Автоматизація Vivado через Jupyter Notebook для вдосконалення проєктів на ПЛІС

УДК 004.274:004.896

Іван Яблоков

ДВНЗ «Донецький національний технічний університет», ivan.yablokov@donntu.edu.ua

За останні кілька років використання штучного інтелекту (ШІ) значно підвищилось у всіх сферах високотехнологічного виробництва. Області застосування варіюються від генерації складних мультимедійних об'єктів до автоматизації розробки програмних та апаратних систем. Зокрема, розробники систем автоматизованого проєктування (EDA – Electronic Design Automation), таких як AMD/Xilinx Vivado, активно впроваджують алгоритми ШІ для переходу від трудомісткого ручного налаштування параметрів ПЛІС (FPGA) до інтелектуальних, повністю автоматизованих процесів. Це значно полегшить роботу інженерів при створенні нових проєктів та оптимізації існуючих архітектур, призначених для захисту рішень у галузі IoT та кіберфізичних систем.

Одним із найбільш перспективних напрямків є використання генеративно-змагальних мереж (GAN) для автоматичної генерації RTL-коду або топологій розміщення компонентів. Однак при генерації великої кількості варіантів дизайну перевірка кожного з них на життєздатність стає критичним «вузьким місцем».

Кожен згенерований нейронною мережею варіант необхідно піддати процедурам синтезу, розміщення (placement) та трасування (routing). Оскільки тривалість цих процесів у Vivado варіюється від декількох хвилин до декількох годин для складних проєктів, загальний час навчання нейронної мережі стає дуже великим. Традиційне використання графічного інтерфейсу (GUI) Vivado не дозволяє ефективно масштабувати цей процес, оскільки вимагає ручного керування та споживає значні ресурси системи на візуалізацію.

Для подолання описаної проблеми пропонується використання середовища Jupyter Notebook для керування усіма процесами. На відміну від стандартного GUI, Jupyter дозволяє реалізувати програмне керування додатком Vivado у режимі batch mode за допомогою TCL-скриптів. Jupyter Notebook у цьому контексті виступає не просто як редактор коду, а як інтерактивне середовище для швидкого прототипування. Його головна перевага полягає у можливості розділити процес на окремі блоки (cells), що дозволяє зберігати проміжні стани обчислень без необхідності повторного запуску всього скрипту.

У сучасних обчислювальних системах для EDA-завдань спостерігається гостра диспропорція між кількістю обчислювальних ядер процесора та доступним об'ємом швидкої оперативної пам'яті. Якщо процесорні потужності у 2026 році дозволяють легко оперувати 64 ядрами та 128 потоками на одну робочу станцію, то оперативна пам'ять перетворилася на основний обмежувач масштабованості.

При розпаралелюванні процесів навчання ШІ споживання RAM зростає лінійно відносно кількості потоків. Кожен процес синтезу для сучасних

складних систем на кристалі (SoC) потребує від 16 до 48 ГБ RAM. Таким чином, для повного завантаження потужного процесора система повинна мати об’єм пам’яті від 215 ГБ до 512 ГБ, що виводить обладнання з розряду персональних комп’ютерів у розряд серверів високої щільності.

Якщо під час піку хоча б одному процесу забракне фізичної пам'яті, ОС активує Swap-файл (підкачку на диск). Оскільки швидкість навіть найсучасніших NVMe-накопичувачів у 2026 році все ще на порядки нижча за швидкість DDR4/DDR5, продуктивність усієї системи навчання миттєво деградує. Це явище отримало назву "Thrashing" — стан, коли система витрачає 90% часу на переміщення даних між диском і пам’яттю, а не на реальні обчислення.

Економічний фактор та дефіцит 2026 року

Важливим аспектом є вартість заліза. Ціни на оперативну пам’ять з набранням популярності ШІ дуже сильно підвищились, оскільки дуже багато компаній залучено у процес розробки різноманітних моделей (LLM, GAN, Diffusion). Виробники пам'яті пріоритезують випуск HBM-пам'яті для прискорювачів, що створює дефіцит звичайної серверної RAM.

Jupyter Notebook дозволяє реалізувати інтелектуальні алгоритми управління пам'яттю:

1) Ресурсний моніторинг: перед запуском чергового пакету (batch) даних, скрипт перевіряє доступну RAM через бібліотеку psutil. Якщо вільний об'єм менше 20%, запуск нових процесів призупиняється.
2) Force Cleanup: після завершення кожної ітерації Jupyter примусово завершує дочірні процеси Vivado та очищає системний кеш.

Висновки. Впровадження паралельного запуску через TCL-скрипти дозволяє досягти майже лінійного прискорення процесу навчання у порівнянні з послідовною перевіркою. Jupyter Notebook дозволяє інтегрувати аналітику безпосередньо в процес навчання. При завершенні циклу навчання система виводить результати у вигляді графіків, гістограм та теплових карт розміщення компонентів. Що значно спрощує аналіз результатів для користувача.

Використання Jupyter Notebook у поєднанні з TCL-автоматизацією Vivado є ефективним рішенням для навчання сучасних генеративних моделей у галузі проєктування ПЛІС. Це не тільки зменшує час роботи за рахунок розпаралелювання обчислень, але й забезпечує повну автоматизацію циклу "генерація-перевірка-корекція", що є критично важливим для створення інтелектуальних систем EDA майбутнього. Запропонований підхід робить процес розробки гнучким, масштабованим та мінімізує вплив людського фактора на етапі ітеративної перевірки дизайну.

1. Vivado Design Suite User Guide: Tcl Command Reference Guide (UG835). – San Jose: AMD/Xilinx, 2025. – 1340 p.
2. Goodfellow I., Bengio Y., Courville A. Generative Adversarial Networks: Deep Learning Series. – MIT Press, 2020. – 800 p.
3. Smith J. The Economics of Semiconductor Memory in the AI Era. – Tech Economic Review, 2026. – 45-52 c.