УДК 004.27

КЛИМЕНКО І. А.
ТКАЧЕНКО В.В.
СТОРОЖУК О. М.

МУЛЬТИПРОЦЕСОРНА СИСТЕМА НА БАЗІ ПРОГРАМОВАНИХ ПРОЦЕСОРНИХ ЯДЕР NIOS II ALTERA

Надана загальна інформація щодо методології розробки мультипроцесорних систем-на-кристалі на базі ПЛІС. Розглянуті особливості застосування стандартного потоку проектування засобами *SOPC Builder Altera* для розробки мультипроцесорних систем. Виконано моделювання та дослідження мультипроцесору із програмованими процесорними ядрами *NIOS* II *Altera*, який призначений для високопродуктивного виконання управляючих функцій.

General information of the methodology of multiprocessor systems on chip based on FPGA are provided. The features of a standard design flow facilities SOPC Builder Altera to develop multiprocessor systems are considered. Modeling and research of programmable multiprocessor cores NIOS II Altera, which is designed for high-performance control functions are completed.

1. Вступ

Мультипроцесорні системи-на-кристалі (*MPSoC*, *Multiprocessor System on Chip*) належать до класу убудованих програмованих мультипроцесорних систем (МПС) та очолюють найсучасніші тенденції розвитку цифрових убудованих електронних систем. Основна ознака таких систем – наявність більш ніж одного процесору. Потужні задачі для вирішення яких призначені сучасні вбудовані електронні системи потребують саме мультипроцесорної архітектури для досягнення режиму функціонування в реальному часі за забезпечення інших критичних обмежень, таких як споживна потужність, фізичні розміри та вартість. *MPSoC* є рішенням для реалізації таких складних систем. Широке коло задач, таких як обслуговування мереж, мультимедіа, управління можуть досягнути значного підвищення продуктивності від застосування систем такого класу. Ефективним застосуванням, на яке спрямовані дослідження в даній роботі, є системи управління в реальному часі, як складними технологічними процесами так і різноманітним сучасним устаткуванням: бортовими системами, радарами, супутниковими системами, робототехнікою, біологічними системами, тощо. *MPSoC* пропонують кращі можливості для рішення означеного кола задач у порівнянні з однопроцесорними убудованими електронними системами [1, 2].

Сучасний рівень розвитку програмованої елементної бази обумовлює найновіші та найперспективніші тенденції в оглянутій області застосування вбудованих електронних систем, якими є програмовані мультипроцесорні системи. Програмована технологія спрощує швидке створення прототипів електронних систем та дозволяє проводити дослідження нових архітектур і технологій без проблем, пов’язаних з реалізацією електронних пристроїв на замовлених інтегральних схемах *ASIC (Application-specific Integrated Circuit)* [3]. Ріст можливостей сучасних ПЛІС [3] дозволяє реалізувати надскладні системи на одному кристалі, а ведучі компанії розробники ПЛІС в свою чергу пропонують можливості застосування програмованих ядер процесорів спеціально розроблених для використання в ПЛІС, та апаратних процесорних ядер. Крім того ПЛІС устатковані убудованими блоками пам’яті, периферією та схемами зв’язку.

Слід також зазначити широку доступність процесу проектування та кінцевої розробки електронних пристроїв на ПЛІС, в тому числі складних мультипроцесорних систем, для широкого кола користувачів без притягнення високотехнологічного та дорогого виробничого процесу. Виробники програмованих інтегральних мікросхем разом зі своїми продуктами пропонують безкоштовні та прості в застосуванні засоби автоматизації проектування.

2. Проблеми застосування методології розробки *Design Flow*

Одна з найбільш розповсюджених методологій розробки вбудованих мультипроцесорних систем на базі ПЛІС має назву ручного проектування (*Hand-Tuned Design* – вручну налагоджуване проектування), за застосуван-

ня уніфікованого (стандартного) потоку проектування (*Design Flow*), який пропонують компанії виробники ПЛІС разом зі своїми САПР [3].

Дві найбільш відомі компанії пропонують засоби *EDK* (*Embedded Development Kit*) від компанії *Xilinx* [4] та *SOPC Builder* (*System on a Programmable Chip Builder*) від компанії *Altera* [14, 15 5, 6]. Основні переваги застосування потоку проектування в наступному:

- простота спеціалізованих інструментів;
- наявність великої кількості бібліотек стандартизованих модулів, як постачальників програмного та апаратного забезпечення, так і сторонніх виробників;
- обізнаність розробників електронних пристроїв на ПЛІС різної степені кваліфікації зі спеціальними інструментами та потоком проектування.

Одна з проблем застосування ручного проектування в тому, що складно дослідити та удосконалити весь простір проекту цілком для отримання найкращої можливої архітектури для конкретної задачі. Цей метод може бути гарним вибором для побудови невеликих систем, де відомо, як має бути розроблена архітектура.

Іншою важливою проблемою є те, що за застосування потоків проектування компаній виробників ПЛІС неможливо реалізувати всі потреби проектування мультипроцесорних систем стосовно збільшення їх продуктивності та адаптації до класів вирішуваних задач, в тому числі «на льоту» [2]. Тобто стандартні потоки проектування обмежують в першу чергу таку важливу перевагу ПЛІС, як *гнучкість проектування*. Функціональність систем автоматизації проектування, запропонованих бібліотечних модулів, програмованих процесорних ядер, інтерфейсів заздалегідь зумовлена виробниками ПЛІС, окрім того інструменти для здійснення потоку проектування спочатку створені для розробки монопроцесорних систем, і мають певний ряд недоліків, коли мова йде про високопродуктивні мультипроцесорні системи. Ці проблеми були детально оговорені в роботі [2], де на підставі проведеного всебічного аналізу літературних джерел висунуті найбільш важливі недоліки стандартних потоків проектування.

1. *Обмеження архітектури для реалізації міжпроцесорних комунікацій в мультипроцесорних системах.* Основний проектний потік *SOPC Builder* тільки дозволяє реалізувати загальну шину, та з'єднання типу «крапка-крапка». Та складні *MPSoC* можуть вимагати більш високої пропускної здатності ніж може запропонувати шина, або можуть вимагати більш ефективного використання ресурсів мікросхеми. Так слід зазначити, що така актуальна конфігурація, як мережа на кристалі (*NoC*, *Network-on-Chip*) [2] недоступна за застосування *SOPC Builder* і потребує розробки спеціальних засобів.

2. *Відсутність ефективних механізмів сумісного застосування ресурсів.* Засоби синхронізації від провідних виробників ПЛІС виявляються неефективним як метод синхронізації для високопродуктивних мультипроцесорних систем з великою кількістю процесорних ядер [1, 2, 7].

3. *Обмежена кількість процесорних ядер.* Кількість ядер, що можуть бути убудовані в один дизайн обмежені виробниками бібліотек, зокрема відсутністю в них ефективних механізмів синхронізації та когерентності пам'яті для великої кількості процесорних ядер.

Основним напрямком досліджень в галузі проектування програмованих високопродуктивних мультипроцесорних систем на базі ПЛІС методом налагоджування вручну на сьогодні є подолання зазначених вище недоліків та проблем шляхом втілення нових методів підвищення ефективності функціонування мультипроцесорних систем та створення на їх основі спеціалізованих засобів що дозволять отримати більш ефективні архітектури *FPGA-MPSoC*.

В контексті оглянутих проблем виконаємо дослідження можливостей засобів *SOPC Builder* компанії *Altera* для побудови мультипроцесорної систем-на-кристалі, орієнтованої на рішення задач управління в реальному часі з відповідно ефективною для цього архітектурою [2, 8]. Мультипроцесорна система побудована на базі програмованого процесорного ядра *Nios II Altera* [9, 10].

3. Організація доступу до спільних ресурсів засобами *SOPC Builder*

Спільні ресурси є дуже потужним аспектом ефективного функціонування багатопроцесорних систем, та важливим питанням при цьому є забезпечення сумісного використання цих ресурсів різними процесорами системи. На рис. 1 показано узагальнену блок-схему

багатопроцесорної системи, в якій два процесори розділяють загальну пам'ять (*Shared Memory*) на кристалі [6, 11].

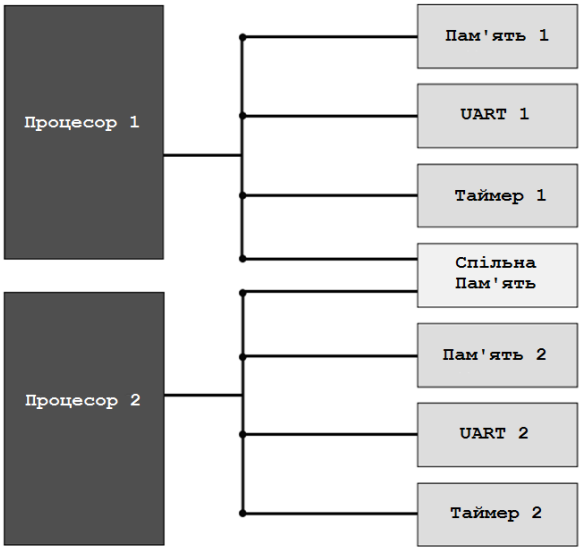

Рис. 1. Мультипроцесорна система із спільними ресурсам

Налагоджування загальної пам’яті виконується шляхом апаратного підключення її до кожної процесорної шини у матриці зв'язку *SOPC Builder*, але це не забезпечує синхронізацію дій процесорів, оскільки доступ до пам’яті процесори матимуть однаковий і можуть одночасно звернутися до пам’яті. Безпосередньо координація доступу до загального ресурсу, покладається на програмне забезпечення кожного процесору, таким чином засоби *SOPC Builder* пропонують програмно-апаратну реалізацію доступу до загального ресурсу.

Якщо під час обміну даними з загальною пам’яттю процесор записує дані в деяку область загальної пам'яті даних, а в той же час інший процесор звертається до цієї ж області пам’яті, відбувається так званий конфлікт доступу до загального ресурсу. Для унеможливлення таких ситуацій під час реалізації пам’яті програм використовують окремий адресний простір для виконавчого простору кожного процесора. Кожен процесор повинен мати свої власні унікальні секції: *.txt, .rodata, .rwdata, .heap і .stack*. Для реалізації пам’яті даних програми обміну даними мають бути написани, так щоб захистити цілісність даних, що зберігаються в загальній пам'яті [11]. Для синхронізації дій процесорів під час роботи із спільною пам’яттю *SOPC Builder* пропонує спеціальний механізм обміну за якого процесори можуть інформувати один одного, коли вони використовують загальний ресурс, так щоб інші процесори переходили у стан очікування доступу. Доступ до спільних ресурсів в *SOPC Builder* відбувається за допомогою функцій апаратних мютексів (*Hardware Mutex Core*).

4. Забезпечення синхронізації засобами Hardware Mutex Core

Середовище *SOPC Builder* пропонує апаратне мютекс ядро, яке дозволяє процесорам спільно працювати та створює умови взаємовиключного доступу до спільних ресурсів в системі.

Мютекс ядро діє як загальний ресурс, який забезпечує набір операцій перевірки та встановлення. Взагалі мютекс можна асоціювати із загальною змінною, яка блокується процесором на час під’єднання до загального системного ресурсу. Дані засоби дозволяють процесору перевірити доступність мютексу та встановити на нього замок одинарною операцією. Коли процесор закінчив використання загального ресурсу, мютекс розблоковується цим процесором і інший процесор може отримати можливість блокування мютекса для використання загальних ресурсів. Важливо відзначити, що мютекс ядро фізично не захищає ресурси в системі від одночасного доступу кількох процесорів. За дотримання правил цілком відповідає програмне забезпечення, що працює на процесорах, яке має завжди перевіряти мютекс перед зверненням до загальних ресурсів. Апаратне мютекс ядро з інтерфейсом *Avalon*, гарантує, що тільки один процесор може встановити замок на мютекс в будь-який момент часу [11]. Мютекс ядро має спеціальний інтерфейс *Avalon-MM*, що складається з двох 32-бітних регістрів (табл. 1), для зв’язку із загальною шиною.

Таблиця 1. Карта регістрів мютекс ядра

Зсув	Назва регістру	R/W	Опис бітів					
			31	16	15	1	0	
0	*mutex*	*RW*	*OWNER*		*VALUE*			
1	*reset*	*RW*	*Reserved*				*RESET*	

Доступ до простого мютекс ядра надається багатьом процесорам, при цьому кожен процесор має унікальний ідентифікатор *ID* [12]. Основний принцип функціонування:

- Коли поле *VALUE* має значення *0x0000*, мютекс є розблокованим і доступним.

В іншому випадку, мютекс заблокований і недоступний.

- Регістр *mutex* завжди доступний для читання. Кожний процесор, підключений до мютексу через інтерфейс *Avalon-MM*, може зчитувати регістр *mutex*, для визначення його поточного стану.
- Процесор, що отримує мютекс, записує свій *ID* в поле *OWNER*, а також відмінне від нуля значення в поле *VALUE*. Під час доступу до загального ресурсу процесор перевіряє набуття змін шляхом перевірки поля *OWNER*.
- Операція запису змінює *mutex* тільки якщо виконуються одна або обидві з наступних умов:
  - Поле *VALUE mutex* регістру встановлена в нуль.
  - Поле *OWNER mutex* регістру відповідає даним *OWNER* поля, які будуть записані процесором.
- Після системного скидання встановлюється біт *RESET* в регістрі скидання. Запис до одного із бітів регістру, очищає його [12].

Під час використання, процесора *Nios II*, *SOPC Builder* пропонує процедури для апаратного доступу до обладнання мютекс ядра. Ці функції безпосередньо управляють мютексом на нижньому апаратному рівні. Мютекс ядро не може бути доступне через стандартні бібліотеки *HAL API* і *ANSI C*. В процесорній системі *Nios II*, процесор блокує мютекс, записуючи значення *cpuid* його регістра управління в поле *OWNER mutex* регістру.

В системі *SOPC Builder* підключається лише апаратне забезпечення тобто мютекс ядро, в САПР від *Altera* містяться бібліотеки які надають процедури та функції для програмного управління мютеком. Тобто при використанні мютексів, велика роль покладається на програмне забезпечення, а саме на доскональність алгоритмів взаємовиключення та синхронізації під час роботі із спільними ресурсами.

Файл *altera_avalon_mutex.h* оголошує структуру *alt_mutex_dev*, що представляє екземпляр мютексу. Деякі процедури для доступу до програмної структури апаратних засобів мютексу, перераховані в таблиці 2.

Таблиця 2. Процедури доступу до апаратних засобів мютекс ядра.

Назва функції	Характеристика
altera_avalon_mutex_open()	Надає дескриптор мютексу, що дозволяє доступу до мютексу всім іншим функціям.
altera_avalon_mutex_trylock()	Намагається блокувати мютекс. Повертає результат як тільки не вдається заблокувати мютекс.
altera_avalon_ mutex_lock()	Блокує мютекс. Не повертає значення, поки успішно не заблокується мютекс.
altera_avalon_mutex_unlock()	Розблоковує мютекс.
altera_avalon_mutex_is_mine()	Визначає, чи заволодів цей процесор мютексом.
altera_avalon_mutex_first_lock()	Перевіряє чи був розблокований скиданням.

Далі наведений приклад програмного коду, що демонструє оголошення мютексу й роботу із ним [9]:

```
    #include <altera_avalon_mutex.h>

/* отримуємо обробник мютексу*/

    alt_mutex_dev* mutex =

    atera_avalon_mutex_open("/dev/mutex");

/* активуємо мютекс, встановивши його
значення в один */

    altera_avalon_mutex_lock(mutex, 1);

/* доступ до спільного ресурсу, знято
блокування */

    altera_avalon_mutex_unlock(mutex);
```

5. Обробка переривань

Процесор *Nios II* підтримує апаратні та програмні переривання, які діляться на наступні категорії: нереалізовані інструкції, переривання під час виникнення непередбаченої ситуації (виключення), змішані переривання від зовнішніх пристроїв. Засоби *SOPC Builder* для процесорного ядра *Nios II* реалізують два різних механізми обробки переривань: внутрішній контролер переривань (*IIC- Internal Interrupt Controller*), та інтерфейс із зовнішнім

контролером переривань (*EIC- External Interrupt Controller*). Конфігурація процесів обробки переривань в *Nios II* залежить від типу контролера переривань, який обирається під час інсталяції процесора *Nios II* в *SOPC Builder*.

Внутрішній контролер переривань. За допомогою контролера *IIC* обробка переривань реалізується за класичною схемою *RISC*. Всі типи переривань – програмні та апаратні, координуються однією програмою-обробником переривань (*funnel*) верхнього рівня. Таким чином, всі переривання обробляються програмним кодом, розташованим в одному місці, за адресою програми-обробника переривань. *Контролер IIC* – це простий невекторний апаратний контролер переривань. Після отримання запиту на будь-яке внутрішнє переривання *IRQ*, контролер *IIC* переміщує управління на основну адресу переривання. Апаратна частина показує, яке саме переривання *IRQ* має оброблятися, що дозволяє програмі-обробнику маскувати окремі переривання [12]. Внутрішній контролер переривань доступний у всіх версіях процесора *Nios II*.

Концепція обробки зовнішніх переривань. Інтерфейс контроллера *EIC* дозволяє процесору *Nios II* працювати з окремими зовнішніми компонентами для обробки переривань, наприклад із зовнішніми контролерами переривань. За допомогою інтерфейсу *EIC*, апаратні переривання можуть оброблятися окремо від програмних. Апаратні переривання мають окремі вектори переривань і обробники. Кожне переривання може мати власний обробник, або обробники можуть бути загальними. Обробка програмних переривань реалізована так само, як за застосування контролеру *IIC*.

Altera пропонує свій власний зовнішній контролер переривань *EIC*, що має назву векторного контролера переривань (*VIC*) *Vectored Interrupt Controller. К*онтролер призначений для обробки переривань від зовнішній пристроїв та забезпечує високу продуктивність з низьким рівнем затримки обробки переривань. Під час зовнішніх переривань, що відбуваються в системі, контролер *VIC* визначає пріоритет переривань, джерело, яке запитує переривання, обчислює адресу програми-обробника переривань (*RHA*), надає процесору відповідну управляючу інформацію, що включає тому числі *RHA*.

Ядро контролера *VIC* містить наступні інтерфейси: до 32 портів введення переривання в *VIC* ядро, один *Avalon-MM* допоміжний інтерфейс для доступу до контроль стану загальних регістрів (*CSR*), один *Avalon Streaming* (*Avalon-ST*), вихідний інтерфейс для передачі інформації про вибрані переривання, один додатковий *Avalon-ST* вхідний інтерфейс для отримання *Avalon-ST* виходів в системах з декількома послідовно підключеними контролерами *VIC*. Модуль ядра *VIC* готовий для використання в *SOPC Builder* і легко інтегрується в будь-яку сформовану засобами *SOPC Builder* систему. Для процесора *Nios II Altera* забезпечує абстрактне представлення апаратних засобів (*HAL*) драйвера програми ядра контролера *VIC* [12].

Бібліотеки програмного забезпечення контролера *VIC* забезпечують апаратний рівень доступу до обладнання та драйверів, які інтегруються з *Nios II*. Макроси для доступу до всіх регістрів визначені в бібліотеці *altera_vic_regs.h*.

Далі в якості прикладу наведений програмний код, що демонструє структуру даних для пристрою *VIC*:

```
#define ALT_VIC_MAX_INTR_PORTS (32)

typedef struct alt_vic_dev {

 void   *base;

/* Базова адреса VIC */

 alt_u32       intr_controller_id;
/* Контролер переривань ID */

 alt_u32       num_of_intr_ports;
/* Кількість портів переривань */

 alt_u32       ril_width;
/* RIL ширина */

 alt_u32       daisy_chain_present;
/* Дейзі-ланцюг вхідного подання */

 alt_u32       vec_size;
/* Розмір вектору */

 void          *vec_addr;
/* Векторна таблиця базових адрес */

 alt_u32

 int_config[ALT_VIC_MAX_INTR_PORTS];
/* INT_CONFIG Налаштування для кожного
переривання */

 } alt_vic_dev;
```

6. Управління кешем

Архітектура *Nios II* не надає апаратної когерентності кешу. Вміст кешу даних всіх процесорів, які мають доступ до загальної пам'яті, управляються програмою, основне призна-

чення якої слідкувати, щоб всі *майстри* читали саме останнє значення і не перезаписували нові дані застарілими. Інструкція *flushd* стежить за тим, щоб кеш даних і пам'ять містили одне і те ж значення в одному рядку. Якщо рядок містить зіпсовані дані, він пишеться в пам'ять. Цей рядок визнається недійсним в кеші даних. Отже, пропуск кешу даних дуже важливий. Процесор не може перевіряти, чи є адреса в кеші даних під час пропуску кешу даних, оскільки при цьому звертається напряму в пам'ять. Нижче наведений приклад програмного коду для запису нові команди в пам'ять:

```
/* Попередньо нова інструкція міститься
в r4 і адреса інструкції в r5 */
/*Записує нову інструкцію в r4, для ад-
реси інструкцій в r5 */
     stw r4, 0(r5)
/* Записує строку кешу даних, асоційо-
вану з адресою в r5 в пам'ять і визнає
недійсною її в кеші даних */
     flushd 0(r5)
/* Визнає недійсною строку кешу інстру-
кцій, асоційовану з адресою в r5 */
     flushi r5
/* Стежить за тим, щоб конвеєр процесо-
ра не використал
стару інструкцію за адресою, визначеною
в r5 */
     flushp
```

Біт 31 пропуск кешу. Використовуючи 31 біт, звичайний набір інструкцій *ld/st* може бути використаний для пропуску кешу даних, якщо останній використовуваний біт адреси (біт 31) встановлений в одиницю. Значення 31 біта використовується тільки внутрішньо процесором; 31 біт примусово обнуляється в фактичному адресі доступу. Це обмежує максимальність адресного простору до 31 біта.

31-ий біт пропуску кешу надається тільки з ядром *Nios ll/f* і не повинен використовуватися з іншими ядрами *Nios II*. Інші ядра *Nios II* зберігають максимальну адресацію до 31 біт, щоб спростити міграцію коду з однієї реалізації ядра на іншу. Вони ефективно ігнорують значення біта 31, що дозволяє коду, написаному для ядра *Nios ll/f*, використовувати біт 31 для пропуску кешу, і коректно запускатися на інших поточних реалізаціях *Nios II*. Таким чином цей засіб залежить від реалізації ядра *Nios II* [11].

7. Шина Avalon

Avalon – проста шина для побудови мультипроцесорної системи на кристалі. Основні цілі проекту шини *Avalon*: простота, мала кількість комірок кристалу задіяна для логіки шини, повністю синхронні операції. Тому специфікація *Avalon* не підтримує такі особливості як, передача даних пакетами (*burst-data transfers*) і можливість роботи в режимі «*multi-master*» [13].

Кожна шина *Avalon* підключає єдиного майстра «*master*» до багатьох периферійних пристроїв «*slave*». Майстром шини *Avalon* може бути, наприклад, ядро мікропроцесора *Nios II*, а периферійними пристрій *Avalon* може бути, наприклад, таймер, або пристрій пам'яті. Всі трансакції шини *Avalon* передають (переміщують) єдиний байт, слово половинної розрядності або слово (8, 16 або 32 біти) між одним з периферійних пристроїв і майстром.

В багатопроцесорній системі всі процесори підключаються до майстер порту шини *Avalon* і кожен із них є головним в спілкуванні через шину із периферійними пристроями та пам’яттю. Тобто кожне процесорне ядро отримує рівноправний доступ до шини і до системних пристроїв – загальної пам’яті та периферії. В цьому контексті слід означити досліджувану систему як децентралізовану.

Шина *Avalon* включає ряд особливостей і домовленостей для автоматичної генерації системи, шини і периферійних пристроїв за допомогою програмного середовища *SOPC Builder*. Середовище *SOPC Builder* відповідає за створення *HDL*-файлів, які здійснюють кожен з компонентів системи в каталозі *Quartus* проекту. Програмне забезпечення може опціонально генерувати синтезований файл *.edf*-список зв'язків, який описує модуль системи *Avalon* як єдиний модуль.

Шина *Avalon* є модулем з великою кількістю виводів. Модуль шини *Avalon* міститиме всю логіку, необхідну для підключення сигналів інтерфейсу до всіх периферійних пристроїв і навпаки.

Програма *SOPC Builder* поставляється як вбудований в програму *Quartus MegaWizard*. Це додаток, заснований на *Java*, який дозволяє створювати і редагувати систему *Avalon*, використовуючи графічний інтерфейс користувача. *SOPC Builder* автоматично виявляє компоненти бібліотеки *Avalon* (які повинні міститися в певних каталогах і відповідати відповідним умовам) і робить їх доступними системному проектувальнику [13].

Зазвичай кінцевим результатом роботи «*SOPC Builder*» є:

- Список зв'язків, який описує модуль системи *Avalon*.

- *HDL*-файл (або *Verilog*, *VHDL* або *AHDL*) який визначає інтерфейс до модуля системи *Avalon*.

- *bsf*- (символ) файл, необхідний для включення модуля системи *Avalon* в схемному проекті.

Програма *SOPC Builder* використовує *PTF*-файл (периферійний файл шаблону) як базу даних, щоб зберегти інформацію про систему *Avalon*. Кожна система *Avalon* має власний опис в *PTF*-файлі [13], який використовують як набір команд, щоб генерувати всю необхідну логіку компонентів системи. Програма *SOPC Builder* на основі *PTF*-файлу створює цю систему.

8. Структурна і програмна розробка в САПР Quartus II та тестування мультипроцесору

Для реалізації системи використовується навчально-дослідницький стенд *DE*2 (*DE*2 *Development and Education Board*) [14] із вбудованою ПЛІС *Cyclone II EP2C35F672C6 Altera* [15], що містить 33 216 логічних елементів на кристалі. Загальний об'єм вбудованої пам'яті (*On-Chip Memory*) на чипі становить 483840 біт, що становить близько 59Кб. На стенді *DE*2 вбудовано зовнішня відносно мікросхеми ПЛІС пам'ять – 8Мб *SDRAM*, 512Кб *SRAM*, 1Мб *Flash Memory* та інші пристрої й інтерфейси [15].

Для розробки мультипроцесору використані програмовані ядра типу *Nios II/s* (*Small core size*) [11]. Обране процесорне ядро займає незначну кількість логічних елементів в кристалі, в порівнянні з більш просунутою версією *Nios II/f*, а саме до 1400 *LEs*, що становить всього 4% від всієї площі кристалу ПЛІС *Cyclone II*; забезпечує високу продуктивність 0,74*DMIPS/MHz*, при частоті роботи ядра 165*MHz*, максимальна продуктивність становитиме 127 *DMIPS*; підтримувана зовнішня пам'ять до 2*Gb*.

Програма *SOPC Builder* дозволяє добавити в проект опціонально до 60 процесорних ядер різної конфігурації, але кількість процесорних ядер обмежується фізичними характеристиками кристалу, зокрема його логічними ресурсами. Так, наприклад мікросхема, що вибрана для реалізації системи, *Cyclone II EP2C35F672C6*, може помістити до 47 економних ядер *Nios II/e (Economy core*), в яких відсутній кеш і які дають продуктивність до 31*DMIPS*, при частоті роботи 200*MHz*. В той же час дана мікросхема може помістити до 18 швидких продуктивних ядер *Nios II/f (Fast core*), в яких присутні кеш інструкцій та кеш даних, а продуктивність роботи становить 218*DMIPS* при частоті роботи 185*MHz*, що в сумі при використанні паралельного програмного забезпечення може досягати 3924 *DMIPS*.

В системі визначено 3 процесорні ядра, які мають кеш інструкцій 4Кб кожне, хоча максимальний розмір пам'яті інструкцій в ядрі може бути до 64Кб, але в даному проекті ми обмежені об'ємом вбудованої пам'яті мікросхеми. До кожного ядра підключений інтервальний таймер з періодом в 1мс.

До складу мікропроцесорної системи входить загальна пам'ять *RAM*, розміром в 10Кб, що знаходиться в кристалі. Загальна пам'ять захищається апаратним мютексом, та використовується як буфер для передачі даних між процесорами. Для використання зовнішньої пам'яті і реалізації зв'язку з зовнішніми пристроями та персональним комп'ютером застосовуються засоби надані *Altera* у складі стенду *DE*2. Так для виконавчого коду використовується зовнішня пам'ять SDRAM розміром 256Кб. Для загрузки конфігурації системи використовується зовнішня енергонезалежна *Flash*-пам'ять розміром 0,3Мб. Інтерфейс *UART* реалізує передачу послідовних символьних потоків між вбудованою системою на ПЛІС і зовнішніми пристроями. Інтерфейс *JTAG UART* реалізує передачу потоків послідовних символів між персональним комп'ютером і системою на ПЛІС. У багатьох проектах, *JTAG UART* використовується для спілкування з ядром на читання і запис даних і управління регістрами. *JTAG UART* ядро використовує схему *JTAG* вбудовану в ПЛІС *Altera*, а також надає доступ через контакти *JTAG* до мікросхеми. ПК може підключатися до ПЛІС через *JTAG* кабель, наприклад, *USB-Blaster*. Програмне забезпечення для *JTAG UART* ядра забезпечує *Altera*. Для процесора *Nios II*, драйвери пристроїв наведені в бібліотеках апаратних абстракцій (*HAL*), що дозволяє писати програму для доступу до системи, використовуючи процедури *ANSI C* стандарт-

ної бібліотеки *stdio.h*. Також на стенді використовується паралельний порт *PIO*, для виведення сигналів на світло-діоди на схемі, і інтерфейс для керування *LCD* дисплеєм під час виведення інформації.

Розроблена система займає на кристалі 32% логічних ресурсів, а саме 10 785 логічних елементів, та використовує 46% вбудованої пам'яті, а саме близько 27Кб.

Після синтезу мультипроцесорної системи в *SOPC Builder* отримали структурний модуль системи, який в САПР *Quartus II* підключається до вхідних та вихідних контактів мікросхеми (рис. 2). Далі модуль мультипроцесорної системи пройшов подальші етапи стандартного потоку проектування в САПР *Quartus II*, а саме функціональне модулювання, розміщення конфігурації на мікросхемі, часове моделювання, прошивка кристалу мікросхеми.

В програмному середовищі *Nios II Software Build Tools for Eclipse*, для створеної системи та кожного її ядра, написано програмне забезпечення для запуску на ядрах мультипроцесору. Виконане тестування мультипроцесорної системи, при цьому можна запускати як всі ядра разом, так і кожне ядро опціонально, обираючи налаштування запуску в меню програми *Nios II SBT*. Результати тестування можливо переглянути в консольному вікні програми *Nios II Software Build Tools for Eclipse* (рис. 3).

9. Висновки

При вирішенні потужних обчислювальних задач вбудовані системи потребують саме мультипроцесорної архітектури для забезпечення високої швидкодії обчислень, зменшення споживної потужності, фізичних розмірів та вартості. Значному підвищенню швидкодії цифрових пристроїв, виконаних на одному кристалі, сприяє застосування програмованих логічних інтегральних схем.

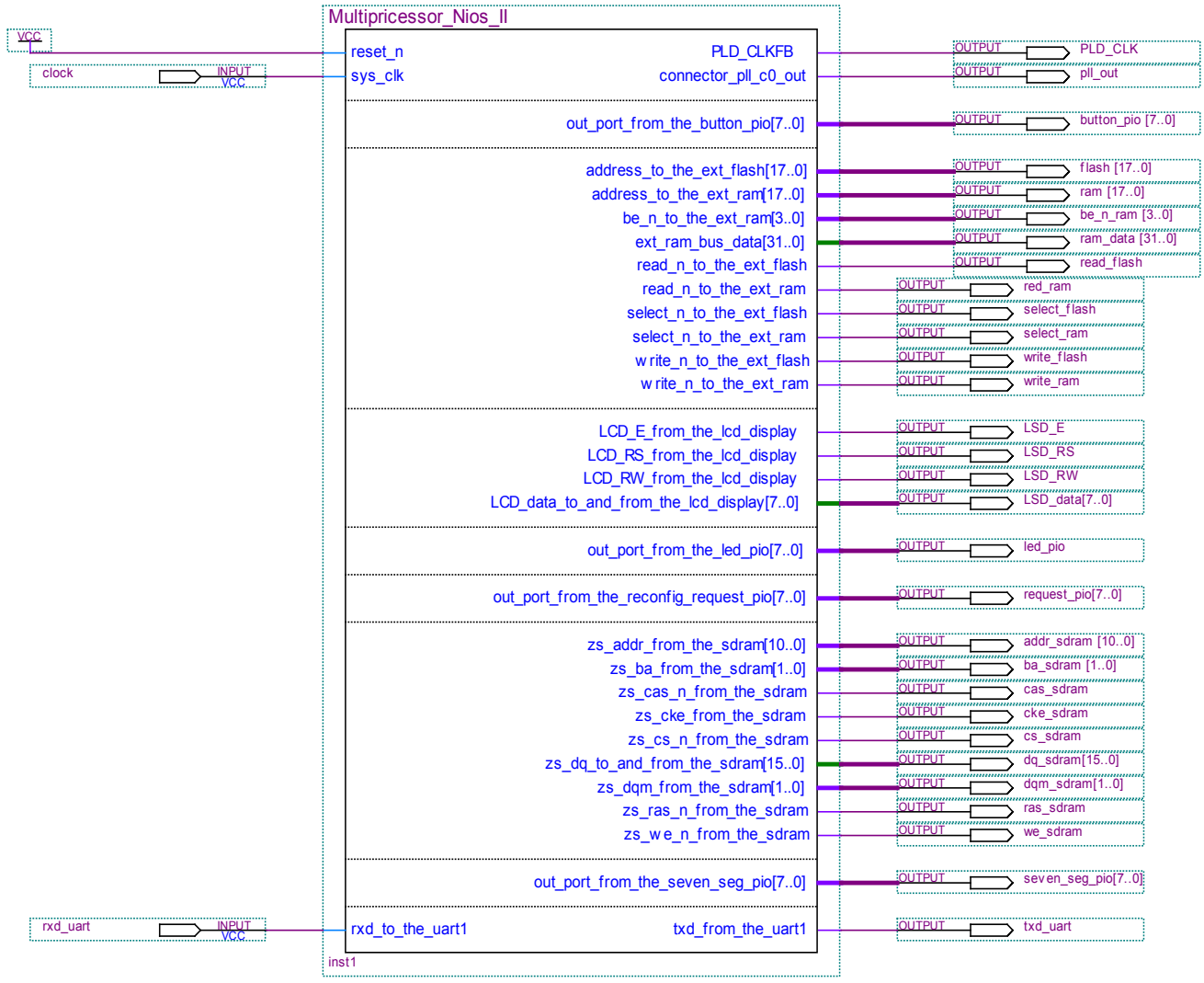

Рис. 2. Блок діаграма мультипроцесору в САПР Quartus II

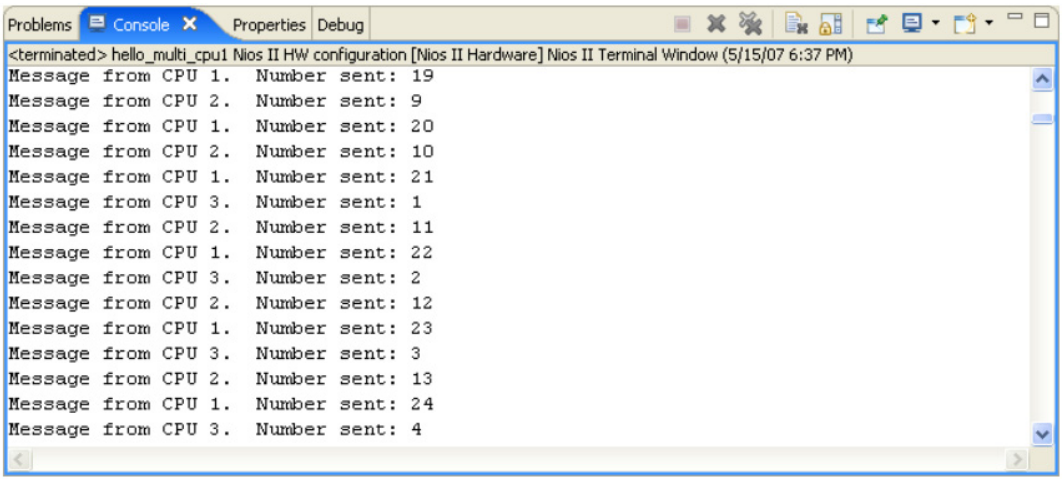

Рис. 3. Консольний вивід повідомлень роботи мультипроцесору

Для створення мультипроцесорних систем на кристалі виробники ПЛІС пропонують готові, вже налагоджені мікропроцесорні ядра та спеціалізовані засоби для здійснення стандартного потоку проектування, які в своїх бібліотеках, окрім засобів налагодження процесорних ядер, пропонують комунікаційні рішення, засоби доступу до загальних системних ресурсів та синхронізації роботи процесорів, засоби обробки переривань.

В роботі розроблена та досліджена мультипроцесорна система класу система-на-кристалі на базі процесорних ядер *Nios II/e* за застосування стандартного потоку проектування засобами середовища *SOPC Builder*.

Дослідження показали, що засоби запропоновані виробником пропонують достатньо ефективні рішення, але мають значні обмеження з точки зору досягнення високої швидкодії обчислень, що вимагають задачі управління в реальному часі, гнучкості проектування та втілення динамічної реконфігурації:

- всі засоби доступу до загальних системних ресурсів, синхронізації роботи процесорів та обробки переривань реалізовані на програмно-апаратному рівні переважно з програмною реалізацією, що значно зменшує швидкодію операцій обміну даними в системі;
- середовище *SOPC Builder* пропонує єдине архітектурне рішення – мультипроцесорна система із загальною шиною, що унеможливлює створення систем з більш ефективною архітектурою комунікаційних з'єднань та способом передачі даних, відносно вимог вирішуваних задач;
- запропоновані засоби синхронізації сумісної роботи процесорів функціонують в режимі децентралізованого управління, що позитивно впливає на швидкодію системи;
- особливості реалізації загальної шини та спосіб підключення до неї процесорів забезпечують можливість реалізації синхронізації за каналами даних, що є одначе низькопродуктивною по причині значної завантаженості загальної шини непродуктивними даними [8], окрім того переважно програмна реалізації механізму доступу до загальної пам'яті нанегативно впливає на швидкодію обміну даними;
- в системі відсутні спеціалізовані засоби для реалізації синхронізації сумісної роботи процесорів за каналами управління [8], що потребують ефективної реалізації системи обробки переривань;
- використання стандартного потоку проектування засобами середовища *SOPC Builder* унеможливлює втілення в систему власних засобів підвищення ефективності, таких як спеціалізовані співпроцесори та інші додаткові пристрої.

В роботі також досліджене важливе питання, що належить до проблеми проектування високопродуктивних мультипроцесорних систем на ПЛІС – обмеженість апаратних ресурсів ПЛІС, зокрема обмеженість логічних комірок, кількість убудованих блоків пам'яті та кількість виводів мікросхеми.:

- процесорне ядро *Nios* належить до класу процесорів з RISC архітектурою, та спрямовані на вирішення широкого кола задач, тому слід зазначити його можливу збиткову функціональність в контексті вирішення специфічних задач, що також негативно впливає на швидкодію та витрату ресурсів мікросхеми;
- кількість убудованих блоків пам'яті фіксовано, що більш обмежує можливу кіль-

кість процесорів, убудованих в один проект, ніж логічні ресурси мікросхеми взагалі, за даними досліджень реалізація системи на мікросхемі широкої доступності та низької вартості дозволить вбудувати не більш шести процесорних ядер *Nios II/e*, це достатньо для функціонування нескладних мультипроцесорних систем с загальною шиною.

Ще одним важливим питанням є розпаралелювання задач. В мультипроцесорних системах розглянутого класу можлива лише програмна реалізація розподілу задач з централізованим управлінням на базі існуючих технологій паралельного програмування. Актуальна та дуже перспективна задача в галузі розробки мультипроцесорних систем – автоматизація процесу розподілу задач та вихідна з цього можливість динамічної реконфігурації системи також унеможливлені обмеженнями потоку проектування та жорстко пропонуємою архітектурою.

В контексті вищесказаного слід зазначити, що стандартні засоби проектування доступні для широкого кола розробників, але мають певні обмеження з точки зору швидкодії створених систем і гнучкості проектування та ефективні для проектування нескладних систем середньої продуктивності.

Основним напрямком досліджень в галузі проектування високопродуктивних мультипроцесорних систем на базі ПЛІС є подолання зазначених вище недоліків та проблем шляхом втілення нових методів підвищення ефективності функціонування мультипроцесорних систем та створення на їх основі спеціалізованих засобів що дозволять отримати більш ефективні архітектури *FPGA-MPSoC*.

Список літератури

1. Dorta T. Reconfigurable Multiprocessor Systems: A Review / T. Dorta, J. Jiménez, J. L. Martín, U.Bidarte, A.Astarloa // International Journal of Reconfigurable Computing [Special issue on selected papers from ReconFig International conference on reconfigurable computing and FPGAs (ReconFig 2009)] – 2010. – Volume 2010. – 11 с.
2. Клименко І.А. Класифікація та архітектурні особливості програмованих мультипроцесорних систем-на-кристалі // Проблеми інформатизації та управління: Зб.наук.пр.– К.: Вид-во нац. авіац. ун-ту «НАУ-друк», 2012.– Вип. 1(36).
3. Клименко І.А. Тенденції застосування сучасної елементної бази для побудови високопродуктивних обчислювальних систем // Проблеми інформатизації та управління: Зб.наук.пр.– К. : Вид-во нац. авіац. ун-ту «НАУ-друк», 2010.– Вип.1(29). – С 90 – 103.
4. MicroBlaze Processor Reference Guide. Embedded Development Kit EDK 13.1 [Електронний ресурс]. – Xilinx Inc., 2011. – Режим доступу: http://www.xilinx.com/support/documentation/sw_manuals/xilinx13_1/mb_ref_guide.pdf.
5. SOPC Builder User Guide. Version 1.0. [Електронний ресурс]. – Altera Corporation, 2010. – Режим доступу: http://www.altera.com/literature/ug/ug_sopc_builder.pdf.
6. Embedded Design Handbook Version 2.9. [Електронний ресурс]. – Altera Corporation, 2011. – Режим доступу: http://www.altera.com/literature/hb/nios2/edh_ed_handbook.pdf.
7. Tumeo A. HW/SW methodologies for synchronization in FPGA / A. Tumeo, C. Pilato, G. Palermo, F. Ferrandi, D. Sciuto // Proceedings of the 7th International Symposium on Field-Programmable Gate Arrays (FPGA '09), [ACM SIGDA], (Monterey, California, USA, 22-24 February 2009). – USA, NY, New York : ACM, 2009. – P. 265–268.
8. Жабин В.И. Архитектура вычислительных систем реального времени / В.И. Жабин. – К. : ВЕК+, 2003. – 176 с.
9. Nios II Processor Reference Handbook. Version 11.0.0. [Електронний ресурс]. – Altera Corporation, 2011. – Режим доступу: http://www.altera.com/literature/hb/nios2/n2cpu_nii5v1.pdf.
10. Nios II Software Developer's Handbook. Version 11.0. [Електронний ресурс]. – Altera Corporation, 2011. – Режим доступу: http://www.altera.com/literature/hb/nios2/n2sw_nii5v2.pdf.
11. Creating Multiprocessor Nios II Systems Tutorial. Version 11.0. [Електронний ресурс]. – Altera Corporation, 2011. – Режим доступу: http://www.altera.com/literature/tt/tt_nios2_multiprocessor_tutorial.pdf
12. Embedded Peripherals IP. User Guide. Version 11.0. [Електронний ресурс]. – Altera Corporation, 2011. – Режим доступу: http://www.altera.com/literature/ug/ug_embedded_ip.pdf
13. Avalon Interface. Specifications Version 11.0. [Електронний ресурс]. – Altera Corporation, 2011. – Режим доступу: http://www.altera.com/literature/manual/mnl_avalon_spec.pdf.
14. DE2 Development and Education Board . [Електронний ресурс]. – Altera Corporation, 2012. – Режим доступу: http://www.altera.com/education/univ/materials/boards/de2/unv-de2-board.html?GSA_pos=3&WT.oss_r=1&WT.oss=DE2 board.
15. Cyclone II Device Handbook [Електронний ресурс]. – Altera Corporation, 2008. – Режим доступу: http://www.altera.com/devices/fpga/cyclone2/cy2-index.jsp.