НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ІНФОРМАТИКИ І ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

КАФЕДРА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

Лабораторна робота №5

з дисципліни **«**Технології проектування комп’ютерних систем**»**

Виконав:

студент групи ІО-53, ФІОТ

Гріщенко Костянтин Сергійович

Перевірив:

Сергієнко Анатолій Михайлович

Київ - 2018 р.

**Тема:** Блок множення.

**Мета та основні завдання** **роботи**: оволодіти знаннями і практичними навичками з проектування обчислювальних блоків послідовної дії, таких як блок множення (MPU), включаючи його керуючий автомат (finite state mashine - FSM). Лабораторна робота також служить для оволодіння навичками програмування та налагодження опису блоків послідовного дії і FSM на мові VHDL.

**Завдання на лабораторну роботу:** розробити блок множення за наведеними нижче умовами:

|  |  |
| --- | --- |
| ***№ завдання*** | ***Функція, яка обчислюється*** |
| 6 | Y=А\*В зі зсувом множника вліво, а множеного вправо |

Розрядність вхідних операндів – 16 біт.

Виконати описання поведінкової операційного блоку, керуючого автомату, моделі, стенду для тестування. Провести аналіз отриманих графіків роботи схем.

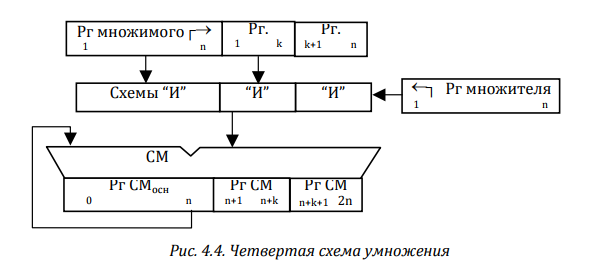
**Хід проектування:**

Оскільки операції знакові, а подання операндів відбувається в прямому коді, то використовуватимемо наступний формат вхідних операндів:

Перші 15 біт – значущі, 16-тий біт – знаковий.

Звідси випливає, що для результату необхідно 30 значущих та 1 знаковий біт. Знаковий біт виводитимемо окремо, а значущі через вихідну шину як 15-бітні слова, за допомогою керуючого біту.

Заданий за варіантом спосіб множення – четвертий. Його структурна схема наступна:



1. ***Проектування операційного блоку:***

Для операційного блоку нам знадобляться:

* 30-бітний регістр для зберігання значущих бітів операнду А, який крім цього має виконувати лівий зсув та запам’ятовування знаку А.
* 15-бітний регістр для зберігання значущих бітів операнду В, який крім цього має виконувати правий зсув, перевірку молодшого біту, перевірку вичерпності, та запам’ятовування знаку А.
* 30-бітний (на виході) суматор (із 15-бітними входами).
* 30-бітний регістр для зберігання проміжних результатів додавання на суматорі та остаточного результату, який крім того має виконувати корекцію\* результату.
* 2-бітна XOR-схема, на якій відбувається визначення знаку результату.
* 30-бітну схему NOR, на якій відбувається перевірка нульового результату.
* Мультиплексор, який залежно від керуючого сигналу забезпечує подачу на вихідну шину 15-ти старших чи 15-молодших значущих бітів результату.

\* - під корекцією результату мається на увазі одноразовий правий зсув, необхідність застосування якого було визначено в процесі відлагодження.

Операційний блок має наступні входи та виходи:

|  |  |  |
| --- | --- | --- |
| **Найменування** | **Тип (in/out)** | **Призначення** |
| ***Виходи назовні*** | | |
| C | in | Синхросигнал |
| RST | in | Скидання |
| DA | in | Шина операнду А |
| DB | in | Шина операнду В |
| OUTHL | in | Керуючий біт мультиплексору (1 – видача 15 молодших бітів, 0 – видача 15 старших бітів) |
| N | out | Знаковий біт результату |
| Z | out | Біт-ознака нульового результату |
| DP | out | Шина результату |
| ***Виходи на автомат*** | | |
| LAB | in | Скидання поточних станів регістрів та бітів ознак, завантаження нових |
| SHIFT | in | Дозвіл зсуву |
| ADD | in | Збудження суматора |
| REZ | in | Дозвіл корекції результату та збудження XOR визначення знаку результату |
| B0 | out | Молодший біт операнду В |
| STOP | out | Біт-ознака вичерпності операнду В |

Задля зменшення кількості тактів формування бітів-ознак B0 та STOP на регістрі RG\_В відбувається завчасно, перед основною дією поточного такту, аби на наступному такті автомат вже мав змогу на основі обробки цих бітів видати нові управляючі сигнали до операційного блоку.

**Виконання:**

***Поведінкова модель:***

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;

library IEEE;

use IEEE.Numeric\_Bit.all;

library cnetlist;

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(23 downto 0); --A bus

DB : in BIT\_VECTOR(23 downto 0); --B bus

ADD : in BIT; --adder start flag

REZ : in BIT; --correction flag

B0 : out BIT; --first bit B flag

STOP : out BIT; --stop flag

Z: out BIT; -- result zero flag

N: out BIT; -- result sign

DP : out BIT\_VECTOR(23 downto 0)); -- result bus

end OB;

architecture BEH of OB is

signal A:bit\_vector(47 downto 0); -- register A data

signal SA:bit; --A sign bit

signal B:bit\_vector(23 downto 0); -- register B data

signal SB:bit; --B sign bit

signal S: unsigned(47 downto 0); -- adder buffer

signal P: unsigned(47 downto 0); -- register P data

begin

-- register A

RG\_A:process(C,RST)

variable high,low:natural;

begin

if RST='1' then

A<="000000000000000000000000000000000000000000000000"; --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(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<="000000000000000000000000"; --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(23 downto 1); --right shift

end if;

end if;

end process;

-- Adder

SM:S(47 downto 0) <= unsigned(P)+unsigned(A) when ADD='1' else P(47 downto 0);

-- register P (result)

RG\_P:process(C,RST,P)

variable zi:bit;

begin

if RST='1' then

P<="000000000000000000000000000000000000000000000000"; --reset

elsif C='1' and C'event then

if LAB='1' then

P<="000000000000000000000000000000000000000000000000"; --reload

elsif REZ='1' then

P<='0'&P(47 downto 1); --result correction

N<=SA xor SB;

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(23 downto 0)) when OUTHL='1' else bit\_vector(P(47 downto 24));

end BEH;

entity MPU is

port(C : in BIT;

RST : in BIT;

START:in BIT;

OUTHL:in BIT;

DA : in BIT\_VECTOR(23 downto 0);

DB : in BIT\_VECTOR(23 downto 0);

RDY : out BIT;

Z: out BIT;

N: out BIT;

DP : out BIT\_VECTOR(23 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(23 downto 0); --A bus

DB : in BIT\_VECTOR(23 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(23 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;

***Тестовий стенд:***

library IEEE;

use IEEE.STD\_LOGIC\_1164.all, CNetwork.all;

entity mpuBench is

port(CLK : in BIT);

end mpuBench;

architecture mpuBenchArch of mpuBench is

constant TIME\_DELTA : time := 100 ns;

signal C : BIT;

signal RST : BIT;

signal START: BIT;

signal OUTHL: BIT;

signal DA : BIT\_VECTOR(23 downto 0);

signal DB : BIT\_VECTOR(23 downto 0);

signal RDY : BIT;

signal Z: BIT;

signal N: BIT;

signal DP : BIT\_VECTOR(23 downto 0);

begin

UUT2:entity MPU(BEH)

port map (C=>CLK, RST=>RST, START=>START, OUTHL=>OUTHL, DA=>DA, DB=>DB, RDY=>RDY, Z=>Z, N=>N, DP=>DP);

simulation : process

begin

DA<="000000000000000000000111";

DB<="000000000000000000000011";

RST<='0';

START <='1';

OUTHL<='1';

wait for TIME\_DELTA;

--wait for TIME\_DELTA;

--AD <= "0000000000001000";

--OE <= '1';

--wait for TIME\_DELTA;

--OE <= '0';

--AD <= "0000000000001000";

--R <= '1';

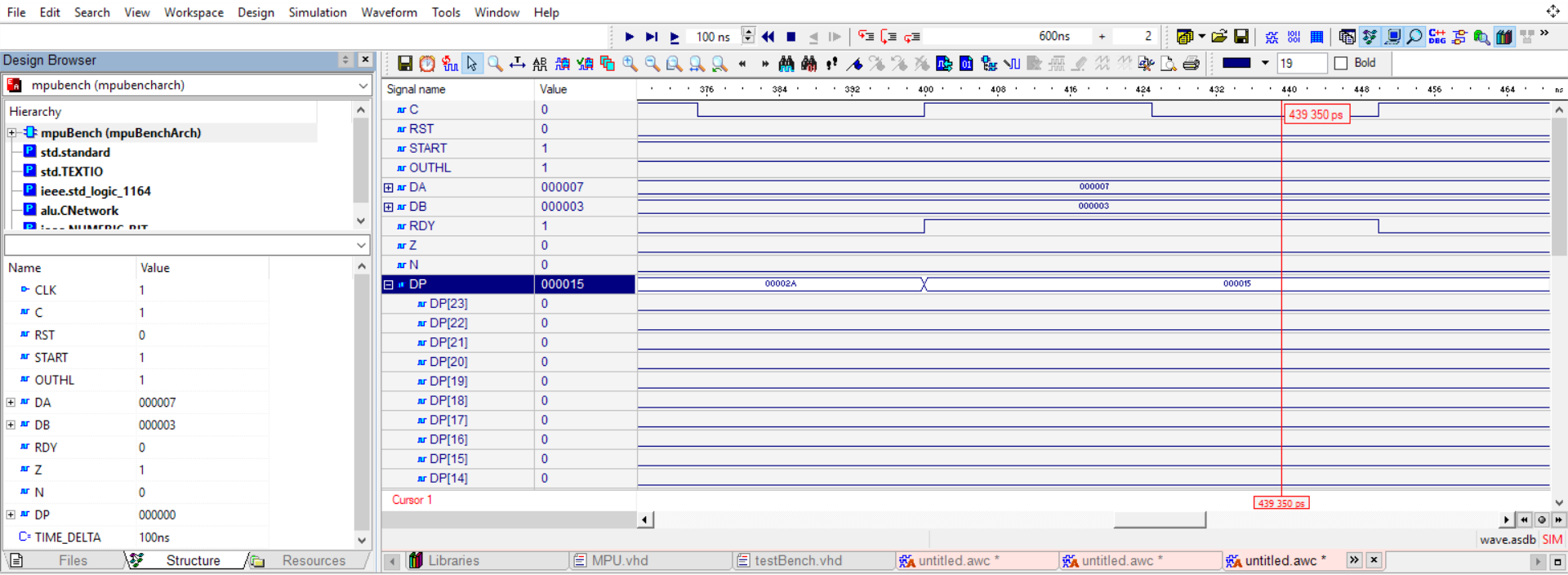
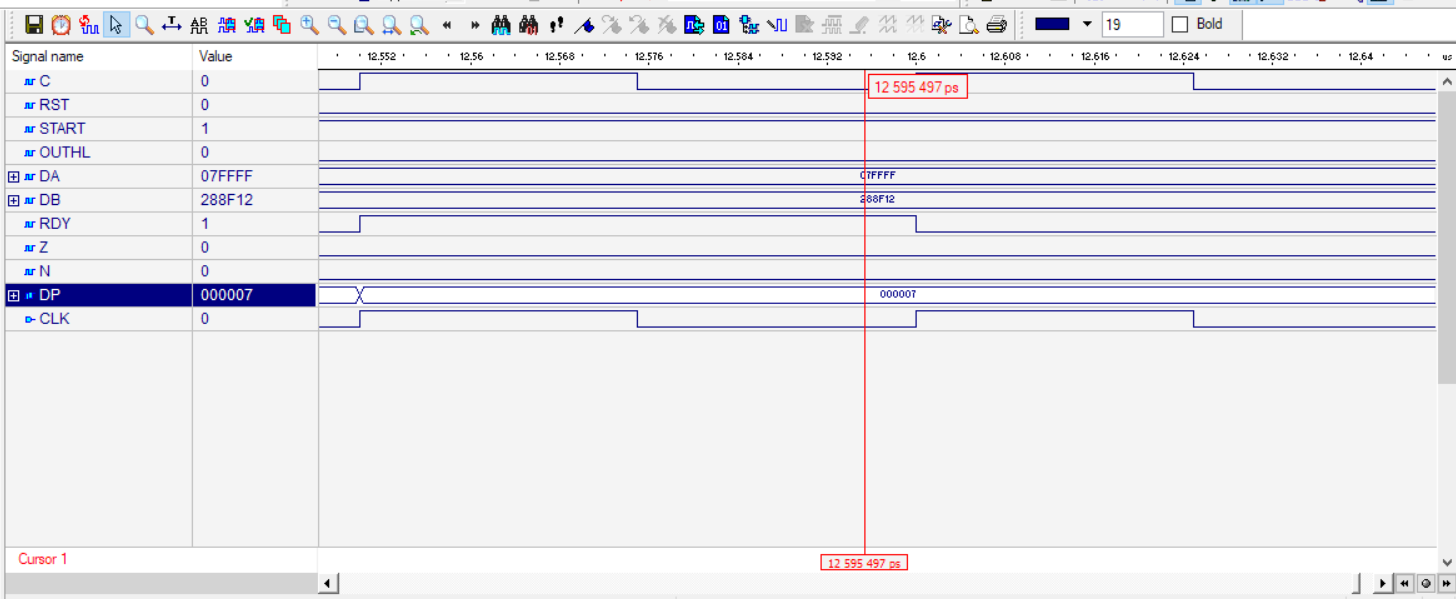
--wait for TIME\_DELTA;

--AD <= "0000000000001000";

--OE <= '1';

end process simulation;

end mpuBenchArch;

**Результати симуляції:**