

## ОРГАНІЗАЦІЯ БАГАТОРІВНЕВОЇ ПАМ'ЯТІ В РЕКОНФІГУРОВАНИХ ОБЧИСЛЮВАЛЬНИХ СИСТЕМАХ

Запропоновані засоби організації багаторівневої пам'яті реконфігурованої обчислювальної системи, для підтримки розкладу зберігання конфігурацій апаратних задач на локальному рівні обчислювального модуля, зменшуючи накладні витрати реконфігурації і вирішуючи проблему обмеженості внутрішньої пам'яті ПЛІС.

The multilevel memory of reconfigurable computing system to support the scheduling of storage hardware tasks configuration at the local level computing module are proposed, reducing the reconfiguration overhead and solving the problem of limited internal memory of the FPGA.

### 1. Вступ

На протязі останніх декількох років реконфігуровані обчислювальні системи все більше проявляють риси паралельних обчислювальних систем загального призначення і носять назву реконфігурованих суперкомп'ютерів або високопродуктивних реконфігурованих комп'ютерів (*High-Performance Reconfigurable Computers, HPRC*) [1, 2, 3].

Ефективним механізмом, що забезпечує гнучкість архітектури високопродуктивних реконфігурованих комп'ютерів є динамічна реконфігурація часу виконання (*RunTime* реконфігурація) [1, 4, 5, 6]. При цьому негативним фактором, що впливає на загальну продуктивність обчислень є накладні витрати реконфігурації, найбільш критичною складовою яких є час, витрачений на реконфігурацію і її підтримку.

Сучасні технології провідних виробників ПЛІС пропонують ефективні механізми зниження затримок реконфігурації за рахунок часткової динамічної реконфігурації, коли перезавантажується лише частина мікросхеми [7]. Вбачається, що саме використання часткової динамічної реконфігурації лежить в основі концепції підвищення продуктивності реконфігурованих суперкомп'ютерів [1, 2, 3, 4, 8].

Надалі, засоби часткової реконфігурації надають можливість внутрішнього конфігурування мікросхеми [4, 9]. Це сприяє значному зменшенню часу реконфігурації, за умови розміщення бітових конфігураційних потоків у внутрішній пам'яті мікросхеми. Але інша проблема обмеженості ресурсів внутрішньої пам'яті мікросхеми ПЛІС не дозволяє зберігати велику кількість конфігураційних даних і все ж вима-

гає залучення зовнішньої пам'яті, що спричинює додаткові часові витрати.

Таким чином, незважаючи на переваги технології часткової динамічної реконфігурації витрати часу на реконфігурацію і обмежені апаратні ресурси внутрішньої пам'яті залишаються значною проблемою, що впливає на досягнення високої продуктивності реконфігурованих обчислювальних систем. Вирішення цієї проблеми є важливим аспектом в сфері високопродуктивних реконфігурованих обчислень, що обумовлює актуальність та доцільність проведених досліджень.

### 2. Огляд відомих рішень

На ліквідацію накладних витрат реконфігурації в останні роки покладено багато дослідницьких та наукових зусиль. В ряді робіт [10, 11, 12, 13] дослідження спрямовані на рішення проблеми зменшення накладних витрат реконфігурації в області вбудованих реконфігурованих систем, які базуються, як на повній так і на частковій реконфігурації.

Також в літературі представлено багато рішень для підвищення ефективності використання часткової динамічної реконфігурації в високопродуктивних реконфігурованих комп'ютерах. В цій області зазвичай дослідження спрямовані на розробки алгоритмів планування і розміщення задач, що водночас реалізують різні методи зменшення витрат реконфігурації. Найбільш розповсюдженими методами є попередня вибірка конфігурацій [1, 14], повторне використання обчислювальних ресурсів [3, 6, 8], поєднання конфігурацій з метою зменшення реконфігураційних витрат і руху даних [1, 2]. Алгоритми представлені авторами робіт [1, 2, 14] випробувані на існуючих

високопродуктивних платформах *Cray XD1*, *SRC-6* [15], обмеження передумовленої архітектури яких негативно вплинули на гнучкість реалізації та отримані результати. Переважна кількість оглянутих розробок базуються на програмному або експериментальному моделюванні процесу реконфігурації, часто абстраговано від фізичної сторони реалізації [3, 6, 8, 10]. Інші розглядають загальну архітектурну модель обчислювальної системи, ігноруючи її реальні особливості, на кшталт структури реконфігурованої області, організації комунікаційного середовища та пам'яті, комунікаційних процесів, конфліктів загальних ресурсів, тощо [2, 8].

Ряд робіт, які пропонують архітектурні та технічні вдосконалення базових технологій часткової реконфігурації, базуються на апаратних методах кешування та віртуальних моделях пам'яті, засобах збільшення пропускної здатності інтерфейсів реконфігурації. В роботі [8] на ряді з існуванням двох процесорів для здійснення управління реконфігурацією і виконання програмних функцій передбачено окремий апаратний контролер реконфігурації, який звільнює центральний управляючий процесор від витрат на реконфігурацію і збільшує пропускну здатність реконфігурації, сприяючи прискоренню обчислень. Автори робіт [16, 17] для підвищення швидкості завантаження конфігураційних потоків пропонують підхід прямого доступу до пам'яті конфігурацій. Розміщення блоку кеш пам'яті поряд з портом *ICAP* призводить до подальшого збільшення продуктивності [17]. Однак, розміщення пам'яті великого об'єму на ПЛІС виявляється неефективним. В роботі [9] прискорення реконфігурації здійснюється за рахунок реалізації пристрою реконфігурації для зв'язку *ICAP* та зовнішньої *SRAM* в режимі прямого доступу, що дозволяє забезпечити швидкість передавання конфігураційного потоку близької до ідеальної – 400 Мбайт/с. Для нових поколінь ПЛІС автори очікують досягнення пропускної здатності в декілька Гбайт/с.

Таким чином, підтримка засобів зменшення накладних витрат, вбудована в алгоритми планування та розміщення задач, зазвичай покладається на рівень програмної або програмно-апаратної надбудови операційної системи [8], що потребує додаткових витрат продуктивності.

З іншого боку, переважно всі відомі архітектурні засоби відокремлені від рішення задач планування та розміщення, що неефективно для

реконфігурованих обчислювальних систем, де всі засоби забезпечення послідовності реконфігурації сильно залежать один від одного і у більшості випадків представляють єдиний механізм.

### 3. Постановка задачі

Визначимо сумарний час виконання задачі  $T_{Sum}$ , як суму двох складових часу, які впливають на загальну продуктивність реконфігурованих обчислень:

$$T_{Sum} = T_{Count} + T_{Rconf},$$

де  $T_{Count}$  – час виконання задачі апаратними засобами,  $T_{Rconf}$  – час реконфігурації апаратної задачі на ПЛІС.

Складова  $T_{Count}$  визначає позитивний ефект від традиційних способів зменшення часу виконання задач, шляхом поглиблення паралелізму та їх апаратного прискорення. Таке прискорення очікуване від застосування реконфігурованих обчислень. Складова  $T_{Rconf}$  є непродуктивною складовою, яка визначає час витрачений саме на процес реконфігурації апаратури. Ця складова обумовлює втрату продуктивності під час реконфігурованих обчислень і вимірюється накладними видатками. При цьому в деяких випадках може бути порівняний час апаратного прискорення задач і час їх конфігурації на апаратурі [1].

Час реконфігурації в свою чергу складається наступних узагальнених компонент: часу виконання логічної та фізичної послідовності реконфігураційного процесу, що забезпечується на програмному рівні алгоритмами планування та розподілу задач; часу передавання конфігураційних даних, які можуть завантажуватись із загальної системної бібліотеки конфігураційних файлів, із зовнішньої локальної пам'яті обчислювального модуля або із внутрішньої пам'яті ПЛІС, часу прошивання мікросхеми. Тоді до найбільш вагомих компонент накладних витрат віднесемо втрати продуктивності на забезпечення обчислювальної складності алгоритмів планування та розміщення задач, та час витрачений на передавання і завантаження конфігураційних даних.

В якості реконфігурованого простору приймемо динамічну частину ПЛІС, призначену для реконфігурації [4], яку визначимо як реконфігуровану область. Реконфігурована область має розмір і структуру. Розмір залежить від можливостей використовуваного сімейства ПЛІС і

архітектури реконфігурованої системи. Структура визначається використовуваною моделлю розміщення задач. Сучасні технології ПЛІС пропонують 1D і 2D моделі розміщення задач [3, 4], коли задачі можуть бути розміщені стовбцями різної ширини на всю висоту реконфігурованої області або прямокутниками фіксованого або довільного розміру та місця розташування.

В якості абстрактної моделі обчислювальної задачі розглядаємо функцією або так зване функціональне ядро, синтезоване в цифрову схему з метою розміщення в реконфігурованій області. Визначимо її, як апаратну задачу. Застосуємо традиційну модель задачі абстрактного рівня [3], коли задача характеризується розміром, формою відображення і часом виконання. Розмір визначає просторові вимоги до реконфігурованої області. Форма відображення залежить від моделі розміщення задач в реконфігурованій області. Для всіх моделей розміщення узагальнимо, що задача має форму прямокутника. Тоді задача визначається як вектор

$$T_i = \{S_i, T_{Sumi} | (T_{Counti} + T_{Rconfigi}), I_i, N_i\}, \quad (1)$$

де  $S_i$  – площа прямокутника, що містить задачу  $T_i$ , а час реконфігурації  $T_{Rconfigi}$  це частина часу її виконання  $T_{Sumi}$ ,  $I_i$  – тип операції, що виконує задача, і  $N_i$  – унікальний ідентифікатор задачі у вихідному графі алгоритму,  $i = 1, N_{max}$ ,  $N_{max}$  – кількість вершин графу.

Вважаємо що задачі розміщуються на поверхні реконфігурованої області згідно деякому алгоритму планування та розміщення задач з урахуванням їх просторових параметрів (1). Основною ціллю досліджень є зменшення, що сприятиме загальному прискоренню обчислень.

Для забезпечення очікуваного ефекту від застосування реконфігурованих обчислень з точки зору продуктивності, має бути вирішена задача зменшення компонент часу реконфігурації  $T_{Rconfig}$ , принаймні в степені порівняній з прискоренням обчислень, що зводитиме до нуля накладні витрати реконфігурації.

#### 4. Вирішення проблеми обмеженого об'єму внутрішньої пам'яті ПЛІС

Алгоритми попередньої вибірки конфігурацій, повторного використання обчислювальних ресурсів, динамічної самореконфігурації, засоби дефрагментації реконфігурованої області і витіснення задач надають максимальні показ-

ники продуктивності забезпечуючи швидкість завантаження конфігурацій, шляхом зберігання їх у внутрішні пам'яті ПЛІС. Але це досягається ціною непродуктивного використання обмежених та коштовних ресурсів внутрішньої пам'яті ПЛІС. Кількісні оцінки використання внутрішньої пам'яті для забезпечення переваг часткової реконфігурації описана в роботі [18], видно, що обмежені ресурси внутрішньої пам'яті ПЛІС є основною перешкодою, яка не дозволяють зберігати велику кількість конфігураційних даних.

Об'єм внутрішньої блочної пам'яті розповсюджених в реалізації реконфігурованих пристроїв сімейств мікросхем Virtex 4 108Кб – 756Кб, Virtex E 4Кб – 16Кб. Розмір повного конфігураційного потоку мікросхем сімейства Virtex E досягає 68Кб – 748 Кб [19]. За даними роботи [18] розмір конфігураційних файлів реконфігурованих лічильників, що відрізняються довжиною лічильного регістру 24Кб – 131Кб. При цьому найменший зконфігурований модуль займає 0.36 % логічного ресурсу мікросхеми Virtex-5 FX70, найбільший – 10 %.

З вищесказаного видно, що ціна використання ресурсів ПЛІС під час зберігання вже зконфігурованих апаратних задач значно менша, ніж за зберігання їх у вигляді конфігураційних файлів у внутрішній пам'яті. На цьому базується запропонована в статті концепція, подолання накладних витрат реконфігурації за рахунок зменшення часу передавання і завантаження конфігураційних даних. При цьому для тимчасового зберігання конфігурацій запропоновано використовувати поверхню реконфігурованої області і зберігати апаратні ресурси в заздалегідь зконфігурованому вигляді, вирішуючи проблему обмежених ресурсів внутрішньої пам'яті.

#### 5. Модель реконфігурованого обчислювального модуля

Представлені в статті дослідження орієнтовані на застосування в масштабованих реконфігурованих суперкомп'ютерах, побудованих за принципом відкритої архітектури та модульної організації. Такі архітектури зазвичай містять центральний процесор і загальну системну пам'ять. Обчислювальні модулі під'єднані до центральних засобів управління та загальної пам'яті за допомогою високошвидкісних мережевих засобів (4).

Структура запропонованого обчислювального модуля представлена на рис. 1.

На відміну від загальноприйнятої моделі обчислювального модуля, у склад якої зазвичай входить управляючий процесор (УП), енергонезалежна пам'яті для збереження локальних програм та даних (ЛП) і реконфігурований модуль, що реалізований на ПЛІС (РМ), у структурі обчислювального модуля запропоновано використання багаторівневої пам'яті. Багаторівнева пам'ять розміщується між основною системною пам'яттю та внутрішніми засобами

управління логікою реконфігурації (Рис.1). Для реалізації багаторівневої пам'яті запропоновано використання традиційної технології кеш-пам'яті. До складу реконфігурованого модуля також додано спеціальний контролер реконфігурації, з метою зменшення часу реконфігурації за рахунок розділення логічного і фізичного рівнів реконфігурації і розвантажування управляючого процесора [8].

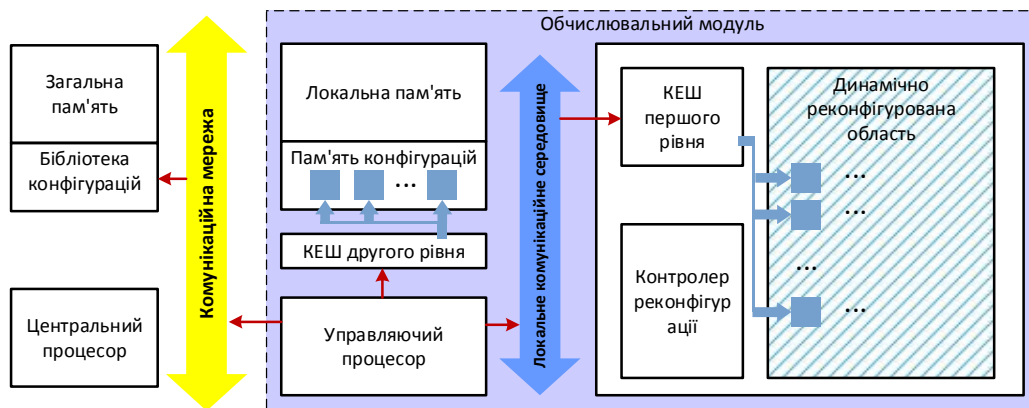


Рис. 1. Архітектура реконфігурованої обчислювальної системи

Реконфігурований модуль складається зі статичної і динамічної області. Динамічна область призначена для розміщення апаратних задач у вигляді відповідно зконфігурованих функціональних блоків (ФБ). Під ФБ ми розуміємо структурне визначення задачі, що є частиною реконфігурованої області в яка містить апаратну задачу. В статичній області ПЛІС розміщується кеш пам'ять першого рівня і контролер управління фізичною послідовністю реконфігурації.

Динамічна область разом із своїм основним обчислювальним призначенням одночасно виконує функцію зберігання функціональних блоків для їх подальшого використання. Конфігурації ФБ зберігаються на поверхні реконфігурованої області за певним розкладом. Кеш пам'ять використовується для обліку управляючої інформації і підтримки розкладу реконфігурації функціональних блоків. Фактично кеш пам'ять містить покажчики на функціональні блоки, що зберігаються в реконфігурованій області, та інформацію щодо їх поточного стану.

Первинне завантаження конфігурацій ще більше уповільнює реконфігурацію за рахунок звернення до загальної системної пам'яті, де зберігається централізована бібліотека конфігурацій функціональних блоків. Для запобігання великої кількості звернень до загальної пам'яті запропонований кеш другого рівня, де зберіга-

ються одного разу вже застосовані конфігураційні файли. Локальна пам'ять конфігурацій реалізована у складі локальної пам'яті обчислювального модуля, яка не так критична до об'єму, ніж внутрішня пам'ять ПЛІС. Але, як вже було зазначено, обмін конфігураційними даними між локальною пам'яттю і реконфігурованою областю буде спричиняти накладні витрати реконфігурації. Кеш пам'ять другого рівня містить покажчики на конфігураційні файли апаратних задач та їх поточний стан.

Алгоритм запуску апаратної задачі на виконання в реконфігурованій області наступний:

Крок 1. Управляючий процесор запускає чергову задачу для обчислення, яка представлена словом команди і словами даних. У форматі команди присутнє поле типу операції, що має бути виконана над вхідними даними. На підставі чого УП формує виконавчу адресу, для пошуку конфігураційних файлів в пам'яті.

Крок 2. Виконавча адреса поступає на вхід кеш-пам'яті першого рівня, здійснюється пошук раніш зконфігурованого функціонального блоку. У випадку «успішного звернення» УП ініціює процедуру запуску задачі в існуючому функціональному блоці.

Крок 3. У випадку «промаху» виконавча адреса поступає на вхід кеш-пам'яті другого рівня в пошуку тимчасово збереженого конфігура-

ційного файлу апаратної задачі. У випадку «успішного звернення» УП ініціює процедуру повторного завантаження конфігураційного файлу із локальної пам'яті на ПЛІС і запуску задачі.

Крок 4. У випадку «промаху» УП завантажує конфігураційний файл на ПЛІС із бібліотеки конфігурацій загальної пам'яті і запускає задачу.

Повернення до кроку 1.

### 6. Модель пам'яті стану функціональних блоків

Виконувана програма у вихідному стані подана у вигляді ациклічного графу  $G$ , в вершинах якого розміщуються обчислювальні задачі. Максимальна кількість вершин графу обмежена величиною  $N_{max}$ . Задачі однозначно визначаються унікальним ідентифікатором задачі у вихідному графі  $N_g | g = \overline{1, N_{max}}$ .

Кожна задача асоціюється з параметром «Тип операції». Тип операції відповідає функції, яку виконує задача, що синтезована в цифрову схему і завантажується в функціональний блок під час реконфігурування обчислень. Апаратні реалізації задач створюються заздалегідь і зберігаються в централізованій бібліотеці конфігурацій у вигляді конфігураційних файлів, кількість яких дорівнює  $M_I$  – по одній для кожного типу операції. Таким чином тип операції  $I_j | j = \overline{1, I_{max}}$ , де  $I_{max}$  – кількість апаратно реалізованих типів операцій, однозначно визначає конфігураційний файл, що завантажується в функціональний блок і використовується як основний параметр для пошуку його конфігурації в пам'яті, або на мікросхемі ПЛІС.

Технологічні обмеження площини реконфігурованої області та розміри конфігурацій обмежують кількість розміщуваних функціональних блоків параметром  $F_{max}$ . Тоді кожний функціональний блок, що розміщений на поверхні реконфігурованої області визначається ідентифікатором  $F_s | s = \overline{1, F_{max}}$ .

Співвідношення між описаними параметрами наведені на рис. 2. Означені співвідношення важливі для формування виконавчої адреси пошуку конфігураційних файлів в локальній та загальній пам'яті і функціональних блоків на поверхні реконфігурованої області. Кожний конфігураційний файл однозначно визначається типом операції. Декілька функціональних блоків можуть виконувати один і той самий тип

операції, відповідно до типу задач. Кожна обчислювальна задача  $N_g$  в визначений момент часу  $T_n$  виконується в окремому функціональному блоці  $F_s$ . Протягом часу в кожному функціональному блоці може бути виконано послідовність задач, у випадку їх повторного використання (рис. 3). Відповідно до цього, конфігураційні файли ідентифікуються значенням типу операції  $I$ . Для ідентифікації зконфігурованих на ПЛІС функціональних блоків використовуємо складений з двох полів ідентифікатор адреси  $I.F$  (рис. 4).

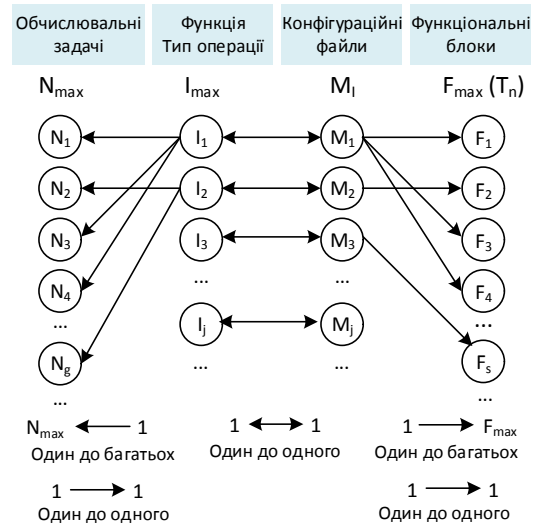


Рис. 2. Співвідношення між основними параметрами

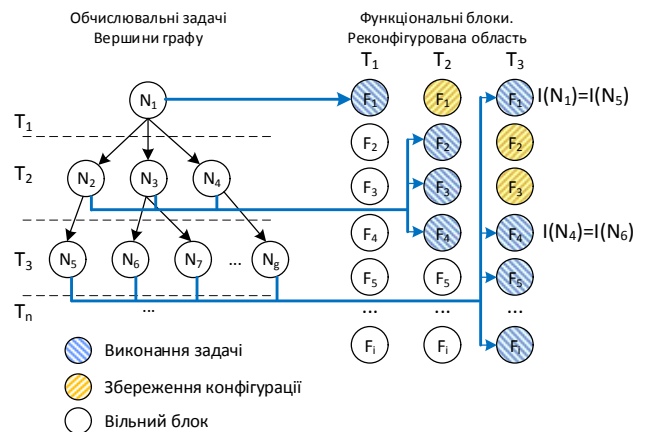


Рис. 3. Принцип завантаження функціональних блоків

Припустимо, що максимальна кількість виконуваних типів операції  $M_{max} = 64$ , максимальна кількість конфігурацій ФБ в одній ПЛІС  $F_{max} = 63$ . Тоді модель пам'яті для зберігання стану розміщених на площині реконфігурованої області ПЛІС знадобиться мінімум 4Кб пам'яті з адресним доступом, за умови мінімального обсягу управляючої інформації, що дорівнює одному байту. За збереження, наприклад чотирьох байтів управляючої інформації знадобить-

ся 16Кб пам'яті. Модель пам'яті з адресним доступом зображена на рис. 5.

Концепцію асоціативного пошуку в багаторівневій пам'яті реалізують класичні технології кеш-пам'яті. Використання їх особливостей, таких як локальність розміщення, прозорість для програмного шару, апаратна реалізація пошуку, швидкість пошуку, відсутність збиткових записів, мобільний розмір, вирішують ряд проблем пов'язаних з накладними видатками реконфігурації і обмеженим ресурсом ПЛІС.

### 7. Вибір моделі кеш пам'яті

Для вирішення поставленої в роботі задачі розглянемо дві типові технології організації

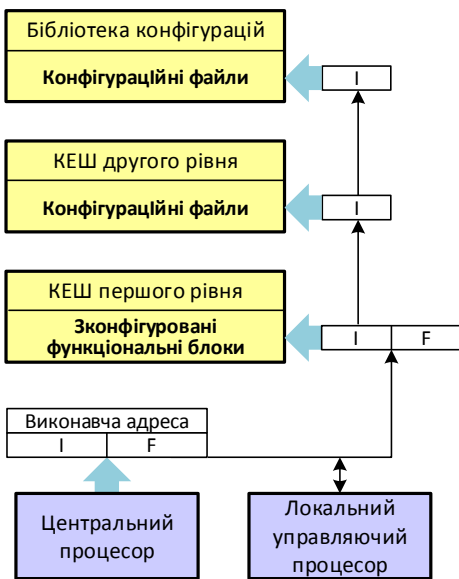


Рис. 4. Формат виконавчої адреси

З огляду на проблему зберігання ресурсів внутрішньої пам'яті ПЛІС, враховуючи, що максимум шістьдесят чотири слова даних із доступних 4Кб будуть використовуватись для збереження станів, в роботі запропоновано використання пам'яті з асоціативним пошуком інформації. Це скоротить об'єм пам'яті станів до 64 байтів. За збільшення довжини слова стану, наприклад до чотирьох байтів об'єм асоціативної пам'яті збільшиться до 256 байт. Що потребуватиме доволі незначних запам'ятовуючих ресурсів мікросхеми ПЛІС.

З іншого боку повторне використання раніш сконфігурованих функціональних блоків та підтримка їх розкладу зберігання, вимагає здійснення пошуку в пам'яті станів шляхом перебору всіх записів пам'яті. Звичайно за застосування пам'яті з адресним доступом час вико-

кеш-пам'яті – повністю асоціативна кеш-пам'ять і кеш-пам'ять із прямим відображенням адрес.

Після формування виконавчої адреси в повністю асоціативній пам'яті її старші біти, що утворюють тег, одночасно порівнюються з всіма тегами записів пам'яті. Якщо всі порівняння призводять до негативного результату, фіксується так званий «кеш-промах», в іншому випадку, коли знайдений один шуканий запис фіксується «кеш-успіх».

		Адреса										Дані		
		Код типу операції (I)					Ідентифікатор ФБ (F)					Поточний стан ФБ		
		ТЕГ					ІНДЕКС							
$f_{64} = 111111$		1	1	1	1	1	1	1	1	1	1	1	Слово стану	4Кб
		1	1	1	1	1	1	1	1	1	1	0	—	
		...	...	...	...	...	...	...	...	...	...	...	...	
$f_0 = 000000$		1	1	1	1	1	0	0	0	0	0	1	Слово стану	0
		1	1	1	1	1	0	0	0	0	0	0	—	
		...	...	...	...	...	...	...	...	...	...	...	...	
		0	0	0	0	0	1	1	1	1	1	1	Слово стану	
		0	0	0	0	0	1	1	1	1	1	0	—	
		...	...	...	...	...	...	...	...	...	...	...	...	
		0	0	0	0	0	0	0	0	0	0	1	Слово стану	
		0	0	0	0	0	0	0	0	0	0	0	—	

Рис. 5. Модель пам'яті

нання такого пошуку буде пропорційний кількості записів, що потребуватиме значного часу.

Асоціативний пошук це типовий засіб спрямований на прискорення пошуку інформації. Для здійснення пошуку в асоціативній пам'яті процесор видає в інтерфейс пам'яті спеціальне слово, яке називається тегом. Під час виконання пошуку заданий тег порівнюється з тегом кожного запису асоціативної пам'яті. Таким чином здійснення пошуку зводиться до одного звернення до пам'яті з боку процесора, що прискорює процедуру пошуку і розвантажує комунікаційне середовище. Сама процедура асоціативного пошуку підтримується апаратно на рівні інтерфейсу пам'яті.

При цьому, якщо в якості тегу використовується ідентифікатор адреси функціонального блоку, складений з двох полів  $I.F$ , ми отриму-



ємо максимальне зменшення об'єму пом'яті. Тоді за видалення збиткових записів об'єм пом'яті дорівнюватиме шістдесят чотири записи, по одній для кожного ФБ. При цьому прискорення пошуку буде досягнуте лише за рахунок зменшення кількості записів. Пошук збереже всі риси пошуку в пам'яті з адресним доступом, таким чином, значне прискорення неможливе.

Пояснимо це наступним чином. Як вже було зазначено між типом виконуваних операцій і ФБ сконфігурованими на ПЛІС і існує зв'язок один-до-багатьох, тобто можлива наявність певної кількості однотипних блоків. Ціллю пошуку є знаходження всіх ФБ на ПЛІС, що виконують певний тип операції. Технологія же повністю асоціативної кеш-пам'яті дозволяє знаходження лише одного запису. В цьому випадку процедуру пошуку доведеться покласти на процесор, що суперечить ідеї досліджень. Використання же в якості тегу лише поля типу операції, можливе, але слова станів всіх однотипних блоків доведеться зберігати в одному рядку кеш-пам'яті. Це потягне за собою ускладнення й уповільнення процедури пошуку та потребуватиме додаткового вдосконалення інтерфейсу кеш-пам'яті. Окрім всього типова апаратура повністю асоціативної кеш-пам'яті будується на застосуванні компараторів, кіль-

кість яких дорівнює кількості записів кеш-пам'яті.

Приведемо також оцінку апаратних ресурсів для реалізації асоціативної кеш-пам'яті. За зроблених припущень знадобиться 64 дванадцятибітні компаратори. Приблизна оцінка затрат устаткування для реалізації одного компаратора складає 10 транзисторів на один біт компаратора [19], таким чином нам знадобиться 7680 транзисторів для реалізації лише схеми порівняння в інтерфейсі кеш-пам'яті. Для реалізації схеми порівняння в типових реалізаціях пам'яті, що використовується в найбільш розповсюджених процесорах, об'ємом 256 Кб з довжиною рядка 32 байти знадобиться 2 211 840 транзисторів.

Поставленим в роботі цілям щодо виконання пошуку найбільш відповідає асоціативна пам'ять з прямим відображенням адрес.

Рядки кеш пам'яті адресуються асоціативним тегом, в якості якого приймаємо тип операції, а ряд індексів, що переадресовується на кожний рядок пам'яті представляє ідентифікатори функціональних блоків. Схема організації кеш-пам'яті наведена на рис. 6. За зроблених припущень кількість рядків, яку визначає розрядність тегу, дорівнює шістдесят чотири.

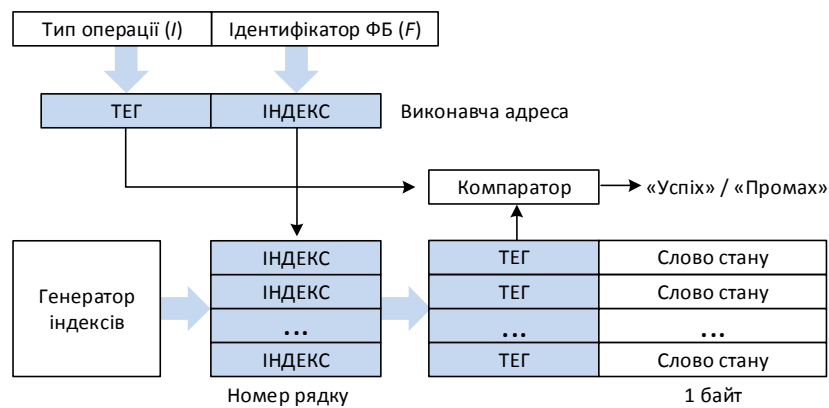


Рис. 6. Структурна схема кеш-пам'яті першого рівня

Стандартний алгоритм функціонування наступний. На адресний вхід інтерфейсу пам'яті поступає виконавча адреса, що складається з полів  $[I.F] = [ТЕГ.ІНДЕКС]$ . Аналізується поле ІНДЕКС, яке вказує на один з ФБ. Надалі старші розряди адреси (ТЕГ), що вказують на шуканий тип операції, порівнюються з тегом, який зберігається в рядку. За збігу формується сигнал «Успіху», інакше – «Проміаху». Така пам'ять потребує мінімального об'єму устатку-

вання. Для порівняння знадобиться лише один компаратор, на вхід якого подається тег КЕШ-рядка обраного за полем «ІНДЕКС».

Таким чином можливо реалізувати наступну логіку цільового пошуку. Фактично задача стоїть в переборі всіх функціональні блоків і пошуку серед них тих, що виконують шуканий тип операції. Надалі розподільувач задач аналізує стан вибраних блоків і обирає серед них блок для завантаження чергової задачі.

Для звільнення процесору від виконання послідовного перебору адрес функціональних блоків, до складу інтерфейсу пам'яті вводиться спеціальний пристрій – генератор адрес (рис. 6). Тоді виконавча адреса, що видається процесором на початку пошуку складається з полів [I.0]. генератор адрес послідовно видає індекси для пошуку, а значення коду операції порівнюється з тегами рядків.

Відомий недолік традиційного застосування кеш-пам'яті з прямим відображенням адрес, коли неможливо одночасно зберігати два рядки з одним і тим самим індексом і різними значеннями тегів, не заважає рішення цільової задачі дослідження, бо така ситуація ніколи не виникає – неможливо виконання декількох операцій одним функціональним блоком.

Записи, що видаляються із кеш-пам'яті потрапляють в кеш-пам'ять другого рівня. Відповідно до цього конфігурації функціональних блоків видаляються з поверхні реконфігурованої області, а конфігураційні дані зберігаються у локальній пам'яті обчислювального модуля.

## 8. Кеш-пам'ять другого рівня

Призначення кеш-пам'яті другого рівня в наданні додаткового часу для зберігання конфігураційних даних на локальному рівні обчислювального модуля. Це сприятиме додатковому зменшенню накладних витрат реконфігурації за рахунок відсутності комунікаційних процедур міжмодульного рівня.

Для реалізації такої кеш-пам'яті застосуємо повністю асоціативну кеш-пам'ять. Обмежена кількість виділеної пам'яті для зберігання конфігураційних файлів обмежує об'єм кеш-пам'яті декількома рядками, що обумовлює доволі незначні апаратні витрати. В якості асоціативного тегу для пошуку застосовується поле типу операції, що однозначно визначає файл конфігурації функціонального блоку.

Розклад зберігання конфігураційних файлів в кеш-пам'яті другого рівня підтримується автоматично стандартними засобами без участі планувальника. В даному випадку доцільна стратегія обмеження кількості рядків пам'яті і витіснення із пам'яті записів або найбільш дав-

ніх записів, або таких що мають меншу ймовірність використання. Для цього існують відомі алгоритми, які можуть бути успішно використані: алгоритм заміщення на основі найбільш давнього застосування (*Least Recently Used, LRU*), - заміщення найменш часто використовуваного рядку (*Least Frequently Used, LFU*).

## 9. Висновки

Накладні витрати реконфігурації є значною проблемою, яка негативним чином впливає на продуктивність реконфігурованих обчислювальних систем. Всі відомі алгоритми планування й розподілу задач, що підтримують різні методи зменшення накладних витрат, реалізовані на абстрактному рівні операційної системи засобами центрального управляючого процесора. Це потребує додаткового часу на їх реалізацію.

Технічна підтримка здійснювана запропонованими засобами спрямована на зменшення часу виконання реконфігурації методом повторного використання ресурсів функціональних блоків.

Запропонована структура обчислювального модуля на базі багаторівневої пам'яті вирішує задачу управління розкладом розміщення та підтримки заздалегідь завантажених функціональних блоків, прозора для програмного шару, локалізовано на рівні апаратури обчислювального модуля. Таким чином, запропоновані засоби зменшують обчислювальну складність алгоритмів управління логічною послідовністю реконфігурації.

Запропонований новий спосіб зберігання конфігурацій апаратних задач на поверхні реконфігурованої області ПЛІС на базі застосування багаторівневої пам'яті, який забезпечує зменшення часу реконфігурації і вирішує проблему обмеженості ресурсів внутрішньої пам'яті ПЛІС. Запропонований спосіб може бути використаний для забезпечення методів повторного використання і попереднього завантаження обчислювальних ресурсів, які широко застосовуються для зменшення накладних витрат реконфігурації.

## Список посилань

1. El-Araby E. Exploiting Partial Runtime Reconfiguration for High-Performance Reconfigurable Computing / E. El-Araby, I. Gonzalez, T. El-Ghazawi // ACM Transactions on Reconfigurable Technology and Systems (TRETs). - USA, NY, New York, ACM, 2009. – Vol. 1, Issue 4, Article № 21.



2. Huang M. Reconfiguration and Communication-Aware Task Scheduling for High-Performance Reconfigurable Computing / M. Huang, V.K. Narayana, H. Simmler, O. Serres, T. El-Ghazawi // Transactions on Reconfigurable Technology and Systems (TRETTS). – USA, NY, New York, ACM, 2010. – Vol. 3, Issue 4, Article № 20.
3. Bassiri M.M. Mitigating Reconfiguration Overhead In On-Line Task Scheduling For Reconfigurable Computing Systems / M.M. Bassiri, S.H. Shahriar // Proceeding of 2nd International Conference on Computer Engineering and Technology (ICCET), (Chengdu, 16-18 April 2010). – IEEE, 2010. – Vol. 4. – P. V4-397. – V4-402.
4. Клименко І.А. Класифікація реконфігурованих обчислювальних систем / І.А. Клименко, М.В. Рудницький // Вісник Вінницького політехнічного інституту. – Вінниця: ВНТУ, 2014. - №5 (116). – С. 120 – 128.
5. Danne K. Periodic Real-Time Scheduling for FPGA Computers / K. Danne, M. Platzner // Published in Third International Workshop on Intelligent Solutions in Embedded Systems (Germany, Hamburg, 2005). – IEEE, 2005. – P. 117 – 127.
6. Panella A. A Design Workflow for Dynamically Reconfigurable Multi-FPGA Systems / A. Panella, F. Redaelli, F. Cancare, D. Sciuto // Published in 18th IEEE/IFIP VLSI System on Chip Conference (VLSI-SoC), (Spain, Madrid, 27-29 Sept. 2010). – IEEE, 2010. – P. 114 – 119.
7. What's New in Xilinx ISE Design Suite 12 [Електронний ресурс]. – Xilinx, 2014. – Режим доступу: [http://www.xilinx.com/support/documentation/sw\\_manuals/xilinx12\\_3/whatsnew.htm](http://www.xilinx.com/support/documentation/sw_manuals/xilinx12_3/whatsnew.htm).
8. Al-Wattar A. Efficient On-line Hardware/Software Task Scheduling for Dynamic Run-time Reconfigurable Systems / A. Al-Wattar, S. Areibi, F. Saffih // Proceeding in 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW). – IEEE, 2012. - P 401 – 406.
9. Liu S. Achieving Energy Efficiency through Runtime Partial Reconfiguration on Reconfigurable Systems / S. Liu, R.N. Pittman, A. Forin, J.-L. Gaudiot // Transactions on Embedded Computing Systems (TECS). – USA, NY, New York, ACM, 2013. – Volume 12, Issue 3, Article № 72. – 21 p.
10. Pellizzoni R. Adaptive allocation of software and hardware real-time tasks for FPGA-based embedded systems / R. Pellizzoni, M. Caccamo // Proceeding of the 12th Real-Time and Embedded Technology and Applications Symposium, 2006. – IEEE, 2006 – P. 208 – 220.
11. Extending Embedded Computing Scheduling Algorithms for Reconfigurable Computing Systems Saha, P. ; El-Ghazawi, T. Proceeding in 3rd Southern Conference on Programmable Logic (SPL '07) (Mar del Plata, 28-26 Feb. 2007). – IEEE, 2007 – P. 87 – 92.
12. Hasan M.Z. Runtime partial reconfiguration for embedded vector processors / M.Z. Hasan, S.G. Ziavras // Journal of Computers. – Vol. 2, №9. – Academy Publisher, 2007. – P. 60 – 66.
13. Jara-Berrocal A. VAPRES: A Virtual Architecture for Partially Reconfigurable Embedded Systems / A. Jara-Berrocal, A. Gordon-Ross A. // Proceeding of Design, Automation & Test in Europe Conference & Exhibition (DATE), (Germany, Dresden, 8-12 March 2010) – IEEE, 2010. – P. 837 – 842.
14. Taher M. Virtual Configuration Management: A Technique for Partial Runtime Reconfiguration / M. Taher, T. El-Ghazawi // IEEE Transactions on Computers. – IEEE, 2009. – Vol. 58, Issue: 10. – P. 1398 – 1410.
15. Buell D. High-Performance Reconfigurable Computing // D. Buell, T. El-Ghazawi, K. Gaj, V. Kindratenko // Journal Computer. – Volume 40 Issue 3 – USA, CA, IEEE Computer Society Press Los Alamitos, 2007. – P. 23 – 27.
16. Koch D. Fine-Grained Partial Runtime Reconfiguration on Virtex-5 FPGAs // D. Koch, C. Beckhoff, J. Torrisson // Proceeding of 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM) (NC, Charlotte, 2-4 May 2010). – IEEE, 2010, - P. 69 – 72.
17. Liu S. Minimizing the runtime partial reconfiguration overheads in reconfigurable systems / S. Liu, R.N. Pittman, A. Forin, J.-L. Gaudiot // The Journal of Supercomputing, 2012. – Volume 61, Issue 3. – P. 894-911
18. Дунець Р. Б. Проблеми побудови частково реконфігурованих систем на ПЛІС / Р. Б. Дунець, Д. Я. Тиханський // Радіоелектронні і комп'ютерні системи, 2010. – № 7 (48). – с. 200 - 204.
19. Virtex TM 2.5 V Field Programmable Gate Arrays. DS003-1 (v4.0) [Електронний ресурс]. – Xilinx, 2013. – Режим доступу: [http://www.xilinx.com/support/documentation/data\\_sheets/ds003.pdf](http://www.xilinx.com/support/documentation/data_sheets/ds003.pdf).