2014 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)
July 6-11, 2014, Beijing, China

Hardware Implementation of a Novel Inference Engine for Interval Type-2 Fuzzy Control on FPGA

Matthew D. Schrieber
School of Engineering
University of Guelph
Ontario, Canada
Email: schriebe@uoguelph.ca

Mohammad Biglarbegian
School of Engineering
University of Guelph
Ontario, Canada
Email: mbiglarb@uoguelph.ca

***Abstract*—Interval type-2 fuzzy logic controllers (IT2 FLCs) have shown a promising potential in handling uncertainties compared to their type-1 counterparts, and as a result, we have witnessed increasing usage of IT2 FLCs in various applications. Due to the complex structures of IT2 FLCs, using them in real-time applications might be computationally expensive. To facilitate real-time implementation of these controllers, hardware with parallel processing abilities are recommended; field-programmable gate arrays (FPGA) are one class of such hardware. In this paper, we propose a structure for implementing a new IT2 FLC inference mechanism called BMM [2] - that has been recently introduced in the literature - on an FPGA. We first demonstrated how the proposed structure can be implemented on software; next, we proposed an implementation architecture for the IT2 FLC mechanism on hardware. We performed simulations and experiments on two different plants and compared the speed of our controllers. The performance speed as well as the tracking of our proposed control structure in simulations and experiments were shown to be very close to each other. Using the BMM engine for the proposed hardware structure proves to be faster than other existing controllers in the literature. Thus, it is expected that IT2 FLCs can be easily implemented on hardware to further enable their real-time applications.**

I. Introduction

Interval type-2 fuzzy logic controllers (IT2 FLCs) are a class of nonlinear controllers that have become widely used in systems featuring uncertainty. Their capability in handling uncertainty allows them to be used in control and identifications of systems where the dynamics are not completely known. IT2 FLCs have proven to be used in many real-world applications such as industrial control or mobile robot control, and in some cases, they out perform traditional controllers [1].

Recently, there has been a significant interest in using IT2 FLCs [2]-[5] largely because of their potential in handling uncertainty. A unique feature of IT2 FLCs lies in their upper and lower membership function structure - called footprint of uncertainty - allowing them to accommodate potential uncertainties. IT2 FLCs also introduce the need for a type reducer which reduces the IT2 output to a type-1 (T1) fuzzy set, which makes them computationally expensive. As such, substantial research on improving and optimizing IT2 FLCs is currently being conducted. For example, Juang and Hsu [3] designed an IT2 FLC for use with a wall-following mobile-robot which was capable of operating without any $a\ priori$ rule sets by using an ant colony optimization technique to generate rules automatically. While Maldonado, Castillo and Melin [4]-[7] have collectively proposed many new optimization techniques for IT2 FLC including incorporating genetic algorithms [6] and particle swarm optimization [7].

Many researchers have concluded that IT2 FLCs are capable of outperforming some traditional controllers as well as T1 FLCs in several applications [8]-[10]. For example, Chen and Tan [9] proposed a T2 FLC that not only was proven to be stable through Lyapunov analysis, but was also found to be more robust and perform better than an adaptive T1 counterpart. Karimi and Safarinejadian [10] proposed a new T2 FLC for the control of a simple nonlinear system, but experienced problems when using the controller on-line due to computational expensive operations featured in T2 FLC. Therefore, despite IT2 FLCs being more capable of handling uncertainty in comparison to T1, their complex structure has limited their ability to be utilized on a larger scale. The computational complexity of any IT2 FLCs invariably leads to a significant increase in processing time, especially for applications required to operate at high speeds.

Researchers have been exploring the implementation of IT2 FLCs on hardware to increase the speed of the control process [11]-[14]. However, the increase in speed as a result of hardware implementation results in a more restrictive design. Melgarejo et al. [11] implemented a Wu-Mendel (W-M) IT2 FLC on a field programmable gate array (FPGA) to take advantage of the parallelism that this hardware offers. The developed controller was implemented successfully with a reported operating speed of $33.3ns$. However, this work was not used in a closed-loop system that includes a plant in which the system performance would have been significantly affected by the controller being implemented over a reported 9 clock cycles. Huang and Tsai [12] applied their controller to an autonomous omnidirectional mobile robot and achieved similar results to [11]. The need for robustness and stability in developing the controller for the omnidirectional mobile robot platform [12] increases the complexity of the controller making it difficult for implementation on hardware, and hence a hardware/software co-design was employed instead. Because of hardware/software co-design, the controller was only able to achieve an operating speed of $150\mu s$, approximately 4500 times slower than what was reported in [11]. Recently, the concept of hardware implementation of IT2 FLCs was further explored in [13] where an IT2 FLC with Mamdani structure was designed for hardware implementation. For speed control of a DC Motor simulation results showed the desired con-

978-1-4799-2072-3/14/$31.00 ©2014 IEEE

troller performance in which the potential operating speed in comparison to a general-purpose industrial computer featuring a quad-core processor was reported as 225,000 to 450,000 times faster. However, this potential speed-up is based on a 5 clock cycle process which will most likely affect the system performance in closed-loop. In another work, an IT2 FLC with Mamdani structure for a mobile robot was implemented on an FPGA [14]; in this work, any indication of change in the operating speed of the controller due to the hardware implementation is missing.

In this paper, we present a hardware implementation of an IT2 FLC by considering the specific hardware limitations in regards to the device area and optimal performance. Using the Karnik-Mendel (K-M) algorithm [15] for hardware implementation is significantly more challenging (than other IT2 FLCs mechanism) due to their iterative nature of these algorithms [13], [16]. Unlike existing approaches that have used Mamdani model structure, we use the Takagi-Sugeno-Kang (TSK) for implementing the controller. Lycnh et al. have successfully implemented both K-M and W-M type reduction using Mamdani model stucture and found W-M to using on average 45% less clock cycles then K-M [17]. Thus, we propose the use of a new closed-form inference engine introduced in [2], called BMM inference engine, because of its simplicity and closed-form structure.

The organization of the rest of this paper is as follows: Section II contains general background on IT2 TSK FLCs and hardware design. Section III presents the design methodology for hardware implementation. Section IV provides simulation and experimental results. Finally, Section V concludes the paper.

II. Background

A. Interval Type-2 TSK Model

The IT2 TSK FLC in this paper deals exclusively with the $A2-C0$ TSK model classification [18]. This means that the antecedent membership functions are type-2 (A2) and the consequents are crisp values (C0). The structure of a rule in a A2-C0 TSK model is defined as follows:

$$\text{Rule } i^{th}: \text{If } x_k \text{ is } \tilde{F}_k^i, \text{ then } y_i = a_0^i + \sum_{k=1}^{n} a_k^i x_k \qquad (1)$$

where $i = 1, \cdots, M$, $k = 1, \cdots, n$, $\tilde{F}_k^i$ represents an interval type-2 fuzzy set of input state k in rule i, x_k are inputs, $a_0^i, \cdots, a_n^i$ are the coefficients of the output polynomial for rule i, y_i is the output of rule i, n is the number of inputs and M is the number of rules. In [2] a new closed-form inference engine for this model was introduced that replaces the type-reduction and is given as follows:

$$Y_{T2-TSK} = m\frac{\sum_{i=1}^{M} \underline{f}^i y_i}{\sum_{i=1}^{M} \underline{f}^i} + n\frac{\sum_{i=1}^{M} \overline{f}^i y_i}{\sum_{i=1}^{M} \overline{f}^i} \qquad (2)$$

where the parameters m and n are used to tune the model variables and the upper and lower firing strengths for rule i, $\underline{f}^i$ and $\overline{f}^i$, are defined as follows:

$$\overline{f}^i = \prod \overline{\mu}_{\tilde{F}_k^i}(x_k) \text{ and } \underline{f}^i = \prod \underline{\mu}_{\tilde{F}_k^i}(x_k) \qquad (3)$$

where $\overline{\mu}_{\tilde{F}_k^i}$ and $\underline{\mu}_{\tilde{F}_k^i}$ represent the k^{th} upper and lower membership functions of rule i. This paper uses (2) to implement a closed-form inference engine in its proposed hardware design.

B. Hardware Design

The most prominent hardware platforms for embedded systems design are classified as: Application Specific Integrated Circuits (ASICs) and FPGAs. An ASIC is designed and manufactured for a specific and unchanging application. Alternatively, an FPGA is a programmable semiconductor device composed of Configurable Logic Blocks (CLBs) in a matrix-like structure connected via programmable interconnections. The hardware components of an FPGAs can be programmed to meet the functionality of any application allowing designers to make the changes to their designs freely, and even while in the field. Due to the flexibility FPGAs offer they are ideal platform for experimental hardware design. An ASIC could potentially offer an increase to operating speed, however; they are expensive to manufacture and would not allow for any additional modification. Although FPGAs have predefined hardware blocks of commonly used functions, such as RAM or DSP blocks, the main components of any FPGA remain the same. As shown in Fig. 1, an FPGA consists of three main components: CLBs, In/Out Blocks (IOBs) and programmable interconnections. CLBs are comprised of a Look-Up Table (LUT), a multiplexor and a flip-flop, which allow them to be configured to operate as anything from simple combination logic circuits to RAM. The programmable interconnections feature their own unique optimization in the form of routing the many interconnecting signals to tie the different CLBs together.

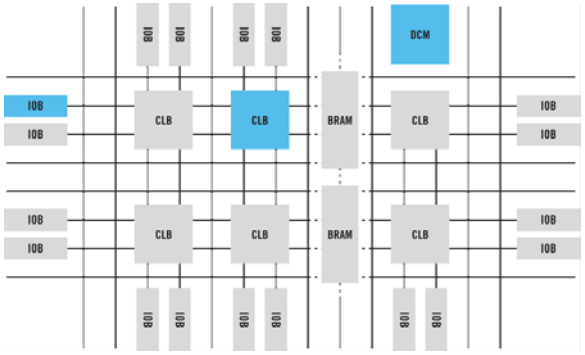

Fig. 1. FPGA Block Structure [19] .

Hardware design for FPGAs is done through a hardware descriptive language (HDL) such as (Very High Speed Integrated Circuit HDL (VHDL). Xilinx System Generator (XSG) improves hardware design by replacing the coding aspect of HDL for function blocks similar to Simulink. The function block interface makes it easy to create complex designs for a wide variety of applications. FPGAs generally operate at a much lower clock rate than a general-purpose processor, most commonly 50-100MHz. This clock rate translates to a clock cycle of 10-20ns meaning that in most FPGAs the more computational expensive processes will require pipelining. While pipelining is a viable option for most applications, it becomes problematic when dealing with real-time applications due to the extra latency introduced. Additional optimization of a system is required to achieve specific application requirements and reduce any unnecessary pipelining.

III. Design Methodology

This section presents the design of the IT2 TSK FLC used for hardware implementation. The specifications for a general purpose controller are given in the Control Design section, while the details of hardware implementation are given in the Hardware Implementation section.

A. Control Design

1) Membership Functions (MF): Due to the performance limitations imposed by the hardware implementation Gaussian and Sigmoid (MFs) could not be used because of their exponential terms making them computationally expensive; instead, triangular and trapezoidal memberships are used. Fig. 2 shows the proposed MFs which are defined within the interval [-1,1] to allow the controller to be as general as possible. By using scaling factors α_{x_i} for each input, they can tuned for specific plants.

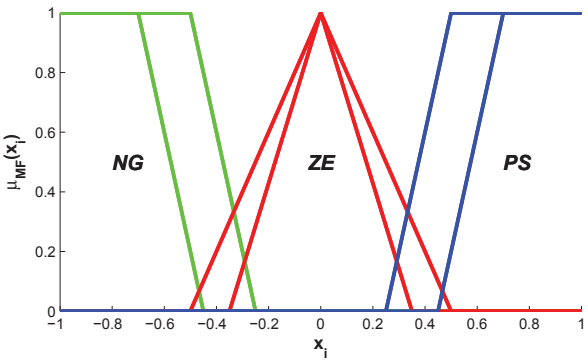

Fig. 2. Proposed IT2 Membership Functions.

2) Rule Base: The linguistic terms that correspond to the MFs shown in Fig. 2 are as follows: Negative (NG), Zero (ZE), and Positive (PS). Each term has an incrementing binary value associated with it that is used extensively in the hardware implementation. We make use of the general MacVicar-Whelan rule base shown in [20] because of its capability in achieving good tracking. Table I presents the rules used in this paper.

TABLE I. Rule Base with 9 rules [20].

$\Delta e/e$	NG	ZE	PS
NG	NG	NG	ZE
ZE	NG	ZE	PS
PS	ZE	PS	PS

3) Inference Mechanism: The inference engine used with this controller was introduced in (2). The advantage of using the BMM inference engine is that it eliminates the need to use a type reducer or any complicated defuzzification techniques. These features are very suitable for hardware implementation due to inherent physical hardware limitations involved. The simple structure of the controller makes it also easy to tune. The parameters m and n as well as the scaling factors α_{x_i} are the controller tuning parameters.

B. Hardware Implementation

This section provides details on the hardware configuration for the implementation of an IT2 TSK FLC. The hardware implementation makes extensive use of the relational operator and multiplexor blocks to ensure the correct flow of signals within the controller. When used together, these blocks form the primary method of decision making used throughout the hardware implementation. This section is organized into fuzzification, inference engine, and output subsections.

1) Fuzzification: The hardware implementation for the fuzzification stage is shown in Fig. 3 in which the inputs to the controller are $x_1, \cdots, x_i, \cdots, x_n$, where n is the number of inputs, and $x_{i,1}up$, $x_{i,2}up$, $x_{i,1}low$ and $x_{i,2}low$ are upper and lower membership grades for input x_i. The external scaling factor α_{x_i} allows identical MF blocks (NG, ZE, and PS) to be used for any input. The individual MF blocks are split between upper and lower blocks, therefore each block produces a membership grade $\mu_{MF}(x_i)$ for the upper and lower portions of the MF. Within each MF block the crisp values representing the first and last values of the support set of the MFs have been hardcoded as constant values, as well as the slope values of lines (of the MFS) represent all the boundary sets. The process for calculating the membership grade from any MF blocks (upper or lower) is given by the following steps:

- Relational operators determine if x_i is within boundary or core set of MF
- If x_i is not within boundary or core set of MF, then $\mu_{MF}(x_i) = 0$
- If x_i is within core set of MF, then $\mu_{MF}(x_i) = 1$
- If x_i is within a boundary set of MF, then $\mu_{MF}(x_i) = m(x_i - x_{\mu_o})$, where m is the slope of the line representing the boundary set and x_{μ_o} represents the x_i value that would produce the smallest non-zero $\mu_{MF}(x_i)$ with the boundary set.

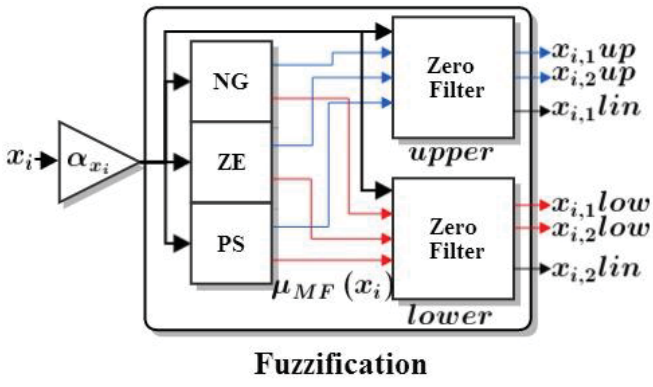

Fig. 3. Hardware Architecture of Fuzzification.

For larger number of MFs, the above methodology can be easily extended to accommodate additional linguistic terms. The "Zero Filters" shown in Fig. 3 were created to eliminate the zero valued membership grades and produce a set of coded linguistic value. The filtering process ("Zero Filters") works by dividing the MF configuration into active regions (ARs). Each AR represents a different combination of overlapping

support sets as shown in Fig. 4. Using these active regions, the upper and lower "Zero Filters" can be used to find the coded linguistic values as well as filter out the zero values. The "Zero Filters" were organized as a simple 3-step process as follows:

- A 3-bit code, representing which of the active regions the input x_i falls into, is created from relational operators and basic logic gates.
- Two 2-bit coded linguistic values ($x_{i,1}lin$ and $x_{i,2}lin$) are produced from a custom dual line 5-to-2 encoder that uses the 3-bit code as input.
- The non-zero membership grades are filtered out by a set of 4-to-1 multiplexers using the two 2-bit coded linguistic values as the selection lines to the multiplexor.

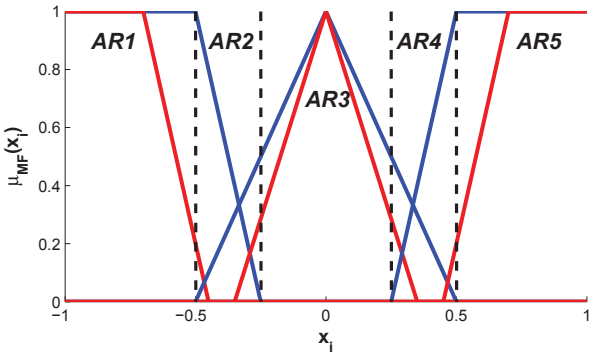

Fig. 4. Active Regions of Membership Functions.

The resulting non-zero membership grades for input x_i are labeled in Fig. 3 as $x_{i,1}up$ $x_{i,2}up$, $x_{i,1}low$ and $x_{i,2}low$. The filtering process results in a total of 4 membership grades (2 upper and 2 lower) meaning that in the current MF configuration, only 2 zero valued membership grades have been eliminated. However, while maintaining the general configuration introduced in Section III.a.1, an increase to the number of linguistic terms would have no change to the total number of non-zero membership grades, while the "Zero Filters" would eliminate 2 additional zero valued membership grades per additional linguistic term. The general configuration and the "Zero Filters" maintain the generality of the controller and ensure that addition linguistic terms would have a minimally impact only on the fuzzification stage.

2) Inference Engine: The implementation of the inference engine is organized into identical upper and lower t-norm function blocks as well as a central rule base. For simplicity, Fig. 5 shows the general architecture of the inference engine for a fuzzy system with two rules. Additional inputs could easily be accommodated by (i) expanding the rule base and (ii) modifying the t-norm function blocks. The central rule base is primarily composed of four 16 input multiplexors driven by a 4-bit concatenated antecedents (ant_1, ant_2, ant_3 and ant_4) of different combinations of the 2-bit linguistic terms ($x_{i,1}lin$ and $x_{i,2}lin$). Each multiplexor produces a 2-bit consequent (con_1, con_2, con_3 and con_4) that is required to calculate the crisp out of the controller. The bit size of con_1 would only be minimally effected by additional linguistic terms as it is a binary logarithm of the total number of linguistic terms. The size and quantity of the multiplexors will be doubled for each addition input in addition to the concatenated selection lines expanding by 2-bits. The t-norm function blocks consist of 4 identical custom-build minimization functions comprised of a single relational operator and multiplexor to select the smaller of the two input values. The inputs to the t-norm function blocks are arranged to match the different combinations of the 2-bit linguistic terms that form antecedents of the rule base.

3) Output: Fig. 6 shows a hardware representation of the BMM inference mechanism on a controller with 2 inputs. The Rule Outputs block features the hardcoded TSK consequent coefficients. Selecting which rules is activated is done through a bank of multiplexors driven by the 2-bit consequent linguistic terms (con_i). The inference mechanisms implementation is organized into upper and lower arithmetic function blocks, as shown in Fig. 6. These blocks feature the hardcoded tuning parameters m and n, and use y_i and the upper and lower consequent firing strengths ($\overline{f^i}$ and $\underline{f^i}$). The results of these arithmetic blocks are summed to produce the final output Y_{T2-TSK}. For each additional input, the number of multiplexors within the Rule Output block will be doubled, as well as the total number of addition and multiplier blocks used in each of the arithmetic blocks.

Each arithmetic block contains a single division operation. To reduce any unnecessary latency that may accompany the predefined XSG divider blocks, a Newton-Raphson division algorithm is implemented instead. This method commonly used

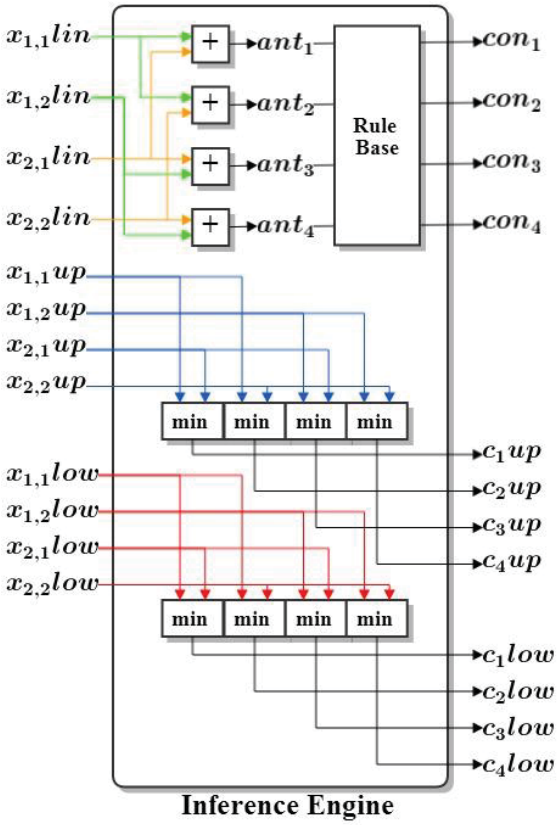

Fig. 5. Hardware Architecture of Inference Engine.

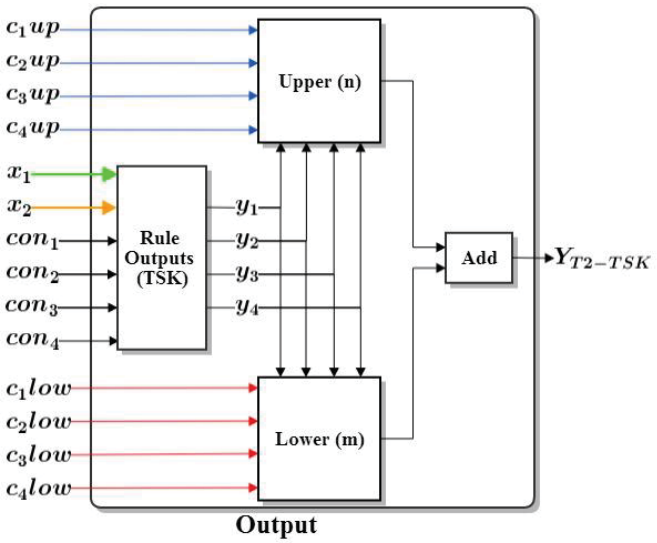

Fig. 6. Hardware Architecture of the Output.

for solving nonlinear equations, uses iterative approximations based on the following formula [21]:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \tag{4}$$

where $f(x) = 0$ and $f'(x_i)$ is the derivative of f at x_i. The division algorithm finds the reciprocal of the divisor (D) by solving the following nonlinear equation [22]:

$$f(x_i) = \frac{1}{x} - D. \tag{5}$$

Substituting (5) and its derivative into (4) yields the following iterative division algorithm:

$$x_{i+1} = x_i(2 - Dx_i) \tag{6}$$

where x_0 is an initial approximation. The accuracy of the division is dependent on the number of iterations, with each iteration involving only two multiplication operations.

IV. Results

This section presents the simulation and experimental results. First, we define a structure for the proposed IT2 FLC to test its effectiveness. Next, we present comparison results between the proposed hardware implementation of the controller and a Simulink model of the controller in a discrete environment. Lastly, we present the results of the hardware implementation on a Nexys3 Spartan-6 FPGA.

A. Controller Stucture

A PD IT2 TSK FLC is considered for implementation. The controller inputs are error (e) and the rate of change of the error (Δe). The rule structure for this controller is defined as

$$\text{If } e \text{ is } \tilde{F}_1^i \text{ and } \Delta e \text{ is } \tilde{F}_2^i, \text{ then } y_i = a_1^i e + a_2^i \Delta e \tag{7}$$

where i, $\tilde{F}_k^i$, y, M, a_k^i have all been defined in Section II Part A.

To evaluate the controller performances, two different plants were considered.

1) The model of the first plant in the Z-domain is given as follows:
$$\frac{z}{z-1}. \tag{8}$$

2) The model of the second plant in the Z-domain is given as follows:
$$\frac{0.4323z + 4.8 \times 10^{-17}}{z^2 - 1.135z + 0.1353}. \tag{9}$$

B. Simulations

The proposed hardware for implementing the IT2 FLC on hardware was constructed using XSG. The resolution of the arithmetic operations for the controller was specified to be 16 bits, with the decimal point after the 8 least significant bit. This bit resolution is large enough to accurately represent most decimal numbers, but small enough to maintain low operating times for most arithmetic functions. General-purpose processors generally operate with a bit resolution of 32-64 bits. With only a 16 bit resolution, there is a potential for small discrepancies occurring in a direct comparison between the controller output values. Another potential source for discrepancy is in the NR division algorithm used. In this implementation, two iterations of the NR algorithm are used to produce the crisp output.

The Simulink model used to perform the comparison was constructed to match the hardware implementation directly. The simulation was performed in a discrete environment. A Simulink model was tuned using the tuning parameters α_e, $\alpha_{\Delta e}$, n and m. Once the controller was properly tuned, these values were using in the hardware implementation to perform a proper comparison. The simulation results for the two plants are reported in the following subsections:

1) Plant 1: The comparison results of plant 1 are show in Fig. 7 where the tuning parameters used are as follows: $\alpha_e = 10$, $\alpha_{\Delta e} = 0.3$, $n = -0.1$ and $m = 0.2$. These results show the controller is capable of a fast response time as well as near perfect tracking.

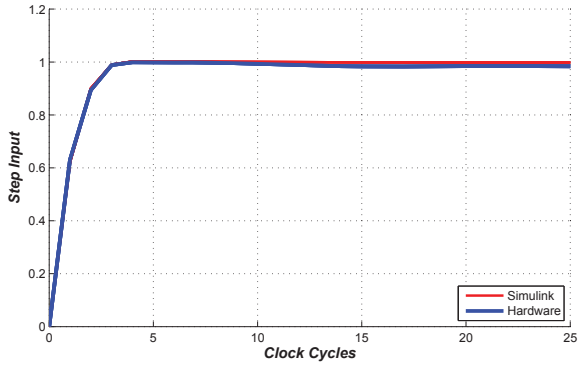

Fig. 7. Comparison between XSG and Simulink of proposed IT2 FLC hardware architecture for plant 1.

TABLE II. SPARTAN-6 FPGA RESOURCES.

Resource	Used	Available	Percentage(%)
Number of DSP48A1s	10	32	31
Number of CLB Slices	872	2178	38
Number of Slice Registers	485	18224	2
Number of Slice LUTs	2712	9112	29

2) Plant 2: The results of this comparison are show in Fig. 8 where the tuning parameters used are as follows: $\alpha_e = 12$, $\alpha_{\Delta e} = 0.3$, $n = -0.3$ and $m = 0.4$. These results show the controller is still capable of a fast response time as well as perfect tracking even with a second-order plant.

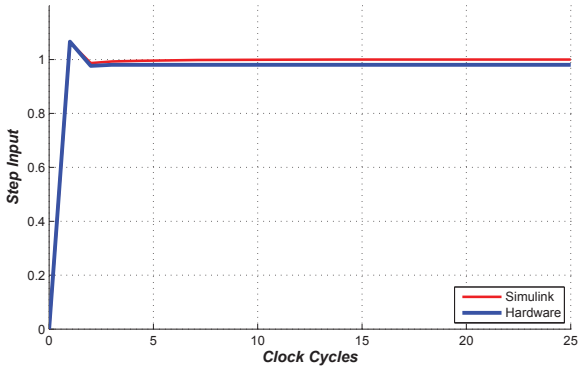

Fig. 8. Comparison between XSG and Simulink of proposed IT2 FLC hardware architecture for plant 2.

C. Experiments

In this section, using the same controllers in the Simulations section, we implement the IT2 FLC on an actual FPGA and report the controller performance in real-time. The specific model of the FPGA used to perform the experiments is a Spartan-6 XC6SLX16 shown in Fig. 9. The resources available for this FPGA are shown in Table II along with their usage (in percentage) by the proposed hardware implementation structure. This table makes reference to CLB slices and DSP48A1. The Spartan-6 family of FPGAs feature CLB slices that contain four LUT and eight flip-flops each, while the DPS48A1 are dedicated arithmetic functions that feature an 18 by 18 multiplier, an adder and an accumulator [23]. The DSP48A1 blocks are key features making this controller design possible. For example, by using these function blocks, multiplication between two unknown signals at a rate of approximately 8-9ns is possible regardless of how computationally expensive a task might be. However, because of the rate of a single multiplication, the hardware implementation is required to be run at a clock rate of 50MHz to allow multiple multiplication operations occurring within a single 20ns clock cycle. In addition, to reduce the number of multiplication blocks that are in serial, the NR algorithm was reduced to a single iteration.

The proposed IT2 FLC hardware implementation was broken down into a three stage pipeline, resulting in a total operation time of 60ns. However, due to the added latency introduced by the pipelining the controller was required to be re-tuned. The results of the re-tuned FPGA implementation are shown in Fig. 10 & 11 where the tuning parameters are now set to

Fig. 9. Nexys3 Spartan-6 FPGA Board [24].

$\alpha_e = 4$, $\alpha_{\delta e} = 0.25$, $n = 0.01$ and $m = 0.04$ for plant 1 and $\alpha_e = 6$, $\alpha_{\delta e} = 0.35$, $n = 0.01$ and $m = 0.05$ for plant 2. The three stage pipeline is responsible for the small latency in the system response as shown in Fig. 10 & 11. Comparisons with the simulation results from Fig. 7 & 8 show that despite a larger rise and settling time the controller is still capable of perfect tracking. When properly tuned, the controller is capable of reaching the steady state set point in 15 clock cycles achieving a real time value of 0.3μs, which is considerably faster then what is being reported in literature for steady state responses [6], [8], [10], [12], [13]. Therefore, the proposed control structure using the BMM inference engine can outperform other IT2 FLC implemented and hence can be a viable structure for real-time implementation

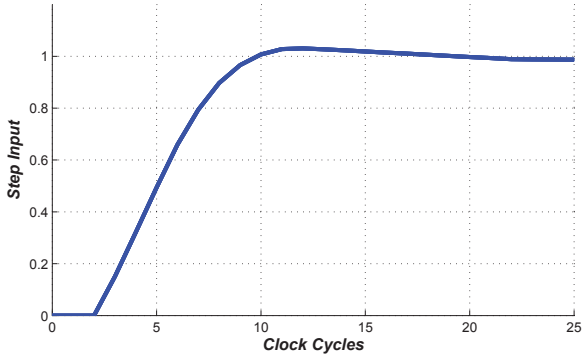

Fig. 10. Spartan-6 FPGA results of proposed IT2 FLC hardware architecture for plant 1.

V. CONCLUSION

We presented a hardware architecture for implementing an IT2 FLC designed for any real-time application. We developed a methodology for designing the controller in regards to the limitations of a hardware implementation. The IT2 FLC was implemented in XSG as well as Simulink and the results

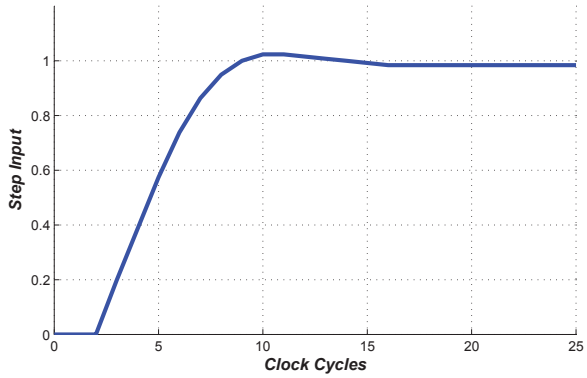

Fig. 11. Spartan-6 FPGA results of proposed IT2 FLC hardware architecture for plant 2.

were compared to verify the effectiveness of the controller for two different plants. Finally, the hardware implementation was tested experimentally on a Spartan-6 FPGA on the two plants. The controller was capable of responding to a change of input after only $60ns$ which was easily compensated for by retuning the controller. By using the BMM inference engine, the implemented controllers were capable of performing at a much higher rate than what is currently being reported in literature. In addition, due to the straight forward implementation of (2), the FPGA area used by the IT2 FLC is considerably smaller than what any other type reducers would use. Furthermore, we have shown that the FPGA hardware implementation is capable of controlling plants in real time with a fast response and near perfect tracking.

REFERENCES

[1] H. Hagras, "Type-2 FLCs: A new generation of fuzzy controllers," *IEEE Comput. Intell. Mag.*, vol. 2, no. 1, pp. 3044, Feb. 2007.

[2] M. Biglarbegian, W. W. Melek, and J. M. Mendel, "On the stability of interval type-2 TSK fuzzy logic control systems," *IEEE Trans. Systems, Man, Cybern. B*, vol. 40, no. 3, pp. 798818, Jun. 2010.

[3] C.-F. Juang and C.-H. Hsu, "Reinforcement ant optimized fuzzy controller for mobile-robot wall-following control," *IEEE Trans. Ind. Electron.*, vol. 56, no. 10, pp. 39313940, Oct. 2009.

[4] O. Castillo and P. Melin, " A review on the design and optimization of interval type-2 fuzzy controllers," *Applied Soft Computing*, pp. 1267-1278, 2012

[5] Y. Maldonado and O. Castillo, "Design and optimization of type-2 fuzzy system in FPGAs," *World Automation Congress*, pp. 24-28, 2010.

[6] Y. Maldonado, O. Castillo and P. Melin, "Optimal Design of Type-2 Fuzzy Controllers with a Multiple Objective Genetic Algorithm for FPGA Implementation," *North American Fuzzy Information Processing*, pp. 1-6, 2010

[7] Y. Maldonado, O. Castillo and P. Melin, " Particle swarm optimization of interval type-2 fuzzy systems for FPGA applications," *Applied Soft Computing*, pp. 496-508, 2013

[8] O. Linda and M. Manic, "Uncertainty-robust design of interval type-2 fuzzy logic controller for delta parallel robot," *IEEE Trans. Ind. Informat.*, vol. 7, no. 4, pp. 661671, Nov. 2011.

[9] X. T. Chen and W. W. Tan, "An adaptive type-2 fuzzy logic controller for dynamic positioning, *IEEE Int. Conf. on Fuzzy Syst.*, Taipei, Taiwan, pp. 21472154, 2011.

[10] M. Karimi and B. Safarinejadian, "On-line control of the inverted pendulum with type-2 fuzzy logic controller," *2nd Int. Conf. on Control, Instrumentation and Automation (ICCIA)*, Shiraz, Iran, pp. 429-433, 2011.

[11] M. A. Melgarejo R. and C. A. Peña-Reyes,"'Hardware architecture and FPGA implementation of a type-2 fuzzy system." *Proc. 14th ACM Great Lakes Symp. VLSI*, pp. 458-461, 2004.

[12] H. C. Huang and C. C. Tsai, "FPGA Implementation of an Embedded Robust Adaptive Controller for Autonomous Omnidirectional Mobile Platform," *IEEE Trans. Ind. Electron.*, vol. 56, no. 5, pp. 1604-1616, May 2009.

[13] R. Sepúlveda, O. Montiel, O. Castillo, and P. Melin, "Embedding a high speed interval type-2 fuzzy controller for a real plant into an FPGA," *Applied Soft Computing 12*, no. 3, pp. 988-998, 2012.

[14] L. Leottau, M. and Melgarejo, "A simple approach for designing a type-2 fuzzy controller for a mobile robot application," *North American Fuzzy Information Processing Society (NAFIPS)*, pp. 12-14, 2010.

[15] J. M Mendel, *Uncertain Rule-Based Fuzzy Logic Systems: Introduction and New Directions*, Upper Saddle River, NJ: Prentice-Hall, 2001.

[16] C. Lynch, H. Hagras and V. Callaghan, "Using uncertainty bounds in the design of an embedded real-time type-2 neuro-fuzzy speed controller for marine diesel engines", *IEEE International Conference Fuzzy Systems*, pp. 1446-1453, July 2006

[17] C. Lynch, H. Hagras and V. Callaghan, "Parallel type-2 fuzzy logic co-processors for engine management," *Fuzzy Systems Conference*, pp. 1-6, July 2007.

[18] M. B. Begian, W. W. Melek, and J. M. Mendel, "Stability analysis of type-2 fuzzy systems," *Proc. IEEE World Congress on Computational Intelligence (WCCI08)*, Hong Kong, pp. 1305-1310, June 2008.

[19] Xilinx. (2013). What is a FPGA [Online]. Available: http://www.xilinx.com/fpga/

[20] S. Chopra, R. Mitra, and V. Kumar, "Fuzzy controller: chossing an appropiate and smallest rule set," *Int. Journal of Computational Cognition*, vol. 3, no. 4, pp. 73-79, 2005.

[21] G. B. Thomas, *Calculas and Analytic Geometry*, Reading, MA: Addison-Wesley, 1962.

[22] M. J. Flynn, "On division by functional iteration," *IEEE Trans. on Computers*, vol. C-19, pp. 702-706, Aug. 1970.

[23] Xilinx. (2011, Oct. 25). Spartan-6 Family Overview(v2.0) [Online]. Available: http://www.xilinx.com/support/documentation/data_sheets/ds160.pdf

[24] Digilent, Inc. Nexys3^{TM} Spartan-6 Family FPGA Board [Online]. Available: http://www.digilentinc.com/Products/Detail.cfm?Prod=NEXYS3