Łukasz SURDEJ, Lesław GNIEWEK
RZESZOW UNIVERSITY OF TECHNOLOGY, DEPARTMENT OF COMPUTER AND CONTROL ENGINEERING
12 Powstańców Warszawy Ave., 35-959 Rzeszów, Poland

Synchronous and asynchronous structural implementation of Łukasiewicz norms in Spartan-6 FPGAs

Abstract

Fast time to market, high performance and low cost make new FPGAs a competition for dedicated VLSI device in many area. Their array architecture with lots of programmable resources and IO pins is attractive hardware platform for implementation a complex fuzzy systems. The article discusses the realization of fuzzy Łukasiewicz operations in Xilinx Spartan-6 FPGAs, which in addition to Zadeh operations, are basic elements in fuzzy systems. Safe behavioral description of these operations that define functionalities independent of the hardware platform are presented. Structural descriptions of both synchronous and asynchronous fuzzy operations are shown, to carry out their primitive level realization and the effective utilization of basic elements of the FPGA structure. As the result the area optimized implementation of Łukasiewicz operations are obtained.

Keywords: fuzzy hardware, fuzzy Łukasiewicz norms, FPGA.

1. Introduction

Fuzzy combinatorial circuits with semantics based on [0, 1] interval of real number have been created by replacing the classical conjunction, disjunction and alternative by t-norm, t-conorm, and strong negation. To generalize the classical systems many different operators can be used i.e. Zadeh (min, max), Łukasiewicz, algebraic, drastic as well as Yager, Hamacher, Dombi, Dubois-Prade, Frank parametric operations [1]. Practical application have found mainly Zadeh norms:

$$a \vee b = \max(a, b)\,, \tag{1}$$

$$a \wedge b = \min(a, b)\,, \tag{2}$$

and Łukasiewicz norms:

$$a \oplus b = \min(a + b, \mathbf{1})\,, \tag{3}$$

$$a \otimes b = \min(a + b - 1, \mathbf{0})\,, \tag{4}$$

with a standard negation:

$$\sim a = 1 - a. \tag{5}$$

These norms have been used to build the fuzzy sequential circuits i.e. JK, SR, T, D flip-flops [2]. Flip-flops, in turn, have been used in the fuzzy memory modules, as elements of the fuzzy neural network [3], fuzzy Petri nets [4] and other fuzzy systems.

The development of combinational and sequential fuzzy circuits began with the implementation of the nine fuzzy logic function by Yamakawa [5] and the concept of fuzzy JK flip-flop proposed by Hirota and Ozawa [6] with its hardware implementation in a VLSI. These chips appeared to be an appropriate platform to cope with the complexity of fuzzy systems. Since then analog and discrete realizations of fuzzy operations were discussed in the literature. In analog hardware fuzzy variables from interval [0, 1] were replaced mostly by a voltage in a range [0 V, 5 V] [6] or a current, normalized to it's max value (a few μA) [5]. In discrete realizations fuzzy variables were recorded on *n*-bits and took one of 2^n values from a discrete set of $\{0..00,\ 0..01,\ \ldots\ , 2^n - 1\}$. In this way logic low state corresponded to the series of zeros $\mathbf{0} = 00\ldots00_2$, and the logic high state to a series of ones $\mathbf{1} = 2^n - 1 = 11\ldots11_2$.

Modern FPGAs, with lots of programmable logic and attractive timing performance make an alternative to dedicated VLSI device in implementation of discrete fuzzy systems. Examples of schematic implementation of Łukasiewicz norms in the FPGA can be found in [7, 8, 9], where these norms were used for the construction of fuzzy flip-flops and fuzzy Petri nets.

The article presents the structural descriptions (both synchronous and asynchronous) which enable the realization of Łukasiewicz operations at the level of primitives in FPGA circuits. This allowed to obtain optimal utilization of hardware resources, compared with the use of behavioral descriptions.

2. Behavioral realization

FPGAs can be designed with a schematic or a hardware description language (VHDL, Verilog). Projects based on a HDL may be constructed in a behavioral or structural style. A behavioral description reflect functionality of the device. It has no direct reference to physical resources (e.g. a specific LUT or multiplexer) so that it can be transferred to any hardware platform with no changes. Necessary FPGA resources are established upon a code during a synthesis and implementation process.

Functional diagrams of fuzzy Łukasiewicz operations from equation (3) and (4) are shown in Fig. 1.

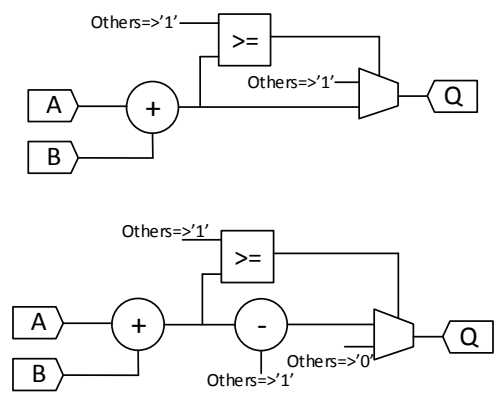


Fig. 1. a) simplified diagram of Łukasiewicz sum, and b) product diagram

The synthesizable VHDL code, describing the behavioral architecture of Łukasiewicz t-norm and t-conorm, programmed according to the diagrams from Fig. 1 is shown in Listing 1. The parameter *n* reflects the bit resolution.

Listing 1. The behavioral description of generic Łukasiewicz t-norm and t-conorm

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
entity Tnorm is
   Generic(n: positive :=4);
   Port ( A : in  STD_LOGIC_VECTOR (n-1 downto 0);
          B : in  STD_LOGIC_VECTOR (n-1 downto 0);
          Q : out  STD_LOGIC_VECTOR (n-1 downto 0));
end Tnorm;
architecture Behavioral of Tnorm is
 constant Jeden:std_logic_vector(n-1 downto 0):=(others=>'1');
 constant Zero :std_logic_vector(n-1 downto 0):=(others=>'0');
begin
   process (A,B)
      variable suma: std_logic_vector(n downto 0);
```

```
   begin
      suma := conv_std_logic_vector((conv_integer(A) +
              conv_integer(B)),n+1);
      if suma >= Jeden then
         suma := conv_std_logic_vector((conv_integer(suma) -
                 conv_integer(Jeden)),n+1);
         Q <= suma(n-1 downto 0);
      else
         Q <= (others => '0');
      end if;
   end process;
end Behavioral;

architecture Behavioral of Cnorm is
 constant Jeden:std_logic_vector(n-1 downto 0):=(others=>'1');
begin
   process (A,B)
      variable suma: std_logic_vector(n downto 0);
   begin
      suma := conv_std_logic_vector((conv_integer(A) +
              conv_integer(B)),n+1);
      if suma >= Jeden then
         Q <= Jeden;
      else
         Q <= suma(n-1 downto 0);
      end if;
   end process;
end Behavioral;
```

If the input variables *a* and *b* are presented in the NB code:

$$a = a_{n-1}2^{n-1} + \cdots + a_2 2^2 + a_1 2^1 + a_0 2^0, \quad (6)$$

$$b = b_{n-1}2^{n-1} + \cdots + b_2 2^2 + b_1 2^1 + b_0 2^0, \quad (7)$$

their sum is an *n*+1 -bit:

$$s = a + b = s_n 2^n + s_{n-1}2^{n-1} + \cdots + s_2 2^2 + s_1 2^1 + s_0 2^0. \quad (8)$$

In this case Łukasiewicz fuzzy operations for 4-bit numbers *a* and *b* can be described by equations:

$$a \oplus b = \begin{cases} a+b & \text{if } s < 10000_2 \\ 1111_2 & \text{if } s \geq 10000_2 \end{cases} = \begin{cases} a+b & if\ s_4 = 0_2 \\ 1111_2 & if\ s_4 = 1_2 \end{cases} \quad (9)$$

$$a \otimes b = \begin{cases} a+b-1111_2 & \text{if } s \geq 10000_2 \\ 0000_2 & \text{if } s < 10000_2 \end{cases} = \\ = \begin{cases} a+b-10000_2+1_2 & \text{if } s_4 = 1_2 \\ 0000_2 & \text{if } s_4 = 0_2 \end{cases}. \quad (10)$$

It can be seen that the result is a product of *n*–1 less significant bits of the sum ($s_{n-1}, \ldots, s_1, s_0$), and the most significant bit s_n imposes appropriate limits.

Adders in Spartan-6 FPGAs are implemented in Configurable Logic Blocks (CLB) as a ripple-carry adders consisted of full adders (Fig. 2), whose operation is characterized by the equations:

$$s_i = c_{i-1}\, Xor\, p_i, \quad c_i = g_i + c_{i-1}p_i, \quad (11)$$

$$p_i = a_i\, Xor\, b_i, \quad g_i = a_i b_i, \quad (12)$$

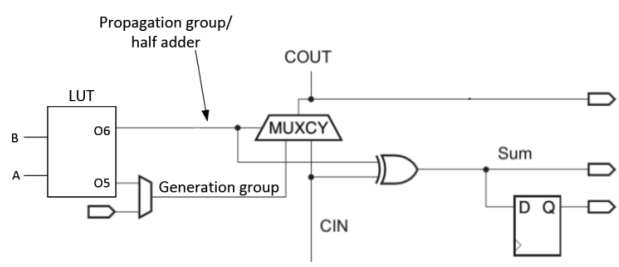


Fig. 2. Spartan-6 FPGA implementation of one bit full adder [10]

A propagation group p_i is generated in Look-Up Table (LUT) - the functions generator. A sum bit s_i is created in a dedicated XORCY gate from a propagation group and a carry of the previous bit. The carry bit c_i is formed in the MUXCY, that is part of carry logic and implements the function:

$$c_i = (a_i\, xor\, b_i)c_{i-1} + a_i\overline{(a_i\, xor\, b_i)}. \quad (13)$$

Full adders are connected to form multi-bit ripple-carry adders by fast carry chain (Fig. 2) being much faster than a switch matrix.

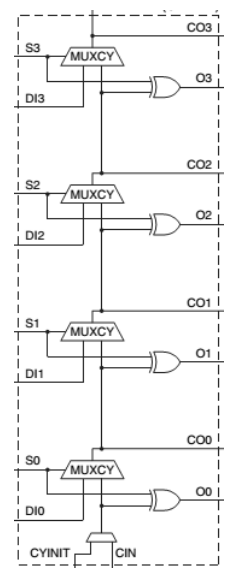


Fig. 3. Fast carry chain in Spartan-6 [10]

Using the carry chain, a fuzzy Łukasiewicz sum and a product can be constructed based on Eqs (9) and (10) in the form of an adder having at the outputs the OR gate (for t-conorm) or AND gate (for t-norm) which is presented in Fig. 4. The most significant bit of the sum s_n, imposing appropriate limits, corresponds to a signal COUT from bit *n*–1.

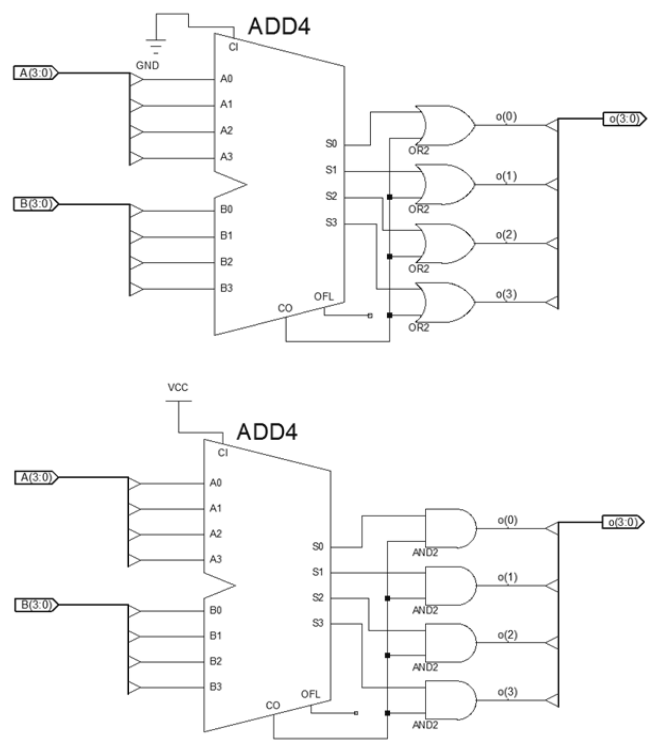


Fig. 4. a) Logical diagram of 4-bit Łukasiewicz t-conorm, and b) t-norm diagram

In the fuzzy product, described by the equation (10), addition of 1_2 and subtraction of 10000_2 is realized in Fig. 3 by connecting the input CIN of the adder to a high logic state and omission the s_4 bit. Diagram realizations of Łukasiewicz norms, build with adders and multiplexers can be found in [10]. A generic code of such a Łukasiewicz t-norm and t-conorm is presented in Listing 2.

Listing 2. A Behavioral architecture of generic Łukasiewicz t-norm and t-conorm based on Eqs (9) and (10)

```
architecture Behavioral of Tnorm is
begin
   process (A,B)
      variable suma: std_logic_vector(n downto 0);
   begin
      suma := conv_std_logic_vector((conv_integer(A) +
              conv_integer(B) + conv_integer('1')),n+1);
      if suma(n) = '0' then
         suma(n-1 downto 0) := (others => '0');
      end if;
      Q <= suma(n-1 downto 0)
   end process;
end Behavioral;

architecture Behavioral of Cnorm is
begin
   P1: process (A,B)
      variable suma: std_logic_vector(n downto 0);
   begin
      suma := conv_std_logic_vector((conv_integer(A) +
              conv_integer(B)),n+1);
      if suma(n) = '1' then
         suma(n-1 downto 0) := (others => '1');
      end if;
      Q <= suma(n-1 downto 0);
   end process;
end Behavioral;
```

Gates at the output of the adder may be replaced by flip-flops to receive the synchronous realization (Fig. 5).

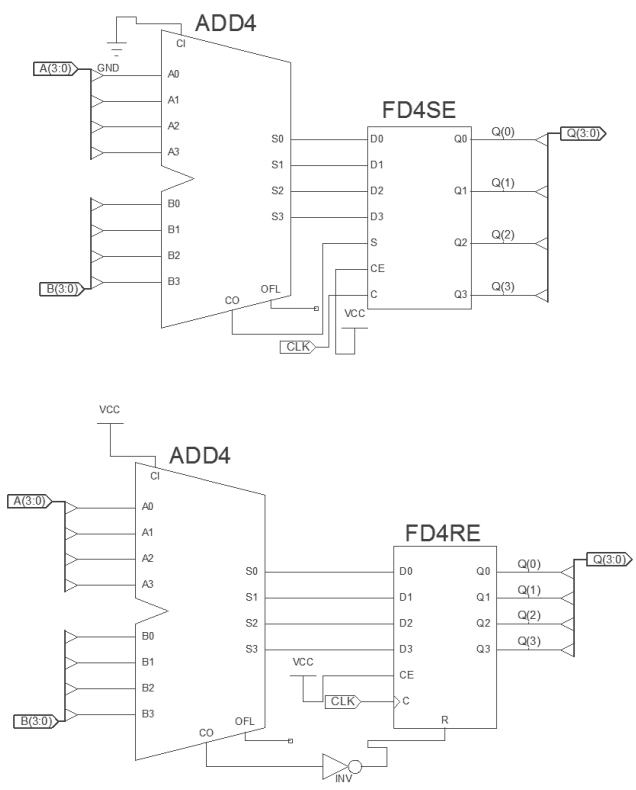


Fig. 5. a) Logical diagram of 4-bit synchronous Łukasiewicz t-conorm, and b) t-norm

Listing 3 describes behavioral code of a synchronous t-norm and t-conorm synchronized by a clock CLK_IN.

Listing 3. A Behavioral architecture of generic synchronous Łukasiewicz t-norm and t-conorm

```
entity TnormSynch is
   Generic(n: positive :=4);
   Port ( A : in  STD_LOGIC_VECTOR (n-1 downto 0);
          B : in  STD_LOGIC_VECTOR (n-1 downto 0);
      CLK_IN: in  STD_LOGIC;
          Q : out  STD_LOGIC_VECTOR (n-1 downto 0));
end TnormSynch;
architecture Behavioral of TnormSynch is
begin
   process (CLK_IN)
      variable suma: std_logic_vector(n downto 0);
   begin
      if rising_edge(CLK_IN) then
         suma := conv_std_logic_vector((conv_integer(A) +
                 conv_integer(B) + conv_integer('1')),n+1);
         if suma(n) = '1' then
            Q <= suma(n-1 downto 0);
         else
            Q <= (others => '0');
         end if;
      end if;
   end process;
end Behavioral;

architecture Behavioral of CnormSynch is
begin
   process (CLK_IN)
      variable suma: std_logic_vector(n downto 0);
   begin
      if rising_edge(CLK_IN) then
         suma := conv_std_logic_vector((conv_integer(A) +
                 conv_integer(B)),n+1);
         if suma(n) = '1' then
            Q <= (others => '1');
         else
            Q <= suma(n-1 downto 0);
         end if;
      end if;
   end process;
end Behavioral;
```

3. Structural realization

A behavioral code is more intuitive, but not always produces an optimum and desired structure. For example, for synchronous norms shown in Fig. 4 flip-flops may be placed in the same slices as adders. This significantly reduces occupied resources - for *n*-bit Łukasiewicz norm to the area of adder (*n*/4 slice). The specific structure can be obtained with a structural style.

The structural description is composed of instances of components and connections between them reflecting the device's diagram. Components can be functional and logical blocks or physical models of system elements supplied by the manufacturers in the form of a primitives library.

A structural code of *n*-bit t-conorm for Spartan-6 device is presented in Listing 4.

Listing 4. The structural description of generic synchronous Łukasiewicz t-conorm for Spartan-6 FPGA

```
LIBRARY IEEE;
LIBRARY UNISIM;
USE IEEE.STD_LOGIC_1164.ALL;
USE UNISIM.VCOMPONENTS.ALL;
entity CnormSynchS is
   Generic(n: positive :=8;
     lutInit: bit_vector := x"6");
   Port ( A : in  STD_LOGIC_VECTOR (n-1 downto 0);
          B : in  STD_LOGIC_VECTOR (n-1 downto 0);
      CLK_IN: in  STD_LOGIC;
          Q : out  STD_LOGIC_VECTOR (n-1 downto 0));
   attribute RPM_GRID: string;
   attribute RPM_GRID of CnormSynchS : entity is "GRID";
end CnormSynchS;
architecture Structural of CnormSynchS is
   signal carry : std_logic_vector(n downto 0);
   signal lutOut : std_logic_vector(n-1 downto 0);
   signal xorOut : std_logic_vector(n-1 downto 0);
   attribute RLOC : string;
```

```
begin
   carry(0) <= '0';
   L1:
      for i in 0 to n-1 generate
         attribute RLOC of xorLut: label is "X0Y" &
            integer'image(integer(4*(i/4)));
         attribute RLOC of xorSum: label is "X0Y" &
            integer'image(integer(4*(i/4)));
         attribute RLOC of carryMux: label is "X0Y" &
            integer'image(integer(4*(i/4)));
         attribute RLOC of flipFlopD: label is "X0Y" &
            integer'image(integer(4*(i/4)));
      begin
         xorLut : LUT2
            generic map (INIT => lutInit)
            port map (O => lutOut(i), I0 => A(i), I1 => B(i));
         xorSum : XORCY
            port map(O => xorOut(i), CI => carry(i),
               LI => lutOut(i));
         carryMux : MUXCY
            port map(O => carry(i+1), CI => carry(i),
               DI => A(i), S => lutOut(i));
         flipFlopD : FDS
            port map(Q => Q(i), C => CLK_IN, D => xorOut(i),
               S => carry(n));
      end generate;
end Structural;
```

The main element of a structural t-conorm description is *for* … generate loop replicating n-times a one bit full adder shown in Fig. 2 with a flip-flop at the output to form a ripple-carry adder. Single-bit full adder is composed of elements labeled xorLut, xorSum, carryMux. For the creation of a propagation group p_i (12) a LUT is initiated according to the truth table of Xor function with hexadecimal value x"6" (Tab. 1).

Tab. 1. The truth table of Xor function

a_1	a_0	q
1	1	0
1	0	1
0	1	1
0	0	0

Attributes RLOC (Relative Location) enforce proper distribution of instantiated primitives to each other. To locate objects the RPM_GRID system is used. The Y value of attributes RLOC is integer(4*(i/4)) which deploys adders in a column and allows to use the carry chain. This value is a result of the assumed grid system and a structure of Spartan-6 carry chain in CLB. Outputs of the ripple-carry adder are terminated with flip-flops labeled flipFlopD as in Fig. 5. In the Spartan-6 there is no need to set the initial state of flip-flops (or latches), because it is closely related to the used set/reset signal. The area occupied by t-conorm is n/4 slices.

A structural code of n-bit t-norm for Spartan-6 FPGA is shown in Listing 5.

Listing 5. The structural architecture of generic synchronous Łukasiewicz t-norm for Spartan-6 FPGA

```
architecture Structural of TnormSynchS is
   signal carry : std_logic_vector(n downto 0);
   signal notCarryN : std_logic;
   signal lutOut : std_logic_vector(n-1 downto 0);
   signal xorOut : std_logic_vector(n-1 downto 0);
   attribute RLOC : string;
   attribute RLOC of inverter: label is "X0Y" &
      integer'image(integer(4*((n-1)/4)));
begin
   carry(0) <= '1';
   L1:
      for i in 0 to n-1 generate
         attribute RLOC of xorLut: label is "X0Y" &
            integer'image(integer(4*(i/4)));
         attribute RLOC of xorSum: label is "X0Y" &
            integer'image(integer(4*(i/4)));
         attribute RLOC of carryMux: label is "X0Y" &
            integer'image(integer(4*(i/4)));
         attribute RLOC of flipFlopD: label is "X0Y" &
            integer'image(integer(4*(i/4)));
      begin
         xorLut : LUT2
            generic map (INIT => lutInit)
            port map (O => lutOut(i), I0 => A(i), I1 => B(i));
         xorSum : XORCY
            port map(O => xorOut(i), CI => carry(i),
               LI => lutOut(i));
         carryMux : MUXCY
            port map(O => carry(i+1), CI => carry(i),
               DI => A(i), S => lutOut(i));
         flipFlopD: FDR
            port map( Q => Q(i), C => CLK_IN, D => xorOut(i),
               R => notCarryN);
      end generate;
   inverter: LUT1
         generic map (INIT => x"1")
         port map (O => notCarryN, I0 => carry(n));
end Structural;
```

Flip-flops in Spartan-6 CLB have only high an active signal level on set/reset port. T-norm in Fig. 5 requires active low level on the flip-flop's reset so that additional instance of the inverter for a carry(n) signal is necessary. This inverter can be placed in the same LUT as a xorLut instance because LUTs in Spartan-6 have two outputs. The last slice configuration of 8-bit synchronous t-norm is presented in Fig. 6.

An asynchronous realization of Łukasiewicz norms can be obtained by replacing gates (Fig. 4) at the adder's outputs with latches. Listing 6 shows a structural code of asynchronous t-conorm with latches for Spartan-6 device.

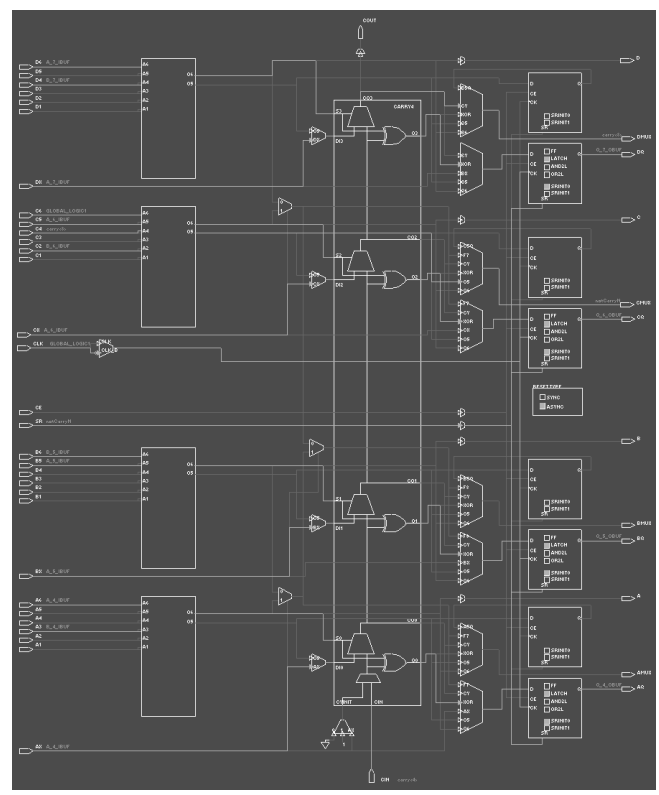


Fig. 6. Configuration of the last slice of 8-bit t-norm in Spartan-6 FPGA

Listing 6. The structural architecture of generic asynchronous Łukasiewicz t-norm for a Spartan-6 FPGA

```
architecture Structural of CnormS is
   signal carry : std_logic_vector(n downto 0);
   signal lutOut : std_logic_vector(n-1 downto 0);
   signal xorOut : std_logic_vector(n-1 downto 0);
   attribute RLOC : string;
begin
   carry(0) <= '0';
   L1:
      for i in 0 to n-1 generate
      attribute RLOC of xorLut: label is "X0Y" &
```

```
         integer'image(integer(4*(i/4)));
      attribute RLOC of xorSum: label is "X0Y" &
         integer'image(integer(4*(i/4)));
      attribute RLOC of carryMux: label is "X0Y" &
         integer'image(integer(4*(i/4)));
      attribute RLOC of latch: label is "X0Y" &
         integer'image(integer(4*(i/4)));
      begin
         xorLut : LUT2
            generic map (INIT => lutInit)
            port map (O => lutOut(i), I0 => A(i), I1 => B(i));
         xorSum : XORCY
            port map(O => xorOut(i), CI => carry(i),
               LI => lutOut(i));
         carryMux: MUXCY
            port map(O => carry(i+1), CI => carry(i),
               DI => A(i), S => lutOut(i));
         latch : LDP
            port map( Q => Q(i), D => xorOut(i), G => '1',
               PRE => carry(n));
      end generate;
end Structural;
```

Latches should be used very carefully because for the Spartan-3A FPGA it was observed in the time simulation (after place and route - PAR) that short change at a set/reset signal, caused by propagation delays, could entail unstable condition (transient at their output). In Spartan-6 time simulations similar phenomenon was not detected.

A structural description allows to achieve the implementation that uses precisely planed area. However, it is much more difficult, less intuitive and therefore prone to errors. It is also platform-dependent. For example, the implementation of t-norm and t-conorm in a Spartan-3A requires a modification of the method of calculating the relative location of elements. In the case of t-norm this FPGA can absorb the inverter in flip-flop and change it to low-level triggered set/reset signal.

4. Test results

The correct operation of the implementation of fuzzy Łukasiewicz norms was confirmed during functional (behavioral) and timing (after PAR) simulation. Listing 7 illustrates testbench file used in a timing simulation of 4-bit synchronous t-conorm.

Listing 7. The testbench file for timing simulation of 4-bit synchronous Łukasiewicz t-conorm.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.NUMERIC_STD.ALL;
USE IEEE.STD_LOGIC_TEXTIO.ALL;
USE STD.TEXTIO.ALL;
entity TimingTestKonormySynch is
end TimingTestKonormySynch;
architecture behavioral of TimingTestKonormySynch is
   component CnormSynchS is
      Port ( A : in  std_logic_vector (3 downto 0);
             B : in  std_logic_vector (3 downto 0);
         CLK_IN: in  std_logic;
             Q : out  std_logic_vector (3 downto 0));
   end component KnormSynchS;
   constant n : positive := 4;
   signal Q       :        std_logic_vector (n-1 DOWNTO 0);
   signal A       :        std_logic_vector (n-1 DOWNTO 0);
   signal B       :        std_logic_vector (n-1 DOWNTO 0);
   signal clock : std_logic;
   constant Period : Time := 40 ns;
   constant Delay : Time := Period /4;
   file RaportFile: text is out "RapartTestowy.txt ";
begin
   UUT: KnormSynchS port map(A => A, B => B,
                        CLK_IN => clock, Q => Q);
   ProcClock : process
      begin
         clock <= '0';
         wait for Period/2;
         clock <= '1';
         wait for Period/2;
     end process ProcClock;
   ProcData : process
         variable result : std_logic_vector (n-1 downto 0);
         variable tx_out: line;
         variable errorCounter: integer := 0;
      begin
         wait for 5 * Period;
         for i in 0 to 2**n-1 loop
            for j in 0 to 2**n-1 loop
               wait for Delay;
               A <= conv_std_logic_vector(i,n);
               B <= conv_std_logic_vector(j,n);
               if (i+j) < (2**n-1) then
                  result := conv_std_logic_vector(i+j,n);
               else
                  result := conv_std_logic_vector(2**n-1,n);
               end if;
               wait for 3 * Delay;
               if Q /= result then
                  write(tx_out,string'("blad: "));
                  write(tx_out,errorCounter);
                  writeline(RaportFile,tx_out);
                  write(tx_out,string'("a: "));
                  write(tx_out,a);
                  write(tx_out,string'(" b: "));
                  write(tx_out,b);
                  write(tx_out,string'(" wyn: "));
                  write(tx_out,result);
                  writeline(RaportFile,tx_out);
                  errorCounter := errorCounter + 1;
                  assert false report "blad!" severity error;
               end if;
            end loop;
         end loop;
         if errorCounter > 0 then
            assert false report "Process ends with errors !!!"
               severity failure;
         else
            assert false report "No errors detected"
               severity note;
         end if;
      end process;
end;
```

In the prepared testbench an instance of t-conorm UUT (unit under test) was created, generated clock signal and test vectors, and formed a validation routine that checks the correctness of responses with appropriate time delays. During the test any detected errors are drawn up to the report text file.

In Tab. 2 and Tab. 3 the area occupied by asynchronous implementations of Łukasiewicz t-conorm and t-norm in a Spartan-6 is summarized. In all cases a structural description gave the best results.

Tab. 2. The area occupied by asynchronous Łukasiewicz t-conorm in Spartan-6

T-conorm with architecture	4-bit	8-bit	12-bit	16-bit	24-bit
behavioral	4 LUT/ 2 Slice	12 LUT/ 6 Slice	18 LUT/ 8 Slice	24 LUT/ 11 Slice	41 LUT/ 20 Slice
behavioral Eqs (9) and (10)	4 LUT/ 2 Slice	12 LUT/ 6 Slice	18 LUT/ 8 Slice	24 LUT/ 11 Slice	36 LUT/ 15 Slice
structural	4 LUT/ 1 Slice	8 LUT/ 2 Slice	12 LUT/ 3 Slice	16 LUT/ 4 Slice	24 LUT/ 6 Slice

Tab. 3. The area occupied by asynchronous Łukasiewicz t-norm in Spartan-6

T-norm with architecture	4-bit	8-bit	12-bit	16-bit	24-bit
behavioral	7 LUT/ 2 Slice	15 LUT/ 6 Slice	38 LUT/ 18 Slice	44 LUT/ 18 Slice	65 LUT/ 24 Slice
behavioral Eqs (9) and (10)	4 LUT/ 1 Slice	12 LUT/ 6 Slice	18 LUT/ 8 Slice	24 LUT/ 11 Slice	36 LUT/ 15 Slice
structural	4 LUT/ 1 Slice	8 LUT/ 2 Slice	12 LUT/ 3 Slice	16 LUT/ 4 Slice	24 LUT/ 6 Slice

Post-PAR Static Timing Report indicated that presented structural descriptions can be implemented in a Spartan-6 with a frequency above 400 MHz.

5. Conclusions

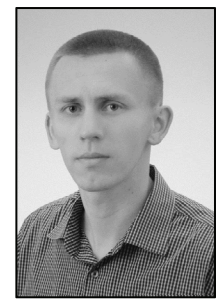
FPGA devices enable the implementation of high-speed and complex fuzzy systems. Diagrams as well as behavioral and structural descriptions allowing to create synchronous and asynchronous Łukasiewicz t-norms and t-conorms have been presented. It has been discussed how to use effectively Spratan-6 FPGA programmable resources to construct these norms. The structural description has been shown to provide implementation of the presented method of FPGA device configuration and to lead to the best area utilization. In this case resources occupied by fuzzy Łukasiewicz t-norm and t-conorm are equal to the multi-bit adder. Tests results expose a big difference in area saving between structural and behavioral description, especially in higher bit resolution.

6. References

[1] Fodor J., Yager R.: Fuzzy Set-Theoretic Operators and Quantifiers. In: Dubois, D., Prade, H. (eds.) Fundamentals of Fuzzy Sets. Kluver, pp. 125-193, 2000.
[2] Atmaca H., Yavuz H. S.: The Implementation of Fuzzy Flip-Flops as Memory Modules. International Conference on Electrical and Electronics Engineering ELECO99, pp. 423-427, 1999.
[3] Kowalski Piotr A.: Evolutionary strategy for the Fuzzy Flip-Flop neural networks supervised learning procedure. Artificial Intelligence and Soft Computing. Springer Berlin Heidelberg, p. 294-305, 2013.
[4] Gniewek L.: Sequential Control Algorithm in The Form of Fuzzy Interpreted Petri Net. IEEE Transactions on Systems, Man, and Cybernetics: Systems, vol. 43, no. 2, pp. 451–459, March 2013.
[5] Yamakawa T, Miki T.: The Current Mode Fuzzy Logic Integrated Circuits Fabricated by the Standard CMOS Process. IEEE Transactions on Computers, vol. 35, no. 2, pp. 161-167, 1986.
[6] Hirota K., Ozawa K.: The Concept of Fuzzy Flip-Flop. IEEE Transactions on Systems, Man, and Cybernetic, vol. 19, no. 5, pp. 980-997, 1989.
[7] Lovassy R., Zavala A., Gál L., Nieto O. C., Kóczy L., Batyrshin I.: Hardware Implementation of Fuzzy Flip-Flops Based on Łukasiewicz Norms. Proceedings of the 9th WSEAS Int. Conference on Applied Computer and Applied Computational Science, Wisconsin, pp. 196-201, 2010.
[8] Lovassy R., Gál L., Tóth Á., Kóczy L. T., Rudas I. J.: Fuzzy Flip-Flop based Neural Networks as a novel implementation possibility of multilayer perceptrons. Instrumentation and Measurement Technology Conference (I2MTC), IEEE, 2012, pp. 280 – 285.
[9] Hajduk Z., Wojtowicz J.: Hardware Implementation of Fuzzy Petri Nets with Lukasiewicz Norms for Modelling of Control Systems. Lecture Notes in Computer Science, vol. 9621. Springer Berlin Heidelberg, pp. 449-458, 2016.
[10] Spartan-6 FPGA Configurable Logic Block User Guide, UG384, Xilinx, 2010.

Received: 03.08.2016 *Paper reviewed* *Accepted: 03.10.2016*

Łukasz SURDEJ, MSc, eng.



Łukasz Surdej received the MSc degree in electronics and telecommunications from the Military University of Technology, Warsaw, Poland, in 2011. Currently, he is a second year PhD student at the faculty of computer science at the Rzeszow University of Technology. The area of his interests includes FPGAs, microcontrollers and fuzzy logic hardware.

e-mail: lukasz.surdej@gmail.com

Lesław GNIEWEK, DSc, PhD, eng.

Lesław Gniewek received the PhD and DSc degrees in computer science from Wroclaw University of Technology, in 1999 and 2014, respectively. From 2014 he is Associate Professor at the Rzeszow University of Technology. His research interests are in area of fuzzy logic hardware, fuzzy Petri nets, and programmable logic controllers.

e-mail: lgniewek@prz-rzeszow.pl