МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»
КАФЕДРА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

Лабораторна робота №6

з дисципліни «Технології проектування
комп'ютерних систем»
на тему: «Арифметичний пристрій»

Виконав:
студент 4-го курсу
факультету ІОТ
групи ІО-41
Демчик В. В.
НЗК 4111

Перевірив:
проф. Сергієнко А. М.

Київ 2017

Тема: Арифметичний пристрій.

Мета та основні завдання роботи: оволодіти знаннями і практичними навичками з проектування обчислювальних блоків, таких як арифметичний пристрій (AU). Лабораторна робота також служить для оволодіння навичками програмування і налагодження опису AU на мові VHDL.

Завдання на лабораторну роботу: розробити арифметичний пристрій за наведеними нижче умовами:

Забезпечити виконання наступних операцій на AU:

ADD
SUB
AND
NOT
ABS
SRL
MUL

До складу AU включити наступні основні частини:

Складова частина:	Характеристики:
Блок пам’яті	Тип – регістровий, FM. Розрядність слів – 16 біт. Кількість слів – 64. Кількість каналів – 3. Заборонено запис в перший регістр.
LSM	Операції: ADD, SUB, AND. Розрядність операндів – 16.
MPU	Операція: MUL. Розрядність операндів – 16. Розрядність результату – 32. Розрядність вихідної шини – 16.

Виконати описання поведінкової моделі. Забезпечити контроль бітів переповнення в результаті, знаку результату, ознаки нульового результату. Провести аналіз отриманих графіків роботи схем.

Хід проектування:

Виконаємо опис та кодування операцій:

Операція	**Мнемоніка**	**Код операції АСОР**	**Код LSM**	**Біт переповнення**
Сума	ADD	000	00	+
Різниця	SUB	011	11	+
Логічне «І»	AND	001	01	-
Логічне «АБО»	OR	010	10	-
Заперечення	NOT	100	-	-
Модуль	ABS	101	-	-
Зсув вправо логічний	SRL	110	-	-
Множення	MUL	111	-	-

Операція «АБО» була додана для повноти використання всіх розрядів трьохбітового кодування.

Код програми:

Модифікований блок регістрової пам'яті:

```
Library IEEE;
use IEEE.NUMERIC_BIT.all;

entity FM is port(C: in BIT; -- synchro
 WR: in BIT; -- write
 INCQ: in BIT;-- inc Q address
 CALL: in BIT;-- write return address and CNZ
 AB: in BIT_VECTOR(5 downto 0);-- address B
 AD: in BIT_VECTOR(5 downto 0);-- address D
 AQ: in BIT_VECTOR(5 downto 0);-- address Q
 ARETC: in BIT_VECTOR (15 downto 0);--in address CNZ
 Q: in BIT_VECTOR (15 downto 0);-- data Q
 B: out BIT_VECTOR(15 downto 0);-- data B
 D: out BIT_VECTOR(15 downto 0);-- data D
 ARETCO: out BIT_VECTOR (15 downto 0));--out address CNZ
end FM;

architecture BEH of FM is
type MEM64X16 is array(0 to 63) of BIT_VECTOR(15 downto 0);
constant FM_init: MEM64X16:= -- initial memory status
 (X"0000",X"0000",X"0000",X"0000",X"0000",X"0000",X"0000",X"0000",others=> X"0000");
signal RAM: MEM64x16:=FM_init;
```

```
begin
 FM16:process(C,AD,AB)
 variable addrq,addrd,addrb:natural;
begin
 addrq:=TO_INTEGER(UNSIGNED(AQ));
 if INCQ='1' then
          addrq:=addrq+1;
 end if;

 addrd:=TO_INTEGER(UNSIGNED(AD));
 addrb:=TO_INTEGER(UNSIGNED(AB));

 if C='1' and C'event then
          if WR = '1' and (addrq /= 0) then
                   RAM(addrq)<= Q;
          end if;
          if CALL = '1' then
                   RAM(63)<= ARETC;
          end if;
 end if;

 B<= RAM(addrb);
 D<= RAM(addrd);
 ARETCO<= RAM(63);

end process;
end BEH;
```

Модифікований блок LSM:

```
library ieee;
use ieee.numeric_bit.all;
use CNetwork.all;

entity LSM is
port(F : in BIT_VECTOR(1 downto 0);-- function
          A : in BIT_VECTOR(15 downto 0);-- first operand
          B : in BIT_VECTOR(15 downto 0);-- second operand
          Y : out BIT_VECTOR(15 downto 0);-- result
          C15: out BIT; -- transfer output
          Z: out BIT; -- bit of zero result
          N: out BIT);-- sign bit
end LSM;

architecture BEH of LSM is

signal atcc, btcc, ytcc  : BIT_VECTOR(15 downto 0);
signal uatcc,ubtcc,uytcc : BIT_VECTOR(14 downto 0);
signal ai, bi, yi: integer;
signal sb: BIT;
signal usa,usb : unsigned(15 downto 0);
signal uy: unsigned (16 downto 0);
signal yb: BIT_VECTOR (15 downto 0);

begin
--for SUB--
SUB:sb<=not B(15) when F="11" else B(15) when F="00";

--two's complement code of operands--
```

```
INV_A: uatcc(14 downto 0)<=not A(14 downto 0) when A(15)='1' else A(14 downto 0) when A(15)='0';
INV_B: ubtcc(14 downto 0)<=not B(14 downto 0) when sb='1' else B(14 downto 0) when sb='0';
INC_A: atcc(14 downto 0)<=INT_TO_BIT(BIT_TO_INT(uatcc(14 downto 0))+1 , 15) when A(15)='1' else
uatcc(14 downto 0) when A(15)='0';
INC_B: btcc(14 downto 0)<=INT_TO_BIT(BIT_TO_INT(ubtcc(14 downto 0))+1 , 15) when sb='1' else
ubtcc(14 downto 0) when sb='0';
SIGN_A:         atcc(15)<=A(15);
SIGN_B:         btcc(15)<=sb;

-- Adder --
ADDER: uy(16 downto 0) <= RESIZE(unsigned(atcc(15 downto 0)),17) + RESIZE(unsigned(btcc(15 downto
0)),17) when (F = "00" or F = "11");

-- Result correction --
INV_Y:uytcc(14 downto 0) <= not BIT_VECTOR(uy(14 downto 0)) when uy(15)='1' else
BIT_VECTOR(uy(14 downto 0));
INC_Y:yb(14 downto 0) <= INT_TO_BIT(BIT_TO_INT(uytcc(14 downto 0))+1 , 15) when uy(15)='1' else
uytcc(14 downto 0);
SIGN_Y:yb(15)<=BIT(uy(15));

-- Results multiplexer --
MUX: with F select
Y <= yb(15 downto 0) when "00"|"11", -- adder
        (A and B) when "01", --AND
         (A or B) when "10"; --OR

 C15 <= not BIT(uy(16)) when BIT(uy(15))='0' else BIT(uy(16)); -- output transfer
   N <= yb(15); --sign bit
ZERO_DEF: Z <= '1' when yb (15 downto 0) = X"0000" else '0'; -- zero sign
end BEH;
```

MPU:

```
library IEEE;
use IEEE.Numeric_Bit.all;
use CNetwork.all;

entity OB is
port(C : in BIT; --synchro
 RST : in BIT; --reset
 LAB : in BIT; -- load A,B, reset P
 SHIFT : in BIT; --shift A and B flag
 OUTHL : in BIT; --get first(1) or last(0) result word
 DA : in BIT_VECTOR(15 downto 0); --A bus
 DB : in BIT_VECTOR(15 downto 0); --B bus
 ADD : in BIT; --adder start flag
 REZ : in BIT; --correction flag
 B0 : out BIT; --first bit B
 STOP : out BIT; --stop flag
 Z: out BIT; -- result zero flag
 N: out BIT; -- result sign
 DP : out BIT_VECTOR(15 downto 0)); -- result bus
end OB;

architecture BEH of OB is
signal A:bit_vector(29 downto 0); -- register A data
signal SA:bit; --A sign bit
signal B:bit_vector(14 downto 0); -- register B data
signal SB:bit; --B sign bit
```

```
signal S: unsigned(29 downto 0); -- adder buffer
signal P: unsigned(31 downto 0); -- register P data
begin
 -- register A
 RG_A:process(C,RST)
 variable high,low:natural;
begin
 if RST='1' then
        A<="00000000000000000000000000000000"; --reset all 30 bits
 elsif C='1' and C'event then
        if LAB='1' then
                A(29 downto 15)<="000000000000000"; --reset 15 last bits
                SA<=DA(15); --load A sign bit
                A(14 downto 0)<=DA(14 downto 0); -- load A data bits
                high:=14; --start values of pointers
                low:=0;
        elsif SHIFT='1' then
                if high/=29 then
                        --A(29 downto 0)<=A(29 downto 1)&'0'; --left shift
                        A(high+1 downto low+1)<=A(high downto low); --left shift
                        A(low)<='0';
                        high:=high+1;
                        low:=low+1;
                end if;
        end if;
 end if;
end process;

 -- register B
 RG_B:process(C,RST)
begin
 if RST='1' then
        B<="000000000000000";          --reset   register
        STOP<='0'; --reset checks bits
        B0<='0';
 elsif C='1' and C'event then
        if LAB='1' then
                STOP<='0';       --reset checks bits
                B0<='0';
           if DB(0)='1' then B0<='1'; else B0<='0'; end if; --check first bit of B
                if BIT_TO_INT(DB(14 downto 0))=0 then STOP<='1'; else STOP<='0'; end if; --if B=0 then
stop
                SB<=DB(15); --load B sign bit
                B(14 downto 0)<=DB(14 downto 0); --load B data bits
        elsif SHIFT='1' then
                if B(1)='1' then B0<='1'; else B0<='0'; end if;
                if BIT_TO_INT('0'&B(14 downto 1))=0 then STOP<='1'; else STOP<='0'; end if;
                B<='0'& B(14 downto 1); --right shift
    end if;
 end if;
end process;

 -- Adder
 SM:S(29 downto 0) <= unsigned(P(29 downto 0))+unsigned(A) when ADD='1' else P(29 downto 0);

 -- register P (result)
 RG_P:process(C,RST,P)
 variable zi,ni:bit;
begin
```

```
 if RST='1' then
        P<=X"00000000"; --reset
 elsif C='1' and C'event then
        if LAB='1' then
                P<=X"00000000"; --reload
        elsif REZ='1' then
                ni:=SA xor SB;
                P<=ni&"00"&P(29 downto 1); --result correction
                N<=ni;
        else
                P(29 downto 0)<=S(29 downto 0);
        end if;
 end if;
 zi:='0'; --zero result check
 for i in P'range loop
        zi:=zi or P(i);
 end loop;
 Z<= not zi; -- zero result flag
end process;

 --output multiplexor
 MUX_P:DP<= bit_vector(P(15 downto 0)) when OUTHL='1' else bit_vector(P(31 downto 16));
end BEH;

entity FSM is
port(
 C : in BIT; --synchro
 RST : in BIT; --reset FSM
 START : in BIT; --start FSM
 B0 : in BIT;
 STOP: in BIT; -- check '1' bits of B
 LAB : out BIT; -- load operands
 SHIFT : out BIT; --shift A and B
 ADD: out BIT; -- add
 REZ: out BIT;
 RDY : out BIT); --finish
end FSM;

architecture BEH of FSM is
type STATES is (ststart,st1,st2,st3,st4,stfinish); --machine states
signal st:STATES; --current state
begin
 STATE:process(C,RST) -- state register
begin
 if RST='1' then
 st<=ststart;
 elsif C='1' and C'event then
        if st=ststart and START='1' then
                st<=st1; --from start to load-operands-state
        elsif st=st1 and STOP='1' then
                st<=stfinish; --if B=0 then finish without other states
        elsif st=st1 and STOP='0' and B0='1' then
                st<=st2; --if B(0)=1 then ADD
        elsif st=st1 and STOP='0' and B0='0' then
                st<=st3; --if B(0)=0 then SHIFT
    elsif st=st2 then
                st<=st3; --after ADD always SHIFT
        elsif st=st3 and STOP='0' and B0='0' then
                st<=st3; --if after SHIFT B(0)='0' then one more SHIFT
```

```
            elsif st=st3 and STOP='0' and B0='1' then
                  st<=st2; --if after SHIFT B(0)='1' then ADD
            elsif st=st3 and STOP='1' then
                  st<=st4; --if after SHIFT B=0 then RSH result
            elsif st=st4 then
                  st<=stfinish; --always go to finish after RSH
            elsif st=stfinish and START='1' then
                  st<=ststart;
            end if;
 end if;
end process;
-- output signals logic
 LAB<='1' when st=st1 else '0';
 ADD<='1' when st=st2 else '0';
 SHIFT<='1' when st=st3 else '0';
 REZ<='1' when st=st4 else '0';
 RDY<='1' when st=stfinish else '0';
end BEH;

entity MPU is
port(C : in BIT;
 RST : in BIT;
 START:in BIT;
 OUTHL:in BIT;
 DA : in BIT_VECTOR(15 downto 0);
 DB : in BIT_VECTOR(15 downto 0);
 RDY : out BIT;
 Z: out BIT;
 N: out BIT;
 DP : out BIT_VECTOR(15 downto 0) );
end MPU;

architecture BEH of MPU is
component OB is port(
 C : in BIT; --synchro
 RST : in BIT; --reset
 LAB : in BIT; -- load A,B, reset P
 SHIFT : in BIT; --shift A and B flag
 OUTHL : in BIT; --get first(1) or last(0) result word
 DA : in BIT_VECTOR(15 downto 0); --A bus
 DB : in BIT_VECTOR(15 downto 0); --B bus
 ADD : in BIT; --adder start flag
 REZ : in BIT; --correction flag
 B0 : out BIT; --first bit B
 STOP : out BIT; --stop flag
 Z: out BIT; -- result zero flag
 N: out BIT; -- result sign
 DP : out BIT_VECTOR(15 downto 0)); -- result bus
end component;

component FSM is port(
 C : in BIT; --synchro
 RST : in BIT; --reset FSM
 START : in BIT; --start FSM
 B0 : in BIT;  --first bit B flag
 STOP: in BIT; -- check '1' bits of B
 LAB : out BIT; -- load operands
 SHIFT : out BIT; --shift A and B
 ADD: out BIT; -- add
```

```
 REZ: out BIT; --correction flag
 RDY : out BIT); --finish
end component ;

signal lab,shift,add,b0,stop,rez:bit;
begin
 --operation block
 U_OP:OB port map(C,RST,
 LAB=>lab, SHIFT=>shift,
 ADD=>add,B0=>b0,OUTHL=>OUTHL,
 DA=>DA, DB=>DB, STOP=>stop, REZ=>rez,
 Z=>Z,N=>N,DP=>DP);

 --final state machine
 U_FSM:FSM port map(C,RST,
 START=>start, B0=>b0, LAB=>lab, STOP=>stop, REZ=>rez,
 ADD=>add, SHIFT=>shift, RDY=>RDY);
end BEH;
```

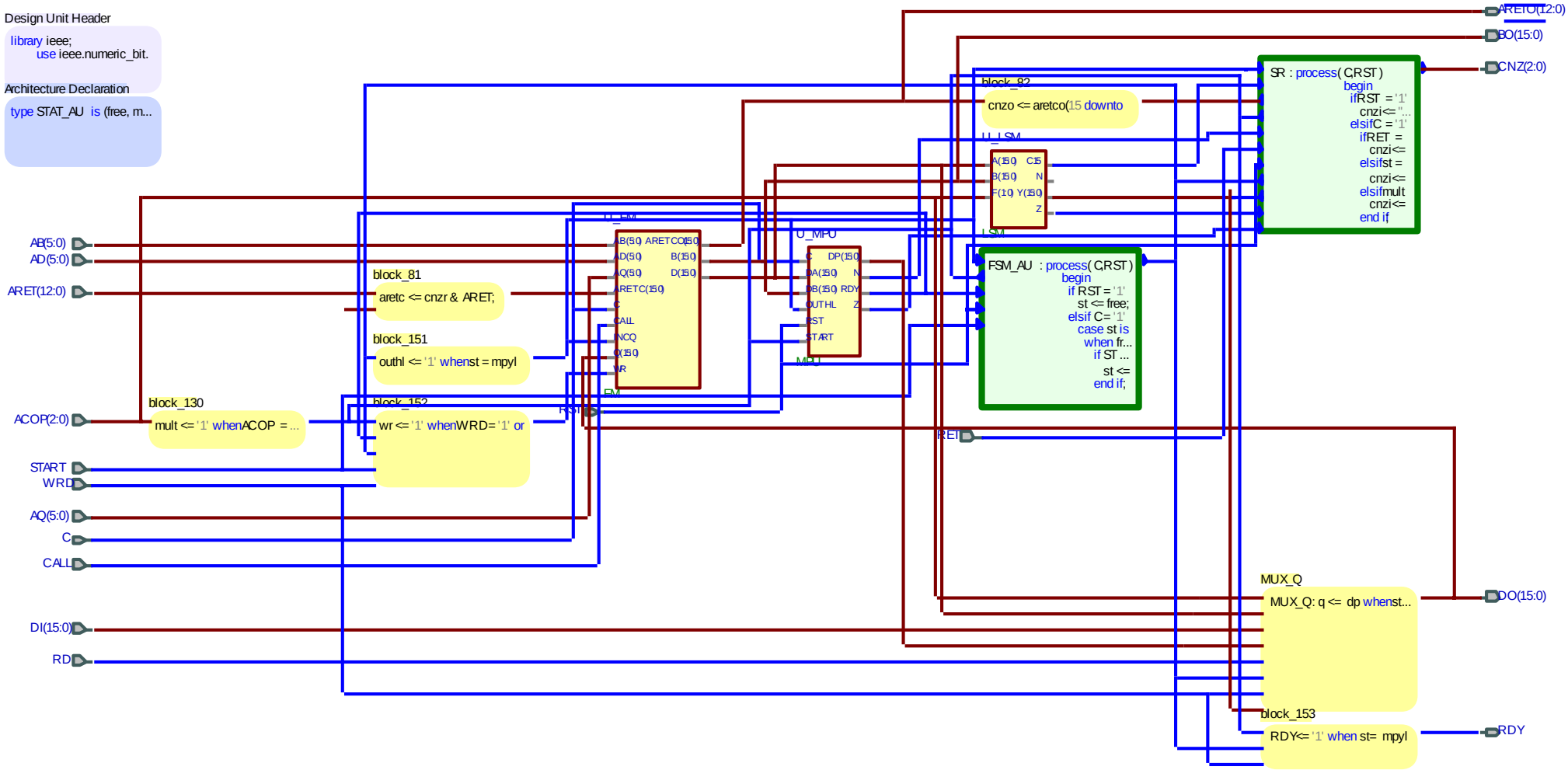
AU:

```
Library IEEE;
use IEEE.NUMERIC_BIT.all;

entity AU is port(
 C : in BIT; --sycnhro
 RST : in BIT; --reset
 START : in BIT; --start operation in AU
 RD: in BIT; -- read from FM to DO bus
 WRD : in BIT; -- write from DI bus
 RET : in BIT; -- return from subprogram
 CALL: in BIT; -- start subprogram
 DI : in BIT_VECTOR(15 downto 0); --in data bus
 AB : in BIT_VECTOR(5 downto 0); -- register B address
 AD : in BIT_VECTOR(5 downto 0); -- register D address
 AQ : in BIT_VECTOR(5 downto 0); -- register Q address
 ARET : in BIT_VECTOR(12 downto 0); --in return address
 ACOP : in BIT_VECTOR(2 downto 0); -- AU operation code
 RDY : out BIT; --result ready
 ARETO : out BIT_VECTOR(12 downto 0);--out return address
 DO : out BIT_VECTOR(15 downto 0); --out data bus
 BO : out BIT_VECTOR(15 downto 0); --out data bus D
 CNZ: out BIT_VECTOR(2 downto 0)); --out state reg
end AU;

architecture BEH of AU is

component FM is port(
 C: in BIT; -- synchro
 WR: in BIT; -- write
 INCQ: in BIT;-- inc Q address
 CALL: in BIT;-- write return address and CNZ
 AB: in BIT_VECTOR(5 downto 0);-- address B
 AD: in BIT_VECTOR(5 downto 0);-- address D
 AQ: in BIT_VECTOR(5 downto 0);-- address Q
 ARETC: in BIT_VECTOR (15 downto 0);--????? ???????? ? CNZ
 Q: in BIT_VECTOR (15 downto 0);-- ?????? ?????? Q
 B: out BIT_VECTOR(15 downto 0);-- ?????? ?????? ?
 D: out BIT_VECTOR(15 downto 0);-- ?????? ?????? D
 ARETCO: out BIT_VECTOR (15 downto 0));-- ????? ???????? ? CNZ
```

```
end component ;

component LSM is port(
 F : in BIT_VECTOR(1 downto 0);-- function
          A : in BIT_VECTOR(15 downto 0);-- first operand
          B : in BIT_VECTOR(15 downto 0);-- second operand
          Y : out BIT_VECTOR(15 downto 0);-- result
          C15: out BIT; -- transfer output
          Z: out BIT; -- bit of zero result
          N: out BIT);-- sign bit
end component ;

component MPU is port(
 C : in BIT;
 RST : in BIT;
 START:in BIT;
 OUTHL:in BIT;
 DA : in BIT_VECTOR(15 downto 0);
 DB : in BIT_VECTOR(15 downto 0);
 RDY : out BIT;
 Z: out BIT;
 N: out BIT;
 DP : out BIT_VECTOR(15 downto 0) );
end component ;

type STAT_AU is (free,mpy,mpyl);-- fsm states
signal st:STAT_AU;
signal b,q,d,y,dp,aretc,aretco:BIT_VECTOR(15 downto 0);
signal c0,c15,csh,zlsm,wr,mult,outhl:BIT;
signal rdym,zmpy,nmpy:BIT;
signal cnzr,cnzo,cnzi:BIT_VECTOR(2 downto 0);

begin
---------FM unit---------
 U_FM: FM port map(C,
 WR=>wr, INCQ=>outhl, CALL=>CALL,
 AB=>AB, AD=>AD, AQ=>AQ,
 ARETC=>aretc,
 Q=>q, B=>b, D=>d,
 ARETCO=>aretco);

 aretc<=cnzr&ARET;
 cnzo<=aretco(15 downto 13); --read control bits (C, N, Z) from high bits of last register of FM
 ARETO<=aretco(12 downto 0); --read return address from low bits of register of FM

 --MUX_C:c0<='1' when ACOP(1 downto 0)="10" else --ióëüòèïëåêñîð Ñ0
 --   cnzi(2) when ACOP(1 downto 0)="01" else '0';

----------LSM unit----------
 U_LSM:LSM port map(F=>ACOP(1 downto 0),
 A=>d,B=>b,Y =>y,
 C15=>c15, Z =>zlsm );

----------MPU----------
 U_MPU:MPU port map(C,RST,
 START=>mult, OUTHL=>outhl,
 DA=>d, DB=>b,
 RDY=>rdym,Z=>zmpy,
 N=>nmpy, DP=>dp);
```

```
 ---------MUL multiplexor---------
 --MUX_CI:csh<=cnzi(2) when ACOP(1 downto 0)="01" else --çàääâèã. ðàçð.
 --           cnzi(1) when ACOP(1 downto 0)="11" else '0';

 ----- result multiplexor ---------
 MUX_Q:      q<=dp when st/=free else --multiply result
 '0'&d(15 downto 1) when ACOP="110" else --logical right shift
 '0'&d(14 downto 0) when ACOP="101" else --abs
 not d(15 downto 0) when ACOP="100" else --not
               DI when WRD='1' else --result subprogram
                d when RD='1' else --data from AD in AD address
                    y; --LSM result

-------state refister with multiplexor--------------
 SR:process(C,RST)
 begin
          if RST='1' then
                  cnzi<="000";
          elsif C='1' and C'event then
                  if RET='1' then
                          cnzi<=cnzo;
                  elsif st=mpyl then
                          cnzi<='0'&nmpy&zmpy;
                  elsif mult='0' then
                          cnzi<=c15&y(15)&zlsm;

                  end if;
          end if;
 end process;

 mult<='1' when ACOP="111" else '0';--multiply decoder

---------AU FSM-----------
 FSM_AU:process(C,RST)
 begin
          if RST='1' then
                  st<=free; -- fsm state register
          elsif C='1' and C'event then
                  case st is
                          when free => if START='1'and mult='1'then --not use
                                  st<=mpy;
                                  end if;
                          when mpy=> if rdym='1' then -- there is a multiplication
                                  st<=mpyl ;
                                  end if;
                          when mpyl=> st<=free; --multiplication finish
                  end case;
          end if;
 end process;

--fsm outs
 outhl<='1' when st=mpyl else '0';
 wr<='1' when WRD='1' or st=mpyl or (st=mpy and rdym='1') or (START='1' and mult='0') else '0';
 RDY<='1'when st=mpyl or (WRD='0' and st/=mpy and mult='0') else'0';
 DO<=q; --output data
 BO<=B;
 CNZ<=cnzi; --state reg out
end BEH;
```

Згенерована схема на основі описаної архітектури:



Як бачимо, отримана схема відповідає класичній схемі арифметичного блоку.

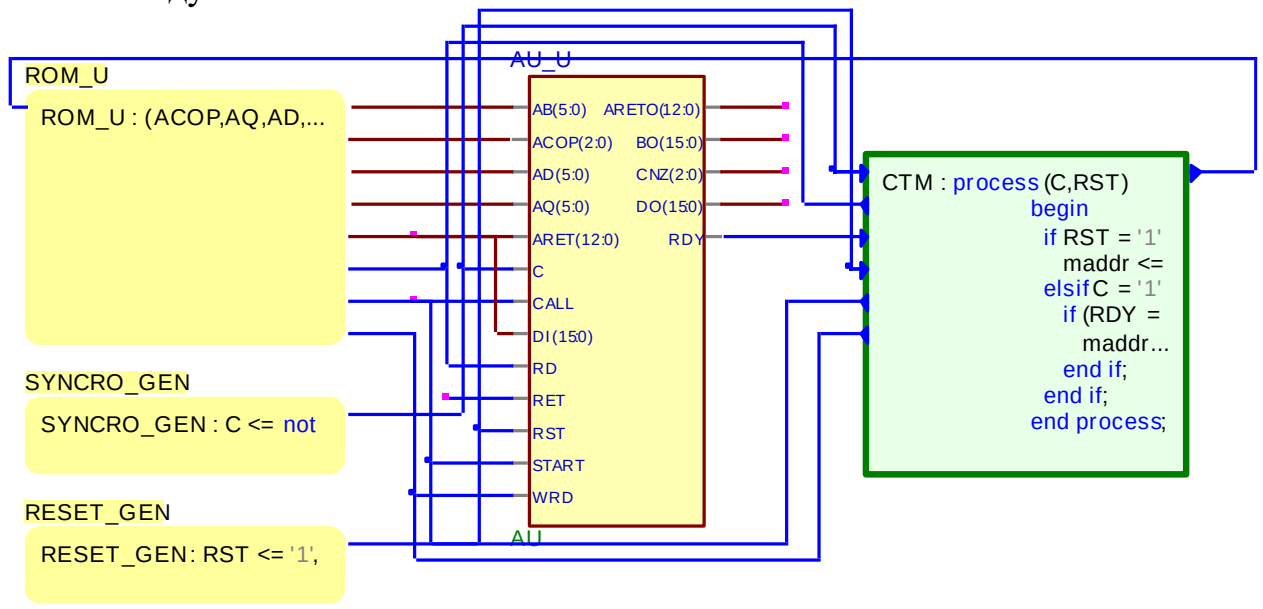
Результати тестування:

Для тестування було розроблено окремий стенд, на якому розміщено арифметичний блок, блок ROM із завчасно завантаженою в нього тестовою мікропрограмою, простий регістр-лічильний адреси по модулю n, та генератори синхросерії та сигналів скидання.

```
Library IEEE;
use IEEE.NUMERIC_BIT.all;

entity au_tb is
end au_tb;

architecture TB_ARCHITECTURE of au_tb is

component AU is port(
 C : in BIT; --sycnhro
 RST : in BIT; --reset
 START : in BIT; --start operation in AU
 RD: in BIT; -- read from FM to DO bus
 WRD : in BIT; -- write from DI bus
 RET : in BIT; -- return from subprogram
 CALL: in BIT; -- start subprogram
 DI : in BIT_VECTOR(15 downto 0); --in data bus
 AB : in BIT_VECTOR(5 downto 0); -- register B address
 AD : in BIT_VECTOR(5 downto 0); -- register D address
 AQ : in BIT_VECTOR(5 downto 0); -- register Q address
 ARET : in BIT_VECTOR(12 downto 0); --in return address
 ACOP : in BIT_VECTOR(2 downto 0); -- AU operation code
 RDY : out BIT; --result ready
 ARETO : out BIT_VECTOR(12 downto 0);--out return address
 DO : out BIT_VECTOR(15 downto 0); --out data bus
 BO : out BIT_VECTOR(15 downto 0); --out data bus D
 CNZ: out BIT_VECTOR(2 downto 0)); --out state reg
end component ;

signal c,rst,rdy,start,wrd,rd,ret,call:BIT;
signal acop,cnz:BIT_VECTOR(2 downto 0);
signal aq,ad,ab:BIT_VECTOR(5 downto 0);
signal di,do,bo:BIT_VECTOR(15 downto 0);
signal aret,areto:BIT_VECTOR(12 downto 0);


signal maddr:natural;
type MICROINST is record -- ?????? ???????????
 ACOP:bit_vector(2 downto 0); -- ??? ???????? AU
 AQ,AD,AB:bit_vector(5 downto 0); -- ?????? FM
 DI:BIT_VECTOR(15 downto 0); -- ??????? ??????
 START,WRD,RD:bit; -- ???? ??????????
end record;
constant n: positive:=13; --????? ???????????
type MICROPROGR is array(0 to n-1) of MICROINST;
----------microprogram----------
constant mp:MICROPROGR:=(

 ("100","000001","000000","000000",X"0000",'0','1','0'), --NOT 0000h, result = FFFFh in 000001
 ("101","000010","000001","000000",X"0000",'0','1','0'), --ABS FFFFh, result = 7FFFh in 000010
 ("110","000011","000010","000000",X"0000",'0','1','0'), --LSR 7FFFh, result = 3FFFh in 000011
 ("110","000011","000011","000000",X"0000",'0','1','0'), --LSR 3FFFh, result = 1FFFh in 000011
```

```
 ("001","000100","000011","000001",X"0000",'1','0','0'), --AND 3FFFh, 1FFFh, result = 1FFFh in 000100
 ("011","000101","000010","000100",X"0000",'1','0','0'), --SUB 7FFFh, 1FFFh, result = 6000h in 000101
 ("110","000110","000101","000000",X"0000",'0','1','0'), --LSR 6000h, result = 3000h in 000110
 ("111","001000","000001","000110",X"0000",'1','0','0'), --MUL FFFFh, 3000h, result = 97FFh in 001000 and
D000h in 001001
 ("000","000111","000110","000101",X"0000",'1','0','0'), --ADD 3000h, 6000h, result = 9000h in 000111
 ("111","001010","000010","000110",X"0000",'1','0','0'), --MUL 7FFFh, 3000h, result = 17FFh in 001010 and
D000h in 001011
 ("000","001110","001000","001010",X"0000",'1','0','0'), --ADD 97FFh, 17FFh, result = 0000h in 001110
 ("011","001111","001011","001001",X"0000",'1','0','0'), --SUB D000h, D000h, result = 0000h in 001111
 ("001","000000","001111","001110",X"0000",'1','0','0')  --AND 0000h, 0000h, result = 0000h in out bus
 );

 begin

 SYNCRO_GEN: C<=not C after 5 ns; -- ????????? ?????????????
 RESET_GEN: RST<='1' ,'0' after 25 ns; -- ????????? c?????? ??????

CTM:process(C,RST)
begin -- ??????? ???????????
 if RST='1' then
 maddr<=0;
 elsif C='1' and C'event then
 if (RDY='1' and START='1') or WRD='1'or RD='1' then
 maddr<=(maddr+1) mod n; -- +1 ? ????????
 end if;
 end if;
end process;

ROM_U:(ACOP,AQ,AD,AB,DI,START,WRD,RD)<=mp(maddr);
AU_U : AU port map (C,RST, --??????????? ?U
 START => START,
 RD => RD, WRD => WRD,
 RET => RET, CALL => CALL,
 DI => DI,
 AB => AB, AD => AD, AQ => AQ,
 ARET => ARET, ACOP => ACOP,
 RDY => RDY, ARETO => ARETO,
 DO => DO, BO=>BO, CNZ => CNZ);
end TB_ARCHITECTURE;
```

Схема стенду:



Design Unit Header

```
library ieee;
    use ieee.numeric_bit.all;
```

Architecture Declaration

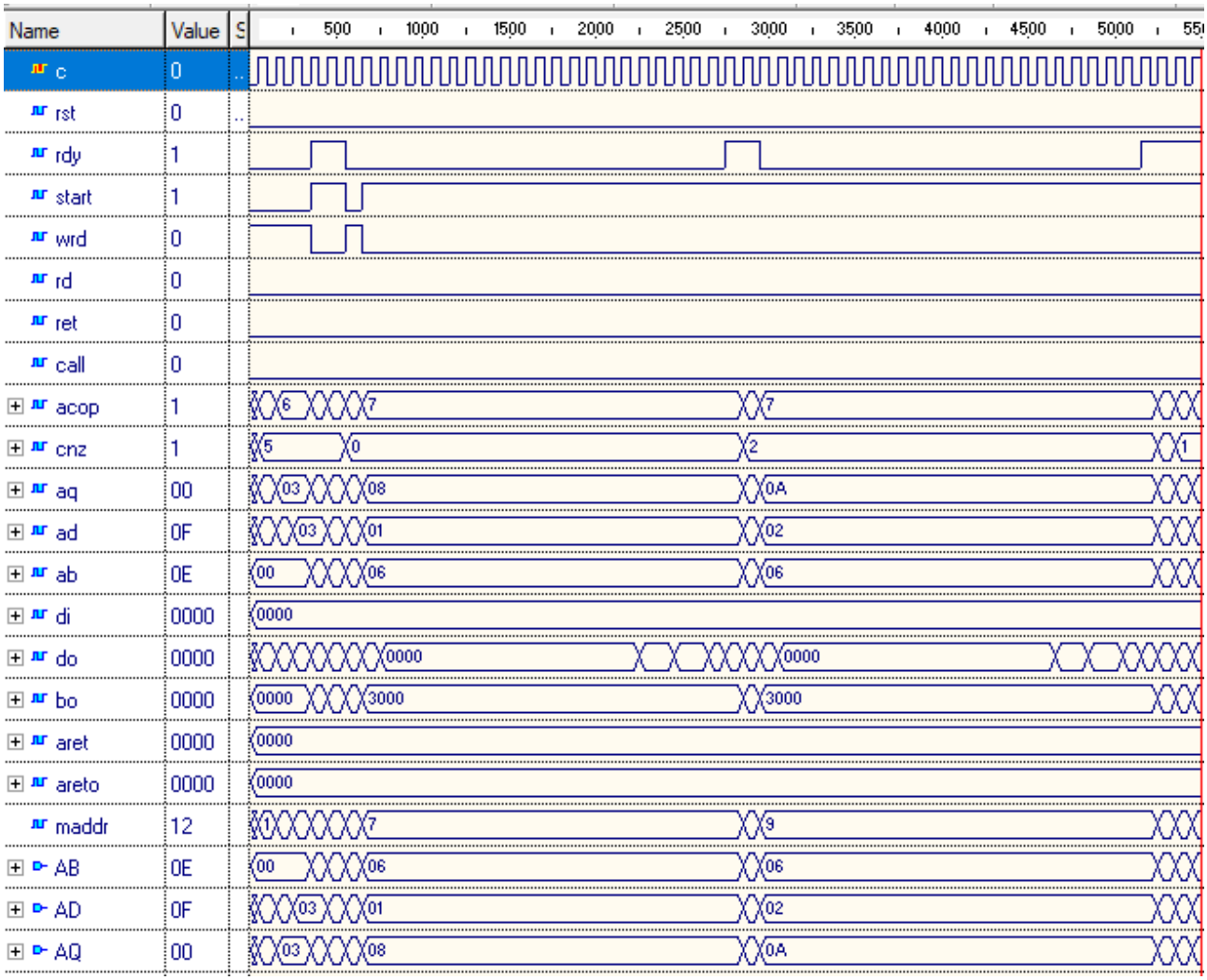
```
constant n : POSITIVE := 13;
constant mp : MICROPROGR := (("100","000001","000000","000000",X"0000",'0','1','0'),("101","000010","00...
type MICROINST is record
  ACOP : BIT_VECTOR(2 downto 0);
  AQ : BIT_VECTOR(5 downto 0);
  AD : BIT_VECTOR(5 downto 0);
  AB : BIT_VECTOR(5 downto 0);
  DI : BIT_VECTOR(15 downto 0);
  START : BIT;
  WRD : BIT;
```

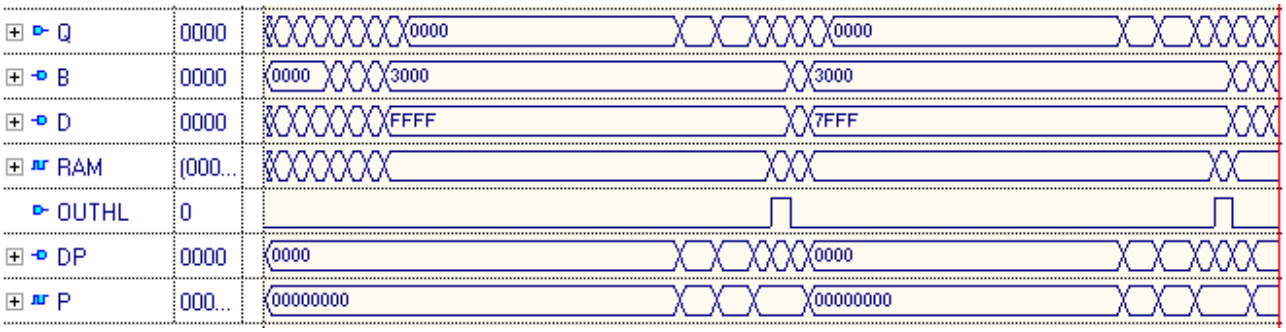
Написана мікропрограма працює за наступним алгоритмом:

1. Зчитати по вхідній шині 0000h, інвертувати, результат (FFFFh) занести в регістр 000001;
2. Знайти модуль числа з регістру 000001 (FFFFh), результат (7FFFh) занести в регістр 000010;
3. Виконати логічний зсув вправо числа в регістрі 000010 (7FFFh), результат (3FFFh) занести в регістр 000011;
4. Виконати логічний зсув вправо числа в регістрі 000011 (3FFFh), результат (1FFFh) залишити в цьому ж регістрі;
5. Виконати логічне «І» над операндами з регістрів 000011 (1FFFh) та 000001 (7FFFh), результат (1FFFh) занести в регістр 000100;
6. Виконати логічне «І» над операндами з регістрів 000011 (1FFFh) та 000001 (7FFFh), результат (1FFFh) занести в регістр 000100;
7. Виконати віднімання над операндами з регістрів 000010 (7FFFh) та 000100 (1FFFh), результат (6000h) занести в регістр 000101;
8. Виконати логічний зсув вправо числа в регістрі 000101 (6000h), результат (3000h) занести в регістр 000110;

9. Виконати множення над операндами з регістрів 000001 (FFFFh) та 000110 (3000h), старші 16 біт результату (97FFh) занести в регістр 001000, молодші 16 біт результату (D000h) занести автоматично в наступний регістр (001001);
10. Виконати додавання над операндами з регістрів 000110 (3000h) та 000101 (6000h), результат (9000h) занести в регістр 000111;
11. Виконати множення над операндами з регістрів 000010 (7FFFh) та 000110 (3000h), старші 16 біт результату (17FFh) занести в регістр 001010, молодші 16 біт результату (D000h) занести автоматично в наступний регістр (001011);
12. Виконати додавання над операндами з регістрів 001000 (97FFh) та 001010 (17FFh), результат (0000h) занести в регістр 001110;
13. Виконати віднімання над операндами з регістрів 001001 (D000h) та 001011 (D000h), результат (0000h) занести в регістр 001111;
14. Виконати логічне «І» над операндами з регістрів 001110 (0000h) та 001111 (0000h), результат (0000h) вивести на вихідну шину.

**Невідповідність даних результатів до результатів, отриманих на звичайних калькуляторах, пояснюється тим, що в нашому випадку старший біт числа є знаковим, а не значимим.*





Як бачимо, після виконання останньої (12) мікрокоманди на вихідній шині маємо очікуваний 0000h. Виконання всієї мікропрограми зайняло 55 тактів.