, 1994.

[5] Wilamowski B.M. and J. Binfet, "Do Fuzzy Controllers Have Advantages over Neural Controllers in Microprocessor Implementation" Proc. of. 2-nd *International Conference on Recent Advances in Mechatronics - ICRAM'99*, Istanbul, Turkey, pp. 342-347, May 24-26, 1999.

[6] Choi J., B.J.Sheu, and J.C.F. Chang, (1994) A Gaussian Synapse Circuit for Analog VLSI Neural Networks. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. **2**, no. 1, pp. 129-133.

[7] Rodriguez-Vazquez A. and F. Vidal-Verdu, Learning in Neuro/Fuzzy Analog Chips, *IEEE International Symposium on Circuits and Systems*, Seattle WA, vol. **3**, pp. 2325-2328, April 30-May 3 1995.

[8] Yamakawa, A fuzzy Inference Engine in Nonlinear Analog Mode and its Application to a Fuzzy Logic Control, *IEEE Trans. on Neural Networks*, vol. 4, pp. 496-522, 1993.

[9] Ota, Y. and B. M. Wilamowski, " CMOS Implementation of a Voltage-Mode Fuzzy Min-Max Controller", Journal of Circuits, Systems and Computers, vol. 6, No 2, pp. 171-184, April 1996.

[10] Wilamowski B. M. and R. C. Jaeger, "Neuro-Fuzzy Architecture for CMOS Implementation" *IEEE Transaction on Industrial Electronic*s vol. 46, No. 6, pp. 1132-1136, Dec. 1999.