УДК 681.3

А.О. Мельник, А.М. Сало, В.А. Клименко
Національний університет "Львівська політехніка",
кафедра електронних обчислювальних машин

АПАРАТНА РЕАЛІЗАЦІЯ ЦИКЛІВ ПРОГРАМОВАНИХ КОНФІГУРОВАНИХ ПРОЦЕСОРІВ

Проаналізовано виконання циклів у конвеєрі команд процесора. З метою підвищення продуктивності пропонується використовувати апаратні цикли. Розглянуто існуючі реалізації апаратних циклів та сигнальних процесорів з підтримкою апаратних циклів. Наведена реалізація блока апаратних циклів з підтримкою вкладених циклів.

In the given paper the execution of loops in the processor's instruction pipeline is analysed. To increase the perfomance, hardware loop utilization is suggested. The existing hardware loops and signal processors realization with the support of hardware loops is considered. The hardware loops realization unit with the nested loops support is presented.

Вступ. Сучасні задачі, які по'являються перед процесорами, вимагають від них все більшої продуктивності. Особливо це стосується вбудованих процесорів, призначених для виконання складних задач з обробки сигналів. Більше того, для вбудованих процесорів є неприпустимим збільшення споживаної потужності та тепловіддачі. Отже, необхідне рішення, яке б дозволило підвищити продуктивність процесорів без значного підвищення потужності. Таким архітектурним рішенням є конвеєризація та паралельне виконання інструкцій у процесорі. Для того, щоб досягти підвищення продуктивності, необхідно максимально заповнити конвеєр та виконувати максимальну кількість інструкцій паралельно.

Під час розробки процесора необхідно брати до уваги ціну його розробки. Конфігуровані процесори дозволяють значно зменшити ціну та час розробки. Для таких процесорів на етапі розробки можливо змінювати систему команд, ширину шин та інші параметри.

Велику частину програмного коду займають різноманітні переходи та цикли. Цикли є однією з причин простоювань конвеєрів та зменшення паралелізму на рівні інструкцій. Існують різноманітні підходи оптимізації циклів, одним з яких є розгортка циклу. Розгортка циклу (loop unrolling) – метод, який дозволяє оптимізувати частини програми.

Основними наслідками такого підходу є [3] :

1. Збільшення числа регістрів для зберігання проміжних результатів, що може викликати зменшення продуктивності.

2. Збільшення коду програми, що є особливо несприятливим для вбудованих систем.

Загалом цей підхід є програмним – розгортка циклу виконується на етапі компіляції. Підвищити ефективність цього підходу можливо за допомогою апаратної підтримки циклів. Це дозволить підвищити ефективність конвеєра і створить підґрунтя для динамічного розгортання циклу.

У сучасних процесорах обробки сигналів реалізовано апаратну підтримку виконання циклів. Це досягається за допомогою спеціальних інструкцій для роботи з циклами. Як приклад можна навести процесори ADSP 219x та процесори TMS320 [1,2].

Огляд існуючих рішень. Як вже було наведено вище, існують сучасні процесори, які містять блоки апаратних циклів. У них введено спеціальні інструкції для роботи з циклами.

У процесорах ADSP 219x реалізовано три стеки для циклів. Під час виконання інструкції DO UNTIL автоматично заносяться дані в такі три стеки.

1. Стек початку адреси циклу. Заноситься початкова адреса циклу та поточне значення програмного лічильника.

2. Стек кінця циклу. Заноситься кінцева адреса циклу.

3. Стек ітерацій. Заноситься значення з регістра CNTR для визначених циклів. Якщо цикл не визначений, то дані заносяться і декрементуються, але результат ігнорується [1].

У процесорі TMS 320 для реалізації апаратної підтримки циклів реалізовані такі інструкції:

1. Повторення блока інструкцій без умови.

2. Повторення інструкції без умови.

3. Повторення інструкції з умовою.

Для реалізації цих інструкцій передбачено декілька регістрів, які зберігають такі дані: кількість ітерацій циклу (BRC) , адресу початкової інструкції циклу (RSA), адресу кінцевої інструкції циклу (REA). Цикл повторюється доки BRC> 0. Передбачена можливість вкладеного циклу. Тіло циклу має бути не більшим за 64К , виконуватись мінімально за 2 цикли, кількість ітерацій має бути не меншою за 3 для того, щоб не було простоювань в конвеєрі циклу. Не дозволяється існування інструкцій галуження в тілі циклу , крім одного [2].

Недоліками такого підходу є те, що не усувається проблема з пам'яттю, визначення параметрів циклу є статичним та не усувається проблема залежності даних.

Ще один підхід пропонується в [5]. Для роботи апаратного циклу вводиться спеціальна інструкція

LSETUP (ПЛ вершина, ПЛ кінець)

Лічильник = X

Для роботи апаратного блока циклу введено декілька регістрів, які названі архітектурними регістрами. Особливістю цього підходу є застосування додаткових регістрів. Оскільки занесення даних відбувається, коли підтверджується інструкція LSETUP, тобто на останній стадії конвеєра WB, то виникає проблема підтвердження інструкції LSETUP. Вона виникає через те, що інструкція тіла циклу може почати оброблятися в конвеєрі до підтвердження інструкції LSETUP. Проблема постає гостріше з ростом довжини конвеєра. Для вирішення цієї проблеми вводяться так звані ранні конвеєри – EBot, ETop, ECnt. Введення таких регістрів дозволяє уникнути погіршення швидкодії через зупинення конвеєра для встановлення архітектурних регістрів. У цьому разі значення початкової та кінцевої адреси обраховуються на стадії обрахування адреси. На стадії виконання ці дані заносяться в ранні регістри. Обрахунок ведеться відносно значення програмного лічильника (ПЛ). Отже, обраховуються початкова та кінцева інструкції тіла циклу та кількість ітерацій циклу. У цьому підході також пропонуються рішення, пов'язані з перериваннями. Недоліками такого підходу є відсутність роботи з вкладеними циклами та невирішення проблеми з пам'яттю.

Інший підхід в побудові апаратного пристрою циклів розглянуто в [6]. В цьому підході пропонується використовувати спеціальний буфер. У цей буфер заноситься цикл, доки він не буде виконаний. Поки виконується цикл в буфері, пам'ять відключається, за допомогою чого заощаджується енергія. Недоліками такого підходу є те, що, якщо тіло циклу більше за буфер, то виконання відбувається звичайним неоптимізованим чином. Крім того, в цьому підході не підтримуються вкладені цикли.

У [7] наведена реалізація апаратного блока циклу мовою VHDL. Цю реалізацію можна використовувати для порівняння та аналізу, моделюючи в пакеті Xilinx Web Pack. Структура цього блока циклу зображена на рис. 1.

Головними перевагами запропонованого блока апаратного циклу є простота та можливість підтримки будь-якої кількості вкладених циклів. Користувач може згенерувати даний блок з різною кількістю вкладених циклів. Індекси циклів генерують кожен такт відповідно до меж кожного вкладеного циклу. Пріоритетний декодер, схема якого зображена на рис. 2, виконує керування інкрементуванням індексів кожного циклу. На вхід даний вузол отримує значення порівняння значення попереднього індексу з граничним значенням циклу. Декодер генерує сигнали скиду для індексів та сигнали дозволу на збільшення значення. Цей пріоритетний декодер видає сигнал закінчення циклу при проходженні всіх ітерацій зовнішнього циклу. Блок апаратного циклу може використовуватися для генерування значень змінних циклу, які можна зберігати в регістровому файлі. На вхід даному блоку даються значення границь для кожного циклу (кількість ітерацій) [7].

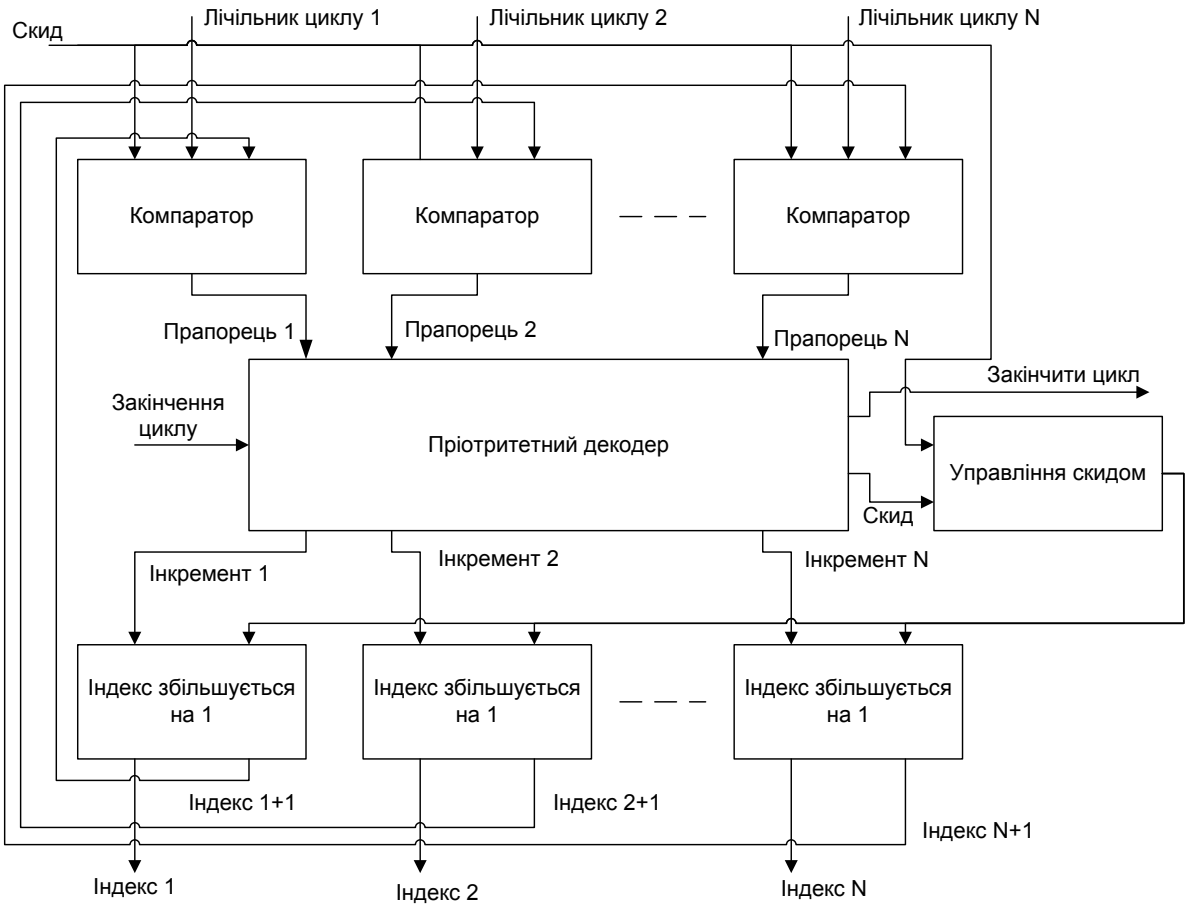

Рис. 1. Схема блока апаратного циклу для порівняння

Недоліками такого блока є те, що він працює за тактовими імпульсами, а отже, він може обробляти цикли, тіло яких складається лише з однієї інструкції. Іншим недоліком є те, що вкладеність має бути одна за одною, тобто на одному ступені вкладеності може бути лише один цикл.

На рис. 2 зображена схема пріоритетного декодера блока апаратного циклу.

Постановка проблеми. Аналізуючи виконання циклу в конвеєрі, можна побачити проблему завантаження конвеєра інструкціями, оскільки цикли є послідовністю інструкцій з умовним переходом в кінці циклу на початок циклу. Цей перехід буде викликати незаповненість конвеєра та появу так званих “бульбашок”. Відбувається багатократне вибирання одних і тих же інструкцій з пам’яті. Це є не раціональним адже звернення до пам’яті погіршує швидкодію і збільшує споживану потужність [4].

Отже можна виділити такі негативні наслідки існування циклів у програмі:

1. Існування умовного переходу і залежності за керуванням.
2. Багаторазові звернення до пам’яті для вибирання однакових інструкцій.
3. Збільшення споживаної потужності через звернення до пам’яті.
4. Повторне декодування інструкцій.
5. Існування залежностей за даними, які важко усунути, змінюючи послідовність інструкцій.

У цій роботі пропонується вирішити проблему залежності потоку керування для того, щоб зняти основний негативний наслідок – зниження продуктивності за рахунок простоювання конвеєра. Розглядаються добре структуровані цикли – цикли, для яких на етапі компіляції відомі адреси початку і кінця циклу та кількість ітерацій, в тілі циклу немає інших переходів. Цикли можуть бути вкладеними.

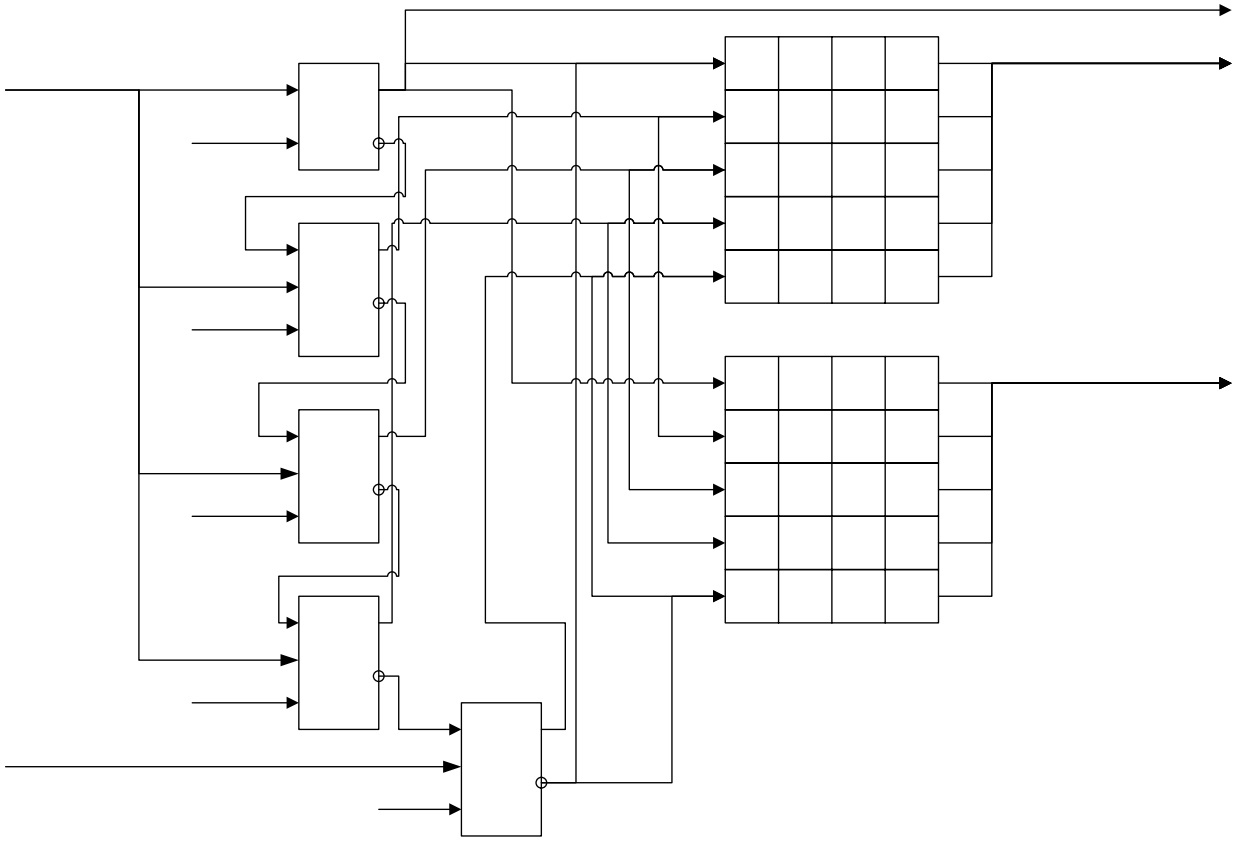

Рис. 2. Схема пріоритетного декодера апаратного циклу аналога

Пропонована організація блока апаратної реалізації циклів. Наведемо приклад циклу мовою високого рівня С.

Прапорець

For(i=0;i<5;i++)

For(j=0;j<6;j++)

A[i][j]=B[i][j]+C[j]

У цьому випадку наведено приклад додавання двох матриць. Нижче наведено два варіанти реалізації цих циклів на асемблері.

Код для процесора
без апаратної підтримки циклів

```
0x000c    ld   r1, 0
0x0010    ld   r2, 0
0x0014    mul r3, r2, r1
0x0018    ld    r5, 0x0100(r3)
0x001c    ld   r6, 0x0200(r2)
0x0020    add r4, r5, r6
0x0024    sw  r4, 0x0300(r3)
0x0028    inc  r2
0x002c    inc  r1
0x0030    sub r7, r2, 6
0x0034    bnez r7, 0x0014
0x0038    sub r8, r1, 5
0x003c    bnez r8, 0x0010
```

Код для процесора
з апаратною підтримкою циклів

```
0x000c    lst  0x14, 0x24, 5
0x0010    ld   r1, 0
0x0014    lst  0x1c, 0x18, 6
0x0018    ld   r2, 0
0x001c    mul r3, r2, r1
0x0020    ld    r5, 0x0100(r3)
0x0024    ld   r6, 0x0200(r2)
0x0028    add r4, r5, r6
0x002c    sw  r4, 0x0300(r3)
0x0030    inc  r2
0x0034    inc  r1
0x0038
```

У коді використані такі регістри загального призначення:
r1 – індекс першого масиву i
r2 – індекс другого масиву j
r3 – індекс зі зміщенням для двомірного масиву
r4 – масив A
r5 – масив B
r6 – масив C
r7, r8 – для перевірки кінця циклу (у варіанті без апаратної підтримки циклів)

Вище показано приклад коду на асемблері без використання апаратних циклів. Цикл реалізується за допомогою умовного переходу в кінці тіла кожного з циклів. Порівняння з границею циклу реалізовано через віднімання поточної ітерації з границею, яка задається безпосередньо числом.

Зображено код на асемблері для процесора з підтримкою апаратних циклів. Як видно з коду для цього вводиться додаткова інструкція.

lst start address, offset, count.

Операнди цієї інструкції надсилаються в спеціальний блок процесора – блок апаратного циклу.

Пропонується реалізація цього блока апаратного циклу, який ініціалізується блоком управління при надходженні інструкції **lst**. Операндами інструкції lst є адреса початку циклу, адреса кінця циклу, кількість ітерацій. Формат інструкції виглядає так.

Код операції	Адреса початку	Адреса зміщення кінця	Кількість ітерацій

Код операції – 6 біт
Адреса початку – 8 біт (64 інструкції)
Адреса кінця – 8 біт (64 інструкції зміщення)
Кількість ітерацій – 10 біт (максимальна кількість ітерацій 1024)
Загальна схема пристрою зображена на рис. 3.

При декодуванні інструкції **lst** ініціалізується блок циклу, тобто передаються операнди на вхід. Операндами цієї інструкції є початкова адреса, кінцева адреса циклу та кількість ітерацій. Коли виконується цикл у блок керування з блока циклу надсилається сигнал Looping, та формується наступна адреса вибірки інструкції. Ця адреса передається на вхід програмного лічильника. Програмний лічильник керується сигналами від блока керування. Блок керування, отримуючи сигнал про виконання циклу Looping, надсилає команду програмному лічильнику, щоб він сформував наступну адресу на основі NPC від блока циклу. Адже блок циклу форму є лише 8 молодших розрядів.

На вхід блока циклу передаються Context – операнди інструкції lst, WRContext – сигнал дозволу записування даних з Context, PC – 8 молодших байт наступної адреси інструкції, RST – сигнал скиду блока. Вихідними сигналами блока є NPC – скоректоване значення – 8 молодших байт наступної адреси інструкції, Looping – сигнал про обробку циклів у пристрої. Блок керування дає дозвіл на зчитування інструкції з пам’яті інструкції та даних з пам’яті даних. Після надходження даних у блок декодера вона там декодується та відповідно до того, чи це була інструкція циклу блок керування генерує сигнал WRContext.

У результаті зображений код буде виконуватись так:

1. Після декодування інструкції lst 0x14, 0x24, 5, адреса початку 0x14, та зміщення 0x24 та кількість ітерацій 5 надходить на вхід блока циклу. Блок циклу формує сигнал виконання циклу Looping. І в якості наступної адреси передає 0x0010.

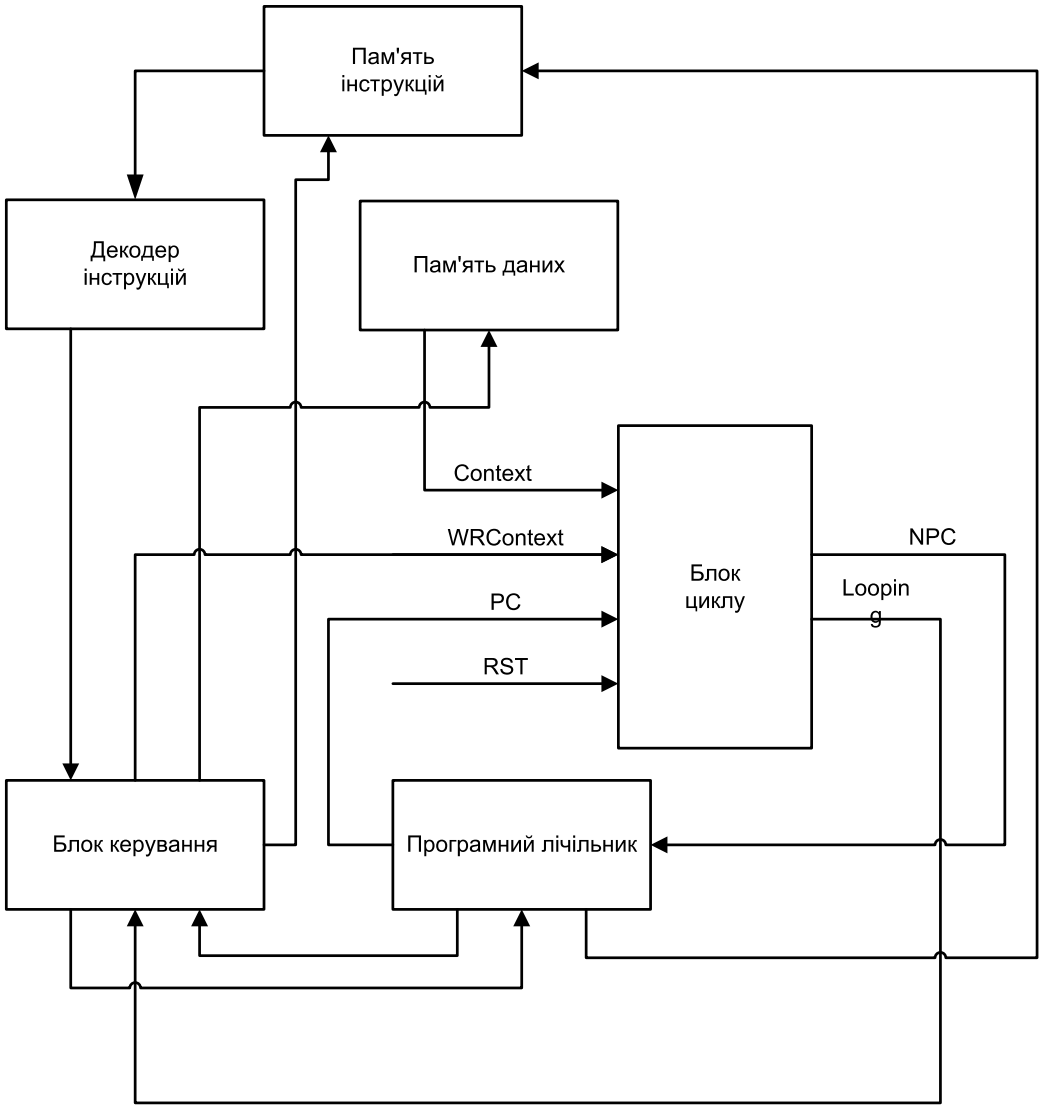

Рис. 3. Загальна схема вибірки команд з блоком апаратних циклів

2. Після декодування інструкції lst 0x1c, 0x18, 6 адреса початку 0x1c, та зміщення 0x18 та кількість ітерацій 6 надходить на вхід блока циклу.

3. Коли досягається адреса 0x1c, а кількість ітерацій першого циклу не виконано, блок циклу формує адресу 0x0018.

4. Коли досягається адреса 0x0034, а кількість ітерацій першого циклу виконано, а другого не виконано, блок циклу формує адресу 0x0014 та виштовхує дані про перший цикл. Далі повертаємось на пункт 2.

5. Коли досягається адреса 0x1c, а кількість ітерацій першого і другого циклу виконано, блок циклу формує адресу 0x0038 та знімає сигнал Looping.

На рис. 4 показано схему пристрою апаратного циклу, а на рис. 5 – блок-схему роботи.

Цей пристрій працює так. У разі появи сигналу про надходження циклу WriteContext відбувається записування даних у стек Reg1-Reg4. Після чого вказівник стеку переміщується вгору, а зчитування відбувається з комірки з на 1 меншою адресою. В стеку зберігаються дані про поточну ітерацію, початкову та кінцеву адреси. При досягненні кінця циклу відбувається зменшення вмісту лічильника ітерацій, коли програмний лічильник = кінцевій адресі. Коли вміст лічильника ітерацій починає дорівнювати 0, дані виштовхуються зі стеку, та вказівник перемішується на одиницю вниз. Якщо він досягнув останньої адреси, тобто 1, то вказівник на зчитування буде = 0. У цьому випадку формується сигнал завершення обробки циклу Looping. Коли досягається кінець циклу встановлюється наступна адреса, що дорівнює початковій адресі циклу.

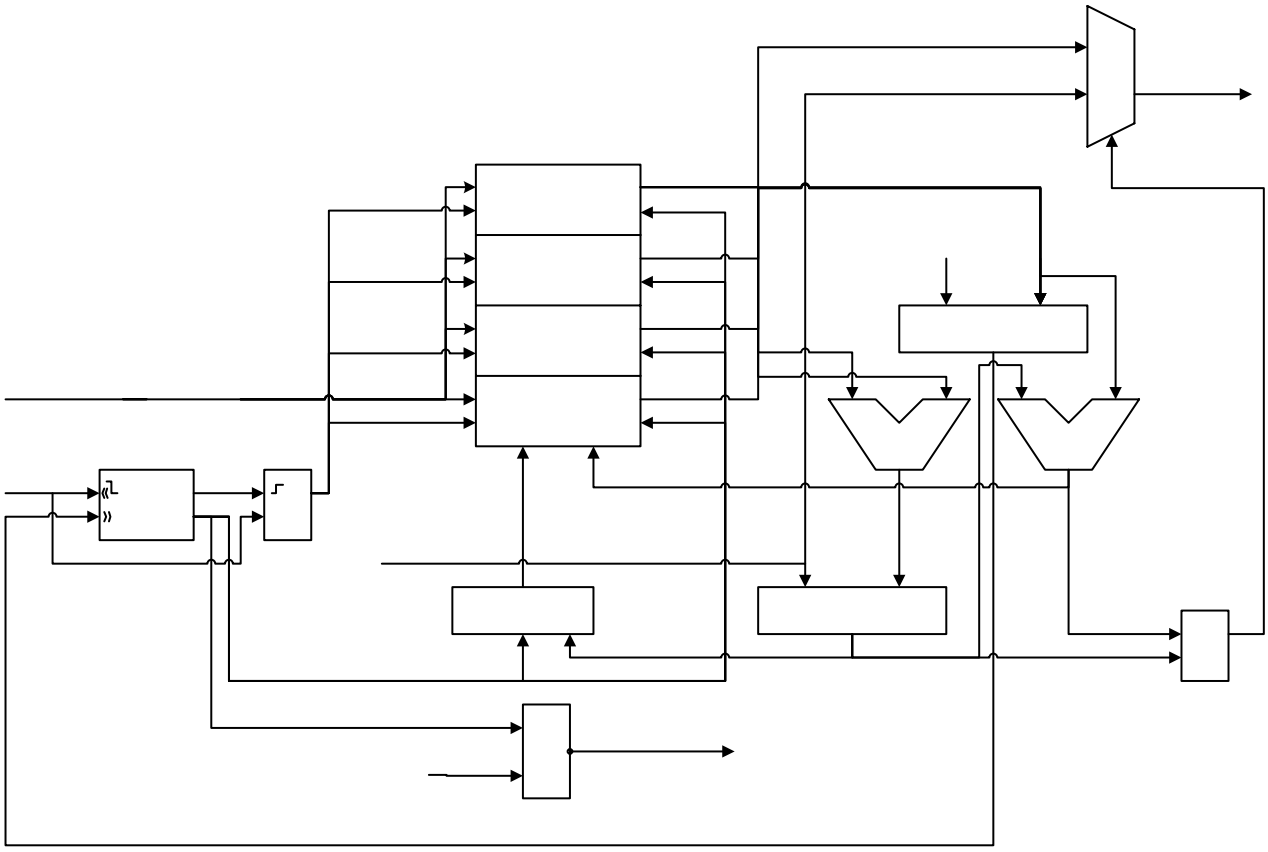

Рис. 4. Структурна схема досліджуваного пристрою апаратного циклу

Цей пристрій спроектовано так, щоб можна було змінити його конфігурацію залежно від поставленої задачі. Змінюючи розмір стеку можна змінювати глибину вкладеності циклів. Стек інструкцій являє собою регулярну структуру, яка може генеруватися VHDL оператором generate. Константою VKLADENIST можна задати кількість елементів стеку. Елементом стеку є комірка пам'яті, яка складається з полів – початкова, кінцева адреса та лічильник ітерацій. Крім того константа VKLADENIST – значення розрядності елементу зсуву та елементів І. Отже, змінивши лише одну константу, можна розширити ступінь вкладеності циклів для цього пристрою.

Змінюючи константи, які визначають розрядність полів стеку інструкцій STARTADDR, ENDADDR та COUNT, можна змінювати розрядність операндів інструкції. Ці константи визначають розрядність шин даних у пристрої, які будуть автоматично змінюватись при зміні констант.

Отже, для зміни глибини вкладеності циклів та розрядності інструкції необхідно змінити лише три константи в VHDL коді і перекомпілювати його.

Результати, отримані в пакеті Xilinx Web Pack під час реалізації цього пристрою на ПЛІС Virtex II, наведено в таблиці.

Результати реалізації блоків апаратних циклів

	Блок аналог	Запропонований блок.
Максимальна частота	125.549MHz	169.578MHz
Кількість Slices	16%	38%
Кількість Flip Flops	7%	31%

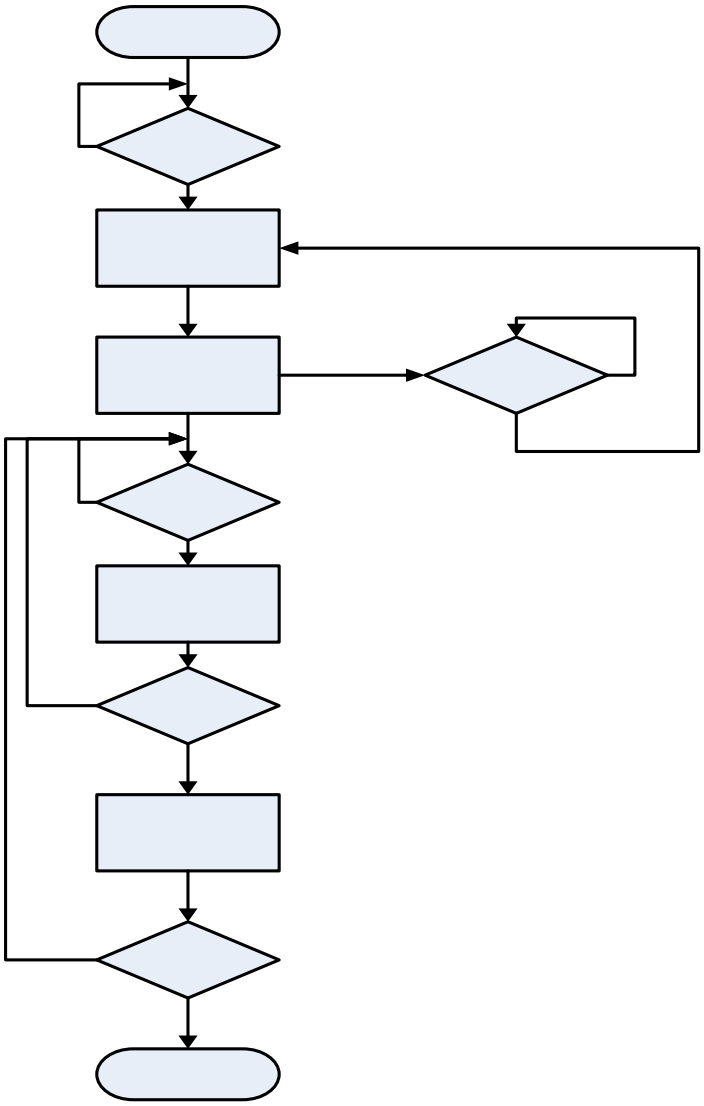

Рис. 5. Блок схема алгоритму роботи пристрою апаратних циклів

Висновки. У роботі запропоновано підхід до покращання продуктивності процесорів під час виконання циклів. Для цього пропонується використовувати апаратні цикли. Розглянуто існуючі процесори з підтримкою апаратних циклів. Проаналізовано апаратні цикли та описання блока апаратних циклів мовою VHDL. Запропоновано блок апаратних циклів з підтримкою вкладеності та можливістю зміни ступеня вкладеності.

Запропонований блок має такі переваги:

1. Звернення відбувається лише до однієї комірки пам’яті, а не до всіх, що може істотно позначитись на продуктивності у разі збільшення вкладеності циклів.

2. З першого пункту також випливає більша енергозаощадливість.

3. Дуже простий спосіб конфігурації пристрою без використання додаткових інструментів.

4. Запропонований пристрій не тільки обраховує лічильники циклів, а визначає адресу наступної інструкції тіла циклу. Збільшення лічильника відбувається не по тактовому імпульсу, а при досягненні кінця циклу.

1. ADSP-219x DSP Instruction Set Reference. 2. TMS320C55x DSP Mnemonic Instruction Set Reference Guide. 3. http://en.wikipedia.org/wiki/Loop_unrolling. 4. Шнитман В.З. Современные высокопроизводительные компьютеры. 5. Ravi P. Singh, Charles P. Roth, Gregory Overkamp: Hardware Loops – United States Patent – 2005 6. Lauren Wojcieszak, Andrew Cofler: Multiple Execution Of Instruction Loops Within A Processor Without Accessing Program Memory – United States Patent – 2005. 7. Nikolaos Kavvadias: Hardware looping Unit – www.opencores.org, April 13, 2004