РЕАЛІЗАЦІЯ ДЕКОДЕРУ РІДА-СОЛОМОНА У ПЛІС

Сергієнко А.М., Лепеха В.Л., Виноградов Ю.М., Франко Р.А.

Національній Технічний Університет України «КПІ»,
Кафедра обчислювальної техніки
info@comsys.kpi.ua

Введення

Сучасні технології збереження і передачі даних неможливі без адекватних засобів їх захисту від втрат. Одним з найбільш ефективних методів завадостійкого кодування даних є метод Ріда-Соломона [1]. При цьому декодування кодів Ріда-Соломона представляє собою розв'язання складної неоднорідної математичної задачі з застосуванням нестандартної арифметики полів Галуа. У результаті цього програмна реалізація декодера Ріда-Соломона, як правило, має низьку швидкість реалізації. Тому в переважній кількості випадків такий декодер реалізовано апаратно у вигляді спец. обчислювача.

Застосування програмованих логічних інтегральних схем (ПЛІС) дає можливість у стислі строки з мінімальними витратами розробити складну систему на кристалі (СНК, SoC). Реалізація декодера Ріда-Соломона в ПЛІС дає можливість розробляти на його основі нові СНК (SoC) для таких застосувань, збереження баз даних великого об'єму, прийом та передача супутникових, мультимедійних і телевізійних сигналів та багато інших.

Така реалізація СНК (Soc) в ПЛІС дає можливість перенастроювати пристрій на різні стандарти кодування інформації, його модернізувати без зміни схеми, змінювати довжину посилок, кількість виправляємих помилок і т.п.

При цьому елементний базис ПЛІС істотно відрізняється від базису замовних НВІС. Це призводить до того, що блоки, розроблені для НВІС, будуть реалізовані в ПЛІС неефективно.

У статті досліджуються питання реалізації алгоритму Ріда-Соломона з урахуванням елементного базису сучасних ПЛІС, можливостей переналаштування алгоритму. В якості змінних параметрів декодера виступає довжина кодованого слова, кількість помилок, що виправляються, характеристичний поліном поля Галуа.

Алгоритм декодування Ріда-Соломона

Декодування кодів Ріда-Соломона являє собою досить складне завдання. Типова схема декодування, що одержала назву авторегресійного спектрального методу декодування, складається з наступних кроків [2, 3].

– прийом закодованого кадру
– обчислення синдрому помилки (синдромний декодер);
– побудова полінома помилок, здійснювана або за допомогою високоефективного, але складного алгоритму Берлекемпа-Мессі, або за допомогою простого, але повільного алгоритму Евкліда;
– знаходження коренів полінома помилок, яке зазвичай виконується перебором (алгоритм Ченя);
– визначення характеру помилки, яке зводиться до побудови маски, що обчислюється на основі звернення алгоритму Форне або іншого алгоритму звернення матриці;
– виправлення помилкових символів шляхом накладання маски на інформаційне слово і послідовне інвертування всіх спотворених біт операцією XOR.

Структурна схема такого алгоритму представлена на рис. 1.

Рис.1. Алгоритм авторегресійного спектрального декодера Ріда-Соломона

Вибір елементної бази

Аналіз вищенаведеного алгоритму показує, що основними операціями є векторне множення, ділення та зведення в ступінь у полі Галуа. Це досить трудомісткі операції, особливо зведення у ступінь та ділення, так як вони виконуються в нестандартній системі числення. Зазначимо, що в більшості стандартних декодерів, як і в прикладах нижче, обчислення виконуються в полі Галуа GF (28).

Нижче буде наведено порівняння реалізації декодеру на звичайному мікропроцесорі і ПЛІС. Базова операція множення A = B*C може бути виконана на спеціалізованому комбінаційному помножувачі чисел Галуа, який має порівняно великі апаратні витрати. Крім того, для зведення в m-у ступінь необхідно більш $\log_2 m$ множень, а для ділення - 16 множень [1]. Якщо виконувати обчислення за допомогою таблиць, то як множення, так і інші операції можна виконувати істотно швидше. Тоді обчислення операції множення має наступний вигляд під час запису на мові VHDL:

```
B0 = tabl0 (B) - вибір операнда з таблиці
C0 = tabl0 (C) - вибір операнда з таблиці
A0 = B0 + C0 - підсумовування в G2 ^ 8
A = tabl1 (A0) - вибір операнда з таблиці
```

Тут tabl0 і tabl1 – таблиці логарифмів і антілогарифмів чисел у полі Галуа розміром 256 входів. Як бачимо, операція множення буде відбуватися за 4 макрокоманди, а з урахуванням організації мікропроцесору - за 10 - 15 команд. Враховуючи, що, наприклад, для виправлення 8 помилок (часто зустрічаються випадки) потрібне множення векторів з 16 елементами, така векторна операція може зайняти 160 - 240 тактів роботи мікропроцесора.

При реалізації на ПЛІС ці операції можна распаралелити. Сучасні ПЛІС мають досить велику кількість незалежних блоків пам'яті, що дозволяє зробити таке векторне множення за 1 - 3 такти. Крім того, можливі й інші способи прискорення обчислень, які у сукупності дають можливість значно прискорити реалізацію алгоритму.

Виходячи з цих міркувань, зроблено висновок про доцільність реалізації декодера Ріда-Соломона на ПЛІС з арифметикою, реалізованою на таблицях.

Реалізація декодера

Як було зазначено вище, основна операція даного алгоритму - векторне множення в полі Галуа. Реалізація помножувача, що виконує операцію множення (ділення) в G8 за 2 такти, наведена нижче:

```
d0 <= conv_std_logic_vector (rom1 (conv_integer (a)), 8);
d1 <= conv_std_logic_vector (rom1 (conv_integer (b)), 8);
sm1 <= ext (d0, 9) + ext (d1, 9) + 1
   when m_d = '0 'else ext (d0, 9) - ext (d1, 9) - 0;
b1 <= not (m_d xor sm1 (8));
sm2 <= sm1 - ext (('0 '& b1), 9);
process (clk, rst)
begin
        if rst = '1 'then
                sm <= (others => '0 ')';
        elsif clk = '1 'and clk'event then
                sm <= sm2;
        end if;
end process;
a2 <= sm (7 downto 0);
d2 <= conv_std_logic_vector (rom0 (conv_integer (a2)),
8);
process (clk, rst)
begin
        if rst = '1 'then
                z0 <= '0 ';
        elsif clk = '1 'and clk'event then
                if a = x "00" or b = "00" then
                        z0 <= '1 ';
                else
                        z0 <= '0 ';
                end if;
        end if;
end process;
res <= x "00" when (z0 = '1 ') else d2;
```

Тут rom1.– двохпортове ПЗУ таблиці логарифмів, а rom0 – ПЗУ таблиці антілогарифмів

Параметричний векторний помножувач має вигляд:

```
U_md: for i in 0 to G_range - 1 generate
mul: mul_g8
         port map (
                clk => clk,
                rst => rst,
                m_d => m_d,
                a => a (i),
                b => b (i),
                res => c (i)
);
 end generate;
```

Структурно декодер розбитий на три блоки, які виконують (див. рис.1):

– прийом кадру і знаходження синдрому помилки;

– визначення характеру помилки і формування корегуючої маски;

– корекцію помилки.

Прийом кадру завдовжки N слів здійснюється в зсувний регістр. Надалі ці дані беруть участь у формуванні синдрому помилок, який обчислюється послідовно на одному помножувачі і накопичуючому суматорі. Результатом є синдром помилок, на підставі якого знаходиться поліном помилки.

Поліном помилки обчислюється за допомогою алгоритму Берлекемпа-Мессі. Для цього використовується описаний вище векторний помножувач. Після цього на тому ж векторному помножувачі реалізуються алгоритм Ченя – для пошуку коренів даного полінома і алгоритм Форне – для формування корегуючої маски.

У кінцевому блоці прийнятий кадр послівно корегується шляхом підсумовування з корегуючою маскою.

Всі модулі працюють у старт-стопному режимі (наступний модуль починає працювати за сигналом закінчення роботи попереднього модуля). Така реалізація дозволяє організувати конвеєрну роботу декодера, що може поліпшити тимчасові характеристики системи.

Модуль параметризований для операцій з байтовими послідовностями (довжина слова - 8 біт). В якості параметру виступає кількість виправляємих помилок t (2 - 8 помилок), довжина кадру N (до 255 слів), що породжує поліном.

Технічні характеристики декодера

Основною характеристикою декодера є час виконання операції декодування. Оскільки аналітично подання досить складне, покажемо деякі дані отримані експериментальним шляхом.

Табл. 1. Число тактів декодування

t	Довжина кадру *N*, слів			
	32	64	128	255
2	472	632	1012	1647
3	618	842	1290	2179
4	704	992	1568	2711
5	790	1142	1846	3243
6	876	1292	2124	3775
7	962	1442	2402	4307
8	1048	1592	2680	4839

У табл.1 вказано кількість тактів, необхідних для виконання декодування, починаючи від сигналу початку послідовності та закінчуючи сигналом закінчення роботи, тобто час виконання операції декодування при різних параметрах. При цьому можливість конвеєрних обчислень не враховується.

Істотне зростання часу декодування при збільшенні кількості виправляємих помилок пов'язане з тим, що при послідовному знаходженні синдрому помилки тут використовується тільки один помножувач. Можна прискорити цю операцію, використовуючи векторний помножувач, за рахунок пропорційного зростання апаратних витрат.

Табл. 2. Апаратні витрати декодера

t	Довжина кадру *N*, слів			
	32	64	128	255
2	8 1324	8 1399	8 1417	8 1567
3	11 1879	11 1882	11 1888	11 1998
4	14 2389	14 2546	14 2638	14 2755
5	17 3052	17 3086	17 3127	17 3196
6	20 3667	20 3714	20 3741	20 3941
7	23 4163	23 4204	23 4252	23 4327
8	26 4425	26 4663	26 4697	26 4712

У табл. 2 зазначені апаратні витрати на декодер, параметризований для різного числа слів що виправляються, та довжини пакета при його конфігуруванні в ПЛІС Xilinx серій Virtex-4 і Spartan-3E. При цьому в осередках вказано кількість блоків пам'яті (BRAM) і кількість конфігурованих логічних блоків (CLB slices). Дані отримані при синтезі проекту декодера в середовищі ISE 9.2.

Табл. 3. Максимальна тактова частота декодеру, МГц

Серія ПЛИС	N = 32, t=2	N =255, t = 8
Spartan 3 xc3s1500-5	74	66
Spartan 3E xc3s1200e-5	91	72
Spartan 3A xc3sd1800a-4	80	67
Virtex4 xc4vlx40-12	149	126
Virtex5 xc5vlx50-3	184	143

Максимальна тактова частота, з якою працює декодер, визначає як його пропускну здатність, так і можливості його впровадження в різних проектах СНК (SoC). У табл. 3 показана максимальна тактова частота роботи в різних ПЛІС фірми Virtex.

Як по тактовій частоті, так і по апаратним витратам синтезований декодер відповідає кращим світовим зразкам декодерів Ріда-Соломона, реалізованим в ПЛІС.

Висновок

Розроблений декодер Ріда-Соломона, завдяки врахуванню структурних властивостей сучасних ПЛІС, реалізується в них з високою тактовою частотою при помірних апаратних витратах, які можна порівняти з параметрами кращих світових зразків.

Разом з тим, є можливості істотного поліпшення декодера щодо збільшення швидкодії. Для цього потрібно розпаралелити процес знаходження синдрому помилки, застосовуючи векторний помножувач в полі Галуа. Також доцільно збільшити тактову частоту роботи декодера при реалізації декодера в серіях ПЛІС Xilinx Spartan 3 та Altera Cyclone, які відрізняються високим відношенням ефективність-ціна.

Список літератури

[1]. Вернер М. Основы кодирования. –М.: Техносфера. –2004. –288с.

[2] Касперски К. Могущество кодов Рида-Соломона или информация, воскресшая из пепла // Системный администратор. –2004. –с.88-94.

[3] Блейхут Р. Теория и практика кодов, контролирующих ошибки. –М.:Мир. –1986. –566с.