МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

Кафедра ЕОМ

Звіт з лабораторної роботи №4
з дисципліни «Основи проектування цифрових засобів на ПЛІС»
на тему:

«РОЗРОБКА КОНВЕЄРНОГО ПРОЦЕСОРА»

Виконав:
студент групи КІ-41
Кушнір Д.О.
Прийняв:
викл. Цигилик Л.О.

Львів – 2016

1. **Тема:** Розробка конвеєрного процесора.
2. **Мета:** Розробити конвеєрний процесор

3. **Постановка задачі:**

Для перетворення прототипу комбінаційної схеми в потокову структуру та отримання VHDL коду цієї процедури, зручно використовувати такі кроки:

1) Створити блок-схему оригінальної комбінаційної схеми та організувати її компоненти в вигляді каскадної послідовності.

2) Визначити основні компоненти блок-схеми та оцінити відносні тривалості їх виконання.

3) Поділити послідовність компонентів на етапи так, щоб тривалість етапів була приблизно рівною.

4) Визначити дані й сигнали необхідні на кожному етапі опрацювання та вибрати ті, які надходять з інших етапів опрацювання.

5) Розмістити регістри для зберігання даних й сигналів, які находять з інших етапів опрацювання.

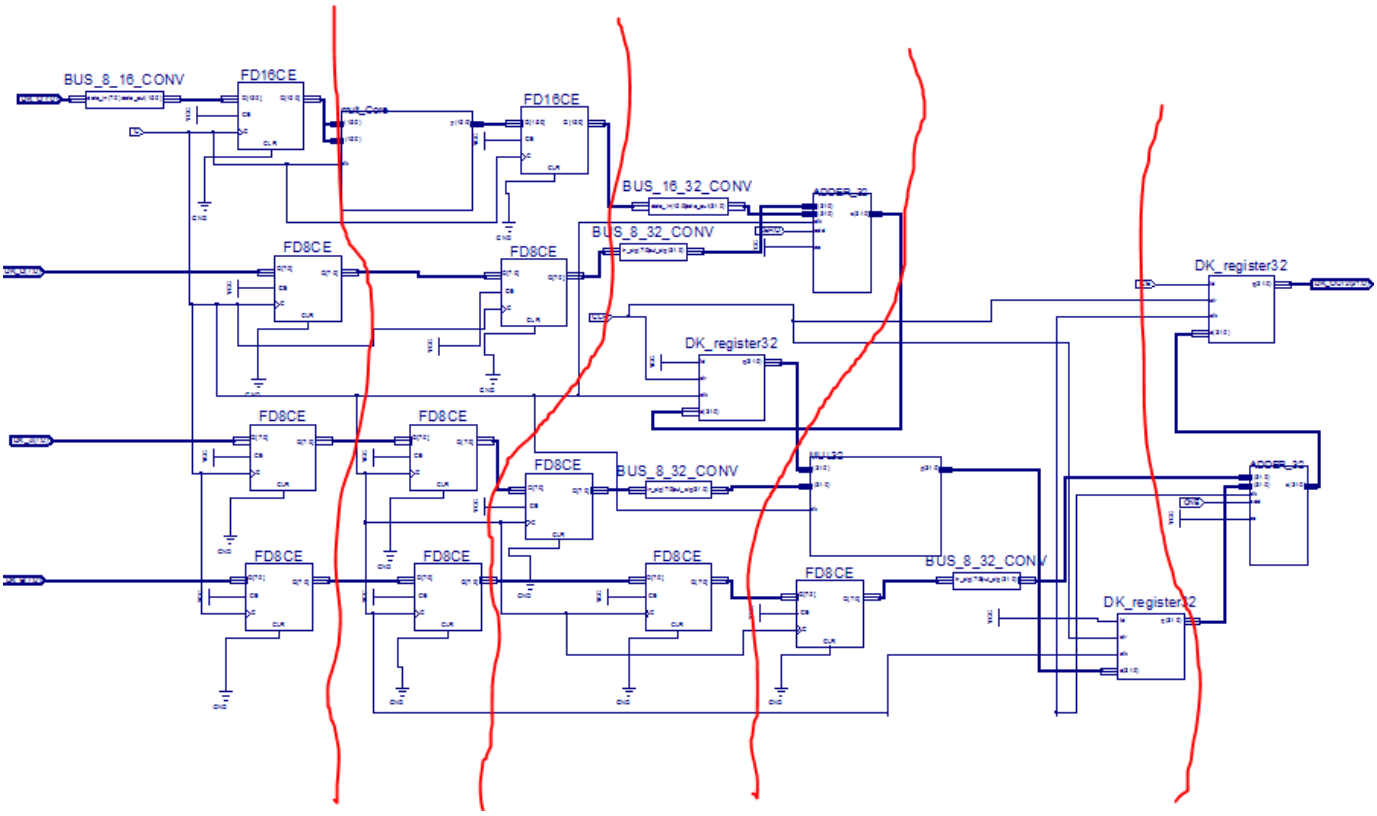
Завдання:

11. $a \cdot (d - c^2) + b$

Для створення конвеєра потрібно поділити функцію КОП на дрібніші кроки, а саме:

c*c
d - c*c
a * (d - c*c)
a * (d - c*c) + b

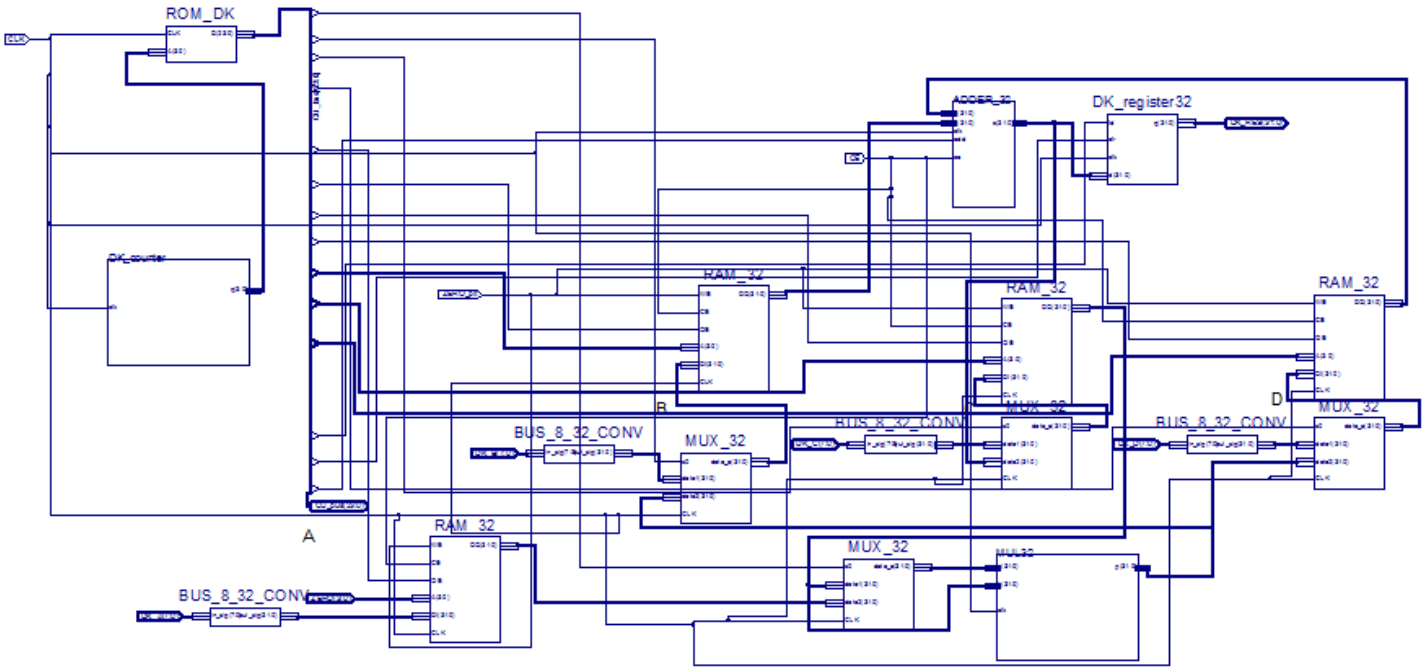
4. Схема конвеєрного обчислення виразу



5. vhdl код реалізації системи

Оскільки система була розроблена за допомогою вбудованих елементів Xilinx та створеного елемента Ipcore то VHDl код був згенерований автоматично.

6. Схема неконвеєрного обчислення виразу



7. **vhdl код реалізації системи**
   Оскільки система була розроблена за допомогою вбудованих елементів Xilinx та створеного елемента Ipcore то VHDl код був згенерований автоматично.
8. **таблиця з порівнянням результатів синтезу обох варіантів**

```
=========================================================================
*                            Design Summary                             *
=========================================================================

Top Level Output File Name         : DK_conv.ngc

Primitive and Black Box Usage:
------------------------------
# BELS                             : 1724
#      GND                         : 10
#      INV                         : 20
#      LUT2                        : 234
#      LUT3                        : 42
#      LUT4                        : 232
#      MULT_AND                    : 284
#      MUXCY                       : 513
#      VCC                         : 8
#      XORCY                       : 448
# FlipFlops/Latches                : 267
#      FD                          : 66
#      FDCE                        : 201
# Clock Buffers                    : 1
#      BUFG                        : 1
# IO Buffers                       : 96
#      IBUF                        : 72
#      OBUF                        : 24


    Total                      5.416ns (3.441ns logic, 1.975ns route)
                                       (65.5% logic, 35.2% route)

    Frequency                  184.63 MHZ
```

конвеєр

```
1
2  =========================================================================
3  *                            Design Summary                             *
4  =========================================================================
5
6  Top Level Output File Name         : DK_sch.ngc
7
8  Primitive and Black Box Usage:
9  ------------------------------
10 # BELS                             : 3675
11 #      GND                         : 3
12 #      INV                         : 2
13 #      LUT1                        : 3
14 #      LUT2                        : 258
15 #      LUT3                        : 68
16 #      LUT4                        : 604
17 #      LUT5                        : 258
18 #      LUT6                        : 120
19 #      MULT_AND                    : 528
20 #      MUXCY                       : 787
21 #      MUXF7                       : 148
22 #      MUXF8                       : 74
23 #      VCC                         : 2
24 #      XORCY                       : 820
25 # FlipFlops/Latches                : 516
26 #      FD                          : 32
27 #      FDE                         : 32
28 #      FDR                         : 4
29 #      FDRE                        : 32
30 #      LDE_1                       : 416
31 # Clock Buffers                    : 5
32 #      BUFG                        : 4
33 #      BUFGP                       : 1
34 # IO Buffers                       : 93
35 #      IBUF                        : 37
36 #      OBUF                        : 32
37 #      OBUFT                       : 24
38
39  Total                      6.246ns (3.150ns logic, 3.095ns route)
40                                     (50.4% logic, 49.6% route)
41  Frequency                  160.2 MHZ
```

ітерації

9. VHDL код тестовго стенду конвеєрного способу:

```
ARCHITECTURE behavioral OF DK_Conv_Scheme_DK_Conv_Scheme_sch_tb IS

   COMPONENT DK_Conv_Scheme
   PORT( ZERO   :  IN STD_LOGIC;
         ONE    :  IN STD_LOGIC;
         CLR    :  IN STD_LOGIC;
         C :  IN STD_LOGIC;
         CE     :  IN STD_LOGIC;
         DK_OUT2 :  OUT   STD_LOGIC_VECTOR (31 DOWNTO 0);
         DK_C :  IN STD_LOGIC_VECTOR (7 DOWNTO 0);
         DK_D :  IN STD_LOGIC_VECTOR (7 DOWNTO 0);
         DK_A :  IN STD_LOGIC_VECTOR (7 DOWNTO 0);
         DK_B :  IN STD_LOGIC_VECTOR (7 DOWNTO 0));
   END COMPONENT;

   SIGNAL ZERO :  STD_LOGIC := '0';
   SIGNAL ONE  :  STD_LOGIC := '1';
   SIGNAL CLR  :  STD_LOGIC := '0';
   SIGNAL C :  STD_LOGIC;
   SIGNAL CE   :  STD_LOGIC := '1';
   SIGNAL DK_OUT2 :  STD_LOGIC_VECTOR (31 DOWNTO 0);
   SIGNAL DK_C :  STD_LOGIC_VECTOR (7 DOWNTO 0);
   SIGNAL DK_D :  STD_LOGIC_VECTOR (7 DOWNTO 0);
   SIGNAL DK_A :  STD_LOGIC_VECTOR (7 DOWNTO 0);
   SIGNAL DK_B :  STD_LOGIC_VECTOR (7 DOWNTO 0);
constant clk_c : time := 10 ns;
      SIGNAL  buf:  integer := 0;
BEGIN
   UUT: DK_Conv_Scheme PORT MAP(
      ZERO => ZERO,
      ONE => ONE,
      CLR => CLR,  C => C,
      DK_OUT2 => DK_OUT2,
      DK_C => DK_C,
      DK_D => DK_D,
      DK_A => DK_A,
      DK_B => DK_B
   );
RESET: process
begin
CLR <= '1';
wait for clk_c;
CLR <= '0'; wait;
end process;
CLC_process :process
   begin
   buf <= buf + 1;
   DK_C <= std_logic_vector(to_unsigned(buf,DK_C'length));
   DK_D <= std_logic_vector(to_unsigned(buf+1,DK_D'length));
   DK_A <= std_logic_vector(to_unsigned(buf+2,DK_A'length));
   DK_B <= std_logic_vector(to_unsigned(buf+3,DK_B'length));
   C <= '0'; wait for clk_c/2; C <= '1'; wait for clk_c/2; end process;
```

10.VHDL код тестовго стенду неконвеєрного способу:

```
ARCHITECTURE behavioral OF DK_sch_DK_sch_sch_tb IS

   COMPONENT DK_sch
   PORT( ZERO   :  IN STD_LOGIC_VECTOR (3 DOWNTO 0)  ;
         ZERO_bit: IN STD_LOGIC;
          CE    :  IN STD_LOGIC;
          DK_D  :  IN STD_LOGIC_VECTOR (7 DOWNTO 0);
          DK_C  :  IN STD_LOGIC_VECTOR (7 DOWNTO 0);
          DK_B  :  IN STD_LOGIC_VECTOR (7 DOWNTO 0);
          CU_bus  :  INOUT STD_LOGIC_VECTOR (23 DOWNTO 0);
          CLK   :  IN STD_LOGIC;

          DK_A  :  IN STD_LOGIC_VECTOR (7 DOWNTO 0);
          DK_RES  :  INOUT STD_LOGIC_VECTOR (31 DOWNTO 0));

   END COMPONENT;

   SIGNAL ZERO :  STD_LOGIC_VECTOR (3 DOWNTO 0) := "0000";
   SIGNAL ZERO_bit: STD_LOGIC := '0';
   SIGNAL CE: STD_LOGIC := '1';
   SIGNAL DK_D :  STD_LOGIC_VECTOR (7 DOWNTO 0) :=x"19";
   SIGNAL DK_C :  STD_LOGIC_VECTOR (7 DOWNTO 0) :=x"04";
   SIGNAL DK_B :  STD_LOGIC_VECTOR (7 DOWNTO 0) :=x"01";
   SIGNAL CU_bus  :  STD_LOGIC_VECTOR (23 DOWNTO 0);
   SIGNAL CLK   :  STD_LOGIC;
   SIGNAL DK_A :  STD_LOGIC_VECTOR (7 DOWNTO 0):=x"02";
   SIGNAL DK_RES  :  STD_LOGIC_VECTOR (31 DOWNTO 0);
constant clk_c : time := 1000 ns;
BEGIN

   UUT: DK_sch PORT MAP(
      ZERO => ZERO,
      ZERO_bit => ZERO_bit,
      DK_D => DK_D,
      CE => CE,
      DK_C => DK_C,
      DK_B => DK_B,
      CU_bus => CU_bus,
      CLK => CLK,
      DK_A => DK_A,
      DK_RES => DK_RES
   );
CLC_process :process

   begin
   CLK <= '1';
   wait for clk_c/2;

   CLK <= '0';
   wait for clk_c/2;
   end process;
```

11.Діаграма для конвеєрного способу:

Декілька вхідних наборів:

$2*(25-4*4)+1 = 2*(25-16)+1 = 2*(9)+1 = 18+1 =$ **19**

$3*(26-5*5)+2 = 3*(26-25)+2 = 3*(1)+2 =$ **5**

$4*(27-6*6)+3 = 4*(27-36)+3 = 4*(-9)+3 = -36+3 =$ **-33**

$5*(28-7*7)+4 =$ **-101**

$6*(29-8*8)+5 =$ **-205**

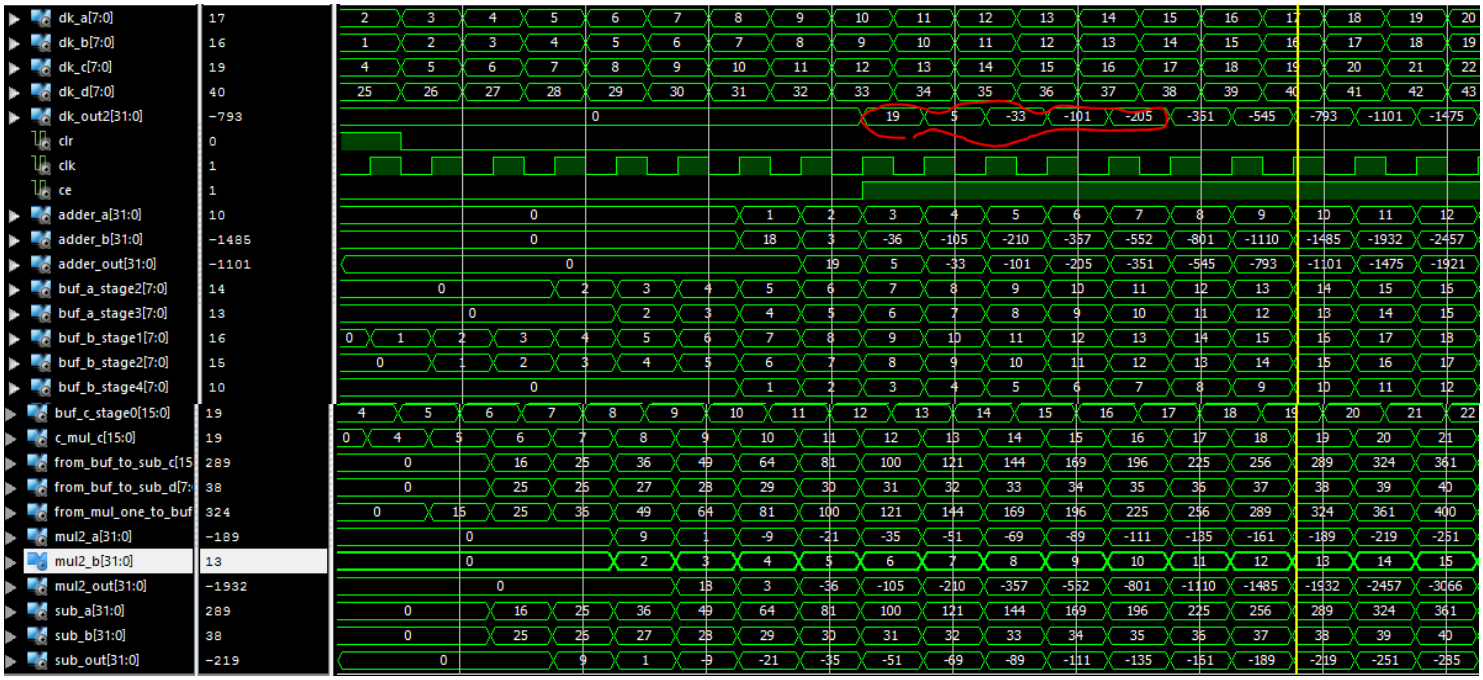
$255*(255-255*255)+255 = -16\,516\,095 =$ FF03 FC01

Етапи конвеєра	Операції
1	c*c
2	d - c*c
3	a*(d - c*c)
4	a*(d - c*c) + b

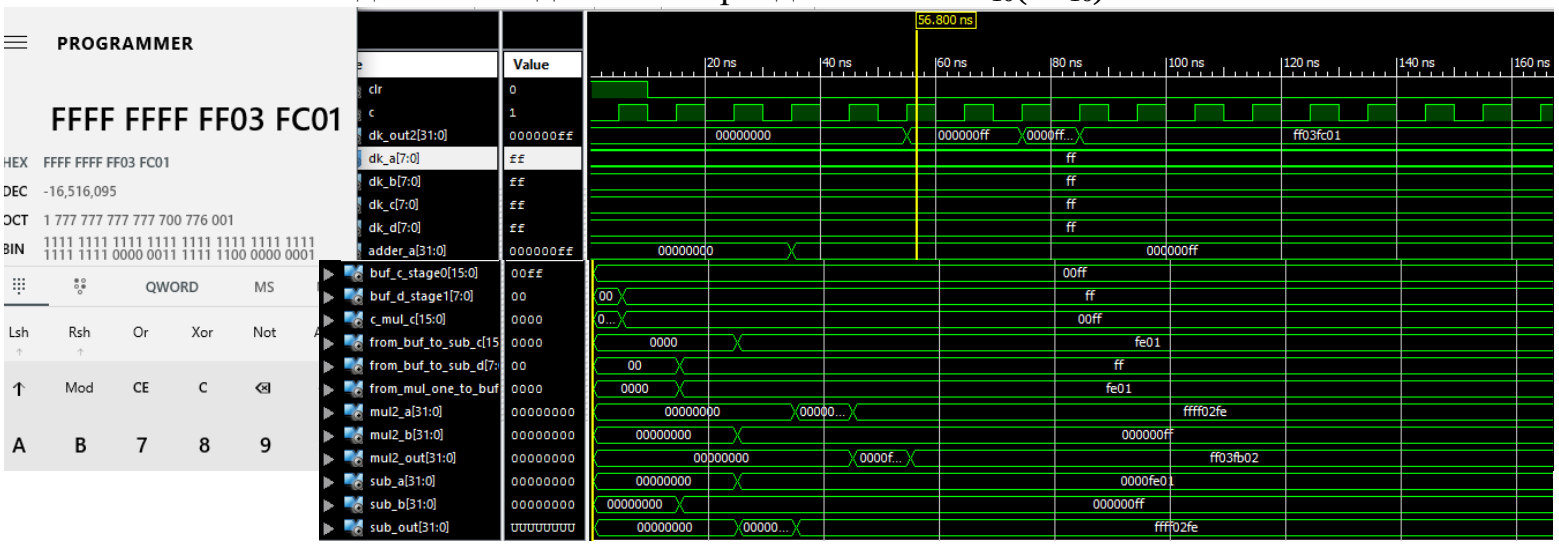
a	2	3	4	5	6
b	1	2	3	4	5
c	4	5	6	7	8
d	25	26	27	28	29

c*c	4	25	36	49	64
d - c*c	9	1	-9	-21	-35
a*(d - c*c)	18	3	-36	-105	-210
result	19	5	-33	-101	-205

Вивід результатів відбувається на **9** такті CLK(оскільки операнди в сумі проходять 9 регістрів)



Перевіримо чи наша схема здатна виводити максимальне число:
Подамо на вхід всіх 4 операндів число $255_{10}(FF_{16})$



12.Діаграми для неконвеєрного способу:

Таблиця прошиття для неконвеєрного способу:

bits	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
iteration	MUX1	MUX2	MUX3	MUX4	o1	o2	o3	o4	ADDR2				ADDR3				ADDR4				ce	ld	clr	pl/min
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0
1	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0
2	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0
3	0	1	0	0	0	1	1	1	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	6
4	0	0	1	0	0	1	0	1	0	0	0	1	0	0	0	1	0	0	0	0	1	1	0	2
5	1	0	0	0	0	1	1	0	0	0	0	1	0	0	0	1	0	0	0	0	1	0	0	0
6	1	0	0	1	1	0	1	0	0	0	0	1	0	0	0	1	0	0	0	1	1	1	0	1
7	1	0	0	1	1	0	1	0	0	0	0	1	0	0	0	1	0	0	0	1	1	0	1	0
8	1	0	0	0	1	1	1	1	0	0	0	1	0	0	0	1	0	0	0	1	1	0	0	1
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	0	1
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	0	1

0,1 ітерація: ініціалізація змінних. Запис в RAM1,2,3,4(b,c,d через MUX). Подача на вихід RAM3

2,3 ітерація:. C*C. З виходу RAM3 подати на перемножувач. Результат в RAM2(адреса1) (через MUX)

4,5 ітерація:. d – C*C. З виходу RAM2 та RAM4 результат подати на вхід суматора. Результат в RAM3 (через MUX)

6,7 ітерація:. a*(d-c*c). З виходу RAM3 та RAM2 подати на суматор. Результат на вхід RAM4 (через MUX)

8-10. ітерація: З виходу RAM4 та RAM2 подати на суматор. Вивід результату.

Приклади виконання:

a = 2 b = 1 c = 4 d = 25

iteration	MUX1_OUT	MUX2_OUT	MUX3_OUT	MUX4_OUT	RAM1_OUT	RAM2_OUT	RAM3_OUT	RAM4_OUT	Multiplier	ADD/SUB
0	0	1	4	25	0	0	0	0	0	0
1	4						4			
2		16							16	
3						16		25		
4			9							9
5							9			
6	2				2					
7				18					18	
8					1			18		
9										19
10										19(RES_SIG)

a = 3 b = 2 c = 5 d = 26

iteration	MUX1_OUT	MUX2_OUT	MUX3_OUT	MUX4_OUT	RAM1_OUT	RAM2_OUT	RAM3_OUT	RAM4_OUT	Multiplier	ADD/SUB
0	0	2	5	26	0	0	0	0	0	0
1	5						5			
2		25							25	
3						25		26		
4			1							1
5							1			
6	3				3					
7				3					3	
8					2			3		
9										5
10										5(RES_SIG)

a = 4 b = 3 c = 6 d = 27

iteration	MUX1_OUT	MUX2_OUT	MUX3_OUT	MUX4_OUT	RAM1_OUT	RAM2_OUT	RAM3_OUT	RAM4_OUT	Multiplier	ADD/SUB
0	0	3	6	27	0	0	0	0	0	0
1	6						6			
2		36							36	
3						36		27		
4			minus 9							minus 9
5			4				minus 9			
6	4				4					
7				minus 36					minus 36	
8					3			minus 36		
9										minus 33
10										min33(RES_SIG)

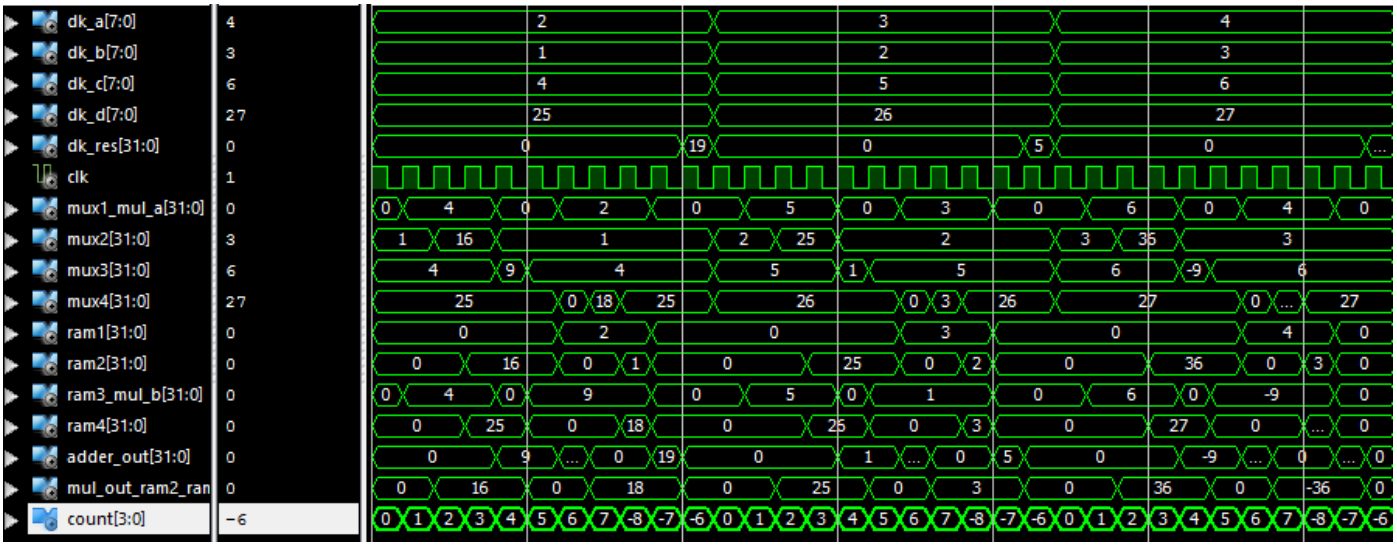
2*(25 – 4*4) + 1 = 2* (25 -16) + 1 = 2 * (9) + 1 = 18 + 1 = **19**

3*(26 – 5*5) + 2 = 3 * (26 - 25) + 2 = 3 * (1) + 2 = = **5**

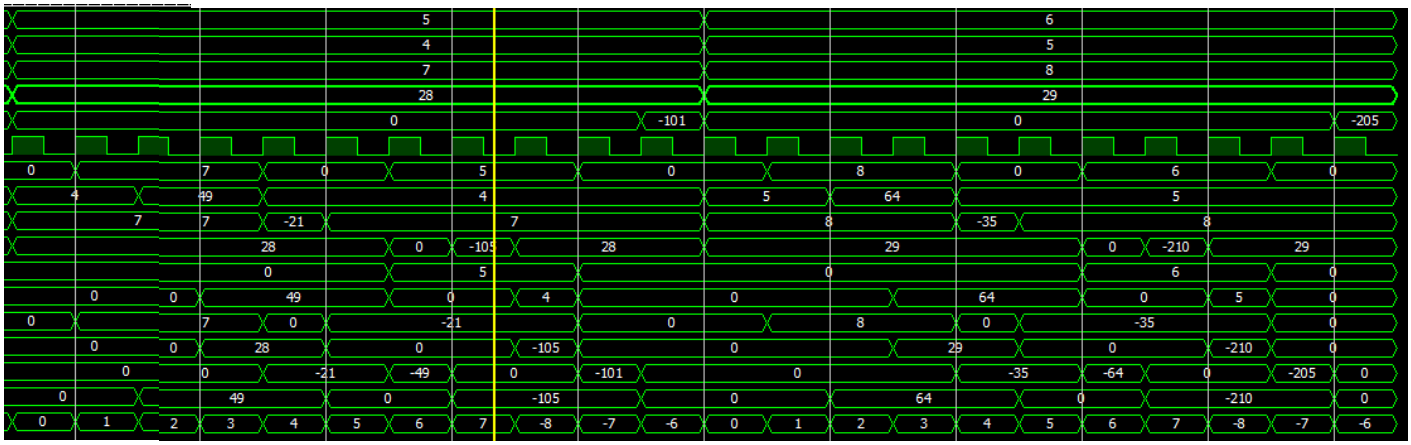
4* (27 – 6*6) + 3 = 4 * (27 - 36) + 3 = 4* (-9) + 3 = -36 + 3 = **-33**

Діаграми:

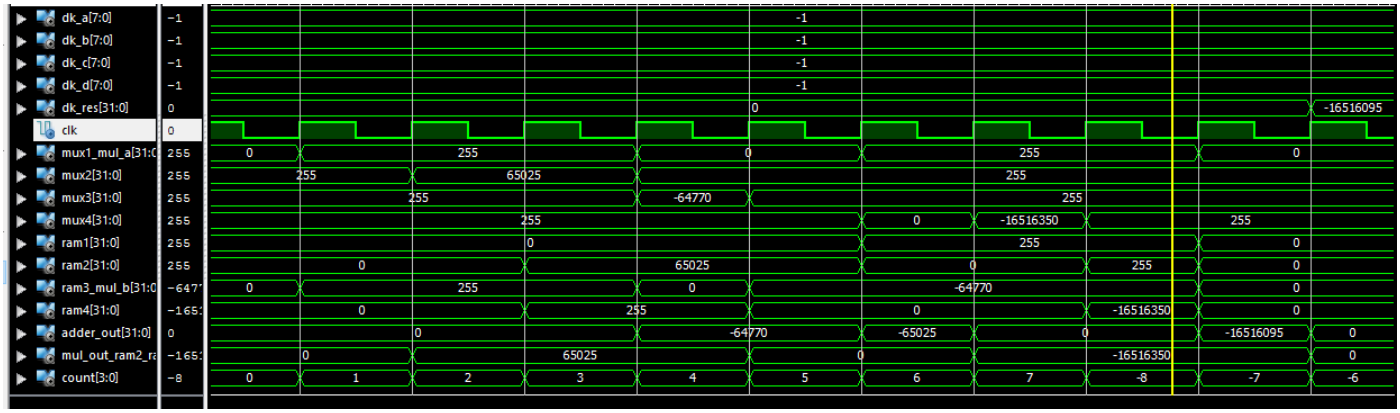
3 вхідні набори



2 вхідні набори(продовження попередньої схеми)



Максимально можливе число:



255* (255 – 255*255) + 255 = - 16 516 095 = FF03 FC01

13. Висновок:

виконуючи дану лабораторну роботу я спроектував роботу спеціалізованого конвеєрного процесора (конвеєрним та ітераційним способом) під задану функцію, шляхом її розбиття на дрібніші кроки.

Як виявилось ітераційний метод реалізувати було складніше за конвеєрний і він не приніс очікуваного результату. До того ж він потребує більше апаратних затрат ніж конвеєрний метод. Результати вийшли коректними і дані схеми можна використовувати ждля подальших досліджень.