A Fast Digital Fuzzy Logic Controller: FPGA Design and Implementation

K. M. Deliparaschos, F. I. Nenedakis, S. G. Tzafestas
Intelligent Robotics and Automation Laboratory
Dept. of Electrical Eng. and Comp. Science
National Technical University of Athens
Zographou Campus, Athens, GR 157 73
kdelip@mail.ntua.gr

Abstract

*This paper describes an improved approach to design a Takagi-Sugeno zero-order type fast parameterized **digital fuzzy logic controller** (DFLC) 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 DFLC 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 (FPGA) chip with the use of a very high-speed integrated-circuits hardware-description-language (VHDL) and advanced synthesis and place and route tools.

1. Introduction

Typical fuzzy logic controller architectures found in the literature [2][3] (assuming overlap of two between adjacent fuzzy sets) consume 2^n clock cycles, since they process one active rule per clock cycle, ,where n is the number of inputs. Using the proposed Odd-Even method, we manage to process for the same model case scenario, two active rules per clock cycle or 2^{n-1}, thus increasing significantly the input data processing rate of the system. 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, while processing only the active rules. The presented DFLC is based on a simple algorithm similar to the Takagi-Sugeno zero-order inference and defuzzification method [1].

2. DFLC hardware implementation

The parameters characterizing the presented DLFC are summarized in Table 1.

Table 1. DLFC characteristics.

Fuzzy Inference System	Takagi-Sugeno zero order
Inputs	2
Input resolution	8-bit
Outputs	1
Output resolution	12-bit
Antecedent MF's	7 Trapezoidal shaped per fuzzy set
Antecedent MF degree of truth (α value) resolution	4-bit
Consequent MF's	49 Singleton type
Consequent MF resolution	8-bit
Aggregation method	MIN
Implication method	PROD (product operator)
MF overlapping degree	2
Defuzzification method	Weighted average

The architecture (Fig. 1) is mainly broken in two major blocks: "Fuzzification & Aggregation" and "Inference & Defuzzification". The dotted lines on the Fig. 1 indicate the number of pipeline stages for each block. Signal bus sizes are noted as well.

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 corresponds to. As a result, for the 2-input DFLC with 7 membership functions per input instead of processing all 49 rule combinations in the rule base, only 4 active rules remain that need to be taken into account.

Fig. 2, 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 ($ip_{0,1}$) with the rising start points ($rise\_start_{0,1}$) of the fuzzy sets. It is also shown for the selected pair (even-odd), whether $ip_{0,1}$ lies in the rising (value 0) or falling

0-7803-9402-X/05/$20.00 © 2005 IEEE

part (value 1) of each FS. The algorithm used in the ARS block is shown in Fig. 3.

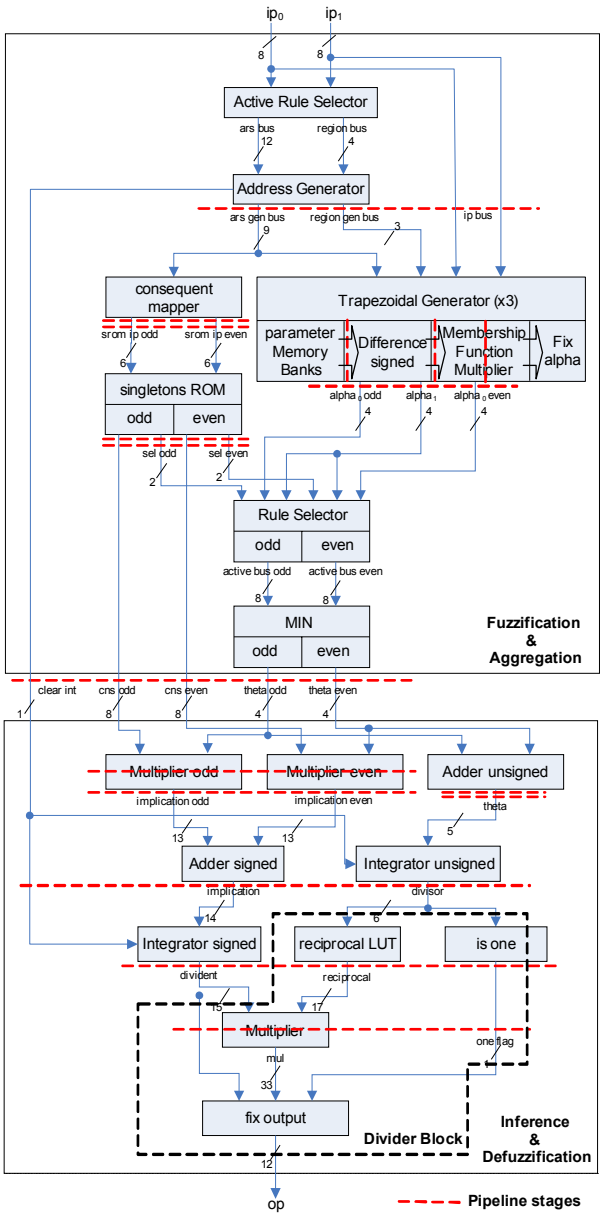

Figure 1. DLFC architecture.

The address generator (ADG) block (Fig. 4) outputs the indicated by the ARS block addresses, needed for the active fuzzy rules ($ars\_even_{0,1}$, $ars\_odd_{0,1}$, $reg\_even_{0,1}$, $reg\_odd_{0,1}$). Normally, for 4 active rule addresses, the ADG would require 4 clock periods to generate the active rule addresses [2], consequently increasing the data input sampling rate period to 4 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.

The flow chart for the trapezoidal membership function generator (TMFG) is shown in Fig. 5. Three TMFG block instances required in the architecture and accept as inputs the controller input (ip0, ip0, ip1), region (reg_0, reg_1, reg_2), start point and slope from the corresponding Parameter Memory Bank ROM (addressed by $ars_0\&reg_0$, $ars_1\&reg_1$, $ars_2\&reg_2$ concatenated signals), for the 1st, 2nd and 3rd TMFG respectively. The output named alpha represents the degree of truth of the current fuzzy set. We treat rise and fall sections of the trapezoidal shape in the same way but distinguished with a logical NOT for the fall section.

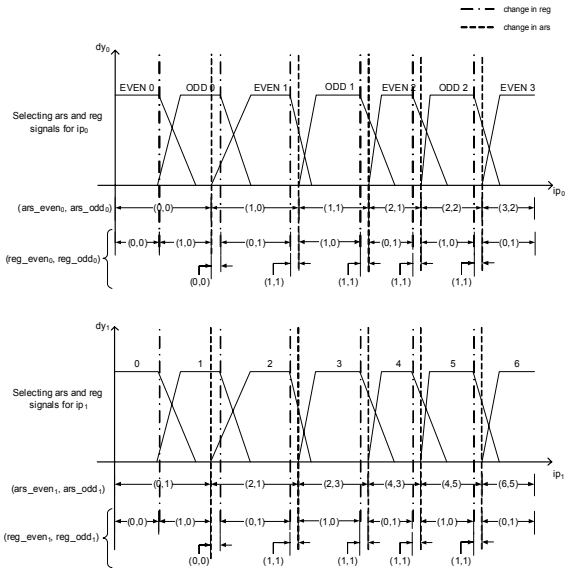

Figure 2. FS coding and Odd-Even regions.

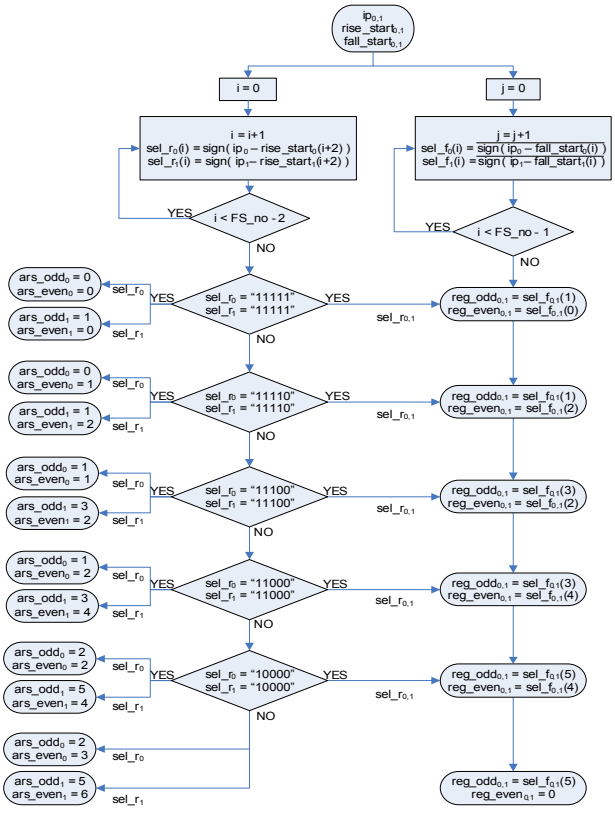

Figure 3. ARS block algorithm.

The TMFG parameter memory banks are organized as follows in Table 2.

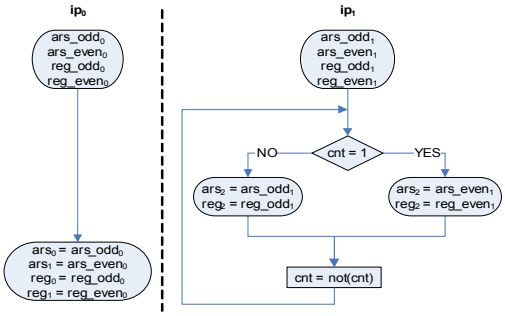

Figure 4. Address generator algorithm.

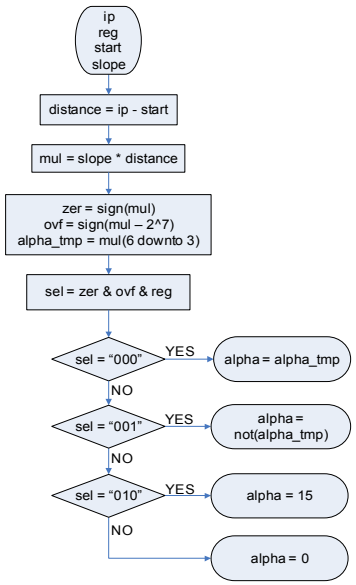

Figure 5. TMFG block flow chart.

Table 2. Parameter memory banks.

ip0 ODD		ip0 EVEN		ip1	
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

The consequent address mapper block (CAM) is simply used to address the singletons ROM by generating two addressing signals, srom_ip odd and srom_ip even.

The purpose of the Rule selection block (Fig. 6) 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. 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:

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

Similarly the Even Rule selector accepts as inputs:

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

Table 3. Singletons Rom.

ODD		EVEN	
$cns_{ODD0\&0}$	$active\_sel_{ODD0\&0}$	$cns_{EVEN0\&0}$	$active\_sel_{EVEN0\&0}$
$cns_{ODD0\&1}$	$active\_sel_{ODD0\&1}$	$cns_{EVEN0\&1}$	$active\_sel_{EVEN0\&1}$
$cns_{ODD0\&2}$	$active\_sel_{ODD0\&2}$	$cns_{EVEN0\&2}$	$active\_sel_{EVEN0\&2}$
$cns_{ODD0\&3}$	$active\_sel_{ODD0\&3}$	$cns_{EVEN0\&3}$	$active\_sel_{EVEN0\&3}$
$cns_{ODD0\&4}$	$active\_sel_{ODD0\&4}$	$cns_{EVEN0\&4}$	$active\_sel_{EVEN0\&4}$
$cns_{ODD0\&5}$	$active\_sel_{ODD0\&5}$	$cns_{EVEN0\&5}$	$active\_sel_{EVEN0\&5}$
$cns_{ODD0\&6}$	$active\_sel_{ODD0\&6}$	$cns_{EVEN0\&6}$	$active\_sel_{EVEN0\&6}$
$cns_{ODD1\&0}$	$active\_sel_{ODD1\&0}$	$cns_{EVEN1\&0}$	$active\_sel_{EVEN1\&0}$
$cns_{ODD1\&1}$	$active\_sel_{ODD1\&1}$	$cns_{EVEN1\&1}$	$active\_sel_{EVEN1\&1}$
$cns_{ODD1\&2}$	$active\_sel_{ODD1\&2}$	$cns_{EVEN1\&2}$	$active\_sel_{EVEN1\&2}$
$cns_{ODD1\&3}$	$active\_sel_{ODD1\&3}$	$cns_{EVEN1\&3}$	$active\_sel_{EVEN1\&3}$
$cns_{ODD1\&4}$	$active\_sel_{ODD1\&4}$	$cns_{EVEN1\&4}$	$active\_sel_{EVEN1\&4}$
$cns_{ODD1\&5}$	$active\_sel_{ODD1\&5}$	$cns_{EVEN1\&5}$	$active\_sel_{EVEN1\&5}$
$cns_{ODD1\&6}$	$active\_sel_{ODD1\&6}$	$cns_{EVEN1\&6}$	$active\_sel_{EVEN1\&6}$
$cns_{ODD2\&0}$	$active\_sel_{ODD2\&0}$	$cns_{EVEN2\&0}$	$active\_sel_{EVEN2\&0}$
$cns_{ODD2\&1}$	$active\_sel_{ODD2\&1}$	$cns_{EVEN2\&1}$	$active\_sel_{EVEN2\&1}$
$cns_{ODD2\&2}$	$active\_sel_{ODD2\&2}$	$cns_{EVEN2\&2}$	$active\_sel_{EVEN2\&2}$
$cns_{ODD2\&3}$	$active\_sel_{ODD2\&3}$	$cns_{EVEN2\&3}$	$active\_sel_{EVEN2\&3}$
$cns_{ODD2\&4}$	$active\_sel_{ODD2\&4}$	$cns_{EVEN2\&4}$	$active\_sel_{EVEN2\&4}$
$cns_{ODD2\&5}$	$active\_sel_{ODD2\&5}$	$cns_{EVEN2\&5}$	$active\_sel_{EVEN2\&5}$
$cns_{ODD2\&6}$	$active\_sel_{ODD2\&6}$	$cns_{EVEN2\&6}$	$active\_sel_{EVEN2\&6}$
		$cns_{EVEN3\&0}$	$active\_sel_{EVEN3\&0}$
		$cns_{EVEN3\&1}$	$active\_sel_{EVEN3\&1}$
		$cns_{EVEN3\&2}$	$active\_sel_{EVEN3\&2}$
		$cns_{EVEN3\&3}$	$active\_sel_{EVEN3\&3}$
		$cns_{EVEN3\&4}$	$active\_sel_{EVEN3\&4}$
		$cns_{EVEN3\&5}$	$active\_sel_{EVEN3\&5}$
		$cns_{EVEN3\&6}$	$active\_sel_{EVEN3\&6}$

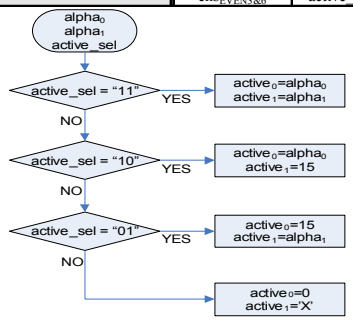

Figure 6. Rule selector flow chart.

The Spartan 3 family of FPGA [5] devices used to implement the proposed architecture provides embedded multipliers, an asynchronous version called MULT18X18 and a version with a register at the outputs called MULT18X18S. 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.

Division remains one of most hardware consuming operations. Several methods have been proposed [4]. 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. 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. The precision of the reciprocal was estimated at 17-bit with the use of a

Simulink model. 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). Thus we set the output to zero (1/0≡0). We have chosen to add two extra circuit (is_one and fix output blocks in Fig. 1) to detect the (1/1≡1) case.

3. Results

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. The Trapezoidal Generator block requires four pipe stages to compute the values of the membership functions. The computation occurs while the system addresses and reads the Singletons ROM. A clock later, after the Rule selection and the Minimum operator have taken place for the pair of the active rules, their θ values (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 along with the addition of the θ values. In the next clock, the sum of the implications is computed. During that time the unsigned integrator outputs the divisor 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 12^{th} pipe stage. Finally, the output is clocked at the output of the chip in the rising edge of the external clock (13^{th}). 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 (Spartan-3 1500-4fg676 features a 326 MHz system clock rate [5]) 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 Fig. 1) 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 1. 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 MF's are ROM based any type and shape could be implemented.

The design summary of the FPGA device used (Spartan-3 1500-4fg676) for both DFLC models (Arithmetic and ROM MF) are summarized in Table 4.

Table 4. FPGA Design Summary for DFLC Models.

Logic Utilization	DFLC Arithmetic MF Generator	DFLC 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 16x1 ROMs		**128**
bonded IOBs	**30**	**30**
MULT18X18s	**6**	**3**
GCLKs	**2**	**2**
DCMs	**1**	**1**
Internal clock		
period	**5ns**	**5ns**

4. 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 MF Generators. 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] T. Takagi & M. Sugeno, Derivation of Fuzzy Control Rules from Human Operator's Control Actions, Proc. of the IFAC Symp. on Fuzzy Information, Knowledge Representation and Decision analysis, pp.55-60, July 1983

[2] A. Gabrielli, E. Gandolfi, "A Fast Digital Fuzzy Processor", *IEEE Micro,* Vol.17, pp. 68-79, 1999.

[3] C. J. Jiménez, A. Barriga. S. Sánchez-Solano, "Digital Implementation of SISC Fuzzy Controllers", *Proc. Int. Conf. on Fuzzy Logic, Neural Nets and Soft Computing*, pp. 651-652, Iizuka, 1994.

[4] Derek Wong, Michael Flynn, "Fast Division Using Accurate Quotient Approximations to Reduce the Number of Iterations", IEEE Trans. Comp., vol. 41, no. 8, Aug. 1992.

[5] Xilinx, Spartan-3 FPGA Family: Complete Data Sheet, DS099 Dec. 24, 2003