Abstract: Fuzzy Logic (FL) was developed by Zadeh to deal with the uncertainty involved in decision making and system modeling and control of real–life systems, and is an extension of the two–valued logic defined by the binary pair {false, true} or {0,1} to the entire continuous interval [0,1] of logic values between false=0 and true=1. The purpose of this paper is to design and implement a zero-order Takagi-Sugeno (T-S) parameterized digital fuzzy logic processor (DFLP), in which only the active rules (i.e. rules that give a non–null contribution for a given data set) are considered, at high speed of operation, without significant increase in hardware complexity. The DFLP discussed in this paper achieves an internal core processing speed of at least 100 MHz, and based on the chosen parameters is featuring four 12-bit inputs and one 12-bit output, with seven trapezoidal shape membership functions per input and a rule base of up to 2401 rules. The proposed architecture was implemented in a Field Programmable Gate Array (FPGA) chip with the use of a very high-speed integrated-circuit hardware-description-language (VHDL) and advanced synthesis and place and route tools.

Key-Words: - Fuzzy Logic processor (FLP), digital FLP (DFLP), Takagi-Sugeno controller, field programmable gate array (FPGA) chip, very high-speed integrated-circuits description language (VHDL).

1. INTRODUCTION

Fuzzy chips are distinguished into two classes 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 presents the design of a parameterized digital fuzzy logic processor (DFLP). By the term parameterized we mean that the DFLP facilitates scaling and can be configured for different number of inputs and outputs, number of triangular or trapezoidal fuzzy sets per input, number of singletons per output, antecedent method (t-norm¹, s-norm²), divider type, number of pipeline registers for the various components in the model. This parameterization enables the creation of a generic Fuzzy Controller (FC) Core than can be used to produce fuzzy processors of different specifications without the need of redesigning the FC Core from the beginning. The fuzzy logic processor architecture assumes overlap of two fuzzy sets between adjacent fuzzy sets and requires 2n clock cycles (input data processing rate), where n is the number of inputs, since it processes one active rule per clock cycle. The architecture of the design allows one to achieve a core frequency speed of 100 MHz, while the input data can be sampled at a clock rate equal to 1/2n of the core frequency speed (100 MHz), while processing only the active rules. To achieve this timing result the latency of the chip architecture involves 11 pipeline stages each one requiring 10 ns. The proposed DFLP is based on a simple algorithm similar to the Takagi-Sugeno of zero-order type inference and weighted average defuzzification method, and based on the chosen parameters employs four 12-bit inputs and one 12-bit output, with up to 7 trapezoidal or triangular shape membership functions per input with a degree of truth resolution of 8 bits and a rule base of up to 2401 rules.

2. FUZZY LOGIC

2.1 Theory of fuzzy sets

Fuzzy Set Theory was developed by Lotfi Zadeh in 1965 [1], where he introduced the fuzzy set concept. Zadeh, has extended 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 of discourse 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

¹ t-norm: T_{min}(a,b)=\min{a,b}, T_{prod}(a,b)=a \cdot b
² s-norm: \_\_max(a,b)=\max{a,b}, \_\_sum(a,b)=a+b-a \cdot b
element \((x)\) belongs or not to a set \((A)\) is gradual rather than crisp (or binary). In other words, fuzzy logic replaces “true or 1” and “false or 0” with a continuous set of 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:

\[
A = \{(x, \mu_A(x)) \mid x \in U\}
\]

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. The Takagi-Sugeno Method

The DFLP proposed in this paper is based on the T-S zero-order type fuzzy model [2][3]. It is well known that the 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 actually quite simple as a method, leads to fast calculations and is relatively easy to apply. Moreover, a fuzzy processor 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. T-S MIMO rules have the following form:

Rule \(R_i\): IF \(x_1\) IS \(A_1^i\) AND \(x_2\) IS \(A_2^i\) AND \(\ldots\) AND \(x_n\) IS \(A_n^i\) THEN \(y_1^i\) AND \(y_2^i\) AND \(\ldots\) AND \(y_l^i\) where \(y_j^i = c_{i0}^j + c_{ij}^k x_k + \ldots + c_{in}^j x_n\)

\(i=1,2,\ldots,m\) where \(m\) is the total number of rules, \(x_k\), \(k=1,2,\ldots,n\) represents the \(k^{th}\) input variable, \(A_k\) is the input fuzzy sets, \(y_{pj}^i, j=1,2,\ldots,l\) represents the \(j^{th}\) output variable of the \(i^{th}\) rule, and \(c_{ij}^k\) are all constants. The above rule is linguistically (logically) equivalent to a number of MISO rules, such as:

Rule \(R_j\): IF \(x_1\) IS \(A_1^j\) AND \(x_2\) IS \(A_2^j\) AND \(\ldots\) AND \(x_n\) IS \(A_n^j\) THEN \(y_1^j\)

Rule \(R_j\): IF \(x_1\) IS \(A_1^j\) AND \(x_2\) IS \(A_2^j\) AND \(\ldots\) AND \(x_n\) IS \(A_n^j\) THEN \(y_l^j\)

The logical “AND” in the consequent of the MIMO rule still exists in the MISO rules case since in the two fuzzy inferences the one set of rules is true “AND” another is true [20]. A zero-order model arises (simplified T-S functional reasoning), if now we only use the constant \(c_{ij}^k\) at the output and therefore \(y_j^i = c_{ij}^k\) (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 defuzzified output is the weighted average of the contribution of each rule, as shown in the following equation:

\[
y_j = \frac{\sum_{i=1}^{m} w_i y_j^i}{\sum_{i=1}^{m} w_i}
\]

where \(w_j\) is the weight contribution of the left part of the \(i^{th}\) rule and for AND method connection is given by

\[
w_j = \prod_{j=1}^{k} \mu_{A_j}(x_j) \text{ or } w_j = \mu_{A_j}(x_j)
\]

The T-S mechanism is illustrated in Figure 1:

![Figure 1: T-S mechanism.](image)

### 2.3. DFLP Characteristics

In this section, the characteristics of our DFLP are presented based on the chosen (generic) parameters (defined on a VHDL package file) for the parameterized FC CORE. These are summarized in Table 1 and Table 2 respectively.

<table>
<thead>
<tr>
<th>Parameters (VHDL generics)</th>
<th>Value</th>
<th>Generic Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ip_no</td>
<td>4</td>
<td>number of inputs</td>
</tr>
<tr>
<td>ip_sz</td>
<td>12</td>
<td>input bus width (bits)</td>
</tr>
<tr>
<td>op_no</td>
<td>1</td>
<td>number of outputs</td>
</tr>
<tr>
<td>op_sz</td>
<td>8</td>
<td>output bus width (bits)</td>
</tr>
<tr>
<td>FS_no</td>
<td>7</td>
<td>number of membership functions (same for all inputs)</td>
</tr>
<tr>
<td>dy</td>
<td>8</td>
<td>dy degree of truth width</td>
</tr>
<tr>
<td>sel_op</td>
<td>0</td>
<td>divider model type 0: restoring array, 1 : LUT reciprocal approximation [19]</td>
</tr>
<tr>
<td>div_type</td>
<td>0</td>
<td>divider model type 0: restoring array, 1 : LUT reciprocal approximation [19]</td>
</tr>
<tr>
<td>Path Synchronization Registers</td>
<td>Signal Path Route</td>
<td></td>
</tr>
<tr>
<td>par1_no</td>
<td>1</td>
<td>ip set -&gt; par1 no -&gt; trap gen p</td>
</tr>
<tr>
<td>par2_no</td>
<td>4</td>
<td>s rom -&gt;par2 no -&gt; mult</td>
</tr>
<tr>
<td>par3_no</td>
<td>2</td>
<td>s rom -&gt;par3 no -&gt; rul sel p</td>
</tr>
<tr>
<td>par3 no</td>
<td>0</td>
<td>par5 -&gt;par5 -&gt; int uns</td>
</tr>
<tr>
<td>Component Pipeline Registers</td>
<td>Component (Entity) Name</td>
<td></td>
</tr>
<tr>
<td>cpr1_no</td>
<td>1</td>
<td>addr gen p</td>
</tr>
<tr>
<td>cpr2_no</td>
<td>1</td>
<td>cons map p</td>
</tr>
<tr>
<td>cpr3_no</td>
<td>3</td>
<td>trap gen p</td>
</tr>
<tr>
<td>cpr4_no</td>
<td>0</td>
<td>rule sel p</td>
</tr>
<tr>
<td>cpr5_no</td>
<td>2</td>
<td>minmax p</td>
</tr>
<tr>
<td>cpr6_no</td>
<td>2</td>
<td>mult</td>
</tr>
<tr>
<td>cpr7_no</td>
<td>2</td>
<td>int uns</td>
</tr>
<tr>
<td>cpr8_no</td>
<td>0</td>
<td>int sig</td>
</tr>
<tr>
<td>cpr9_no</td>
<td>3</td>
<td>div array</td>
</tr>
</tbody>
</table>
The necessary parameters are assigned on the generics on the entity definition of every parameterized VHDL component in the design hierarchy. On the same VHDL package file, the number of pipeline stages for every functional block in the design as well as path synchronization registers is defined.

Table 1: DFLP Soft Core chosen parameters

<table>
<thead>
<tr>
<th>Fuzzy Inference System (FIS) type</th>
<th>Takagi-Sugeno zero-order type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inputs</td>
<td>4</td>
</tr>
<tr>
<td>Input resolution</td>
<td>12 bit</td>
</tr>
<tr>
<td>Outputs</td>
<td>1</td>
</tr>
<tr>
<td>Output resolution</td>
<td>12 bit</td>
</tr>
<tr>
<td>Antecedent Membership Functions (MF’s)</td>
<td>7 Triangular or Trapezoidal shaped per fuzzy set</td>
</tr>
<tr>
<td>Antecedent MF degree of truth (α value) resolution</td>
<td>8 bit</td>
</tr>
<tr>
<td>Consequent MF’s</td>
<td>2401 Singleton type</td>
</tr>
<tr>
<td>Consequent MF resolution</td>
<td>8 bit</td>
</tr>
<tr>
<td>Maximum number of fuzzy inference rules</td>
<td>2401 (number of fuzzy sets * number of inputs)</td>
</tr>
<tr>
<td>AND method</td>
<td>MIN (T-norm operator implemented by minimum)</td>
</tr>
<tr>
<td>Implication method</td>
<td>PROD (product operator)</td>
</tr>
<tr>
<td>MF overlapping degree</td>
<td>2</td>
</tr>
<tr>
<td>Defuzzification method</td>
<td>Weighted average</td>
</tr>
</tbody>
</table>

Table 2: DFLP characteristics

### 3. DFLP HARDWARE IMPLEMENTATION

This section provides an analytical description of the various hierarchical blocks of the DFLP architecture, and explains how these blocks are designed in a parameterized fashion to result in a fully parametric processor core.

#### 3.1 DFLP Architecture

The present DFLP architecture is shown in Figure 2. The architecture is mainly broken in three major hierarchical blocks: “Fuzzification”, “Inference” and “Defuzzification.” The “CPR” blocks on the figure indicate the number of pipeline stages for each component (component pipeline registers), the “PSR” blocks indicate the path synchronization registers, while the “U” blocks represent the different components of the DFLP. The component identified as DCM is the Digital Clock Manager and is one of four components available in the specific FPGA library chosen (Spartan-3 1500-4FG676) [15]. The latter accepts the FPGA board (Memec 3SMB1500) clock (clk) of 75 MHz and is configured to generate a divided clock signal (clkdv) of 75 MHz/12=6.25 MHz and a multiplied clock signal (clkfx) which is 6.25·16=100 MHz for the DFLP Core operation. It should be mentioned here that the above frequency values can vary by changing the DCM generic parameters. A control logic component (control_logic_p) is implemented on the top structural entity (binds FC Core, DCM, R1, R2 and control logic) of the design that provides two signals named “ready_in” and “ready_out”, which are used to provide input and output data-ready signals for handshaking with external devices.
Generally, FS_start_addr0,…,4 denotes the active FS pair block. The generate statement available in VHDL and use of parameterization is mainly achieved by making use of parameterization is illustrated. The provided, showing the VHDL code style followed to achieve parameterization.

The initial FS pair is always the 1st and 2nd and we only remain that need to be taken into account. Fig. 3, the 3rd FS is exceeded. The FS section transition is shift right to the next FS pair when the rise_start (rs) of the 4-input DFLP with 7 membership functions per input, instead of processing all 2401 rule combinations in the rule base, only 16 active rules remain that need to be taken into account. Fig. 3, illustrates the fuzzy set (FS) coding used for the inputs (ip0,…,4), as well as the FS area split up that occurs by repeatedly comparing the four input data values (ip0,…,4) with the rising start points (rise_start0,…,4) of the fuzzy sets. Since we assume overlapping of two fuzzy sets between adjacent fuzzy sets, it is obvious that the comparison of the 1st and 2nd FS is not necessary, since the initial FS pair is always the 1st and 2nd and we only shift right to the next FS pair when the rise_start (rs) of the 3rd FS is exceeded. The FS section transition is depicted in Figure 3 as line type named ‘change in ars’. Generally, FS_start_addr0,…,4 denotes the active FS pair and its output range is from 0 to 5 in this case.

An analytical block diagram for the parameterized ars_p block is shown in Figure 4. In the same figure a snapshot of the architecture of the ars_p block is provided, showing the VHDL code style followed to achieve parameterization is illustrated. The parameterization is mainly achieved by making use of the generate statement available in VHDL and use of generics in the entity to define the parameters of the block.

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, i.e. those rules that give a non null contribution to the final result. Dealing with the above problem and given the fact that the overlapping between adjacent fuzzy sets is of order 2, we have implemented an active rule selection block (ars_p) to calculate the fuzzy set region in which the input data corresponds to. As a result, for the 4-input DFLP with 7 membership functions per input, instead of processing all 2401 rule combinations in the rule base, only 16 active rules remain that need to be taken into account. Fig. 3, illustrates the fuzzy set (FS) coding used for the inputs (ip0,…,4), as well as the FS area split up that occurs by repeatedly comparing the four input data values (ip0,…,4) with the rising start points (rise_start0,…,4) of the fuzzy sets. Since we assume overlapping of two fuzzy sets between adjacent fuzzy sets, it is obvious that the comparison of the 1st and 2nd FS is not necessary, since the initial FS pair is always the 1st and 2nd and we only shift right to the next FS pair when the rise_start (rs) of the 3rd FS is exceeded. The FS section transition is shift right to the next FS pair when the rise_start (rs) of the 3rd FS is exceeded. The FS section transition is depicted in Figure 3 as line type named ‘change in ars’. Generally, FS_start_addr0,…,4 denotes the active FS pair and its output range is from 0 to 5 in this case.

The address generator (addr_gen_p) block outputs the indicated by the ars_p block addresses, needed for the active fuzzy rules (gen_addr_p) since the ars_p block has already identified previously all the involved fuzzy sets at once (fully combinatorial), the addr_gen_p produces clock by clock period the addresses which correspond to the active fuzzy rules. Normally, for 16 active rule addresses as in our case study, the addr_gen_p would require 16 clock periods to generate the active rule addresses [9], consequently increasing the data input sampling rate period to 16 times the internal clock period. Shown in Figure 5, signal int_zer is used as a zeroing flag for the integrators, when all 16 active rules have been processed to avoid integration over floating. The latter signal depends on the block generic int_zer_delay=cp1_no+cp2_no+ps2_no+cp6_no-1 or with the chosen parameters in Table 1 is 7. The -1 in the last equation is due to int_zer signal operation on integrators on the negative edge of the pulse (otherwise operated on the positive edge, int_zer_delay=8). Signal cnt is an ip_no size incremental counter and index iε[0, ip_no-1].
3.4. Trapezoidal membership function generator

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

![Figure 6: trap_gen_p architecture and RTL synthesis result.](image)

Four trap_gen_p block instances are required in the architecture that accept as inputs the processor inputs (ip0,…,3), the start point and the slope from the corresponding Parameter Memory Bank ROM (addressed by ars0&reg0, ars1&reg1, ars2&reg2 concatenated signals), for the 1st, 2nd and 3rd trap_gen_p respectively. The output named “alpha” represents the degree of truth of the current fuzzy set. Flag, zer depicts the special case where ip0,1 < rise_start0,1, 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 to the universe of discourse axis.

3.5. Memory organization

The ars_p parameters ROM array structure is shown in Figure 7.

![Figure 7: ars_p parameters memory array organization.](image)

The trap_gen_p parameter memory banks are organized as shown in Figure 8.

![Figure 8: trap_gen_p parameters memory organization.](image)

The Singletons ROM (s_rom_p) organization is shown in Figure 9.

![Figure 9: s_rom_p memory organization](image)

3.6. Consequent address mapper

The consequent address mapper block (cons_map_p) shown in Figure 10 is used in order to generate the address (addr_out) for the singletons ROM by collecting the generated addresses (addr_in) from the addr_gen_p component. The input bus signal addr in is multiplied by the number of fuzzy sets of each input, which in our case are 7.
Figure 10: Consequent address mapper block.

3.6. Rule selection block

The purpose of this block is to allow for the selection of the desired rules stored in the rule base, i.e. the rules that will contribute to the final result for an input data set. The rule combinations are stored as the 4 MSBs in the singletons ROM (s_rom_p block) concatenated with the singleton value. The selection is performed in the following manner. The antecedents of each rule are stored in ROM either as 0 or 1, thus enabling or disabling part or the entire rule. The Rule selector block accepts as inputs the alpha values (alpha_val) of the corresponding trapezoidal generators and the selection signal (active_sel) from the singletons ROM. The parameterized VHDL code (entity and architecture) for the rule selection block together with the synthesis result for 4 inputs is shown in Figure 11.

Figure 11: Rule selector VHDL code and synthesis result (RTL view).

More specifically, in reference to Figure 11, 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 \( d=8 \) bit degree of truth resolution (unsigned number since alpha value, \( \alpha \in \mathbb{R} \)) is \( 2^{51}-1=255 \). 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.

3.7. Integrator block

This block (int_sig) simply implements a signed number discrete integrator block (Figure 12). An almost identical architecture with minor changes, not worth mentioning here, is implemented for unsigned numbers (int_uns). Summation overflowing is avoided by clear input signal which is controller from addr_gen_p block described earlier in section 3.

Figure 12: Integrator block and RTL synthesis result

3.8. Multiplier blocks.

The Spartan 3 family of 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. 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 since it leads to better timing results.

3.10. AND/OR method block

This block simply implements different intersection/union (AND/OR) methods for the antecedent connection. It is fully parameterized with respect to the number and resolution of input antecedents and method selection. The block implements the following norms (sel_op = 00 for \( \min \), 01 for \( \text{prod} \), 10 for \( \max \), 11 for \( \text{probor} \) (probabilistic OR or algebraic sum).

3.11. Divider block

Division remains one of the 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. 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:

$$op = \sum w_i y_j \sum w_i$$

The precision of the reciprocal was estimated at 22 bit using the Simulink model shown in Figure 13:

![Figure 13: Reciprocal precision estimation and reciprocal ROM generation Simulink models.](image)

The output error for the model considered above is shown in Figure 14:

![Figure 14: Output error.](image)

Figure 14 clearly shows that the resulting error is not greater than 0.8 and the output is valid for the desired 12 bit output. This error is practically rounded since the output is scaled down to 12 bits. 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 s_rom_p). The latter case consequently means that for the present input data set, there are no defined rules. Thus we set the output to minus one ($1/0=-1$). Adding the case where divd/1=divd, increases the reciprocal memory (reciprocal LUT block in Figure 2) word length from 22 to 23 bit. Since this is not very effective as it increases the reciprocal memory size and leads to bigger multiplication, we have chosen to add two extra circuits to detect these cases. The reciprocal memory contents are shown on the graph of Figure 15:

![Figure 15: Reciprocal LUT contents.](image)

The divider block (div_ppa) synthesis result is shown in Figure 16, where the component named recip_lut_p is the ROM that holds the reciprocal LUT contents (Figure 15).

![Figure 16: Divider block RTL synthesis result view](image)

4. DESIGN FLOW FOR DFLP

This section presents the design flow (Figure 17) in a top-down manner, followed in our DFLP design. In a top-down design, one first deals with the specification and then with the design of the circuit functionality. The starting point of the design process is the system level modeling (Simulink model) of the proposed DFLP. The Simulink model enabled us to evaluate the proposed DFLP, and to extract valuable test vector values to be used later for RTL and timing (VITAL) simulation. The VHDL language was used for the description of the circuit in RTL (Register Transfer Level). Special attention was paid on the coding of the different components, since we aimed on writing a fully parameterized code. The DFLP 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, number of outputs and bit size of the output resolution, antecedent connection method, divider type, as well as in terms of the number of pipeline stages each block has. The DFLP presented here uses the generic parameters chosen in Table 1. 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 [14] 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 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) [14] 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 was run successfully, the map program maps the logical design to the Xilinx FPGA device. Finally the PAR program accepts the mapped design, places and routes the FPGA, and produces the output for the bitstream (BitGen) generator. The latter program receives the placed and routed design and generates a bitstream (*.bit file) for Xilinx device configuration. Before programming the FPGA file, a VITAL simulation was performed to ensure that the circuit meets the timing requirements set and works correctly.

![Figure 17: Design Flow Map.](image)

5. RESULTS

Figure 18 shows the data flow for all the signals specified in Figure 2. A new input data set is clocked on the rising edge of the clkdv_out clock, with 1/16 the frequency of the clkfx_out clock, providing the necessary time (16 algorithmic clock cycles) for the Address Generator (pipeline stage, cpr1) to generate the signals corresponding to all active rules (2nd-17th clock cycles). Here, we remind that by using the ars_p block we effectively identify and process 16 active rules per clock cycle instead of 2401 [8].

The Trapezoidal Generator (trap_gen_p) block requires three pipeline stages (cpr3) to compute the α values of the membership functions. The computation occurs while the system addresses and reads the Singletons ROM. Two clocks later (cpr5), after the rule selection (rule_sel_p) and the Minimum operator (andor_meth_p) 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 (mult), occurs two clocks later along with the addition of the 0 values. In the next clock, the sum of the implications is computed. During that time the unsigned integrator (div uns) outputs the divisor value (cpr7), while the dividend is computed by the signed integrator (div sig), three clocks later (cpr9) the LUT reciprocal division estimation (div_ppa) becomes available. 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 27 pipe stages or 270 ns which is analyzed in 16 algorithmic clock cycles (see the Address Generator - section 3.3) and 11 clock cycles due to pipelining (Table 1). This effectively characterizes the throughput data processing rate of the system (a new input data set can be sampled every 16 clocks or 160 ns), while the internal clock frequency operation of the DFLP Core is 100 MHz or 10 ns period. Along with the proposed DFLP architecture, a modified model with LUT based MF Generator blocks, instead of arithmetic based, has been implemented which is not presented in this paper. The latter DFLP model achieves a better timing result with the same levels of pipelining, but with significant increase in FPGA area utilization compared to the presented model. Moreover, it is obvious that since the MF’s are ROM based any type and shape could be implemented.
6. CONCLUSIONS AND FUTURE WORK

We have managed to design a parameterized and amenable to scaling, digital fuzzy logic processor (DFLP) soft core that processes only the active rules for a given input data set, thus increasing substantially the overall input data processing rate of the system. The soft core model is parametric in terms of the different number of inputs and outputs, number of triangular or trapezoidal fuzzy sets per input, number of singletons per output, antecedent method connection, divider model type, and number of pipeline registers for the individual components in the hierarchy. The parameterized DFLP Core allows the creation of various fuzzy processors without the need of recoding the RTL components description. Two DFLP architectures based on the selected parameters (Table 1) were implemented; one with arithmetic -and one with LUT- based membership function generators (not described in this paper). The parameters in Table 1 were selected so as to create the fuzzy controller required for balancing control of the Pendubot inverted pendulum [18]. A laboratory experiment is in-progress for the evaluation of the capabilities of this controller. The results of this evaluation will appear elsewhere.

REFERENCES


