Journal of Intelligent and Robotic Systems (2006) 45: 77–96
DOI: 10.1007/s10846-005-9016-2

Design and Implementation of a Fast Digital Fuzzy Logic Controller Using FPGA Technology

K. M. DELIPARASCHOS, F. I. NENEDAKIS and S. G. TZAFESTAS
Intelligent Robotics and Automation Laboratory, School of Electrical and Computer Engineering, National Technical University of Athens, Zographou Campus, Athens, GR 157 73, Greece;
e-mail: kdelip@mail.ntua.gr, el99199@mail.ntua.gr, tzafesta@softlab.ntua.gr

(Received 15 June 2005; in final form 8 October 2005)

Abstract. Fuzzy logic controllers (FLCs) are finding increasing popularity in real industrial applications, especially when the available system models are inexact or unavailable. This paper proposes a zero-order Takagi–Sugeno parameterized digital FLC, processing only the active rules (rules that give a non-null contribution for a given input data set), at high frequency of operation, without significant increase in hardware complexity. To achieve this goal, an improved method of designing the fuzzy controller model is proposed that significantly reduces the time required to process the active rules and effectively increases the input data processing rate. The digital fuzzy logic controller discussed in this paper achieves an internal core processing speed of at least 200 MHz, featuring two 8-bit inputs and one 12-bit output, with up to seven trapezoidal shape membership functions per input and a rule base of up to 49 rules. The proposed architecture was implemented in a field programmable gate array chip with the use of a very high-speed integrated-circuits hardware description language and advanced synthesis and place and route tools.

Key words: digital fuzzy logic controller, odd–even method, register transfer level,
very high-speed hardware description language, place and route, synthesis.

1. Introduction

Fuzzy chips are classified into two categories depending on the design techniques employed: digital and analog. The first fuzzy chip was reported in 1986 at AT&T Bell Laboratories (Murray Hill, NJ) [4]. The digital approach originated from Togai and Watanabe's paper [4] and resulted in some interesting chips [5, 6]. Other digital architectures were reported in [11, 12]. Analog chip approaches begun with Yamakawa in [7, 8].

This paper discusses an improved method, referred in the text as the odd–even method, to design digital fuzzy logic controllers (DFLCs). This method reduces the clock cycles required to generate the active rule addresses to be processed. Typical fuzzy logic controller architectures found in the literature [9, 10] assuming overlap of two between adjacent fuzzy sets consume 2^n clock cycles (input data processing rate) where n is the number of inputs, since they process one active rule per clock cycle. Using the proposed odd–even method, we manage to process for the same model case scenario, two active rules per clock cycle, thus increasing significantly the input data processing rate of the system to $2^n/2$. The architecture of the design allows us to achieve a core frequency speed of 200 MHz, while the input data can be sampled at a clock rate equal to the half of the core frequency speed (100 MHz), while processing only the active rules. The presented DFLC is based on a simple algorithm similar to the Takagi–Sugeno order zero inference and defuzzification method and employs two 8-bit inputs and a 12-bit output, with up to seven trapezoidal or triangular shape membership functions per input and a rule base of up to 49 rules. The total latency of the chip architecture involves 13 pipeline stages each one requiring 5 ns.

2. Fuzzy Logic

2.1. FUZZY SET THEORY

Fuzzy set theory was formalized by Lotfi Zadeh in 1965 [1], where he introduced the fuzzy set concept. Zadeh decided to extend the two-value logic, defined by the binary pair {0, 1}, to the whole continuous interval [0, 1], introducing a gradual transition from falsehood to truth. A "crisp" set is closely related to its members, such that an item from a given universe is either a member (1) or not (0). Zadeh suggested that many sets have more than a yes (1) or not (0) crisp criterion for membership and he proposed a *grade of membership* (μ) in the interval [0, 1], such that the decision whether an element (x) belongs or not to a set (A) is gradual than crisp (or binary). In other words, fuzzy logic replaces "true or 1" and "false or 0" with a continuous set membership values ranging in the interval from 0 to 1.

If U defines a collection of elements denoted by x, then a fuzzy set A in U is defined as a collection of ordered pairs:

$$A = \{(x, \mu_A(x)) | x \in U\}$$

The elements (x) of a fuzzy set are said to belong to a *universe of discourse*, U, which effectively is the range of all possible values for an input to a fuzzy system. Every element in this universe of discourse is a member of the fuzzy set to some degree.

The function $\mu_A(x)$ is called a *membership function* and assigns a degree of truth number in the interval [0, 1] to each element of x of U in the fuzzy set A.

2.2. TAKAGI–SUGENO METHOD

The DFLC proposed in this paper is based on the Takagi–Sugeno zero-order fuzzy model [2, 3]. It is well known that the Takagi–Sugeno (T-S) fuzzy model can provide an effective representation of complex nonlinear systems in terms of fuzzy sets and fuzzy reasoning. The T-S method is considered to be quite simple as a method, leads to fast calculations and is relatively easy to apply. Moreover, a fuzzy controller based on the T-S method provides a good trade-off between the hardware simplicity and the control efficiency. In the T-S inference rule, the conclusion is expressed in the form of linear functions. Rules have the following form:

$$\begin{aligned}&\text{Rule } R_i\text{: IF } x_1 \text{ IS } A_i^1 \text{ AND} \ldots \text{AND } x_k \ \varepsilon\acute{\iota}\nu\alpha\iota \ A_i^k\\ &\qquad\qquad \text{THEN } y_i = c_i^0 + c_i^1 x_1 + \ldots + c_i^k x_k\end{aligned}$$

where $x_1, \ldots, x_k$ represent the input variables, $A_i^1, \ldots, A_i^k$ represent the input membership functions, y_i represent the output variable and $c_i^0, \ldots, c_i^k$ are all constants. A zero-order model arises (simplified T-S functional reasoning) if we only use the constant c_i^0 at the output and therefore $y_i = c_i^0$ (singleton outputs). In the T-S model, inference with several rules proceeds as usual, with a firing strength associated with each rule, but each output is linearly dependent on the inputs. The output from each rule is a moving singleton, and the deffuzified output is the weighted average of the contribution of each rule, as shown in the following equation:

$$y = \frac{\sum_{i=1}^{m} w^i y^i}{\sum_{i=1}^{m} w^i}$$

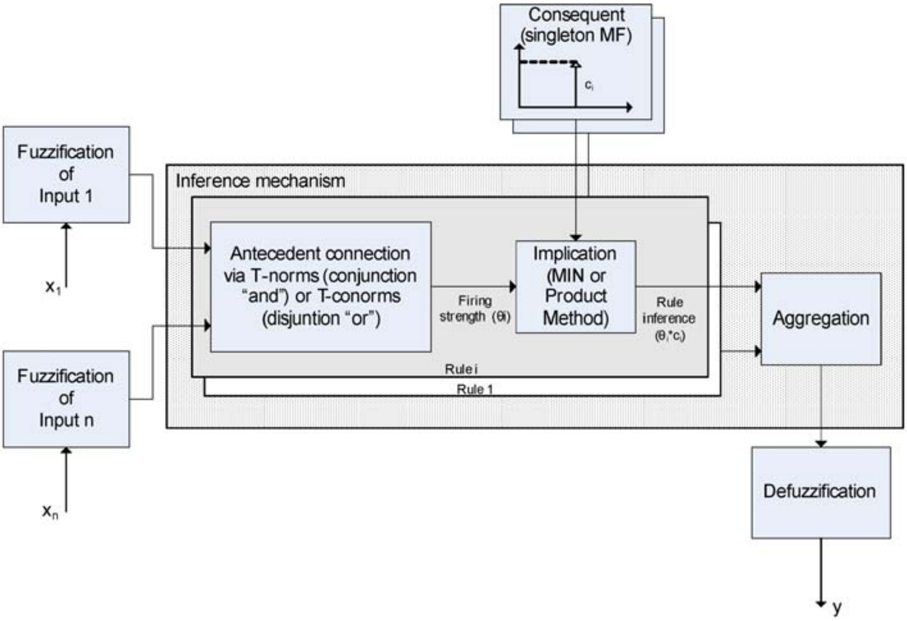

Figure 1. T-S mechanism.

where w^i is the weight contribution of the left part of the ith rule and is given by

$$w^i = \prod_{k=1}^{n} \mu_{A_k^i}(x_k)$$

Figure 1 illustrates the T-S mechanism.

2.3. DFLC CHARACTERISTICS

In this section, the parameters characterizing the proposed DLFC are presented. These are summarized in Table I.

3. DFLC Hardware Implementation

This section attempts an analytical description of the numerous hierarchical blocks of the DFLC architecture and also explains the odd–even method used to increase the input data processing rate of the controller.

3.1. DLFC ARCHITECTURE

The DLFC architecture studied here is shown in Figure 2. The architecture is mainly broken in two major blocks: “Fuzzification and Aggregation” and “Inference and Defuzzification.” The dotted lines on the figure indicate the number of pipeline stages for each block. Signal bus sizes are noted as well.

Table I. DLFC characteristics.

Fuzzy inference system (FIS) type	Takagi–Sugeno zero-order
Inputs	2
Input resolution	8 bit
Outputs	1
Output resolution	12 bit
Antecedent membership functions (MFs)	Seven trapezoidal shaped per fuzzy set
Antecedent MF degree of truth (α value) resolution	4 bit
Consequent MFs	49 singleton type
Consequent MF resolution	8 bit
Maximum number of fuzzy inference rules	49 (number of fuzzy sets $^{\text{number of inputs}}$)
Aggregation method	MIN (T-norm operator implemented by minimum)
Implication method	PROD (product operator)
MF overlapping degree	2
Defuzzification method	Weighted average

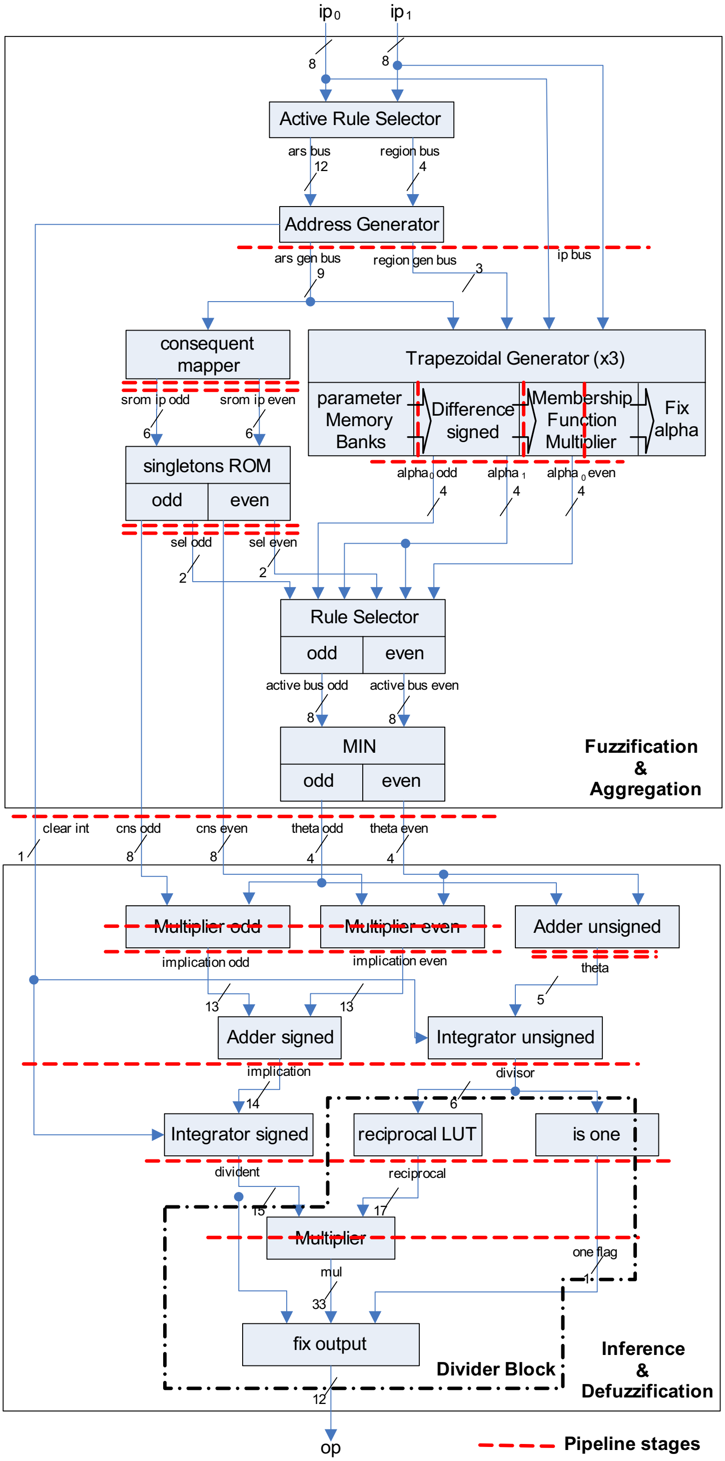

Figure 2. DLFC architecture.

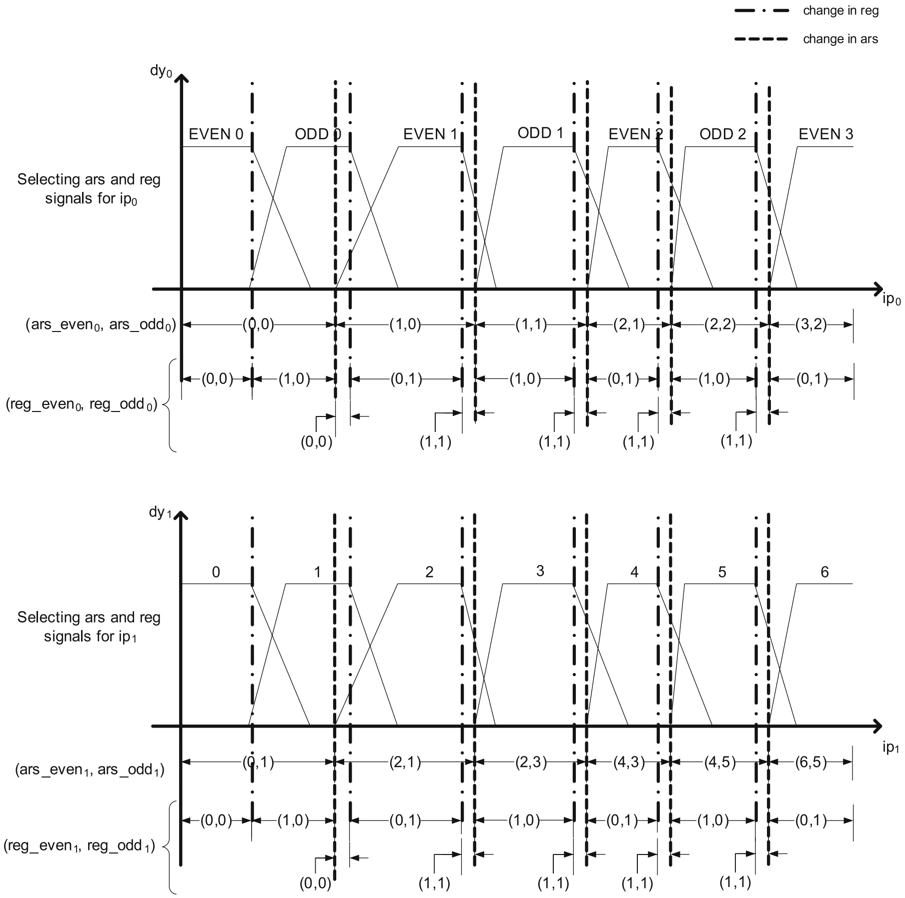

Figure 3. FS coding and odd–even regions.

3.2. ACTIVE RULE SELECTION BLOCK

Instead of processing all the rule combinations for each new data input set, we have chosen to process only the active rules. Dealing with the above problem, and given the fact that the overlapping between adjacent fuzzy sets is 2, we have implemented an active rule selection block (ARS) to calculate the fuzzy sets region in which the input data correspond to. As a result, for the two-input DFLC with seven membership functions per input instead of processing all 49 rule combinations in the rule base, only four active rules remain that need to be taken into account. Figure 3 illustrates the fuzzy set (FS) coding used for both inputs, as well as the FS area split-up that occurs by repeatedly comparing the two input data values ($\mathrm{ip}_{0,1}$) with the rising start points ($\mathrm{rise\_start}_{0,1}$) of the fuzzy sets. Here, it is useful to point out that the comparison of the first and second FS is not necessary, since the initial FS pair is the first and second always, and we only shift right to the next FS pair when the rise_start of the 3rd FS is exceeded. Since we require the transition to the adjacent FS pair when $\text{ip}_{0,1} = \text{rise\_start}_{0,1}$ and not when $\text{ip}_{0,1} > \text{rise\_start}_{0,1}$, the sign (MSB) of $\text{ip}_{0,1} - \text{rise\_start}_{0,1}$ is calculated in this order. The FS section transition is depicted in Figure 3 as line type named "change in ars." It is also shown for the selected pair (even–odd), whether $\text{ip}_{0,1}$ lies in the rising (value 0) or falling part (value 1) of each FS. The sections that regions change occurs are notable as line type named "change in reg." It should be noted here that computation of the regions is identical for both inputs, and also that the last FS is not taken into account since it is an S-shape membership function with no falling start point ($\text{fall\_start}_{0,1}$). Generally,

i. $\text{ars}_{0,1}\text{\_odd}$, $\text{ars}_{0,1}\text{\_even} \rightarrow$ active FS pair

ii. $\text{reg\_odd}_{0,1}\begin{cases} 0, \text{ if } \text{ip}_{0,1} < \text{fall\_start}_{\text{ars\_odd}_{0,1}} \\ 1, \text{ if } \text{ip}_{0,1} \geq \text{fall\_start}_{\text{ars\_odd}_{0,1}} \end{cases}$

iii. $\text{reg\_even}_{0,1}\begin{cases} 0, \text{ if } \text{ip}_{0,1} < \text{fall\_start}_{\text{ars\_even}_{0,1}} \\ 1, \text{ if } \text{ip}_{0,1} \geq \text{fall\_start}_{\text{ars\_even}_{0,1}} \end{cases}$

The algorithm used in the ARS block is shown in Figure 4.

3.3. ADDRESS GENERATOR BLOCK

The address generator (ADG) block (Figure 5) outputs the indicated by the ARS block addresses, needed for the active fuzzy rules ($\text{ars\_even}_{0,1}$, $\text{ars\_odd}_{0,1}$, $\text{reg\_even}_{0,1}$, $\text{reg\_odd}_{0,1}$). Since the ARS block has already identified previously all the involved fuzzy sets at once (fully combinatorial), the ADG produces clock by clock period the addresses, which correspond to the active fuzzy rules. Normally, for four active rule addresses, the ADG would require four clock periods to generate the active rule addresses [9], consequently increasing the data input sampling rate period to four times the internal clock period. Here, by using the odd–even scheme, we manage to reduce the clock period number required by the ADG to only 2.

3.4. TRAPEZOIDAL MEMBERSHIP FUNCTION GENERATOR

The flow chart for the trapezoidal membership function generator (TMFG) is shown in Figure 6.

Three TMFG block instances required in the architecture and accepts as inputs the controller inputs (ip_0, ip_0, ip_1), region (reg_0, reg_1, reg_2), the start point and the slope from the corresponding parameter memory bank ROM (addressed by $\text{ars}_0\&\text{reg}_0$, $\text{ars}_1\&\text{reg}_1$, $\text{ars}_2\&\text{reg}_2$ concatenated signals), for the first, second and third TMFG, respectively. The output named alpha represents the degree of truth of the current fuzzy set. Flag *zer* depicts the special case where $\text{ip}_{0,1} <$

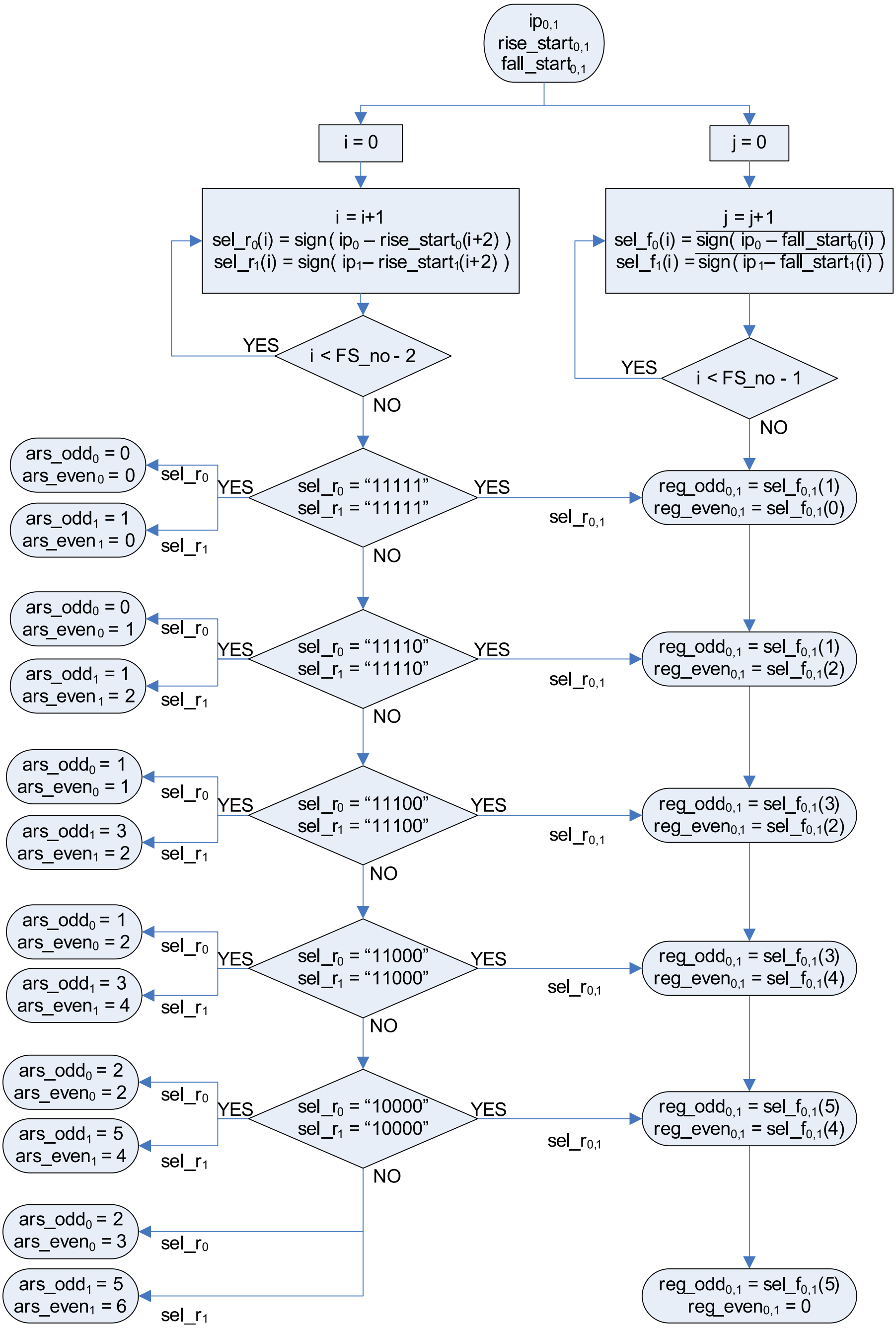

Figure 4. ARS block algorithm.

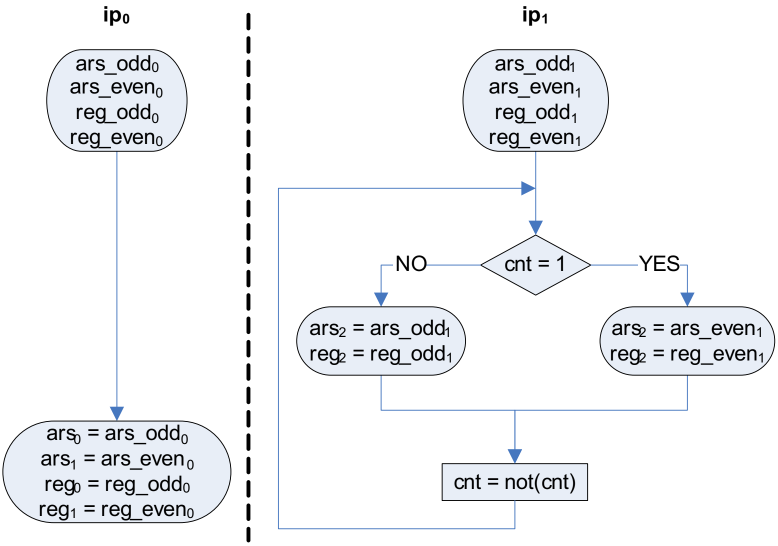

Figure 5. Address generator algorithm.

rise_start$_{0,1}$, and flag *ovf* checks whether alpha value overflow has been reached. We treat the rise and fall sections of the trapezoidal shape in the same way but distinguished with a logical NOT for the fall section. The effect of the last is that the fall section arises as the symmetrical of the rise section with respect of the universe of discourse axis.

3.5. PARAMETER AND SINGLETONS ROMs

The TMFG parameter memory banks are organized as shown in Table II.

The singletons ROM organization is shown in Table III.

3.6. CONSEQUENT ADDRESS MAPPER

The consequent address mapper block is simply used to address the singletons ROM by generating two addressing signals, named in Figure 7 as srom_ip odd and srom_ip even. The input signals ars$_0$ and ars$_1$ are multiplied by the number of fuzzy sets of each input, which in our case is 7.

3.7. RULE SELECTION BLOCK

The purpose of this block (Figure 8) is to allow for the selection of the desired rules stored in the rule base, rules that will contribute to the final result for an input data set. The rule combinations are stored in the singletons ROM. The

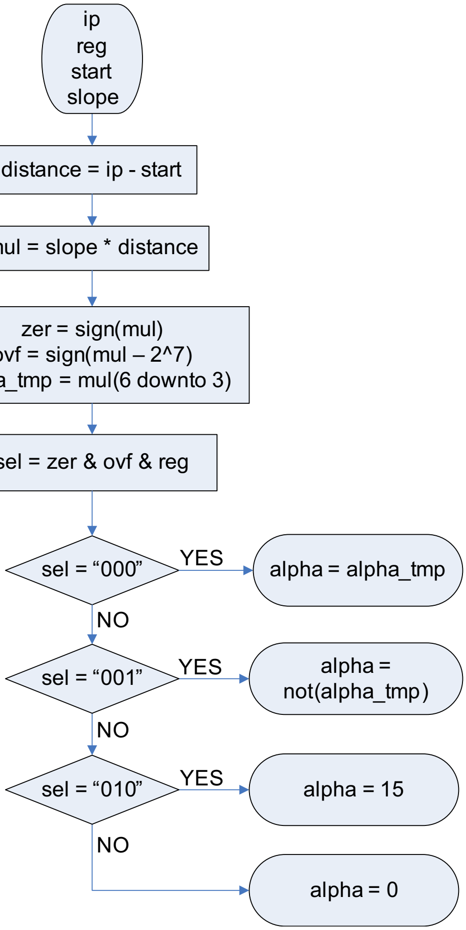

Figure 6. TMFG block flow chart.

selection is performed in the following manner. The antecedents of each rule are either stored in ROM as 0 or 1, thus enabling or disabling part or the entire rule. Due to the odd–even method followed in this architecture, it is obvious that two similar rule selector blocks are needed, since there are two rules to be examined at every clock cycle.

The odd rule selector block accepts as inputs the alpha$_0$even and alpha$_1$ of the corresponding trapezoidal generators and the selection signal active_sel of the odd consequents ROM (sel_odd):

- alpha0 = alpha0odd
- alpha1 = alpha1
- active_sel = sel_odd

Table II. Parameter memory banks.

ip_0 Odd		ip_0 Even		ip_1	
rise_startODD0	rise_slopeODD0	rise_startEVEN0	rise_slopeEVEN0	rise_start0	rise_slope0
fall_startODD0	fall_slopeODD0	fall_startEVEN0	fall_slopeEVEN0	fall_start0	fall_slope0
rise_startODD1	rise_slopeODD1	rise_startEVEN1	rise_slopeEVEN1	rise_start1	rise_slope1
fall_startODD1	fall_slopeODD1	fall_startEVEN1	fall_slopeEVEN1	fall_start1	fall_slope1
rise_startODD2	rise_slopeODD2	rise_startEVEN2	rise_slopeEVEN2	rise_start2	rise_slope2
fall_startODD2	fall_slopeODD2	fall_startEVEN2	fall_slopeEVEN2	fall_start2	fall_slope2
		rise_startEVEN3	rise_slopeEVEN3	rise_start3	rise_slope3
				fall_start3	fall_slope3
				rise_start4	rise_slope4
				fall_start4	fall_slope4
				rise_start5	rise_slope5
				fall_start5	fall_slope5
				rise_start6	rise_slope6

Similarly, the even rule selector accepts as inputs the alpha$_0$even and alpha$_1$ of the corresponding trapezoidal generators and the selection signal active_sel of the even consequents ROM (sel_even):

- alpha0 = alpha0even
- alpha1 = alpha1
- active_sel = sel_even

More specifically, in reference to Figure 8, when the antecedent of a rule is not active, it will not contribute to the final weight of the rule. Provided that the weight contribution of the rule (*theta* value) is extracted using the min operator, we can ignore the *alpha* value of the chosen antecedent by setting it to its maximum value which in our case for 4-bit degree of truth resolution (unsigned number since alpha value $\in R^+$) is $2^4 - 1 = 15$. Finally, if all rule antecedents are inactive, at least one alpha value must be set to a minimum value, so that the weight contribution of the rule is zero.

3.8. MULTIPLIER BLOCKS

The Spartan 3 family of field programmable gate array (FPGA) devices [15] used to implement the proposed architecture provides embedded multipliers, where their number varies depending on the family code. At present, we have used a Spartan 3 1500 FPGA providing a total of 32 dedicated multipliers. The input buses to the multiplier accept data in two's-complement form (either 18-bit signed or 17-bit unsigned). One such multiplier is matched to each block RAM on the die.

Table III. Singletons ROM.

Odd		Even	
cns$_{\text{ODD0\&0}}$	active_sel$_{\text{ODD0\&0}}$	cns$_{\text{EVEN0\&0}}$	active_sel$_{\text{EVEN0\&0}}$
cns$_{\text{ODD0\&1}}$	active_sel$_{\text{ODD0\&1}}$	cns$_{\text{EVEN0\&1}}$	active_sel$_{\text{EVEN0\&1}}$
cns$_{\text{ODD0\&2}}$	active_sel$_{\text{ODD0\&2}}$	cns$_{\text{EVEN0\&2}}$	active_sel$_{\text{EVEN0\&2}}$
cns$_{\text{ODD0\&3}}$	active_sel$_{\text{ODD0\&3}}$	cns$_{\text{EVEN0\&3}}$	active_sel$_{\text{EVEN0\&3}}$
cns$_{\text{ODD0\&4}}$	active_sel$_{\text{ODD0\&4}}$	cns$_{\text{EVEN0\&4}}$	active_sel$_{\text{EVEN0\&4}}$
cns$_{\text{ODD0\&5}}$	active_sel$_{\text{ODD0\&5}}$	cns$_{\text{EVEN0\&5}}$	active_sel$_{\text{EVEN0\&5}}$
cns$_{\text{ODD0\&6}}$	active_sel$_{\text{ODD0\&6}}$	cns$_{\text{EVEN0\&6}}$	active_sel$_{\text{EVEN0\&6}}$
cns$_{\text{ODD1\&0}}$	active_sel$_{\text{ODD1\&0}}$	cns$_{\text{EVEN1\&0}}$	active_sel$_{\text{EVEN1\&0}}$
cns$_{\text{ODD1\&1}}$	active_sel$_{\text{ODD1\&1}}$	cns$_{\text{EVEN1\&1}}$	active_sel$_{\text{EVEN1\&1}}$
cns$_{\text{ODD1\&2}}$	active_sel$_{\text{ODD1\&2}}$	cns$_{\text{EVEN1\&2}}$	active_sel$_{\text{EVEN1\&2}}$
cns$_{\text{ODD1\&3}}$	active_sel$_{\text{ODD1\&3}}$	cns$_{\text{EVEN1\&3}}$	active_sel$_{\text{EVEN1\&3}}$
cns$_{\text{ODD1\&4}}$	active_sel$_{\text{ODD1\&4}}$	cns$_{\text{EVEN1\&4}}$	active_sel$_{\text{EVEN1\&4}}$
cns$_{\text{ODD1\&5}}$	active_sel$_{\text{ODD1\&5}}$	cns$_{\text{EVEN1\&5}}$	active_sel$_{\text{EVEN1\&5}}$
cns$_{\text{ODD1\&6}}$	active_sel$_{\text{ODD1\&6}}$	cns$_{\text{EVEN1\&6}}$	active_sel$_{\text{EVEN1\&6}}$
cns$_{\text{ODD2\&0}}$	active_sel$_{\text{ODD2\&0}}$	cns$_{\text{EVEN2\&0}}$	active_sel$_{\text{EVEN2\&0}}$
cns$_{\text{ODD2\&1}}$	active_sel$_{\text{ODD2\&1}}$	cns$_{\text{EVEN2\&1}}$	active_sel$_{\text{EVEN2\&1}}$
cns$_{\text{ODD2\&2}}$	active_sel$_{\text{ODD2\&2}}$	cns$_{\text{EVEN2\&2}}$	active_sel$_{\text{EVEN2\&2}}$
cns$_{\text{ODD2\&3}}$	active_sel$_{\text{ODD2\&3}}$	cns$_{\text{EVEN2\&3}}$	active_sel$_{\text{EVEN2\&3}}$
cns$_{\text{ODD2\&4}}$	active_sel$_{\text{ODD2\&4}}$	cns$_{\text{EVEN2\&4}}$	active_sel$_{\text{EVEN2\&4}}$
cns$_{\text{ODD2\&5}}$	active_sel$_{\text{ODD2\&5}}$	cns$_{\text{EVEN2\&5}}$	active_sel$_{\text{EVEN2\&5}}$
cns$_{\text{ODD2\&6}}$	active_sel$_{\text{ODD2\&6}}$	cns$_{\text{EVEN2\&6}}$	active_sel$_{\text{EVEN2\&6}}$
		cns$_{\text{EVEN3\&0}}$	active_sel$_{\text{EVEN3\&0}}$
		cns$_{\text{EVEN3\&1}}$	active_sel$_{\text{EVEN3\&1}}$
		cns$_{\text{EVEN3\&2}}$	active_sel$_{\text{EVEN3\&2}}$
		cns$_{\text{EVEN3\&3}}$	active_sel$_{\text{EVEN3\&3}}$
		cns$_{\text{EVEN3\&4}}$	active_sel$_{\text{EVEN3\&4}}$
		cns$_{\text{EVEN3\&5}}$	active_sel$_{\text{EVEN3\&5}}$
		cns$_{\text{EVEN3\&6}}$	active_sel$_{\text{EVEN3\&6}}$

The close physical proximity of the two ensures efficient data handling. The multiplier is placed in a design using one of two primitives: an asynchronous version called MULT18X18 and a version with a register at the outputs called MULT18X18S. In the present design, the latter multiplier type was used along the design as it was proved necessary in order to achieve better timing results.

3.9. DIVIDER BLOCK

Division remains one of the most hardware-consuming operations. Several methods have been proposed [13]. These methods, which target on better timing results, usually start with a rough estimation of the reciprocal = 1/divisor and follow a repetitive arithmetic method, where the number of repetitions depends on the precision difference of the reciprocal with the desirable division precision.

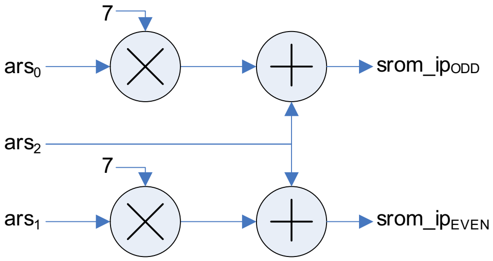

Figure 7. Consequent address mapper block.

A similar method is followed here, but the reciprocal precision is calculated in such a way that it is the minimum possible for achieving the desired precision at the divider output without the need of any repetitions. This way, the division is simplified to a multiplication operation between the reciprocal estimation and the dividend. At this point, it has to be mentioned that we treat the minimum reciprocal precision estimation based on all possible data values that occur from the weighted average defuzzification method:

$$op = \frac{\sum_{i=1}^{m} w^i y^i}{\sum_{i=1}^{m} w^i}$$

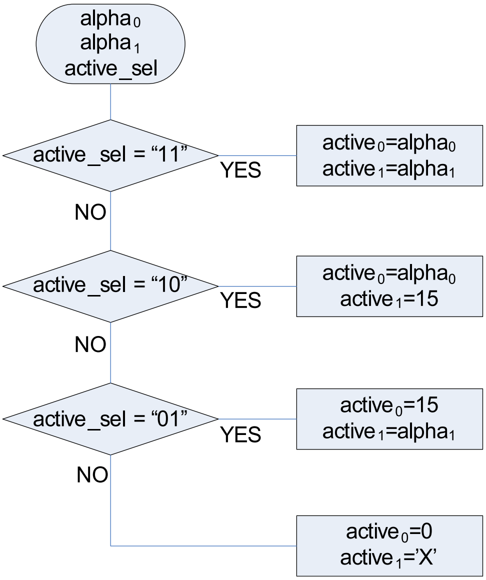

Figure 8. Rule selector flow chart.

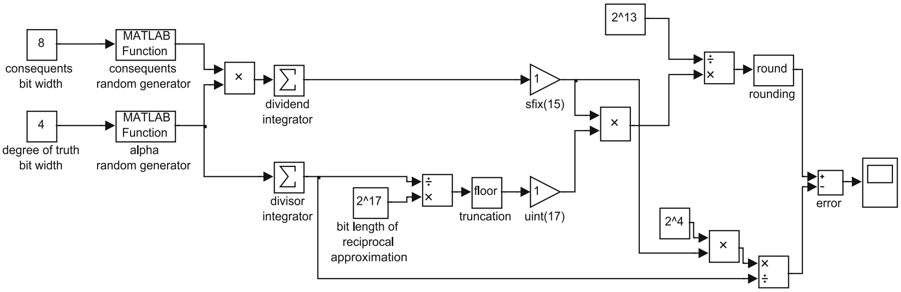

Figure 9. Reciprocal precision estimation.

The precision of the reciprocal was estimated at 17 bit using the Simulink model, shown in Figure 9.

The output error for the model considered above is shown in Figure 10.

Figure 9 clearly shows that the resulting error is no greater than 1, and the output is valid for the desired 12-bit output.

The "divide by zero" case has no practical value since it happens when all the theta values are zero, or when the antecedents of the rules have been disabled (by the rule selector) leading to no contributing consequences (from the sROM). The latter case consequently means that for the present input data set, there are no defined rules; thus, we set the output to zero (1/0 ≡ 0). Adding the case where 1/1 = 1 increases the reciprocal memory (reciprocal LUT block in Figure 2) word length from 17 to 18 bit. Since this is not very effective as it increases the

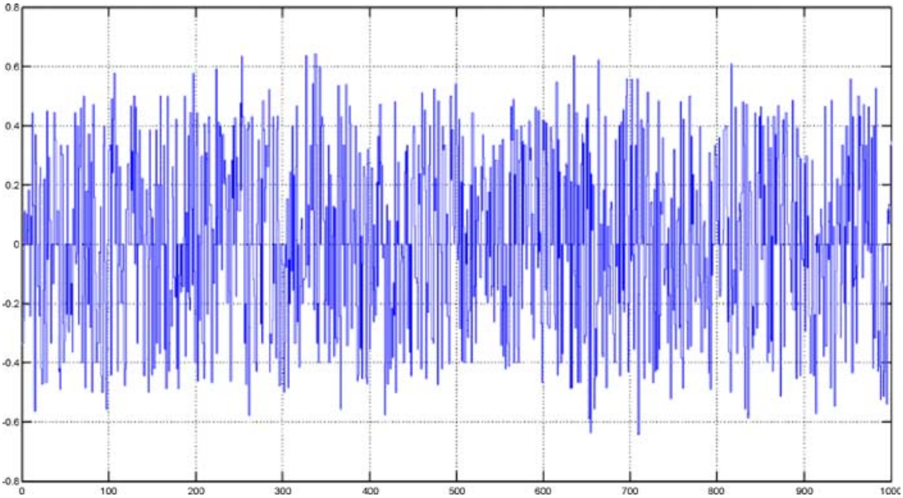

Figure 10. Output error.

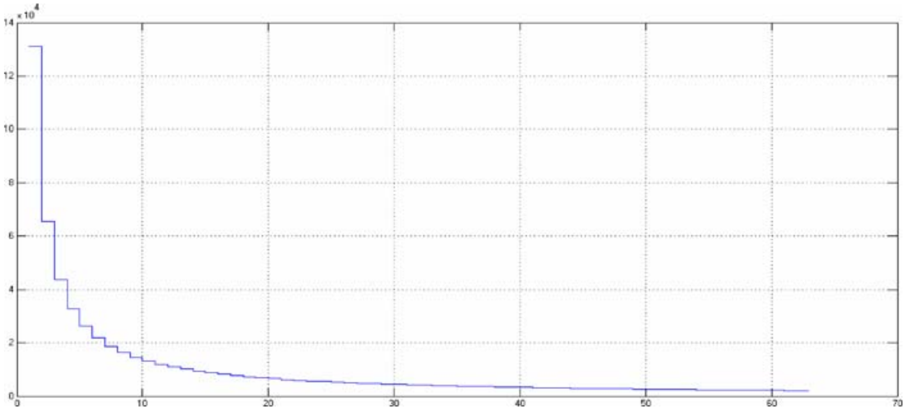

Figure 11. Reciprocal LUT contents.

reciprocal memory size and leads to bigger multiplication, we have chosen to add two extra circuits (is_one and fix output blocks in Figure 2) to detect this case.

The reciprocal memory contents are shown in the graph of Figure 11.

4. Design Flow for DFLC

This section presents the design flow (Figure 12) in a top-down manner, followed in our DFLC design. In a top-down design, one first concentrates on specifying and then on designing the circuit functionality [14]. The starting point of the design process is the system level modeling of the proposed DFLC. The last enabled us to evaluate the proposed model and to extract valuable test vector values to be used later for RTL and timing simulation. The very high-speed integrated-circuits hardware description language (VHDL) was used for the description of the circuit in register transfer level (RTL). Special attention was paid on the coding of the different blocks, since we aimed on writing a fully parameterized DFLC code. The DFLC can be parametric in terms of the number of available inputs and their resolution in bits, number of fuzzy sets per input and resolution of alpha value in bits and bit size of the output resolution, as well as the number of pipeline stages each block has. The DFLC presented here uses 2, 8, 7, 4 and 12, respectively, for the generic parameters described above. A VHDL package stores the above generic parameters together with the number of necessary pipeline stages for each block. An RTL simulation was performed to ensure the correct functionality of the circuit. Next, logic synthesis was done, where the tool first creates a generic (technology-independent) schematic on the basis of the VHDL code and then optimizes the circuit to the FPGA specific library chosen (Spartan-3 1500-4FG676). At this point, area and timing

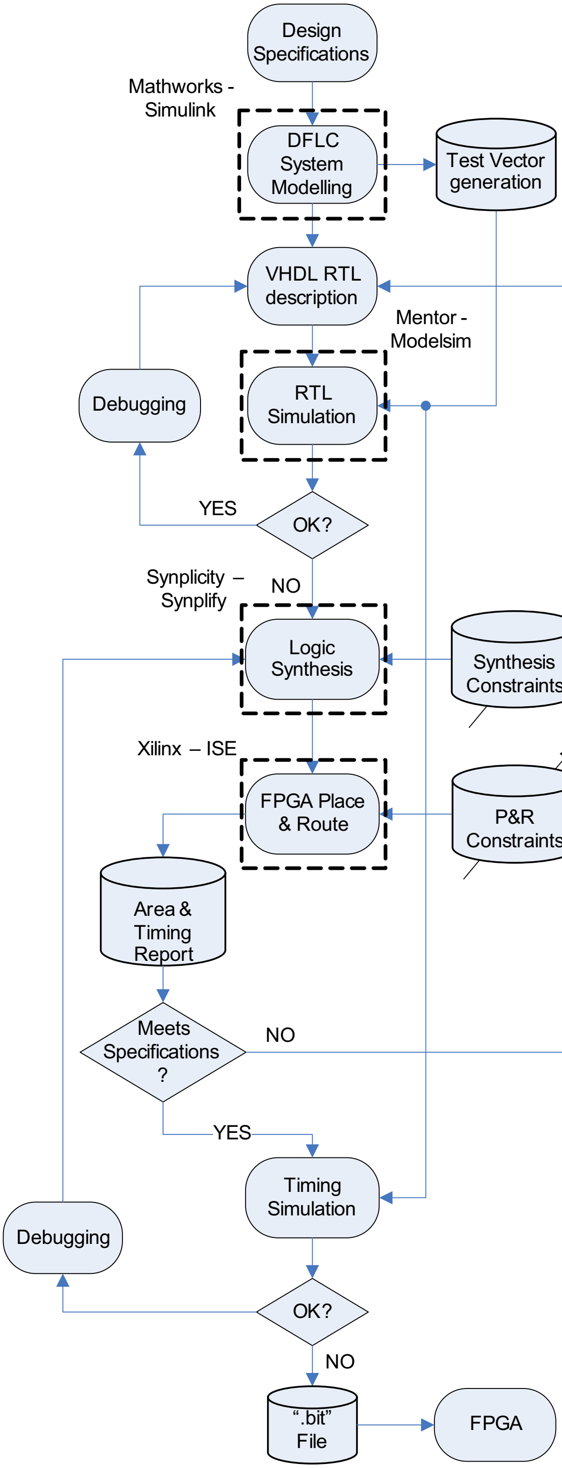

Figure 12. Design Flow Map.

Table IV. FPGA Utilization for DFLC Models.

Logic utilization	DFLC with arithmetic MF generator	DFLC with ROM MF generator
Slice flip flops	344	238
4 input LUTs	406	419
Logic distribution		
Occupied slices	294	368
4 input LUTs	419	560
Used as logic	406	419
Used as route-thru	13	13
Used as 16 × 1 ROMs		128
Bonded IOBs	30	30
MULT18X18s	6	3
GCLKs	2	2
DCMs	1	1

constraints and specific design requirements must be defined as they play an important role for the synthesis result.

Next, the Xilinx place and route (PAR) tool accepts the input netlist file (.edf), previously created by the synthesis tool and goes through the following steps. First, the translation program translates the input netlist together with the design constraints to a Xilinx database file. After the translation program has run successfully, the map program maps the logical design to the Xilinx FPGA device. Lastly, the PAR program accepts the mapped design, places and routes the FPGA and produces output for the bitstream (BitGen) generator. The latter program receives the placed and routed design and produces a bitstream (*.bit file) for Xilinx device configuration.

Before programming the FPGA file, a timing simulation was performed to ensure that the circuit meets the timing requirements set and it works correctly.

5. Results

Figure 13 shows the data flow for all the signals specified in Figure 2. A new input data set is clocked on the falling edge of the external clock, with half the frequency of the internal clock providing the necessary time for the address generator to generate the signals corresponding to all active rules (first to second pipe). Here, we remind that by using the odd–even method, we effectively identify and process two active rules per clock cycle instead of one [8].

The trapezoidal generator block requires four pipe stages to compute the α values of the membership functions. The computation occurs while the system addresses (third to fourth pipe) and reads (fifth to sixth pipe) the singletons ROM. A clock later (sixth to seventh pipe), after the rule selection and the minimum operator have taken place for the pair of the active rules, their θ values

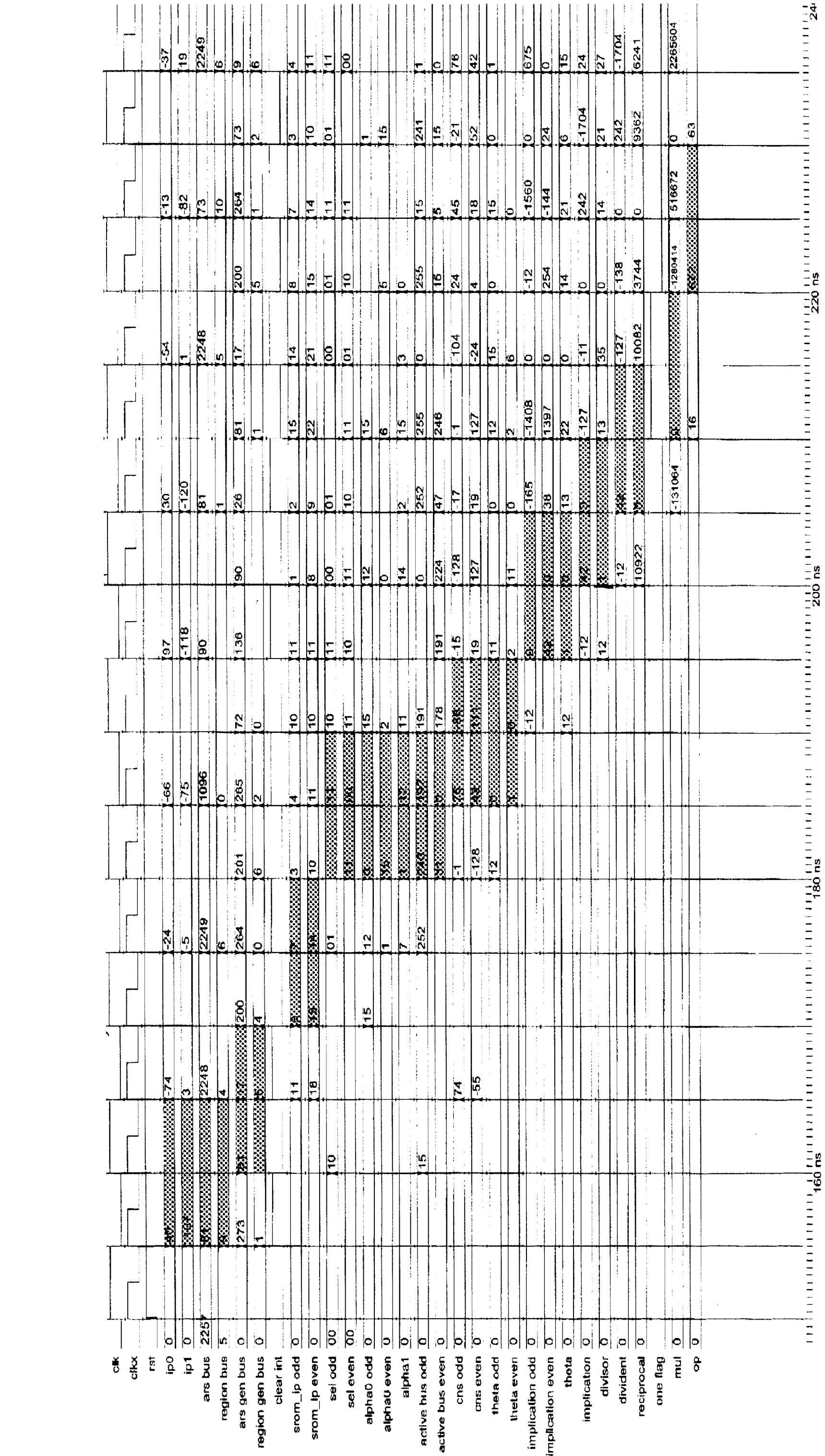

Figure 13. Pipeline structure data flow.

(MIN operation on the α values) along with their consequents and the signal for zeroing the integrators become available for the inference and defuzzification part. The rules implication result occurs two clocks later (eighth to ninth pipe) along with the addition of the θ values. In the next clock (ninth to tenth pipes), the sum of the implications is computed. During that time, the unsigned integrator outputs the divisor (tenth pipe) value, while the dividend is computed by the signed integrator one clock later and at the same time the reciprocal estimation becomes available. Their multiplication and fixing of the output occurs in the twelfth pipe stage. Finally, the output is clocked at the output of the chip in the rising edge of the external clock (thirteenth). The total data processing time starting from the time a new data set is clocked at the inputs until it produces valid results at the output requires a total latency of 13 pipe stages or 65 ns, with an internal clock frequency rate of 200 MHz or 5-ns period and external clock frequency of 100 MHz or 10-ns period, which effectively characterizes the input data processing rate (every 10 ns, new input data can be sampled in). Without the use of the odd–even architecture, we would have a latency of 12 pipe stages (no need for the adders in Figure 2), but the external clock frequency would be 50 MHz, or the input data processing rate would be 20 ns.

Along with the presented DFLC architecture, a modified model with ROM-based MF generator blocks instead of arithmetic-based has been implemented as well with respect to Table I. The latter DFLC model requires 11 pipe stages or 55-ns total data processing time and 10-ns input data processing rate but with increase in FPGA area utilization compared to the first model; moreover, it is obvious that since the MFs are ROM-based, any type and shape could be implemented.

The utilized resources on the FPGA device used for both DFLC models (arithmetic and ROM MF) are summarized in Table IV.

6. Conclusions

We have managed to design a digital fuzzy logic controller (DFLC) that reduces the clock cycles required to generate the active rule addresses to be processed, thus increasing the overall input data processing rate of the system. The latter reduction was possible by applying the proposed odd–even method presented in this paper. Two DFLC architectures were implemented: one with arithmetic and one with ROM-based membership function generators comparing their differences in timing and area utilization in the FPGA device. The first model achieves an internal clock frequency rate of 200 MHz with a total latency of 13 pipeline stages or 65 ns, and the second, for the same frequency rate, requires a latency of 11 or 55 ns The input data processing rate for both models is 10 ns or 100 MHz. Finally, the VHDL codes for DFLC models presented here are fully parameterized, allowing us to generate and test DFLC models with different specification scenarios.

References

1. Zadeh, L. A.: Fuzzy sets, *Inf. Control* **8** (1965), 338–353.
2. Takagi T. and Sugeno, M.: Derivation of fuzzy control rules from human operator's control actions, in: *Proc. of the IFAC Symp. on Fuzzy Information, Knowledge Representation and Decision Analysis*, July 1983, pp. 55–60.
3. Takagi, T. and Sugeno, M.: Fuzzy identification of systems and its application to modeling and control, *IEEE Trans. Syst. Man Cybern.* **20**(2) (1985), 116–132.
4. Togai, M. and Watanabe, H.: A VLSI implementation of a fuzzy inference engine: toward an expert system on a chip, *Inform. Sci.* **38** (1986), 147–163.
5. Watanabe, H., Dettloff, W. D., and Yount, K. E.: A VLSI fuzzy logic controller with reconfigurable, cascadable architecture, *IEEE J. Solid-State Circuits* **25**(2) (Apr. 1990), 376–382.
6. Shimuzu, K., Osumi, M., and Imae, F.: Digital Fuzzy Processor FP-5000, in: *Proc. 2nd Int. Conf. Fuzzy Logic Neural Networks*, Iizuka, Japan, July 1992, pp. 539–542.
7. Yamakawa, T. and Miki, T.: The current mode fuzzy logic integrated circuits fabricated by the standard CMOS process, *IEEE Trans. Comput.* **C-35** (Feb. 1986), 161–167.
8. Yamakawa, T.: High speed fuzzy controller hardware system: the mega-FLIPS machine in fuzzy computing, in: M. M. Gupta and T. Yamakawa (eds.), *Inf. Sci.* **45**(1) (1988), 113–128 Elsevier, Amsterdam, The Netherlands.
9. Gabrielli, A. and Gandolfi, E.: A fast digital fuzzy processor, *IEEE MICRO* **17** (1999), 68–79.
10. Jiménez, C. J., Barriga, A., and Sánchez-Solano, S.: Digital implementation of SISC fuzzy controllers, in: *Proc. Int. Conf on Fuzzy Logic, Neural Nets and Soft Computing*, Iizuka, 1994, pp. 651–652.
11. Patyra, M. J., Grantner, J. L., and Koster, K.: Digital fuzzy logic controller: design and implementation, *IEEE Trans. Fuzzy Syst.* **4**(4) (Nov. 1996), 439–459.
12. Eichfeld, H., Künemund, T., and Menke, M.: A 12b general-purpose fuzzy logic controller chip, *IEEE Trans. Fuzzy Syst.* **4**(4) (Nov. 1996), 460–475.
13. Wong, D. and Flynn, M.: Fast division using accurate quotient approximations to reduce the number of iterations, *IEEE Trans. Comput.* **41**(8) (Aug. 1992).
14. Sjoholm, S. and Lindh, L.: *VHDL for Designers*, 1st edn., Prentice Hall PTR, Jan. 10, 1991.
15. Xilinx, Spartan-3 FPGA Family: Complete Data Sheet, DS099, Dec. 24, 2003.