ДОДАТОК А Графічна частина атестаційної роботи

Призначення та області використання біт-потокових функціональних обчислювачів

Сучасні системи управління є розподіленими системами, які містять у своєму складі функціональні перетворювачі і обчислювачі, що виконують **первинну і вторинну обробку сигналів**, які поступають з сенсорів фізичних величин.

Важливою особливістю в системах є те, що **обробка сигналів проводитися у реальному часі** в темпі надходження вхідних даних.

Функціональні обчислювачі призначені для виконання **вторинної** обробки при здійсненні лінеаризації сигналів за необхідною функцією та передбачає прямі обчислення.

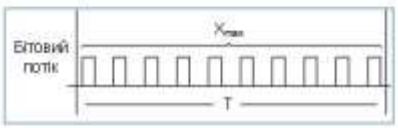
Функціональні модулі біт-потокових перетворень можуть бути використані:

- в розподілених системах управління, в якості функціональних перетворювачів бітових потоків даних, отриманих від вимірювальних сенсорів фізичних величин;
- в системах управління в якості спеціальної апаратури їх спряження з датчиками і виконавчими органами об'єкту керування;
- в якості обчислювальних вузлів інформаційно-вимірювальних систем при обчисленні діючих значень напруг і струмів;
- в якості обчислювальних вузлів в приборах вимірювання вібрації обертового обладнання при оцінюванні середньоквадратичних значень віброшвидкості та віброприскорення агрегатів з обертовими частинами;
- в якості обчислювальних вузлів у вимірювальних системах при здійсненні непрямих вимірювань;.

Бітові потоки даних та їх функціональне перетворення

Бітові потоки даних представляють собою потоки імпульсів одиничної амплітуди частотних, час-імпульсних або широтно-модульованих сигналів.

При представленні інформаційних сигналів бітовими потоками **інформативним параметром** є фіксоване значення імпульсів (біт) за часовий інтервал.



Вхідний і вихідний інформаційні сигнали біт-потокових обчислювачів являють собою **два бітових потоки.**

Періодичність проходження біт вхідного потоку визначається **способом квантування відтворюваної функції по аргументу**, а вихідного потоку - **алгоритмом функціонування пристрою.**

При цьому забезпечується **рівномірне квантування** аргументу **цілочисельними значеннями**.

При **функціональному перетворенні** бітових потоків здійснюється реалізація потокового методу обчислень, що передбачає одночасне паралельно-послідовне виконання перетворень над бітовими потоками частотних або час-імпульсних сигналів відповідно до необхідної функції.

Досліджувана функція

Апаратний обчислювач ірраціональних функцій обчислює апроксимуючу функцію:

$$y = [\frac{1}{p}[\sqrt{\sum_{i=1}^{n} x_i^2} + |\delta_{2max}|] + |\delta_{1max}|] \qquad (1)$$

де x – аргумент функції, що представляє собою серії бітового потоку;

$|\delta_{1max}|$ – граничне значення абсолютної похибки ділення функції $\sqrt{\sum_{i=1}^{n} x_i^2}$ на p;

$|\delta_{2max}|$ – граничне значення абсолютної похибки добування квадратного кореня з $\sum_{i=1}^{n} x_i^2$

Абсолютна похибка обчислення ірраціональної функції виникає при поділі полінома на число p і при добуванні кореня та дорівнює 0,5 одиниці молодшого біту числа x, що є мінімальною похибкою:

$|\delta_{1max}| = |\delta_{2max}| = 0,5$

Отже, **обчислювач реалізує функцію**:

$$y = [\frac{1}{p}[\sqrt{\sum_{i=1}^{n} x_i^2} + 0,5] + 0,5] \qquad (2)$$

При **обчисленні досліджуваної функції** виконуються наступні **операції над серіями вхідного бітового потоку x**:

- на **першому етапі** обчислень виконуються операції піднесення до степеню і добування кореня у виразі $\sqrt{\sum_{i=1}^{n} x_i^2}$, тобто обчислення степеневої функції з дробовим показником;
- на **другому етапі** виконується операція поділу отриманого результату на ціле число **p**.

Метод ступінчастої апроксимації безперервних висхідних функцій

Безперервна висхідна функція $y^* = f(x^*)$, обмеженням якої є умова $y \le x$ і яка має зворотну $x^* = \psi(y^*)$ може бути відтворена на виході біт-потокового обчислювача апроксимуючою функцією:

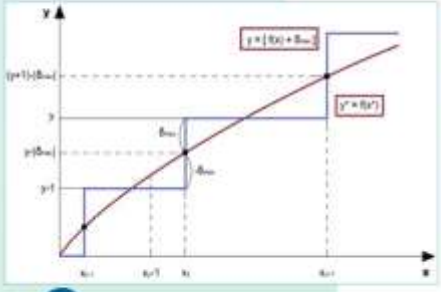
$$y = [f(x) + |\delta_{max}|] \qquad (3)$$

де x, y - вхідний і вихідний бітові потоки даних,

$|\delta_{max}|$ - граничне значення абсолютної похибки відтворення безперервної функції $0,5 \le |\delta_{max}| < 1$

Для будь-якого рівня $y - |\delta_{max}|$, де y приймає цілочисельні значення y = 1, 2, 3, ...k, можна вказати пару цілочисельних значень аргументу $x_y - 1$ та x_y, для яких має місце система нерівностей:

$$\begin{cases} f(x_y - 1) < y - |\delta_{max}|, \\ f(x_y) \ge y - |\delta_{max}|. \end{cases}$$



Формула загального члена x_y числової послідовності $x_1, x_2, x_3, ..., x_k$, що відповідає вузлам апроксимації вихідної функції y має вигляд:

$$\Psi(y - |\delta_{max}|) \le x_y < \Psi(y - |\delta_{max}|) + 1 \qquad (4)$$

де - значення біт, що обираються з вхідного бітового потоку (вибірка), які отримують шляхом підстановки в (4) цілочисельних значень y = 1, 2, 3, ...k.

При абсолютній похибці обчислень $|\delta_{max}| = 0,5$ значення вибірок x_y визначаються за формулою:

$$\Psi(y - 0,5) \le x_y < \Psi(y - 0,5) + 1 \qquad (5)$$

Математична модель обчислювача степеневих функцій

На **першому етапі** ірраціональним обчислювачем реалізується апроксимуюча **степенева функція**, що має вигляд:

$$y = [\sqrt{\sum_{i=1}^{n} x_i^2} + 0,5] \qquad (6)$$

Математична модель обчислювача приведена для однієї серії імпульсних послідовностей.

Для степеневої функції $y = [\sqrt{x^2} + 0,5]$ (7)

була визначена зворотна функція і формула загального члена числової послідовності x_y, що відповідає вузлам апроксимації даної функції

$$2^2 x_y^2 \geq (2y_k - 1)^2. \qquad (8)$$

Було отримано математичну модель обчислювача степеневих функції, що представляє собою систему нерівностей в різницях:

При надходженні на вхід пристрою певного біта з номером x_y на його виході буде сформований вихідний біт y_k при виконанні кожної нерівності системи (9).

$$\begin{aligned} &2^2 x_1^2 \geq (2y_1 - 1)^2 \\ &2^2(x_2^2 - x_1^2) + \Delta_1 \geq (2y_2 - 1)^2 - (2y_1 - 1)^2 \\ &\cdots\cdots\cdots\cdots\cdots\cdots \\ &2^2(x_y^2 - x_{y-1}^2) + \Delta_{y-1} \geq (2y_k - 1)^2 - (2y_{k-1} - 1)^2 \end{aligned} \qquad (9)$$

Δ_{y-1} - різниця, що утворюється при виконанні нерівності на попередньому кроці при порівнянні приростів ґратчастих функцій $2^2 x_y^2$ і $(2y_k - 1)^2$.

Математична модель обчислювача лінійних функцій

На другому етапі обчислень необхідно виконувати ділення результату відтворення проміжної степеневої функції на константу **p**. Абсолютна похибка обчислень $|\delta_{max}| = 0,5$.

Обчислення може бути реалізовано з використанням обчислювача лінійних функцій, який відтворює функцію:

$$y = [\frac{1}{p}x + 0,5] \qquad (10)$$

де x - бітовий потік, що надходить на вхід лінійного обчислювача з виходу степеневого модуля проміжної функції.

На основі зворотної функції отримано **нерівність для вибірок** x_y $2x_y \geq p(2y_k - 1)$ (11)

Отримано систему нерівностей (12), що є математичною моделлю обчислювача лінійної функції.

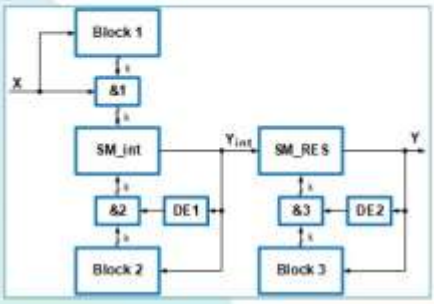
$$\begin{aligned} &2x_1 \geq p \\ &2(x_2 - x_1) + \Delta_1 \geq 2p, \\ &2(x_3 - x_2) + \Delta_2 \geq 2p, \\ &\cdots\cdots\cdots\cdots\cdots \\ &2(x_y - x_{y-1}) + \Delta_{y-1} \geq 2p \end{aligned} \qquad (12)$$

Першому вихідному біту y = 1 відповідає біт з номером x_1, що обирається з вхідного потоку x, при якому буде виконана перша нерівність системи.

Аналогічно другому обраному імпульсу y = 2 відповідає імпульс з номером x_2 при якому виконається друга нерівність.

де $\Delta_{y-1} = 2(x_y - x_{y-1}) + \Delta_{y-2} - 2p$

Узагальнена архітектура обчислювача ірраціональних функцій



Обчислення ірраціональної функції в структурі виконується послідовно у **два етапи**:

На **першому етапі** обчислюється значення проміжної степеневої функції $y_{int} = f(x)$.

2. **На другому етапі** обчислюється значення ірраціональної функції $y = f(x)$. Вона є результатом ділення на константу результату обчислення проміжної функції.

Основними обчислювальними компонентами архітектури є **паралельні суматори SM_int і SM_RES зі зворотним зв'язком**, що використовується в якості компонента порівняння паралельних кодів.

Формування сходинок відтворюваної функції, здійснюється на виході **суматора результату SM_RES**, в який з Block 1 надходять прямі коди чисел, а з Block 2 - додаткові коди чисел.

Block1, Block2 – блок відтворення проміжної степеневої функції з дробовим показником.

Block3 – блок дільника чисел, на виході якого формується апроксимуюча ірраціональна функція у вигляді бітового потоку y.

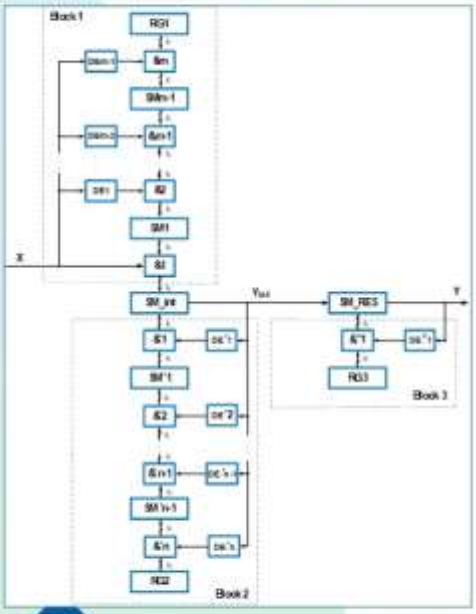
SM_int – суматор результату проміжної функції;

SM_RES – суматор результату обчислення ірраціональної функції

&1, &2, &3 – група елементів &

DE1, DE2 – елемент затримки

Деталізована архітектура біт-потокового ірраціонального обчислювача



В деталізованій загальній архітектурі **Block1, Block2, SM_int** складають структуру обчислювача **степеневих функцій**.

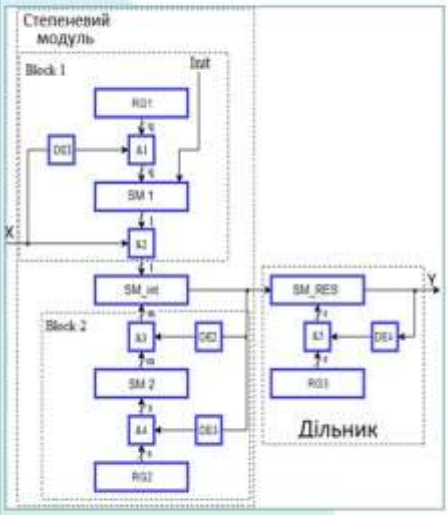
Block1, Block2 - конвеєрні структури біт-потокового обчислювача поліномів, в яких відтворення функції відбувається на основі конвеєрних обчислень.

Block3, суматор SM_RES - **дільник чисел**, побудований на основі архітектури обчислювача лінійних функцій.

Ініціалізація компонентів у **Block1, Block2** здійснюється значеннями перших членів арифметичних рядів m-го та n-го порядків та їх арифметичних рядів різниць відповідно.

Суматор SM_int ініціалізується числом, що враховує числа ініціалізації з боку поліноміальних обчислювачів прямого та зворотного зв'язку.
Суматор SM_RES ініціалізується числом, що враховує числа ініціалізації з боку **Block3.**

Архітектура досліджуваного обчислювача



До складу пристрою входять **2 блоки**:
- **степеневий модуль**,
- **дільник чисел.**

I. В **степеневому модулі** реалізується нерівність

$$2^2 x_y^2 \geq (2y_k - 1)^2.$$

Основний обчислювальний компонент - суматор SUM1, що використовується в якості схеми порівняння паралельних кодів приростів грачастої функції $2^2 x_y^2$ з приростами грачастої функції $(2y-1)^2$ з урахуванням їх різниці отриманої на попередньому кроці обчислень.

Ініціалізація компонентів степеневого модуля:
- суматор **SM_int** ініціалізується числом $2^l - 1$;
- суматор **SM1** ініціалізується числом $1^2 = 1$;
- суматор **SM2** - ініціалізується числом $3^2 - 1^2$
- регістр **RG1** ініціалізується числом 2l;
- регістр **RG2** ініціалізується константою в ряду різниць 2-го порядку функції $(2y-1)^2$.

II. Дільник чисел, побудований на основі обчислювача лінійної функції, виконує ділення значення степеневої функції на число p.

Ініціалізація компонентів дільника чисел:
- суматор **SM_RES** ініціалізується числом $2^l - p$,
- регістр **RG3** ініціалізується числом $2^l - 2p$.

Kharkov National University of Radio Electronics,
Design Automation Department, phone 7021 326

Апаратна реалізація обчислювача ірраціональних функцій

В процесі експериментального дослідження розроблено апаратну реалізацію пристрою, що відтворює ірраціональну функцію:

$$y = [\frac{1}{p}[\sqrt{\sum_{i=1}^{n} x_i^2} + 0,5] + 0,5]$$

Вихідні дані експерименту:

x_1	x_2	x_3	x_4	p	n
7	4	5	6	2	4

На вхід обчислювача надходить 4 серії біт розмірністю:
$x_1 = 7$ $x_1 = 4$ $x_3 = 5$ $x_4 = 6$

В результаті відтворювана функція набуває вигляду:

$$y = [\frac{1}{2}[\sqrt{7^2 + 4^2 + 5^2 + 6^2} + 0,5] + 0,5]$$

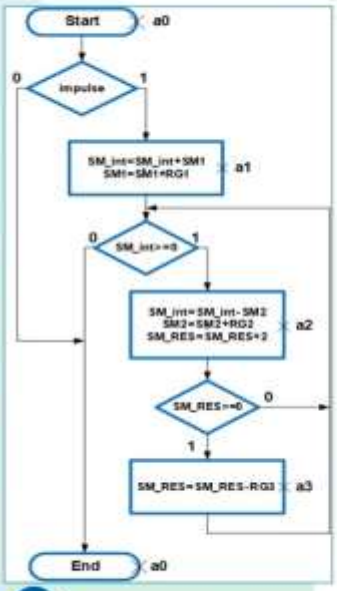
Для функції x_y^2 арифметичний ряд 2-го порядку і арифметичні ряди різниць 1-го і 2-го порядків мають вигляд:
1, 4, 9, 16, 25, 36, 49, 64, 81, 100
Δ: 3, 5, 7, 9, 11, 13, 15, 17, 19
Δ^2: 2, 2, 2, 2, 2, 2, 2, 2.

Для функції $(2y-1)^2$ арифметичний ряд 2-го порядку арифметичні ряди різниць 1-го і 2-го порядків мають вигляд:
1, 9, 25, 49, 81, 121, 169, 225, 289, 361.
Δ: 8, 16, 24, 32, 40, 48, 56, 64, 72
Δ^2: 8, 8, 8, 8, 8, 8, 8, 8.

Ініціалізація компонентів пристрою: **SM_int** ініціалізується числом $2^l - 1$, **SM1** – числом $1^2 = 1$, **SM2** – числом $3^2 - 1^2 = 8$, **RG1** – числом 2l, тобто 2; **RG2** – константою, числом 8; **SM_RES** – числом $2^l - p$, тобто $2^l - 2$, **RG3** – числом $2^l - 2p$, тобто, $2^l - 4$.

Kharkov National University of Radio Electronics,
Design Automation Department, phone 7021 326

ГСА операційного автомату обчислювача



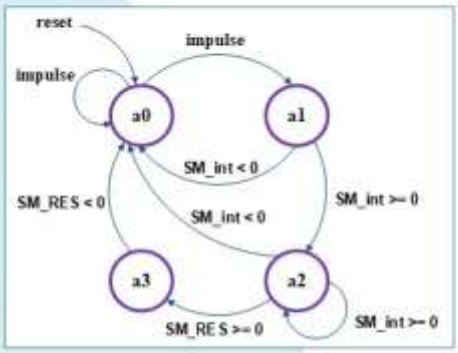
Біт-потоковий обчислювач представлений мікропрограмним автоматом, що є **композицією керуючого і операційного автоматів.** ГСА розмічена для синтезу **автомата моделі Мура.** Містить керуючі стани: a_0, a_1, a_2 , a_3.

Фрагмент HDL-коду операційного автомату обчислювача

```
begin
if (reset_i = '1') then
      counter <= CONV_STD_LOGIC_VECTOR(1,  width);
      sum1 <= CONV_STD_LOGIC_VECTOR(-1,  width);
      sum2 <= CONV_STD_LOGIC_VECTOR(8,  width);
      sum3 <= CONV_STD_LOGIC_VECTOR(-p,  width);
      get_next_bundle <= '0';
else
      if (falling_edge(clock_i)) then
      if (x_imp_i = '1') then
         sum1 <= sum1 + CONV_INTEGER(counter(width-3  downto 0)& "00");
                  if (next_bundle = '1') then
                     counter <= CONV_STD_LOGIC_VECTOR(1,  width);
                     get_next_bundle <= '1';
                  else
                           counter <= counter + step_cnt;
                           get_next_bundle <= '0';
                  end if;
                  elsif (sum1_ovf_i  = '1') then
                              sum1 <= sum1 - sum2;
                              sum2 <= sum2 + step_sum;
                              sum3 <= sum3 + step_cnt;

                  elsif (sum3_ovf_i  = '1') then
                              sum3 <= sum3 - 2*p;
                  end if;
      end if;
   end if;
```

Граф переходів автомата Мура арифметичного блоку

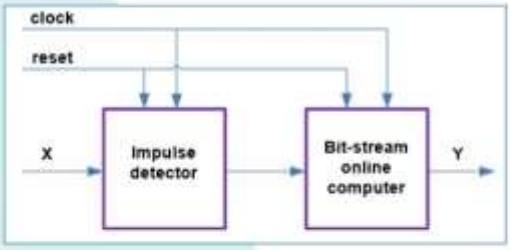


Керуючий автомат арифметичного блоку обчислювача описано графом переходів. Граф було отримано в результаті аналізу і синтезу ГСА для **автомата** обраної **моделі Мура**. Він може приймати **4 стани**: a0, a1, a2, a3

Фрагмент **HDL-коду керуючого автомату ірраціонального обчислювача** з використанням оператору **case**.

```
begin
        case (state) is
                when a_0 =>
                if x_i = '1' then
                        next_state <= a_1;
                 else
                        next_state <= a_0;
                end if;
                when a_1 =>
                if sum1_above_zero_i = '1' then
                        next_state <= a_2;
                else
                        next_state <= a_0;
                end if;
                when a_2 =>
                if sum2_above_zero_i = '1' then
                        next_state <= a_3;
                else
                        next_state <= a_0;
                end if;
                when a_3 =>
                        next_state <= a_0;
                when others =>
                        next_state <= a_0;
        end case;
```

Структурно-блокова схема досліджуваного обчислювача



Структурно-блокова схема пристрою містить **два блоки:**

- **детектор імпульсу;**
- **блок біт-потокового обчислювача.**

Блок **детектору імпульсу** призначений для детектування біт-потокової послідовності на вході пристрою x.

Імпульс детектується за двома сусідніми тактами сигналу clock. Якщо на черговому такті значення сигналу x = 0, а на наступному такті значення x = 1, то вхідний буфер детектує імпульс, і на виході встановлює відповідний сигнал impulse = 1. Сигнал буде отримано пристроєм обчислювача.

Блок **біт-потокового обчислювача** виконує операцію піднесення аргументу x до дробового степеню з заданою похибкою, результат округляється до найближчих цілих чисел.

Обчислювальний процес в компонентах досліджуваного пристрою

В таблиці наведено обчислювальний процес, що відбуваються в компонентах обчислювача при надходженні на вхід пристрою двох серій імпульсів довжиною 7 та 4.

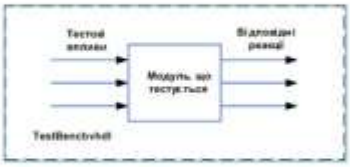
x	Значення функції Y
1	$y=\left[\frac{1}{2}\left[\sqrt{1^2}+0.5\right]+0.5\right]=\left[\frac{1}{2}\cdot[1.5]+0.5\right]=1$
2	$y=\left[\frac{1}{2}\left[\sqrt{2^2}+0.5\right]+0.5\right]=\left[\frac{1}{2}\cdot[2.5]+0.5\right]=1$
3	$y=\left[\frac{1}{2}\left[\sqrt{3^2}+0.5\right]+0.5\right]=\left[\frac{1}{2}\cdot[3.5]+0.5\right]=2$
4	$y=\left[\frac{1}{2}\left[\sqrt{4^2}+0.5\right]+0.5\right]=\left[\frac{1}{2}\cdot[4.5]+0.5\right]=2$
5	$y=\left[\frac{1}{2}\left[\sqrt{5^2}+0.5\right]+0.5\right]=\left[\frac{1}{2}\cdot[5.5]+0.5\right]=3$
6	$y=\left[\frac{1}{2}\left[\sqrt{6^2}+0.5\right]+0.5\right]=\left[\frac{1}{2}\cdot[6.5]+0.5\right]=3$
7	$y=\left[\frac{1}{2}\left[\sqrt{7^2}+0.5\right]+0.5\right]=\left[\frac{1}{2}\cdot[7.5]+0.5\right]=4$

x	Значення функції Y
1	$y=\left[\frac{1}{2}\left[\sqrt{7^2+1^2}+0.5\right]+0.5\right]=\left[\frac{1}{2}\cdot[7.5]+0.5\right]=4$
2	$y=\left[\frac{1}{2}\left[\sqrt{7^2+2^2}+0.5\right]+0.5\right]=\left[\frac{1}{2}\cdot[7.7]+0.5\right]=4$
3	$y=\left[\frac{1}{2}\left[\sqrt{7^2+3^2}+0.5\right]+0.5\right]=\left[\frac{1}{2}\cdot[8.1]+0.5\right]=4$
4	$y=\left[\frac{1}{2}\left[\sqrt{7^2+4^2}+0.5\right]+0.5\right]=\left[\frac{1}{2}\cdot[8.5]+0.5\right]=4$

№ пачки	x	SM_int	Біт на виході SM_int	SM1	SM2	SM_RES	Біт на виході SM_RES
1 Серія біт	1	-1+4=3	1	1+2=3	8+8=16	-2+2=0	1
	2	3-8=-5 -5+12=7	1	3+2=5	16+8=24	0-4=-4 -4+2=-2	
	3	7-16=-9 -9+20=11	1	5+2=7	24+8=32	-2+2=0	1
	4	11-24=-13 -13+28=15	1	7+2=9	32+8=40	0-4=-4 -4+2=-2	
	5	15-32=-17 -17+36=19	1	9+2=11	40+8=48	-2+2=0	1
	6	19-40=-21 -21+44=23	1	11+2=13	48+8=56	0-4=-4 -4+2=-2	
	7	23-48=-25 -25+52=27	1	13+2=15	56+8=64	-2+2=0	1
		27-56=-29				0-4=-4	
2 Серія біт	1	-29+4=-25		1+2=3	64	-4	
	2	-25+12=-13		3+2=5	64	-4	
	3	-13+20=7	1	5+2=7	64+8=72	-4+2=-2	
	4	7-64=-57 -57+28=-29		7+2=9	72	-2	

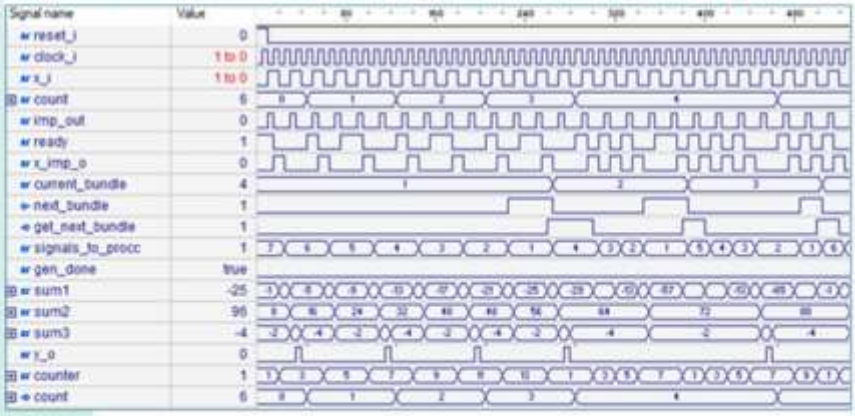
Верифікація та тестування роботи пристрою

Для верифікації даної апаратної реалізації, мовою VHDL було розроблено тестове оточення (test bench).



Верифікацію поведінкової моделі пристрою представлено на часовій діаграмі

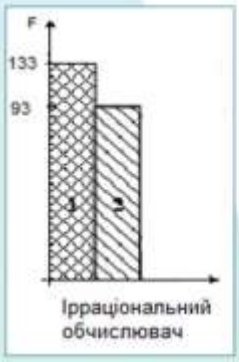
Результати експерименту співпадають з результатами теоретичних обчислень, що підтверджує правильність його роботи.



Порівняльний аналіз частотних характеристик ірраціональних обчислювачів

Порівняльна таблиця частотних характеристик

Біт-потоковий обчислювач	T_F, ns	F_{max}, MHz
Ірраціональний (1)	7,7 ns	133 MHz
Ірраціональний (2)	10,7 ns	93 MHz



Максимальна частота роботи ірраціонального обчислювача (1), побудованого на основі степеневого модуля на 28% вища за максимальну частоту роботи ірраціонального обчислювача (2), побудованого на основі дробово-раціонального модуля.

Висновки

В кваліфікаційній роботі розроблено і досліджено **апаратний біт-потоковий обчислювач ірраціональних функцій**.

- Проаналізовано **форми представлення** бітових потоків даних та **їх перетворення** в функціональних біт-потокових обчислювачах;
- Було розроблено математичну модель обчислювача ірраціональних функцій, що являє собою **декомпозицію математичних моделей обчислювачів степеневої та лінійної функцій**.
- Для отримання математичних моделей степеневого та лінійного обчислювачів був використаний **метод ступінчастої апроксимації безперервних висхідних функцій** на основі визначення **зворотних функцій.**
- Використано переваги **принципу побудови** біт-потокової конвеєрної архітектури обчислювача поліномів, яка є **складовою архітектур ірраціональних обчислювачів**. Принцип реалізує функціональне перетворення бітових потоків на основі обчислення **приростів відтворюваної функції.**

Висновки

- Розроблено архітектуру **обчислювача ірраціональних функцій**, що реалізовано **синтезом двох** послідовно з'єднаних **блоків**: блоком реалізації степеневої функції та блоком дільника чисел.
- **Основним обчислювальним компонентом** в структурі обчислювача є паралельний суматор зі зворотним зв'язком, в якому здійснюється **порівняння гратчастих функцій** нерівностей математичної моделі пристрою.
- В результаті розробки та аналізу математичної моделі та архітектури досліджуваного обчислювача, було розроблено **апаратну модель обчислювача** та здійснено опис проекту для введення в САПР.
- Апаратна модель сформована на основі **кінцевого автомата моделі Мура**.
- Було розроблено змістовну граф-схему алгоритму **операційного автомата**, і, на підставі ГСА, отриманий граф переходів **керуючого автомату обчислювача**.
- За графами переходів з використанням HDL-шаблонів кода розроблено модель пристрою на мові опису апаратури VHDL.
- Для верифікації апаратної реалізації, було розроблено тестове оточення test bench і отримано поведінкову модель пристрою.
- Модель імплементована в технологічну платформу ПЛІС Xilinx Spartan 3E.

ДОДАТОК Б Лістинг коду програм

Лістинг Б.1 – Програма верхнього рівня speccalc_top.vhdl

```
speccalc_top.vhdl

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;
use work.cfg_pkg.all;

entity speccalc_toplevel is
	port(
		reset_i: in std_logic;
		clock_i: in std_logic;
		x_i: in std_logic;
		y_o: out std_logic;
		count: out std_logic_vector(width-1 downto 0));
end speccalc_toplevel;

architecture struct of speccalc_toplevel is
	-- Component declaration of the "aproximator(struct)" unit defined in
	-- file: "./src/aproximator.vhd"
	component speccalc
		port(
			x_i: in std_logic;
		   x_imp_o: out std_logic;
			ready_o: out std_logic;
			done_i: in std_logic;
			get_next_bundle: out std_logic;
			next_bundle: in std_logic;
			clock_i: in std_logic;
			reset_i: in std_logic;
			y_o: out std_logic;
			sum_o: out std_logic_vector(width-1 downto 0));
	end component;

	-- Component declaration of the "inputbuffer(beh)" unit defined in
	-- file: "./src/inputbuffer.vhd"
	component speccalc_imp
		port(
			x_i : in std_logic;
			clock_i : in std_logic;
			reset_i : in std_logic;
			ready_i : in std_logic;
			y_o : out std_logic);
	end component;

	-- Component declaration of the "traceability_unit(struct)" unit
```

```
	-- file: "./src/traceability_unit.vhd"
	component traceability_unit
	port(reset_i: in std_logic;
		x_imp_i: in std_logic;
		done: out std_logic;
		get_next_bundle: in std_logic;
		next_bundle: out std_logic);
	end component;

	signal imp_out, speccalc_out, ready : std_logic;

	signal count_t: std_logic_vector(width-1 downto 0);
	signal sum_t: std_logic_vector(width-1 downto 0);

	signal x_imp_o, done, get_next_bundle, next_bundle: std_logic;
begin

	Impulse_Detector: speccalc_imp
	port map(
		x_i => x_i,
		clock_i => clock_i,
		reset_i => reset_i,
		ready_i => ready,
		y_o => imp_out
		);

	spec_calc: speccalc
	port map(
		x_i => imp_out,
		x_imp_o => x_imp_o,
		ready_o => ready,
		done_i => done,
		get_next_bundle => get_next_bundle,
		next_bundle => next_bundle,
		clock_i => clock_i,
		reset_i => reset_i,
		y_o => speccalc_out,
		sum_o => sum_t
		);

	trace_unit: traceability_unit
	port map (
		reset_i => reset_i,
		x_imp_i => x_imp_o,
		done => done,
		get_next_bundle => get_next_bundle,
		next_bundle => next_bundle);

	process(clock_i, reset_i)
	begin
		if reset_i = '1' then
			count_t <= (others => '0');
		else
			if rising_edge(clock_i) then
				if speccalc_out = '1' then
					count_t <= count_t + 1;
				end if;
```

```
                end if;
            end if;

        end process;

        y_o <= clock_i and speccalc_out;
        count <= count_t;

end struct;
```

Лістинг Б.2 – Програма операційного автомата speccalc.oa.vhdl

```
speccalc.oa.vhdl

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;
use work.cfg_pkg.all;


entity speccalc_oa is
        port(
                clock_i : in std_logic;
                reset_i : in std_logic;
                x_imp_i : in std_logic;
                next_bundle : in std_logic;
                get_next_bundle : out std_logic;
                sum1_ovf_i : in std_logic;
                sum3_ovf_i : in std_logic;
                sum1_above_zero_o : out std_logic;
                sum3_above_zero_o : out std_logic;
                sum_o: out std_logic_vector(width-1 downto 0));
end speccalc_oa;


architecture beh of speccalc_oa is

        signal sum1, sum2, sum3 : std_logic_vector(width-1 downto 0);
        signal counter : std_logic_vector(width-1 downto 0);

begin

        sum_o <= sum1;

        sum1_above_zero_o <= '0' when (sum1 < 0) else '1';
        sum3_above_zero_o <= '0' when (sum3 < 0) else '1';

        process (clock_i, reset_i)
        begin
                if (reset_i = '1') then
                        counter <= CONV_STD_LOGIC_VECTOR(1, width);
                        sum1 <= CONV_STD_LOGIC_VECTOR(-1, width);
                        sum2 <= CONV_STD_LOGIC_VECTOR(8, width);
```

```
                sum3 <= CONV_STD_LOGIC_VECTOR(-p, width);
                get_next_bundle <= '0';
        else
                if (falling_edge(clock_i)) then
                if (x_imp_i = '1') then
                sum1 <= sum1 + CONV_INTEGER(counter(width-3 downto 0)& "00");

                        if (next_bundle = '1') then
                        counter <= CONV_STD_LOGIC_VECTOR(1, width);
                                get_next_bundle <= '1';
                        else
                                counter <= counter + step_cnt;
                                get_next_bundle <= '0';
                        end if;
                elsif (sum1_ovf_i = '1') then
                        sum1 <= sum1 - sum2;
                        sum2 <= sum2 + step_sum;
                        sum3 <= sum3 + step_cnt;
                elsif (sum3_ovf_i = '1') then
                        sum3 <= sum3 - 2*p;
                end if;
                end if;
        end if;
    end process;
end beh;
```

Лістинг Б.3 – Програма керуючого автомату пристрою speccalc.ua.vhdl

```
speccalc.ua.vhdl

library ieee;
use ieee.std_logic_1164.all;

entity speccalc_ua is
    port(
        clock_i: in std_logic;
        reset_i: in std_logic;
        x_i: in std_logic;
        y_o: out std_logic;
        done_i: in std_logic;
        ready_o: out std_logic;
        sum1_above_zero_i: in std_logic;
        sum3_above_zero_i: in std_logic;
        x_imp_o : out std_logic;
        sum1_ovf_o: out std_logic;
        sum3_ovf_o: out std_logic);
end speccalc_ua;


architecture beh of speccalc_ua is
    type state_type is (a_0, a_1, a_2, a_3);
    signal state, next_state: state_type;
    signal control: std_logic_vector(4 downto 0);
begin
```

```
	process(clock_i, reset_i)
	begin
		if (reset_i = '1') then
			state <= a_0;
		else
			if (rising_edge(clock_i)) then
				state <= next_state;
			end if;
		end if;
	end process;
	process(state, x_i, sum1_above_zero_i, sum2_above_zero_i)
	begin
		case (state) is
			when a_0 =>
			if x_i = '1' then
				next_state <= a_1;
			else
				next_state <= a_0;
			end if;
			when a_1 =>
			if sum1_above_zero_i = '1' then
				next_state <= a_2;
			else
				next_state <= a_0;
			end if;
			when a_2 =>
			if sum2_above_zero_i = '1' then
				next_state <= a_3;
			else
				next_state <= a_0;
			end if;
			when a_3 =>
			next_state <= a_0;
			when others =>
			next_state <= a_0;
		end case;
	end process;
	(ready_o, x_imp_o, sum1_ovf_o, sum3_ovf_o, y_o) <= control;
	with state select
	control <= "10000" when a_0,
	"01000" when a_1,
	"00100" when a_2,
	"00011" when a_3,
	"00000" when others;
end beh;

process(state, x_i, sum1_above_zero_i, sum2_above_zero_i)
	begin
		case (state) is
			when a_0 =>
			if x_i = '1' then
				next_state <= a_1;
			else
				next_state <= a_0;
			end if;
			when a_1 =>
			if sum1_above_zero_i = '1' then
				next_state <= a_2;
```

```
                else
                      next_state <= a_0;
                end if;
                when a_2 =>
                if sum2_above_zero_i = '1' then
                      next_state <= a_3;
                else
                      next_state <= a_0;
                end if;
                when a_3 =>
                next_state <= a_0;
                when others =>
                next_state <= a_0;
          end case;
    end process;
```

Лістинг Б.4 – Програма-обгортка Wrapper.VHDL

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.STD_LOGIC_UNSIGNED.all;
use work.cfg_pkg.all;

entity wrapper is
    port(
        rst, St, clk: in STD_LOGIC;
        LED: out STD_LOGIC_VECTOR(width-1 downto 0)
        );
end wrapper;


architecture behav of wrapper is
    component speccalc_toplevel is
    port(
        reset_i: in std_logic;
        clock_i: in std_logic;
        x_i: in std_logic;
        y_o: out std_logic;
        count: out std_logic_vector(width-1 downto 0));
    end component;

    constant N : INTEGER := 16;
    signal COUNT_INT : STD_LOGIC_VECTOR(N-1 downto 0);

    signal s1, s2, s3, St_in, clk_out: std_logic;
    signal y_out : std_logic;


begin
    SPEC1: speccalc_toplevel
    generic map(
        width => width
    )
```

```
	port map(
		reset_i => rst,
		clock_i => clk,
		x_i => St_in,
		y_o => y_out,
		count => LED);

	process(rst, clk)
	begin
		if rst = '1' then
			COUNT_INT <= (others => '0');
		elsif clk'event and clk = '1' then
			COUNT_INT <= COUNT_INT + 1;
		end if;
	end process;

	clk_out <= '1' when COUNT_INT(N-1) = '1' else '0';

	process (clk_out)
	begin
		if (clk_out'event and clk_out = '1') then
			s1<=St;
			s2<=s1;
			s3<=s2;
		end if;
	end process;

	St_in <= s1 and s2 and not s3;
end behav;
```

№	Позначення	Найменування	Дод. відомості
		Текстові документи	
1	ГЮІК 46741Х.009 ПЗ	Пояснювальна записка	77 с.
		Графічні документи	
2		Слайд-презентація	20 слайдів
		Носії інформації	
3		Компакт-диск (CD)	1 шт.

Змін.	Арк	Номер докум.	Підп.	Дата	46741Х.009 ВД			
Розроб.		Кузніченко Т. Б.			Моделі та методи проектування апаратного біт-потокового обчислювача ірраціональних функцій	Літ	Аркуш	Аркушів
Перевір.		Ларченко Л.В				У		
Н.контр.		Шкіль О.С.			Відомість атестаційної роботи	ХНУРЕ Кафедра АПОТ		
Затв.		Чумаченко С.В.						