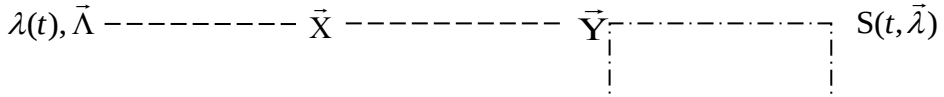
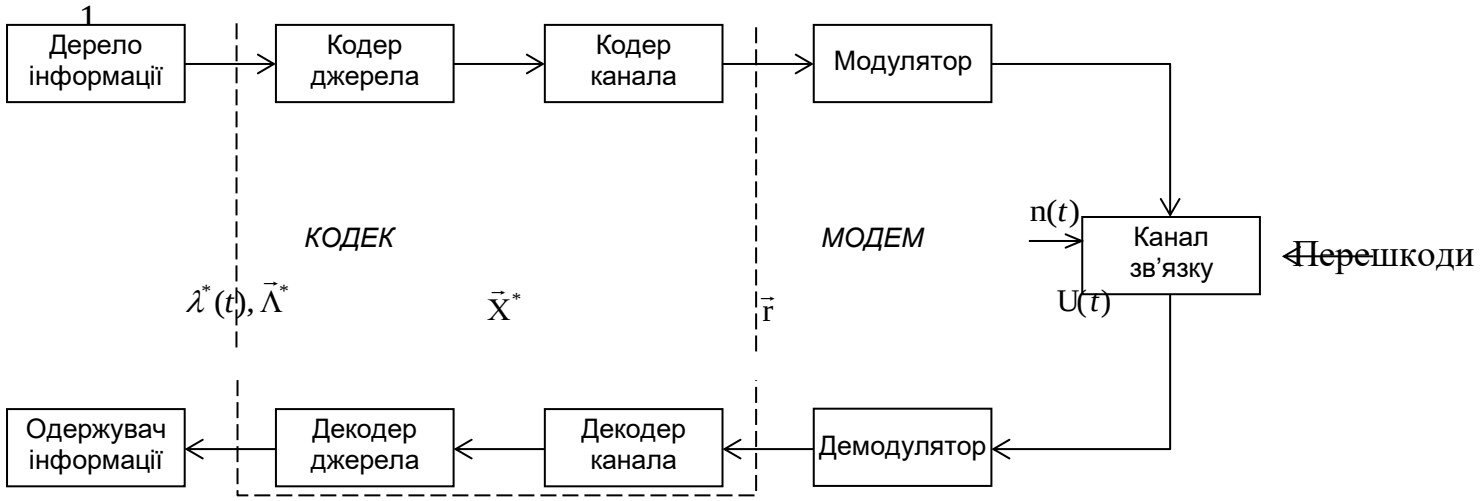
Зміст

ЗАВАДОСТІЙКЕ (ПЕРЕШКОДОСТІЙКЕ) КОДУВАННЯ





Модель системи передачі (і зберігання) інформації

Отже, ми розглянули основи економного кодування даних, або кодування джерела в системах передачі інформації. Завдання кодера джерела – представити дані, що підлягають передачі, в максимально компактній і, по можливості, неспотвореній формі.

При передачі інформації по каналу зв'язку з перешкодами в прийнятих даних можуть виникати помилки. Якщо такі помилки мають невелику величину або виникають досить рідко, інформація може бути використана споживачем. При великому числі помилок отриманою інформацією користуватися не можна.

Для зменшення кількості помилок, що виникають при передачі інформації по каналу з перешкодами, може бути використане кодування в каналі, або перешкодостійке кодування.

Можливість використання кодування для зменшення числа помилок в каналі була теоретично показана К. Шенноном в 1948 році в його роботі "Математична теорія зв'язку". У ній було зроблено твердження, що якщо швидкість створення джерелом повідомлень (продуктивність джерела) не перевищує деякої величини (пропускної спроможності каналу), то при відповідному кодуванні і декодуванні можна звести ймовірність помилок в каналі до нуля.

Незабаром стало ясно, що фактичні обмеження на швидкість передачі встановлюються не пропускною спроможністю каналу, а складністю схем кодування і декодування. Тому зусилля розробників і дослідників в останні десятиліття були направлені на пошуки ефективних кодів, створення схем кодування і декодування, що практично реалізуються і по своїх характеристиках наближалися б до передбачених теоретично.

Надійність — властивість технічних об'єктів зберігати у часі в установлених межах значення всіх параметрів, які характеризують здатність виконувати потрібні функції в заданих режимах та умовах застосування, технічного обслуговування, зберігання та транспортування. Під технічними об'єктами розуміють пристрої, прилади, механізми, машини, комплекси обладнання, будівельні конструкції і споруди, технологічні операції і процеси, системи зв'язку, інформаційні системи, автоматизовані системи управління технологічними процесами тощо.

Приклади надмірності

RAID (англ. *redundant array of independent/inexpensive disks*) — надлишковий масив незалежних/недорогих дисків для комп'ютера. Дисковий масив — це набір дискових пристроїв, що працюють разом, щоб підвищити швидкість і надійність системи вводу/виводу. Цим набором пристроїв управляє спеціальний RAID-контролер (контролер масиву), який забезпечує функції розміщення даних по масиву; а для решти всієї системи дозволяє представляти весь масив як один логічний пристрій вводу/виводу. За рахунок паралельного виконання операцій читання і запису на кількох дисках, масив забезпечує підвищену швидкість обмінів в порівнянні з одним великим диском.

Масиви також можуть забезпечувати надмірне зберігання даних, з тим, щоб дані не були втрачені у разі виходу з ладу одного з дисків. Залежно від рівня RAID, проводиться або дзеркалювання або розподіл даних по дисках.

Каліфорнійський університет в Берклі представив наступні рівні специфікації RAID, які були прийняті як стандарт де-факто:

• RAID 0 - дисковий масив підвищеної продуктивності з чергуванням, без відмовостійкості;

• RAID 1 - дзеркальний дисковий масив;

• RAID 2 зарезервований для масивів, які застосовують код Хеммінга;

• RAID 3 і 4 - дискові масиви з чергуванням і виділеним диском парності;

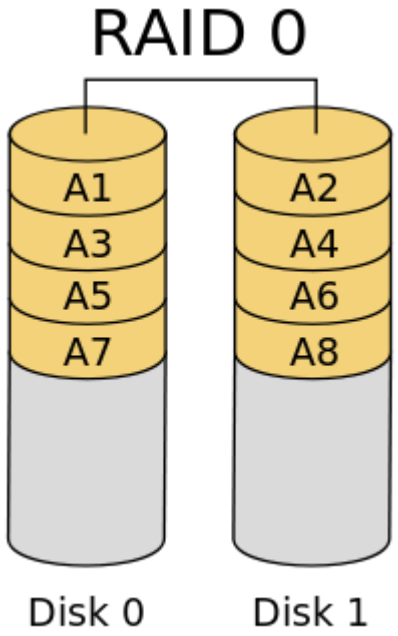
• RAID 5 - дисковий масив з чергуванням і «невиділений диском парності»;

• RAID 6 - дисковий масив з чергуванням, що використовує дві контрольні суми, що обчислюються двома незалежними способами;

• RAID 10 - масив RAID 0, побудований з масивів RAID 1;

• RAID 50 - масив RAID 0, побудований з масивів RAID 5;

• RAID 60 - масив RAID 0, побудований з масивів RAID 6.



Технологія RAID 0 також відома як розподіл даних (англ. *data striping*). Із застосуванням цієї технології інформація розбивається на шматки (фіксовані обсяги даних, зазвичай називаються блоками); і ці шматки записуються на диски

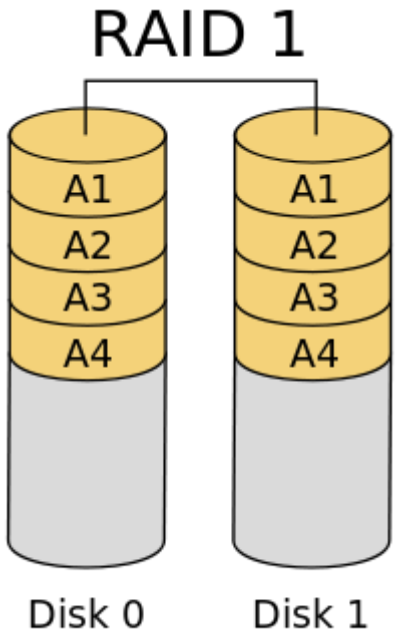
і прочитуються з них паралельно. З погляду швидкодії це означає дві основні переваги:

- підвищується пропускна спроможність послідовного вводу/виводу за рахунок одночасного завантаження кількох інтерфейсів.
- знижується латентність випадкового доступу; декілька запитів до різних невеликих сегментів інформації можуть виконатися одночасно (властивість об'єктів або процесів перебувати у прихованому стані, не виявляючи себе).

Недолік:

- рівень RAID 0 призначений виключно для підвищення швидкодії, і не забезпечує надмірності даних. Тому будь-які дискові збої вимагають відновлення інформації з резервних носіїв.

RAID рівня 1



Технологія RAID 1 також відома як дзеркалювання (англ. *disk mirroring*). В цьому випадку, копії кожного шматка інформації зберігаються на окремому диску; або, зазвичай кожен (використовуваний) диск має «двійника», який зберігає точну копію цього диска. Якщо відбувається збій одного з основних дисків, то він підмінюється своїм «двійником». Продуктивність довільного читання може бути покращена, якщо для читання інформації використовуватиметься той з «двійників», головка якого розташована ближче до необхідного блоку.

Час запису може бути дещо більшим, ніж для одного диска, залежно від стратегії запису: запис на два диски може проводиться або в паралель (для швидкості), або строго послідовно (для надійності).

Рівень RAID 1 добре підходить для додатків, які вимагають високої надійності, низької латентності при читанні, а також якщо не потрібна мінімізація вартості. RAID 1 забезпечує надмірність зберігання інформації, але у будь-якому випадку слід підтримувати резервну копію даних, оскільки це єдиний спосіб відновити випадково видалені файли або директорії.

Обчислювальні системи

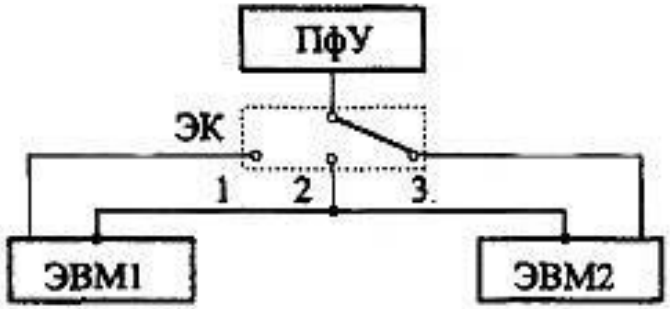
Положення 1 і 3 електронного ключа (ЕК) забезпечувало режим підвищеної надійності. При цьому одна з машин виконувала обчислення, а інша перебувала в "гарячому" або "холодному" резерві, тобто в готовності замінити основну ЕОМ. Положення 2 електронного ключа відповідало нагоди, коли обидві машини забезпечували паралельний режим обчислень. Тут можливі дві ситуації:

а) обидві машини вирішують одну і ту ж задачу і періодично звіряють результати, рішення. Тим самим забезпечувався режим підвищеної достовірності, зменшувалася ймовірність появи помилок в результатах обчислень. Приблизно за такою ж схемою побудовані керуючі бортові обчислювальні комплекси космічних апаратів, ракет, кораблів. Наприклад, американська космічна система "Шатл" містила п'ять обчислювальних машин, що працюють за такою схемою;

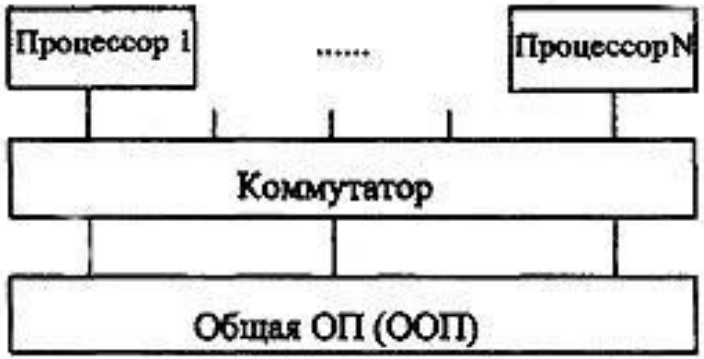
б) обидві машини працюють паралельно, але обробляють власні потоки завдань. Можливість обміну інформацією між машинами зберігається. Цей вид роботи відноситься до режиму підвищеної продуктивності. Вона широко використовується в практиці організації робіт на великих обчислювальних центрах, оснащених декількома ЕОМ високої продуктивності.

Схема, представлена на рис. а, була неодноразово повторена в різних модифікаціях при проектуванні різноманітних спеціалізованих ММС. Основні відмінності ММС полягають, як правило, в організації зв'язку і обміну інформацією між ЕОМ комплексу. Кожна з них зберігає можливість автономної роботи і управляється власною ОС. Будь-яка інша підключається ЕОМ комплексу розглядається як спеціальне периферійне устаткування.

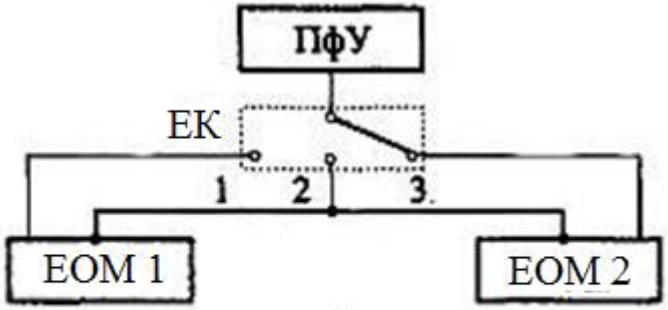
Багатопроцесорні обчислювальні системи (МПС) будуються при комплексуванні декількох процесорів (рис. Б). Як загальне ресурсу вони мають загальну оперативну пам'ять (ООП). Паралельна робота процесорів і використання ООП забезпечуються під управлінням єдиної загальної операційної системи. У порівнянні з ММС тут досягається найвища оперативність взаємодії обчислювачів-процесорів. Багато дослідників вважають, що використання МПС є основним магістральним шляхом розвитку обчислювальної техніки нових поколінь.


ПфУ
ЭК
1
2
3
ЭВМ1
ЭВМ2

а


Процессор 1
.....
Процессор N
Коммутатор
Общая ОП (ООП)

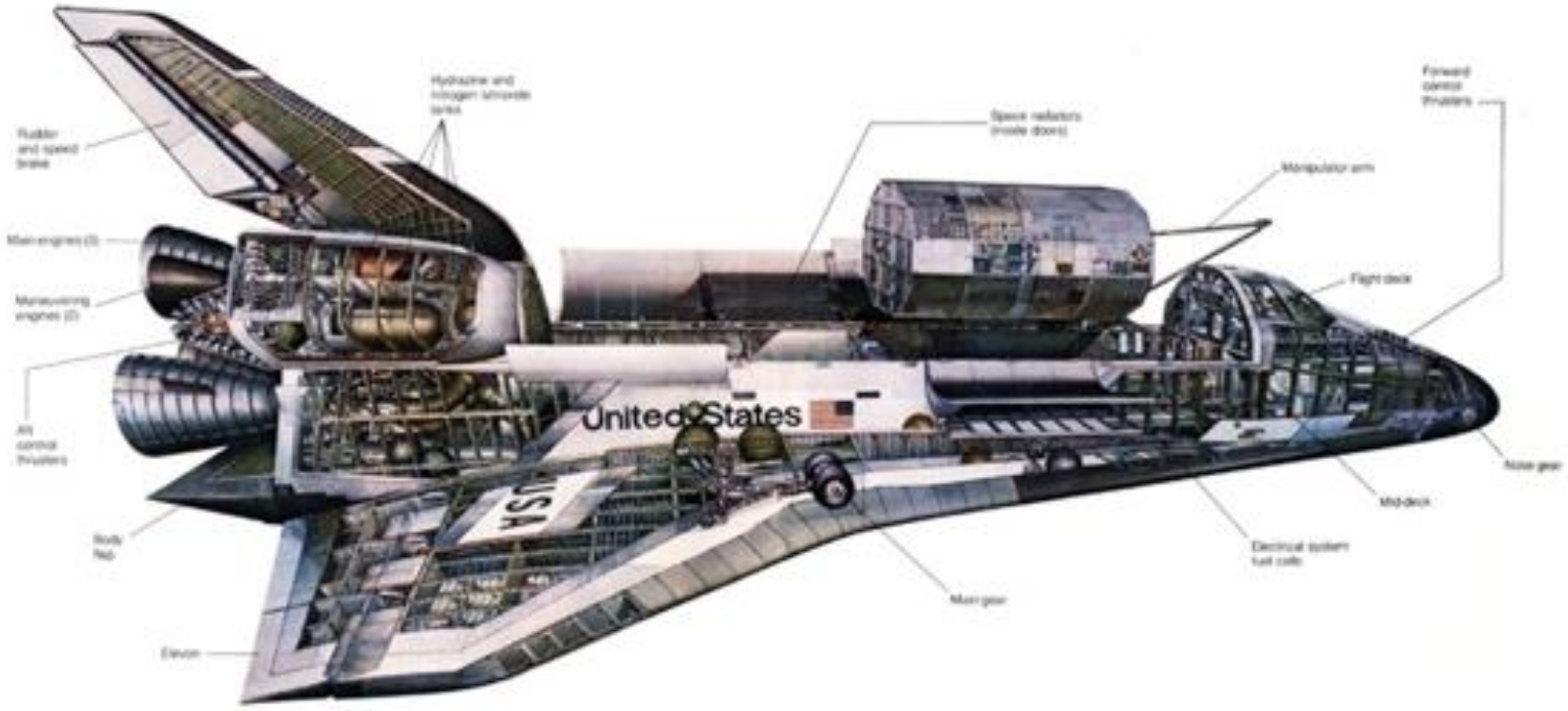
б


ПфУ
ЕК
1
2
3
ЕОМ 1
ЕОМ 2

а


Процесор 1
.....
Процесор N
Комутатор
Загальна ОП

б

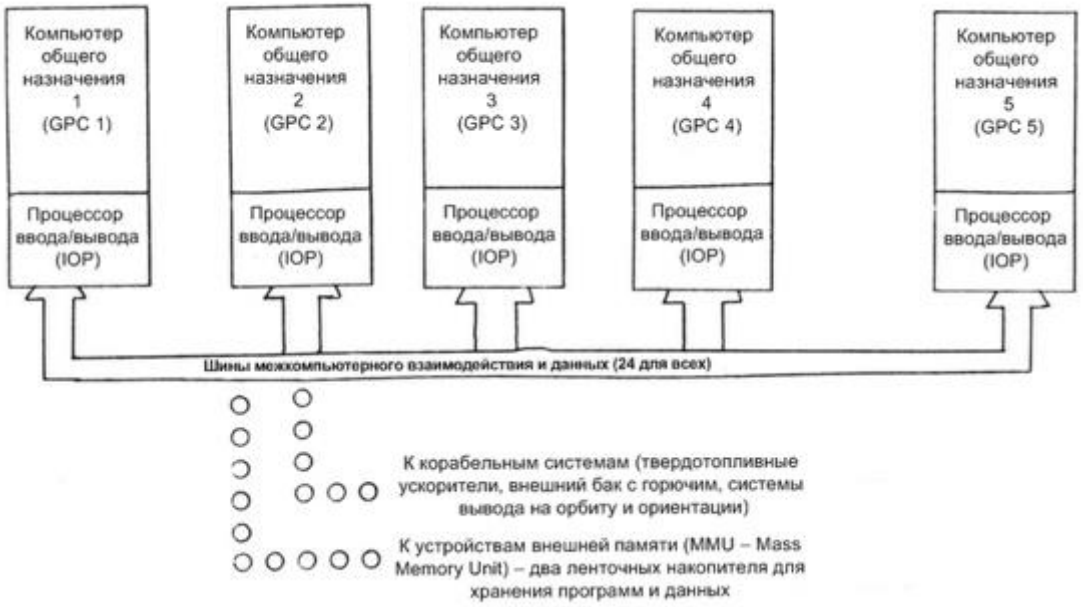

Hydrazine and nitrogen tetroxide tanks
Space radiators (inside doors)
Forward control thrusters
Rudder and speed brake
Manipulator arm
Main engines (3)
Flight deck
Maneuvering engines (2)
United States
USA
Aft control thrusters
Nose gear
Mid-deck
Body flap
Electrical system fuel cells
Main gear
Elevon


USA
NASA
Discovery

Вибір самолетоподобной конструкції космічного корабля поставив перед інженерами ряд складних завдань, однією з яких була комп'ютерна система човника. Очевидно, що комп'ютерне наповнення космічного літака повинно бути не в приклад складніше комп'ютерів, що використовуються в більш ранніх пілотованих місіях. Чому? Та хоча б тому, що комп'ютери шаттла на відміну від комп'ютерів програм Gemini і Appolo, крім навігаційних функцій і систем телеметрії польоту повинні були відповідати за управління самим човником.

Розвивається семимильними кроками реактивна авіація, довела, що самостійно впоратися з сотнями функціональних підсистем мчить на надзвуковий швидкості літака пілот просто не здатний. На допомогу йому була запропонована авіоніка (авіаційна електроніка) - комп'ютерні системи, що підтримують всі фази польоту в заданих параметрах і активно реагують на дії пілота.

Але шаттл - не звичайний реактивний літак. Його комп'ютерні системи повинні працювати не тільки в режимі авіоніки, а й забезпечувати: орієнтацію корабля на орбіті, навігацію, стикування з космічними станціями і запуск в експлуатацію таких об'єктів, як, наприклад, супутники.



Виходячи з добре відпрацьованої в той час троірованої системи з мажоруванням, прийняти рішення про правильний сигнал управління можна було тільки в разі видачі його не менше ніж трьома комп'ютерами. Значить, надлишкова схема GPC, здатна мінімум два рази відмови і зберегти при цьому три працездатні машини, повинна складатися з п'яти ЕОМ. Проста арифметика. Трохи пізніше число надлишкових машин було скорочено до чотирьох, але п'ятий комп'ютер все одно залишався "на підхваті" з резервною копією польотної програми.

Такий підхід в корені відрізнявся від комп'ютерних реалізацій місій Gemini і Appolo, де основна комп'ютерна система просто одноразово дублювалася, і дубль включався тільки при явному відмову основного комп'ютера.

Але ці пілотовані програми працювали в режимі балістичного запуску і некерованої посадки. Комп'ютери же шаттла працювали і на зльоті і на орбіті і при посадці. Будь-який промах в будь-який з цих моментів може виявитися фатальним.

Тому, всі п'ять GPC розкидані по різних кутах човника і повна відмова мінімум двох з них ніяк не вплине на політ корабля.

Але раз комп'ютери GPC (чи то пак, AP-101) голосували за правильність сигналу, значить вони були пов'язані. І ще як! Шинна архітектура, що зв'язує все на шатлі, починаючи від GPC і закінчуючи, наприклад, контролерами закрилків, - навіть більш вражаюче творіння ніж "пятиголова" комп'ютерна система.

Кожен з блоків GPC може працювати з шиною в двох режимах: "командувати" і "слухати". Командувати будь-яким компонентом польотної програми може тільки один з GPC. Решта ж тільки слухають його дії. Таким чином, кожен з п'яти GPC командує п'ятою частиною справ на човнику і слухає

залишилися чотири п'ятих. Тобто, будь-який з них "в курсі" всіх справ, які виконуються іншими обчислювальними побратимами.

Як же відбувається те саме надмірне голосування, "що обчислює" несправний комп'ютер? GPC, слухаючи командувачі ЕОМ, дубльовано виконують їх прошивки. Виконання кожної з них комп'ютери завершують формуванням трехбітного коду, однозначно визначає вид процедури (наприклад, "010 - висновок виконаний без помилки"). Якщо всі комп'ютери з надлишкового набору генерують один і той же код, значить все в нормі, якщо ж один або навіть два з них брешуть, три залишилися з правильним кодом можуть зрозуміти, хто з п'ятірки дає збої.

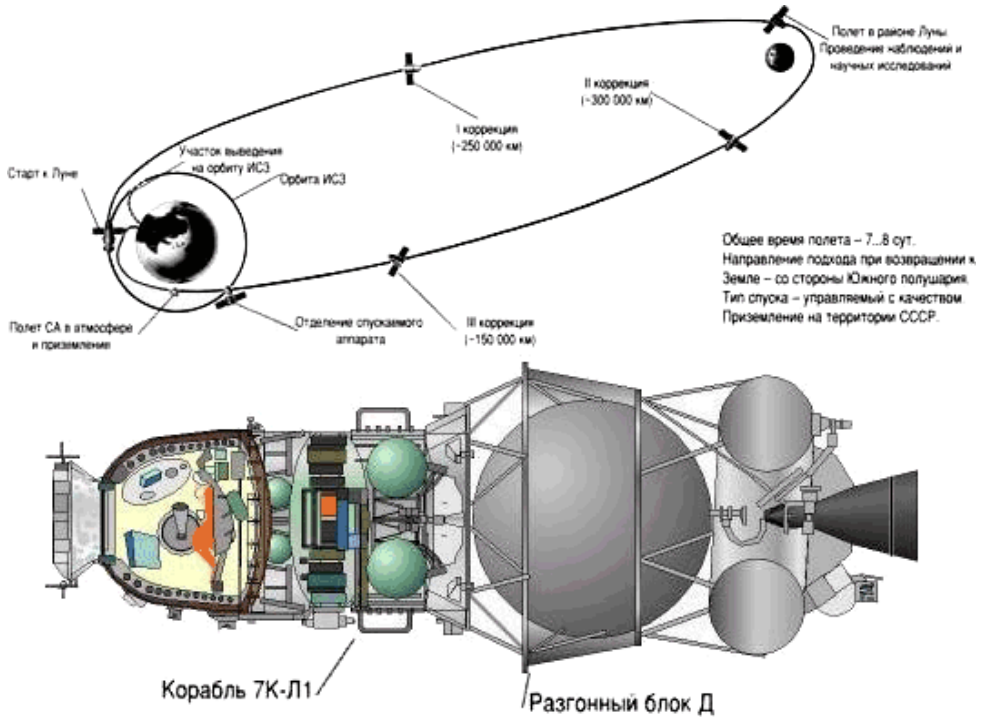
Така схема перевірки створює сильнозв’язану групу GPC. Щоб переконатися, що вся група працює злагоджено, використовується 64-бітна структура під назвою "sumword". Вона відправляється в шину кожним GPC кожні 6,25 секунди і містить біти останніх викликів до ключових систем корабля. Порівнюючи чужі "sumword" зі своїм, кожен GPC раз в 6,25 секунди переконується, що його надлишкові брати працездатні. Чи ні.

Мажоритарні комплекси

"Аргон" на борту. місія здійсненна

Космічна місія, для якої співробітники НІІЕМ створювали бортову ЕОМ, була вельми відповідальною. Апарати серії "Зонд", сконструйовані на основі пілотованого корабля "Союз 7К-Л1", повинні були дослідити можливість висадки на Місяці радянських космонавтів. Завдання це була політично важливою. Програма "Аполлон", відпрацьовує NASA з початку шістдесятих років, до 1968 року увійшла в стадію пілотованих польотів, і керівництво СРСР бажала утерти ніс потенційному супротивникові.

Космічні апарати серії "Зонд", що літали до Місяця, невипадково були засновані на пілотованих кораблях "Союз". На них одного разу повинні були відправитися і радянські космонавти.

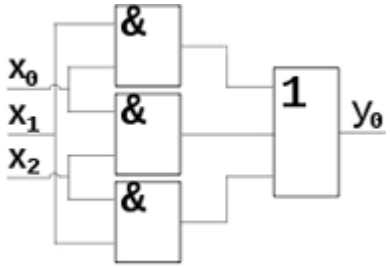


Застосування в польотах подібного типу бортових ЕОМ було надзвичайно важливим. Політ складався з декількох фаз, у кожній з яких був потрібний точний розрахунок в реальному масштабі часу безлічі параметрів роботи систем корабля. Автоматика на основі програмно-тимчасових пристроїв (ПВУ) тут непридатна: аж надто непередбачувані умови польоту. А ось цифрова ЕОМ з її гнучкою програмованої логікою підходила для цих цілей ідеально. Тим більше що БЦВМ серії "Аргон-1", розроблені в НІІЕМ для мобільного оперативно-тактичного ракетного комплексу "Точка", довели свою ефективність в управлінні складними процесами.

Розуміючи це, інженери НІІЕМ зробили "Аргон-11С" ... триголового. У буквальному сенсі цього слова.

В "Аргон-11С" вперше в практиці створення бортових ЕОМ була застосована схема резервування вузлів, яка іменувалася троірованою структурою з мажорітіруванієм. За цим мудрованим назвою ховається елегантна за своєю ідеєю конструкція.

Структурно "Аргон-11С" складався з трьох однакових функціональних блоків, що працюють паралельно і незалежно один від одного. На входи кожного блоку (всього їх було 28) надходила зовсім однакова інформація від безлічі датчиків телеметрії. На її основі кожен блок виробляв більше сорока керуючих впливів.



Мажоритарні логічні схеми відомі досить давно. Сигнал на їх виході залежить від однакових сигналів на більшості входів

Фактично в "Аргон-11С" постійно проходило голосування за найбільш правильне керуючий вплив. А щоб ви не подумали, що трійця обчислювальних блоків постійно прагнула організувати коаліцію проти меншості, знайте, що між їх вхідними та вихідними каналами були зв'язки, що дозволяють обмінюватися інформацією в разі, якщо вона в одному або декількох блоках спотворювалася.

Ще однією важливою особливістю "Аргон-11С" було застосування інтегральних схем. Спеціально для цієї серії фахівцями НІІЕМ спільно з інженерами науково-дослідного інституту точної технології НІІТТ були розроблені гібридні інтегральні схеми серії "Стежка" - фактично перші радянські інтегральні схеми.

Конструкція троірованной схеми "Аргон-11С" була настільки вдалою, що в подальшому вона була повторена з БЦВМ "Аргон-16", яку сміливо можна назвати космічним довгожителем. Ця ЕОМ використовувалася в найрізноманітніших космічних апаратах більше 25 років! Близько трьохсот примірників "Аргон-16" працювали в "Союзах", транспортниках "Прогрес", орбітальних станціях "Салют" і "Мир". Повірте, для ЕОМ космічного базування це велика цифра.

Ось він - космічний Горинич, бортова цифрова обчислювальна машина "Аргон-16"

Зате надійність цієї машини, офіційно зафіксована в її документації, приголомшує. Імовірність відсутності відмов у двох з трьох її модулів (а для голосування за більшістю потрібно мінімум два працездатних модуля) становить 0,999 протягом восьми діб польоту космічного апарату до Місяця і назад.

Обчислювальна керуюча система М6000 АСВТ-М (модифікованої). "Робоча конячка" для багатьох тисяч управляючих систем і комплексів різного призначення 70-ті роки XX століття

1.1. Основні принципи. Типи кодів

Кодування з виправленням помилок є методом обробки повідомлень, призначеним для підвищення надійності передачі по цифрових каналах. Хоча різні схеми кодування дуже несхожі один на одного і засновані на різних математичних теоріях, всім їм властиві дві загальні властивості.

Перше − використання надмірності. Закодовані послідовності завжди містять додаткові, або надлишкові, символи. Кількість символів в кодовій послідовності $\boldsymbol{Y}$ завжди більша, ніж необхідно для однозначного представлення будь-якого повідомлення $\boldsymbol{\lambda_i}$ з алфавіту.

Друге — властивість усереднення, що означає, що надлишкові символи залежать від декількох інформаційних символів, тобто інформація, що міститься в кодовій послідовності $\boldsymbol{X}$, перерозподіляється також і на надлишкові символи.

Існує два великі класи коректуючи кодів − блочні і згортальні. Визначальна відмінність між цими кодами полягає у відсутності або наявності пам'яті кодера.

Кодер для блочних кодів ділить безперервну інформаційну послідовність X на блоки-повідомлення довжиною k символів.

Кодер каналу перетворить блоки-повідомлення X в довші двійкові послідовності Y, що складаються з n символів і звані кодовими словами. Символи (n-к), що додаються до кожного блоку-повідомлення кодером, називаються надлишковими. Вони не несуть жодної додаткової інформації, і їх функція полягає в забезпеченні можливості виявляти (або виправляти) помилки, що виникають в процесі передачі.

Як ми раніше показали, ***k***-розрядним двійковим словом можна представити 2^k можливих значень з алфавіту джерела, їм відповідає 2^k кодових слів на виході кодера.

Така сукупність 2^k кодових слів називається блочним кодом.

Термін "без пам'яті" означає, що кожен блок з n символів залежить лише від відповідного інформаційного блоку з ***k*** символів і не залежить від інших блоків.

Кодер для згортальних кодів працює з інформаційною послідовністю без розбиття її на незалежні блоки. У кожен момент часу кодер з невеликого поточного блоку інформаційних символів розміром в b символів (блоку-повідомлення) утворює блок, що складається з v кодових символів (кодовий блок), причому v > b. При цьому кодовий v-символьний блок залежить не лише від b-символьного блока-повідомлення, присутнього на вході кодера зараз, але і від попередніх m блоків-повідомлень. У цьому, власне, і полягає наявність пам'яті в кодері.

Блочне кодування зручно використовувати в тих випадках, коли вихідні дані за своєю природою вже згруповані в які-небудь блоки або масиви.

При передачі по радіоканалах частіше використовується згортальне кодування, яке краще пристосоване до побітової передачі даних. Окрім цього, при однаковій надмірності згортальні коди, як правило, володіють кращою виправною здатністю.

1.2. Лінійні блочні коди

Для блочного коду з 2^k кодовими словами завдовжки в n символів, якщо він лише не володіє спеціальною структурою, апарат кодування і декодування є дуже складним. Тому обмежимо свій розгляд лише кодами, які можуть бути реалізовані на практиці.

Однією з умов реалізованості блочних кодів при великих k є умова їх лінійності.

Що таке лінійний код?

Блочний код довжиною n символів, що складається з 2^k кодових слів, називається лінійним (n, k) - кодом за умови, що все його 2^k кодових слів утворюють k-мірний підпростір векторного простору n-послідовностей двійкового поля GF(2).

Якщо сказати простіше, то двійковий код є лінійним, якщо додавання по модулю 2 ***(mod2)*** двох кодових слів також є кодовим словом цього коду.

Працюючи з двійковими кодами, ми постійно стикатимемося з елементами двійкової арифметики, тому визначимо основні поняття.

Полем називається безліч математичних об'єктів, які можна додавати, віднімати, множити і ділити.

Візьмемо просте поле, що складається з двох елементів − нуля - 0 і одиниці - 1. Визначимо для нього операції додавання і множення:

$$0+0=0, \qquad 0\cdot 0=0;$$
$$0+1=1, \qquad 0\cdot 1=0;$$
$$1+0=1, \qquad 1\cdot 0=0;$$
$$1+1=0, \qquad 1\cdot 1=1.$$

Визначені таким чином операції додавання і множення називаються додаванням по модулю 2 ***(mod2)*** і множенням по модулю 2.

Відзначимо, що з рівності 1+1 = 0 випливає, що -1 = 1 і, відповідно, 1+1=1-1, а з рівності 1·1=1 − що 1:1=1.

Алфавіт з двох символів 0 і 1 разом із додаванням і множенням по mod2 називається полем з двох елементів і позначається як GF(2). До поля GF(2) застосовні всі методи лінійної алгебри, у тому числі матричні операції.

Ще раз звернемо увагу на те, що всі дії над символами в двійкових кодах виконуються по модулю 2.

Бажаною якістю лінійних блочних кодів **є систематичність.**

Систематичний код має формат, зображений на мал. 1.1, тобто містить незмінну інформаційну частину завдовжки $\boldsymbol{k}$ символів і надлишкову (перевірочну) довжиною $\boldsymbol{n-k}$ символів.

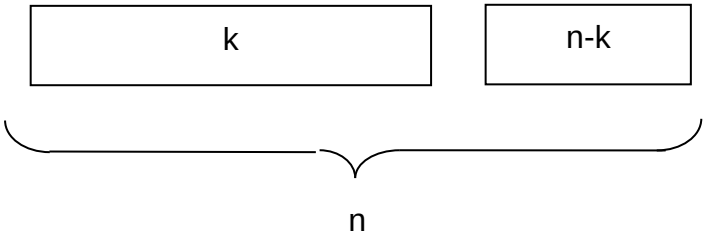


Рис. 1.1

Блочний код, що володіє властивостями лінійності і систематичності, називається лінійним блочним систематичним ($\boldsymbol{n, k}$) -кодом.

1.2.1. Код з перевіркою на парність

Найпростішим лінійним блочним кодом є ***(n, n-1)***- код, побудований за допомогою однієї загальної перевірки на парність. Наприклад, кодове слово (4,3)- коду можна записати у вигляді вектора-стовпця:

$$\overline{U}^T = (m_0, m_1, m_2, m_0+m_1+m_2), \qquad (1.1)$$

де m_i - символи інформаційної послідовності, що набувають значень 0 і 1, а додавання проводиться по модулю 2 ***(mod2).***

Пояснимо основну ідею перевірки на парність.

Нехай інформаційна послідовність джерела має вигляд

$$m = (1\ 0\ 1). \qquad (1.2)$$

Тоді відповідна їй кодова послідовність виглядатиме таким чином :

$$U = (U_0, U_1, U_2, U_3) = (1\ 0\ 1\ 0), \qquad (1.3)$$

де перевірочний символ U_3 формується шляхом додавання по ***mod2*** символів інформаційної послідовності m :

$$U_3 = m_0 + m_1 + m_2. \qquad (1.4)$$

Неважко відмітити, що якщо число одиниць в послідовності ***m*** парне, то результатом додавання буде 0, якщо непарне − 1, тобто перевірочний символ доповнює кодову послідовність так, щоб кількість одиниць в ній була парною.

При передачі по каналах зв'язку в прийнятій послідовності можлива поява помилок, тобто символи прийнятої послідовності можуть відрізнятися від відповідних символів переданої кодової послідовності (нуль переходить в одиницю, а 1 − у 0).

Якщо помилки в символах мають однакову ймовірність і незалежні, то ймовірність того, що в ***n***-позиційному коді станеться лише одна помилка, буде

$$P_1 = n \cdot P_{пом} \cdot (1 - P_{пом})^{n-1} \qquad (1.5)$$

(тобто в одному біті помилка є, а у всіх останніх ***n - 1*** бітах помилки немає).

Ймовірність того, що станеться дві помилки, визначається вже числом можливих поєднань помилок по дві (у двох довільних бітах помилка є, а у всіх останніх ***n - 2*** бітах помилки немає):

$$P_2 = C_n^2 \cdot P_{пом} \cdot (1 - P_{пом})^{n-2}, \qquad (1.6)$$

і аналогічно для помилок вищої кратності.

Якщо вважати, що ймовірність помилки на символ прийнятої послідовності $P_{пом}$ достатньо мала ($P_{пом} << 1$), а інакше передача інформації не має сенсу, то ймовірність випадання рівно l помилок складе $P_l \cong P_{пом}^{\,l}$.

Звідси видно, що найбільш ймовірними є одиночні помилки, менш ймовірними — подвійні, ще меншу ймовірність матимуть трикратні помилки і так далі.

Якщо при передачі даного (4,3) - коду сталася одна помилка, причому не важливо, в якій його позиції, то загальне число одиниць в прийнятій послідовності $\boldsymbol{r}$ вже не буде парним.

Таким чином, ознакою відсутності помилки в прийнятій послідовності може служити парність числа одиниць. Тому такі коди і називаються кодами з перевіркою на парність.

Правда, якщо в прийнятій послідовності $\boldsymbol{r}$ сталися дві помилки, то загальне число одиниць в ній знову стане парним і помилка виявлена не буде. Проте ймовірність подвійної помилки значно менша ймовірності одиночної, тому найбільш ймовірні одиночні помилки таким кодом виявлятися все ж будуть.

На підставі загальної ідеї перевірки на парність і перевірочного рівняння (1.4) легко організувати схему кодування - декодування для довільного коду з простою перевіркою на парність.

Відзначимо наступний момент. Якщо посимвольно скласти два кодові слова, що належать даному (4, 3) - коду:

$$\boldsymbol{a = (a_0, a_1, a_2, a_0 + a_1 + a_2),} \quad \boldsymbol{b = (b_0, b_1, b_2, b_0 + b_1 + b_2),} \qquad (1.7)$$

то отримаємо

$$c = (a_0+b_0, a_1+b_1, a_2+b_2, a_0+b_0+a_1+b_1+a_2+b_2) = (c_0, c_1, c_2, c_0+c_1+c_2), \qquad (1.8)$$

тобто перевірочний символ в новому слові c визначається за тим же правилом, що і в доданках. Тому c також є кодовим словом даного коду.

Цей приклад відображає важливу властивість лінійних блочних кодів — **замкнутість,** що означає, що сума двох кодових слів даного коду також є кодовим словом.

Не дивлячись на свою простоту і не дуже високу ефективність, коди з перевіркою на парність широко використовуються в системах передачі і зберігання інформації. Вони цінуються за невисоку надмірність: досить додати до послідовності, що передається всього один надлишковий символ − і можна взнати, чи є в прийнятій послідовності помилка. Правда, визначити місце цієї помилки і виправити її, поки не можна. Можна лише повторити передачу слова, в якому була допущена помилка, і тим самим її виправити.

1.2.2. Ітеративний код

Ще одна проста схема кодування, яка також часто використовується, може бути побудована таким чином.

Передбачимо, що потрібно передати, наприклад, дев'ять інформаційних символів $m = (m_0 , m_1 , ..., m_8)$. Ці символи можна розташувати у вигляді

квадратної матриці, як це показано в таблиці. 1.1, і додати до кожного рядка і кожного стовпця цієї таблиці по перевірочному символу (перевірка на парність).

Таблиця 1.1

m_0	m_1	m_2	$P_1 = m_0 + m_1 + m_2$
m_3	m_4	m_5	$P_2 = m_3 + m_4 + m_5$
m_6	m_7	m_8	$P_3 = m_6 + m_7 + m_8$
$m_0 + m_3 + m_6$	$m_1 + m_4 + m_7$	$m_2 + m_5 + m_8$	$m_0 + m_0 + m_1 + m_1 + \ldots + m_8 + m_8$

Таким чином, по рядках і по стовпцях цієї таблиці виконуватиметься правило парності одиниць.

Якщо в процесі передачі по каналу з перешкодами в цій таблиці станеться одна помилка (наприклад в символі m4), то перевірка на парність у відповідному рядку і стовпці (у нашому прикладі - P_2 і P_5) не виконуватиметься.

Іншими словами, координати помилки однозначно визначаються номерами стовпця і рядка, в яких не виконуються перевірки на парність. Таким чином, цей код, використовуючи різні перевірки на парність (по рядках і по стовпцях), здатний не лише виявляти, але і виправляти помилки (якщо відомі координати помилки, то її виправлення полягає просто в заміні символу на протилежний: якщо 0, то на 1, якщо 1 – то на 0).

Описаний метод кодування, званий ітеративним, виявляється є корисним у разі, коли дані природним чином формуються у вигляді масивів, наприклад, на шинах ЕОМ, в пам'яті, що має табличну структуру, і так далі При цьому розмір таблиці в принципі не має значення (3х3 або 20х20), проте в першому випадку виправлятиметься одна помилка на 3х3=9 символів, а в другому – на 20х20=400 символів.

Звернемо увагу ще на один момент. Якщо в простому коді з перевіркою на парність для виявлення помилки доводиться додавати до інформаційної послідовності всього один символ, то для того, щоб код став виправляти однократну помилку, знадобилося до дев'яти інформаційних символів додати ще сім перевірочних.

Таким чином, надмірність цього коду виявилася дуже великою, а виправна здатність – порівняно низькою. Тому зусилля фахівців в області перешкодостійкого кодування завжди були направлені на пошук таких кодів і методів кодування, які при мінімальній надмірності забезпечували б максимальну виправну здатність.

1.2.3. Породжуюча матриця лінійного блочного коду

Тільки що як приклад були розглянуті два прості корегуючи коди - код з простою перевіркою на парність, що дозволяє виявляти однократну помилку в прийнятій послідовності, і блочний ітеративний код, що виправляє одну помилку за допомогою набору перевірок на парність по рядках і стовпцях таблиці. Проте формальне правило, по якому здійснюється кодування, тобто перетворення інформаційної послідовності в кодове слово, по-справжньому ще не визначене. Тож як же ж задаються блочні коди? Простим способом опису, або задавання корегуючи кодів є **табличний спосіб**, при якому кожній інформаційній послідовності ставиться у відповідність кодове слово з таблиці коди (таблиця. 1.2)

Таблиця 1.2

M	***U***
000	***0000***
001	***0011***
010	***0101***
011	***0110***
100	***1001***
101	***1010***
110	***1100***

111	***1111***

Такий спосіб опису кодів, до речі, застосовний для будь-яких, а не лише лінійних кодів. Проте при великих ***k*** розмір кодової таблиці виявляється дуже великим, щоб ним користуватися на практиці (для коду з простою перевіркою на парність двобайтового слова розмір таблиці складе $\sim 2^5 * 2^{16} = 2000000$ двійкових символів).

Іншим способом задавання лінійних блочних кодів є використання так званої **системи перевірочних рівнянь**, що визначають правило, по якому символи інформаційної послідовності перетворяться в кодові символи. Для того ж прикладу система перевірочних рівнянь виглядатиме таким чином:

$$\begin{aligned} U_0 &= m_0, \\ U_1 &= m_1, \\ U_2 &= m_2, \\ U_3 &= m_0 + m_1 + m_2. \end{aligned} \qquad (1.9)$$

Проте найбільш зручним і наочним способом опису лінійних блочних кодів є їх завдання з використанням породжуючої матриці, яка являється компактною формою представлення системи перевірочних рівнянь:

$$\underline{G} = \left| \begin{array}{c|c} 1\ 0\ 0 \ldots 0 & P_{00}\ \ P_{01} \ldots .\ \ P_{0,\,n-k-1} \\ 0\ 1\ 0 \ldots 0 & P_{10}\ \ P_{11} \ldots .\ \ P_{1,\,n-k-1} \\ \ldots\ldots\ldots & \ldots\ldots\ldots\ldots\ldots \\ 0\ 0\ 0 \ldots 1 & P_{k-1,\,0}\ \ P_{k-1,\,1} \ldots .\ P_{k-1,\,n-k-1} \end{array} \right| . \qquad (1.10)$$

одинична матриця **I** k∗k

матриця **P** k∗(n- k)

Означення. *Лінійний блоковий систематичний (n,k) - код повністю визначається матрицею* $\underline{G}$ *розміром* $k * n$ *з двійковими матричними елементами. При цьому кожне кодове слово є лінійною комбінацією рядків матриці* $\underline{G}$*, а кожна лінійна комбінація рядків* $\underline{G}$ *- кодовим словом.*

Хай $m = (m_0, m_1, \ldots, m_{k-1})$ буде тим блоком-повідомленням, який необхідно закодувати з використанням даного коду.

Тоді відповідним йому кодовим словом U буде

$$U = m \cdot \underline{G}\,. \qquad (1.11)$$

З урахуванням структури матриці $\underline{\boldsymbol{G}}$ символи кодового слова U будуть такими:

для $\boldsymbol{i = 0, 1, 2,. . . , k- 1}$

$$\boldsymbol{U_i = m_i ;} \qquad (1.12)$$

для $\boldsymbol{i = k, k+1,. . . , n}$

$$\boldsymbol{U_i = m_0 \cdot P_{0j} + m_1 \cdot P_{1j} + m_2 \cdot P_{2j} + \ldots + m_{k-1} \cdot P_{k-1,\,j}\,.} \qquad (1.13)$$

Іншими словами, $\boldsymbol{k}$ крайніх лівих символів кодового слова збігається з символами кодованої інформаційної послідовності, а останні $(\boldsymbol{n - k})$ символів є лінійними комбінаціями символів інформаційної послідовності.

Визначений таким чином код називається лінійним блочним систематичним (n,k) - кодом з узагальненими перевірками на парність, а матриця $\underline{\boldsymbol{G}}$, що його задає називається породжуючою матрицею коду.

Як приклад розглянемо відомий (7,4) - код Хеммінга, що є класичною ілюстрацією простих кодів з виправленням помилок.

Хай $\boldsymbol{m} = (\boldsymbol{m_0, m_1, m_2, m_3})$ буде тим повідомленням, або інформаційною послідовністю, яку потрібно закодувати.

Породжуюча матриця $\underline{G}$, для (7. 4) - коду Хеммінга має вигляд

$$\underline{G}_{(7,4)} = \left| \begin{array}{ccccccc} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{array} \right| . \quad (1.14)$$

Тоді символи відповідного кодового слова визначаються таким чином :

$$U = m \cdot \underline{G} = (m_0 m_1 \; m_2 m_3) \left| \begin{array}{ccccccc} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{array} \right| =$$

$$= (m_0, m_1, m_2, m_3, m_0 + m_2 + m_3, m_0 + m_1 + m_2, m_1 + m_2 + m_3), \quad (1.15)$$

або

$$
\begin{aligned}
U_0 &= m_0, \\
U_1 &= m_1, \\
U_2 &= m_2, \\
U_3 &= m_3, \\
U_4 &= m_0 + m_2 + m_3, \\
U_5 &= m_0 + m_1 + m_2, \\
U_6 &= m_1 + m_2 + m_3.
\end{aligned}
\qquad (1.16)
$$

Наприклад, нехай m = (1 0 1 1), тоді відповідне кодове слово матиме вигляд U = (1 0 1 1 1 0 0). Або інший приклад: нехай

m = (1 0 0 0), тоді U = (1 0 0 0 1 1 0).

Цікаво відзначити, що відповідно до наведеного вище визначення, рядки матриці $\underline{G}$ самі є кодовими словами даного коду, а всі останні кодові слова - лінійними комбінаціями рядків породжуючої матриці.

На підставі породжуючої матриці $\underline{G}_{(7,4)}$ (1.15) або наведеної системи перевірочних рівнянь (1.16) легко реалізувати схему кодування для даного (7,4) - коду Хеммінга (мал. 1.4).

Кодер працює точно так, як і при простій перевірці на парність, але тепер виконує не одну загальну, а декілька часткових перевірок, формуючи, відповідно, декілька перевірочних символів.

1.2.4. Перевірочна матриця

Лінійний систематичний блочний код може бути визначений також з використанням так званої перевірочної матриці **H**, що володіє наступною властивістю:

- *якщо деяка послідовність U є кодовим словом, то*

$$\boldsymbol{U * \underline{H}^T = \underline{0}.} \qquad (1.17)$$

Іншими словами, перевірочна матриця H ортогональна будь-якій кодовій послідовності даного коду. Перевірочна матриця має розмірність (n-к)*n і наступну структуру :

$$\underline{\boldsymbol{H}} = \left| \begin{array}{cccc|ccccc} P_{00} & P_{10} & \dots & P_{k-1,\,0} & 1 & 0 & 0 & \dots & 0 \\ P_{01} & P_{11} & \dots & P_{k-1,\,1} & 0 & 1 & 0 & \dots & 0 \\ P_{22} & P_{12} & \dots & P_{k-1,\,2} & 0 & 0 & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ P_{0,\,n-k-1} & P_{1,\,n-k-1} & \dots & P_{k-1,\,n-k-1} & 0 & 0 & 0 & \dots & 1 \end{array} \right| , \qquad (1.18)$$

P^T — $\underline{I}^1_{(n-k)\times(n-k)}$

де $\underline{P}^T$ - транспонована підматриця P з породжуючої матриці $\underline{G}$;

$\underline{I}^1_{(n-k)\times(n-k)}$ - одинична матриця відповідного розміру.

Видно, що одинична і перевірочна підматриці в $\underline{G}$ и $\underline{H}$ помінялися місцями, крім того, змінився їх розмір.

Для прикладу, який ми розглядали, а точніше для (*7,4*)- коду Хеммінга перевірочна матриця $\underline{H}$ має вигляд

$$H_{(7,4)} = \left|\begin{array}{cccc:ccc} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{array}\right| . \qquad 1.19)$$

Перевірочна матриця дозволяє легко визначити, чи є прийнята послідовність кодовим словом даного коду.

Нехай, наприклад, є послідовність символів c = **(1011001)**, тоді

$$c * H^T = (1011001) \left|\begin{array}{cccc:ccc} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{array}\right|^T = (1\ 1\ 0) \neq \underline{0}.$$

Звідси можна зробити висновок, що послідовність **c = (1011001)** не є кодовим словом даного коду.

Розглянемо інший приклад. Припустимо, прийнята послідовність $\boldsymbol{d}$ = (**0010111**), тоді

$$\boldsymbol{d \cdot \underline{H}^T = (0010111)} \left|\begin{array}{cccc:ccc} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{array}\right|^T = (0\,0\,0) = \underline{0},$$

$$\boldsymbol{c * H^T = (0011001)} \left|\begin{array}{cccc:ccc} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{array}\right|^T = (1\,1\,0) \neq \underline{0}.$$

не код

тобто двійкова послідовність $\boldsymbol{d}$ належить коду з перевірочною матрицею $\underline{\boldsymbol{H}}$.

1.2.5. Дуальні коди

Розглядаючи матриці $\underline{\boldsymbol{G}}$ і $\underline{\boldsymbol{H}}$, можна зробити наступні цікаві висновки. Кожна з них містить безліч лінійно незалежних векторів, тобто кожна з матриць може розглядатися як базис деякого лінійного простору. Крім того, кожен з цих просторів є підпростором векторного простору, що складається зі всіх наборів двійкових символів довжиною n.

Скалярний добуток кожного рядка матриці $\underline{\boldsymbol{G}}$ на кожен рядок матриці $\underline{\boldsymbol{H}}$ дорівнює нулю, тобто

$$\underline{\boldsymbol{H}} \cdot \underline{\boldsymbol{G}}^{T} = \underline{\boldsymbol{0}} \quad \text{и} \quad \underline{\boldsymbol{G}} \cdot \underline{\boldsymbol{H}}^{T} = \underline{\boldsymbol{0}} . \qquad (1.20)$$

Отже, можна "поміняти ролями" ці дві матриці і використовувати $\underline{\boldsymbol{H}}$ як породжуючи матрицю, а $\underline{\boldsymbol{G}}$ як перевірочну матрицю деякого іншого коду.

Коди, зв'язані таким чином, називаються дуальними один одному, тобто, задавши яким-небудь чином лінійний блоковий код, ми автоматично задаємо і другий, дуальний йому код. Правда, якщо вихідний код був отриманий так, щоб мати мінімальну надмірність при заданій виправній здатності, то гарантувати хорошу якість дуального йому коду ми не можемо. Такі коди зазвичай мають виправну здатність, однакову з початковою, але більшу, ніж у них, надмірність.

Наприклад, якщо розглянутий як приклад (***7,4***)- код Хеммінга має надмірність 7/4 і при цьому дозволяє виправляти одну помилку в кодовому слові з 7

символів (про це детально говоритимемо в наступних розділах цього посібника), то дуальний йому код (7,3) також виправляє одну помилку на 7 символів, але вже має надмірність 7/3, тобто на 3 інформаційних символи містить 4 перевірочних.

1.2.6. Синдром і виявлення помилок

Перш ніж говорити про виявлення і виправлення помилок кодами, що коректують, визначимо саме поняття помилки і методи їх опису.

Нехай $U = (U_0 , U_1 , \dots U_n)$ є кодовим словом, переданим по каналу з перешкодами, а $r = (r_0 , r_1 , \dots r_n)$ - прийнятою послідовністю, можливо, U, що відрізняється від переданого кодового слова . Відмінність r від U полягає в тому, що деякі символи r_i прийнятої послідовності можуть відрізнятися від відповідних символів U_i переданого кодового слова. Наприклад, $U = (0\,0\,0\,1\,0\,0\,0)$, а $r = (0\,0\,0\,0\,0\,0\,0)$, тобто сталася помилка в четвертому символі кодового слова, 1 перейшла в 0 . Або інший приклад: передано кодове слово $U = (0\,0\,1\,1\,1\,1)$, а прийнята послідовність має вигляд $r = (1\,0\,1\,1\,1\,1\,1)$, тобто помилка виникла в першому біті кодового слова, при цьому 0 перейшов в одиницю.

Для опису помилок, що виникають в каналі, використовують вектор помилки, що зазвичай позначається як e і є двійковою послідовністю довжиною n з одиницями в тих позиціях, в яких сталися помилки.

Так, вектор помилки $e = (0\,0\,0\,1\,0\,0\,0)$ означає однократну помилку в четвертій позиції (четвертому біті), вектор помилки $e = (1\,1\,0\,0\,0\,0\,0)$ - подвійну помилку в першому і другому бітах і так далі

Тоді при передачі кодового слова $\boldsymbol{U}$ по каналу з помилками прийнята послідовність $\boldsymbol{r}$ матиме вигляд

$$\boldsymbol{r} = \boldsymbol{U} + \boldsymbol{e}, \qquad (1.21)$$

наприклад:

$$\boldsymbol{U} = (\boldsymbol{0\,0\,0\,1\,0\,0\,0}),$$
$$\boldsymbol{e} = (\boldsymbol{0\,0\,0\,1\,0\,0\,0}), \qquad (1.22)$$
$$\boldsymbol{r} = (\boldsymbol{0\,0\,0\,0\,0\,0\,0}).$$

Прийнявши вектор $\boldsymbol{r}$, декодер спочатку повинен визначити, чи є в прийнятій послідовності помилки. Якщо помилки є, то він повинен виконати дії з їх виправлення.

Для того щоб перевірити, чи є прийнятий вектор кодовим словом, декодер обчислює $\boldsymbol{(n\text{-}к)}$ - послідовність, що має наступний вигляд :

$$\boldsymbol{S} = (\boldsymbol{S_0}, \boldsymbol{S_1}, \ldots, \boldsymbol{S_{n-k-1}}) = \boldsymbol{r} \cdot \underline{\boldsymbol{H}}^{\boldsymbol{T}}. \qquad (1.23)$$

При цьому $\boldsymbol{r}$ є кодовим словом тоді, і лише тоді, коли $\boldsymbol{S = (00..0),}$ і не є кодовим словом даного коду, якщо $\boldsymbol{S \neq \underline{0}}$. Отже, $\boldsymbol{S}$ використовується для виявлення помилок, ненульове значення $\boldsymbol{S}$ служить ознакою наявності помилок в прийнятій послідовності. Тому вектор $\boldsymbol{S}$ називається синдромом прийнятого вектора $\boldsymbol{r}$.

Деякі поєднання помилок, використовуючи синдром, виявити неможливо. Наприклад, якщо передане кодове слово $\boldsymbol{U}$ під впливом перешкод перетворилося на інше дійсне кодове слово $\boldsymbol{V}$ цього ж коду, то синдром $\boldsymbol{S} = \boldsymbol{V} * \underline{\boldsymbol{H}}^{\boldsymbol{T}} = \underline{\boldsymbol{0}}$. В цьому випадку декодер помилки не виявить і, звичайно, не спробує її виправити.

Поєднання помилок такого типу називаються такими, що не виявляються. При побудові кодів необхідно прагнути до того, аби вони виявляли найбільш ймовірні поєднання помилок.

Для лінійного блочного систематичного (7,4) - коду Хеммінга, що розглядається як приклад, синдром визначається таким чином:

нехай прийнятий вектор $\boldsymbol{r} = (\boldsymbol{r_0}, \boldsymbol{r_1}, \boldsymbol{r_2}, \boldsymbol{r_3}, \boldsymbol{r_4}, \boldsymbol{r_5}, \boldsymbol{r_6})$, тоді

$$\boldsymbol{S} = \boldsymbol{r} * \underline{\boldsymbol{H}}_{(7,4)}^{\boldsymbol{T}} = (\boldsymbol{r_0}, \boldsymbol{r_1}, \boldsymbol{r_2}, \boldsymbol{r_3}, \boldsymbol{r_4}, \boldsymbol{r_5}, \boldsymbol{r_6}) * \begin{vmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{vmatrix}^{T} =$$

$$= (\boldsymbol{r_0} + \boldsymbol{r_2} + \boldsymbol{r_3} + \boldsymbol{r_4}), (\boldsymbol{r_0} + \boldsymbol{r_1} + \boldsymbol{r_2} + \boldsymbol{r_5}), (\boldsymbol{r_1} + \boldsymbol{r_2} + \boldsymbol{r_2} + \boldsymbol{r_6}), \quad (1.23)$$

або

$$\begin{aligned} \boldsymbol{S_0} &= \boldsymbol{r_0} + \boldsymbol{r_2} + \boldsymbol{r_3} + \boldsymbol{r_4}, \\ \boldsymbol{S_1} &= \boldsymbol{r_0} + \boldsymbol{r_1} + \boldsymbol{r_2} + \boldsymbol{r_5}, \end{aligned} \quad (1.24)$$

$$S_2 = r_1 + r_2 + r_2 + r_6 .$$

Взявши за основу отримані співвідношення, можна легко організувати схему для обчислення синдрому.

1.2.7. Синдромне декодування лінійних блочних кодів

Покажемо, як можна використовувати синдром прийнятого вектора не лише для виявлення, але і для виправлення помилок.

Нехай $U = (U_0, U_1, \ldots, U_{n-1})$, $e = (e_0, e_1, \ldots, e_{n-1})$ і $r = (r_0, r_1, r_2, \ldots, r_{n-1})$ є кодовим словом, що передається, вектором-помилкою і прийнятим вектором відповідно. Тоді

$$r = U + e \qquad (1.25)$$

і синдром

$$S = r \cdot H^T = (U + e) \cdot H^T = U \cdot H^T + e \cdot H^T = 0 + e \cdot H^T = e \cdot H^T , \qquad (1.26)$$

оскільки для будь-якого кодового слова $U \cdot H^T = 0$.

Таким чином, синдром прийнятої послідовності r залежить лише від помилки, що має місце в цій послідовності, і абсолютно не залежить від переданого кодового слова. Завдання декодера, використовуючи цю залежність,

визначити елементи (координати) вектора помилок. Знайшовши вектор помилки можна відновити кодове слово як

$$U^* = r + e\,. \qquad (1.27)$$

На прикладі одиночних помилок при кодуванні з використанням лінійного блочного (7,4) - коду покажемо, як вектор помилки пов'язаний з синдромом, і як, маючи синдром, локалізувати і усунути помилки, що виникли при передачі.

Знайдемо значення синдрому для всіх можливих одиночних помилок в послідовності з семи символів:

$e\_ = (\,0000000\,)$,

$$e\_ \cdot \underline{H}^T = (\,0000000\,) \cdot \begin{vmatrix} 1011100 \\ 1110010 \\ 0111001 \end{vmatrix}^T = (000);$$

$e_0 = (\,1000000\,)$,

$$e_0 \cdot \underline{H}^T = (\,1000000\,) \cdot \begin{vmatrix} 1011100 \\ 1110010 \\ 0111001 \end{vmatrix}^T = (110);$$

$e_1 = (\,0100000\,)$,

$$e_1 \cdot \underline{H}^T = (\,0100000\,) \cdot \begin{vmatrix} 1011100 \\ 1110010 \\ 0111001 \end{vmatrix}^T = (011);$$

$e_2 = (\ 0010000\ )$,

$$e_2 \cdot \underline{H}^T = (0010000) \cdot \begin{vmatrix} 1011100 \\ 1110010 \\ \\ 0111001 \end{vmatrix}^T = (111);$$

$e_3 = (\ 0001000\ )$,

$$e_3 \cdot \underline{H}^T = (0001000) \cdot \begin{vmatrix} 1011100 \\ 1110010 \\ \\ 0111001 \end{vmatrix}^T = (101$$

$e_4 = (\ 0000100\ )$,

$$e_4 \cdot \underline{H}^T = (0000100) \cdot \begin{vmatrix} 1011100 \\ 1110010 \\ \\ 0111001 \end{vmatrix}^T = (100);$$

$e_5 = (\ 0000010\ )$,

$$e_5 \cdot \underline{H}^T = (0000010) \cdot \begin{vmatrix} 1011100 \\ 1110010 \\ \\ 0111001 \end{vmatrix}^T = (010);$$

$e_6 = (\ 0000001\ )$,

$$e_6 \cdot \underline{H}^T = (\ 0000001\ ) \cdot \begin{vmatrix} 1011100 \\ 1110010 \\ \\ 0111001 \end{vmatrix}^T = (001).$$

Всі можливі для (7,4) - коду одиночні помилки і відповідні їм вектори синдрому наведені в таблиці. 1.3.

Таблиця 1.3

Вектор помилки	Синдром помилки	Десятковий код синдрому
1000000	*110*	6
0100000	*011*	3
0010000	*111*	7
0001000	*101*	5
0000100	*100*	4
0000010	*010*	2
0000001	*001*	1

З цієї таблиці видно, що існує однозначна відповідність між поєднанням помилок (при одиночній помилці) і його синдромом, тобто, знаючи синдром, можна абсолютно однозначно визначити позицію коду, в якій сталася помилка.

Наприклад, якщо синдром, обчислений по прийнятому вектору, рівний (111), це означає, що сталася одиночна помилка в третьому символі, якщо S = (001) – то в останньому, і так далі.

Якщо місце помилки визначене, то усунути її вже досить просто.

А тепер подивимося, що станеться, якщо в прийнятій послідовності буде не одна, а, наприклад, дві помилки. Нехай помилки виникли в другій і шостій позиціях $e = (0100010)$. Відповідний синдром визначиться як

$$S = (0100010) \begin{vmatrix} 1011100 \\ 1110010 \\ 0111001 \end{vmatrix}^T = (001).$$

Проте синдром $S = (001)$ відповідає також і одиночній помилці в сьомій позиції (e_6).Отже, наш декодер не лише не виправить помилок в позиціях, в яких вони сталися, але і внесе помилку до тієї позиції, де її не було. Таким чином (7,4) - код не забезпечує виправлення подвійних помилок, а також помилок більшої кратності.

Це обумовлено не тим, яким чином виробляється декодування, а властивостями самого коду. Декілька пізніше буде показано, від чого залежить виправна здатність коду, тобто скільки помилок він може виправити.

Річард Уеслі Хемінг(Гемінг) (1915- 1998)— геній одної ідеї

Річард Уеслі Хеммінга (11 лютого 1915 Чикаго - 7 січня 1998 Монтеррей) - американський математик, роботи якого в сфері теорії інформації мали істотний вплив на комп'ютерні науки ітелекоммунікації. Основний внесок - т. зв. код Хеммінга, а також відстань Хеммінга.

Спочатку Хеммінг мріяв вивчати інженерну справу, але був час Великої депресії - світова економічна криза, і коштів на навчання в іншому вищому закладі просто не було, а в університеті ім. Річарда Крейна таку дисципліну не

викладали. Вибравши факультет природничих наук, математику, в 1937 році отримав ступінь бакалавра з даної дисципліни. Такий поворот подій зіграв доленосну роль в житті Хеммінга, адже будучи інженером ... "я був би просто хлопцем, який лагодить каналізації, а так моє життя геть пов'язана з напрочуд цікавою науково-дослідною роботою, яка хвилює розум".

Хеммінг народився в Чикаго. Він отримав ступінь бакалавра в університеті Чикаго в 1937 році. Потім він продовжив освіту в Університеті Небраска і в 1939 році отримав там ступінь магістра. У 1942 році він захищає дисертацію в університеті Іллінойс і стає доктором філософії. Деякий час числиться професором в Університеті Луїсвілль, де перериває роботу для участі в Манхеттенський проект в 1945.

Проект Манхеттен »(англ. Manhattan Project) - кодова назва програми США по разработкеядерного зброї, здійснення якої почалося 17 вересня 1943 року. Перед цим дослідження велися в «уранових комітеті» (S-1 Uranium Committee, з 1939 року). У проекті брали участь вчені зі Сполучених Штатів Америки, Великобританії, Німеччини та Канади.

В рамках проекту були створені три атомні бомби: плутонієва «Трініті» (підірвана при першому ядерному випробуванні), урановий «Малюк» (скинута

на Хіросіму 6 серпня 1945 года) і плутонієвий «Товстун» (скинута на Нагасакі 9 серпня 1945 года).

Керували проектом американський фізик Роберт Оппенгеймер і генерал Леслі Гровс.

У серпні 1939 року, знаменитий фізик Лео Силард і Юджин Вігнер склали так званий лист Ейнштейна Рузвельту, яке містило попередження про можливу розробку надзвичайно потужної бомби нового типу. Воно спонукало США до отримання уранової руди і прискоренню досліджень Енріко Фермі і інших в області ланцюгових ядерних реакцій. Лист був підписаний Альбертом Ейнштейном і доставлено президентуФранкліну Рузвельту. Рузвельт призначив Лаймана Бріггса з Національного Бюро Стандартів главою Уранового Комітету для дослідження проблем, піднятих в листі. 21 жовтня 1939 року Бріггс скликав збори, на якому були присутні Силард, Вігнер та Едвард Теллер. У листопаді комітет доповів Рузвельту, що уран забезпечить можливе джерело бомби з руйнівною силою, що значно перевершує будь-що відоме.

В рамках цього проекту Хеммінга займається програмуванням одного з перших електронних цифрових комп'ютерів для розрахунку рішення фізичних рівнянь. Мета програми полягала в тому, щоб з'ясувати, чи не призведе вибух атомної бомби до займання атмосфери. Відповідь виявилася негативною, внаслідок чого було прийнято рішення про її використанні.

Незважаючи на недовгий термін роботи в Лос-Аламосі над проектом, саме тут Річард Хеммінга близько познайомився з комп'ютерними методами

обчислювальної математики, які кардинально вплинули на всю його подальшу кар'єру і життя.

Силами союзників було зроблено три плутонієвих заряду, по 500 мільйонів доларів за кожен. Перший заряд використовувався в експерименті «Трініті» 16 липня 1945 року, який вважається початком атомного століття. Другий заряд був підірваний над Нагасакі, а ось третій заряд, який планувалося скинути на Японію, відвезли в Лос-Аламос для подальших досліджень, де він «убив» самих вчених і зараз відомий як «заряд-демон».

У період з 1946 по 1976 року Хеммінга працював в Bell Labs, де співпрацював з **Клодом Шенноном**.

У середині 1940-х років Річард Хеммінг працював в знаменитих Bell Labs на лічильній машині Bell Model V. Це була електромеханічна машина, що використовує релейні блоки, швидкість яких була дуже низька: один оберт за кілька секунд. Дані вводилися в машину за допомогою перфокарт, і тому в процесі читання часто відбувалися помилки. У робочі дні використовувалися спеціальні коди, щоб виявляти і виправляти знайдені помилки, при цьому оператор дізнавався про помилку за світінням лампочок, виправляв і запускав машину. У вихідні дні, коли не було операторів, при виникненні помилки машина автоматично виходила з програми і запускала іншу. Хеммінг часто

працював у вихідні дні, і все більше і більше дратувався, тому що часто був повинен перезавантажувати свою програму через ненадійність перфокарт. Протягом декількох років він проводив багато часу над побудовою ефективних алгоритмів виправлення помилок. У 1950 році він опублікував спосіб, який на сьогоднішній день відомий як код Хеммінга[1].

23 липня 1976 року він переїхав до Монтеррей і очолив там наукові дослідження в області обчислювальної техніки у Вищому військово-морському училищі.

Помер 7 січня 1998 року в віці 82 років.

У його честь Інститут інженерів з електротехніки та електроніки заснував медаль, якою нагороджуються вчені, які внесли значний вклад в теорію інформації - медаль Річарда Хеммінга.

Алгоритм Хемінга

Коди, в яких можливе автоматичне виправлення помилок, називаються такими, що самокоректуються. Для побудови коди, що самокорректирующегося, розрахованої на виправлення одиночних помилок, одного контрольного розряду недостатньо. Як видно з подальшого, кількість контрольних розрядів k повинно бути вибрано так, щоб задовольняти нерівності $2^k \geq k+m+1$ або $k \geq \log_2(k+m+1)$, де m - кількість основних двійкових розрядів кодового слова. Мінімальні значення k при заданих значеннях m, знайдені відповідно до цієї нерівності, приведені в таблиці № 1

Діапазон m	$\mathbf{k_{min}}$
1	2
2-4	3
5-11	4
12-26	5
27-57	6

Маючи m+k розрядів, код, що самокоректується, можна побудувати таким чином.

Привласнимо кожному з розрядів свій номер - від 1 до m+k; запишемо ці номери в двійковій системі числення. Оскільки $2^k > m + k$, то кожен номер можна представити, вочевидь, k-розрядним двійковим числом.

Передбачимо далі, що всі m+k розрядів коду розбиті на контрольні групи, які частково перекриваються, причому так, що одиниці в двійковому представленні номера розряду вказують на його приналежність до певних контрольних груп. Наприклад: розряд № 5 належить до 1-ої і 3-ої контрольних груп, тому що в двійковому представленні його номери $5_{10} = ...000101_2$ - 1-й і 3-й розряди містять одиниці.

Серед m+k розрядів при цьому є k розрядів, кожен з яких належить лише до однієї контрольної групи:

Розряд № 1: $1_{10} = .000001_2$ належить лише до 1-ої контрольної групи.

Розряд № 2: $2_{10} = .000010_2$ належить лише до 2-ої контрольної групи.

Розряд № 4: $4_{10} = .000100_2$ належить лише до 3-ої контрольної групи.

.

Розряд № 2^{k-1} належить лише до k- ої контрольної групи.

Ці *k* розрядів ми і вважатимемо контрольними. Останні m розрядів, кожен з яких належить, щонайменше, до двох контрольних груп, будуть інформаційними розрядами.

У кожній з до контрольних груп матимемо поодинці контрольному розряду. У кожен з контрольних розрядів помістимо таку цифру (0 або 1), аби загальна кількість одиниць в його контрольній групі була парною.

Наприклад, досить поширений код Хемінга **з m=7 і k=4.**

Хай вихідне слово (кодове слово без контрольних розрядів) - $\mathbf{0110101_2}$**.**

Позначимо $\mathbf{p_i}$ - контрольний розряд №i; а $\mathbf{d_i}$ - інформаційний розряд №i, де i = 1,2,3,4…..

№ розряду:	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011
Розподіл контр. і інфор. р-в	p1	p2	d1	p3	d2	d3	d4	p4	d5	d6	d7
Інформаційне кодове слово :			**0**		**1**	**1**	**0**		**1**	**0**	**1**
p1	**1**		0		1		0		1		1
p2		**0**	0			1	0			0	1
p3				**0**	1	1	0				
p4								**0**	1	0	1
Кодове слово з контрольними розрядами:	**1**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**0**	**1**

Цікаво поглянути, як перекриваються контрольні групи в даному випадку (Рис №1). Перша група контролює розряди № 3,7,5 вихідної коди, друга — 3,7,6. (№ групи = № контрольного розряду). Вочевидь, що вони частково перекриваються.

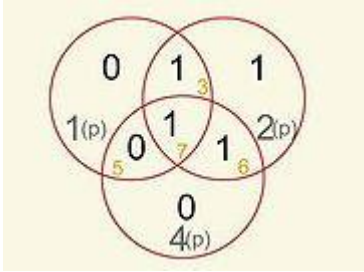


Рис.№1

Передбачимо тепер, для прикладу, що при передачі даного кодового слова 10001100101 сталася помилка в 11–м розряді, так, що було прийнято нове кодове слово 10001100100. Виробивши в прийнятому коді перевірку парності усередині контрольних груп, ми виявили б, що кількість одиниць непарно в 1-й,2-й і 4-й контрольних групах, і парно в 3-ій контрольній групі. Це вказує, по-перше, на наявність помилки, по-друге, означає, що номер помилково прийнятого розряду в двійковому представленні містить одиниці на першому, другому і четвертому місцях і нуль - на третьому місці, так як помилка лише одна, і 3-я контрольна сума виявилася вірною.

№ р-у:	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011		
Розподіл контрольних і інформаційних розрядів	**p1**	**p2**	**d1**	**p3**	**d2**	**d3**	**d4**	**p4**	**d5**	**d6**	**d7**	**Контроль по парності в групі**	**Контрольний біт**
Передане кодове слово:	**1**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**0**	**1**		
Прийняте кодове слово:	**1**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**0**	**0**		
p1 Сума	**1**		0		1		0		1		0	**Fail**	**1**
p2		**0**	0			1	0			0	0	**Fail**	**1**
p3				**0**	1	1	0					**Pass**	**0**
4								**0**	1	0	0	**Fail**	**1**
		p4	p3	p2	p1								
У двійковому представленні		1	0	1	1								
У десятковому представленні		**8**		**2**	**1**	**Σ = 11**							

Pi - по степенях двійки

Канал передачі

Зміна - плюс одиниця !

З таблиці виходить, що помилка сталася в 11-м розряді і її можна виправити. Побудований код, зрозуміло, не розрахований на можливість одночасної помилки в двох розрядах.

Наприклад (Рис №2), коли помилки одночасно прошли в 3-м і 7-м розрядах вихідної коди, перший і другий контрольні біти навіть не помічають підміни.

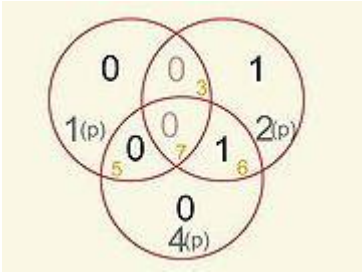


Рис.№2

Коли помилки прошли в 3-м і 7-м розрядах вихідного коду, перший і другий контрольні біти навіть не помічають помилки

Коли в прийнятому коді виробляється перевірка парності усередині контрольних груп, випадок подвійної помилки нічим зовні не відрізняється від випадку одиночної помилки.

1.2.8. Мажоритарне декодування лінійних блочних кодів

Ідею мажоритарного декодування лінійних блочних кодів можна продемонструвати на дуже простому прикладі.

Нехай ***m*** – кодована інформаційна послідовність, що складається всього з одного символу $m_0 = 0$ або 1, а відповідне їй кодове слово перешкодостійкого (надлишкового) коду має вигляд $U = (m_0, m_0, m_0)$, тобто (000), якщо $m_0 = 0$, або (111), якщо $m_0 = 1$ (код з трикратним повторенням).

Припустимо, передано кодове слово $U = (111)$ і в першому символі сталася одна помилка, тобто прийнята послідовність $r = (011)$.

Питання: яка послідовність передавалася, $U = (000)$ *або* $U = (111)$?

Здоровий глузд підказує, що, швидше за все, передавалося кодове слово $U = (111)$, оскільки інакше помилка повинна була б спотворити два символи, щоб кодове слово $U = (000)$ перетворилося на послідовність вигляду $r = (011)$. Можливо, і не віддаючи собі звіту в правилі ухвалення рішення (про те, що передавалося), ми прийняли рішення – мажоритарно.

У загальному випадку, для лінійних блочних кодів із складнішою структурою, рішення буде не таким простим, але ідея мажоритарного декодування – та ж: рішення приймається по більшості. Як і в житті, вважається, що більшість дає правильнішу відповідь.

Розглянемо складніший приклад – мажоритарне декодування для (7,4) -коду Хеммінга.

Хай передано кодове слово (7,4) - коду – $\boldsymbol{U = (U_0, U_1, U_2, U_3, U_4, U_5, U_6)}$, символи якого сформовані відповідно до системи перевірочних рівнянь (правилом кодування) вигляду:

$$\begin{aligned} U_0 &= m_0, \\ U_1 &= m_1, \\ U_2 &= m_2, \\ U_3 &= m_3, \\ U_4 &= m_0 + m_2 + m_3, \\ U_5 &= m_0 + m_1 + m_2, \\ U_6 &= m_1 + m_2 + m_3. \end{aligned} \qquad (1.28)$$

На вході декодера спостерігається прийнята послідовність $\boldsymbol{r = (r_0, r_1, r_2, r_3, r_4, r_5, r_6)}$, і необхідно її декодувати, тобто визначити вигляд інформаційної, що передається послідовності $\boldsymbol{m}$.

Оскільки неможливо бути абсолютно упевненими в правильності декодування, ми можемо говорити лише про оцінку інформаційної послідовності $\boldsymbol{m^*}$.

Спершу передбачимо, що помилок в прийнятій послідовності $\boldsymbol{r}$ немає, тобто $\boldsymbol{r = U}$.

Тоді по прийнятій послідовності $\boldsymbol{r}$ можна легко знайти оцінку переданої інформаційної послідовності $\boldsymbol{m^*}$, причому не єдиним способом.

По-перше, можна відразу записати

$$\begin{aligned} m_0^{*1} &= r_0, \\ m_1^{*1} &= r_1, \\ m_2^{*1} &= r_2, \\ m_3^{*1} &= r_3, \end{aligned} \qquad (1.29)$$

тобто як відповідь або результат декодування, узяти перші чотири символи прийнятої послідовності.

Але це не єдиний спосіб. Враховуючи, що для елементів поля $\boldsymbol{GF(2)}$ справедлива умова $m_i + m_i = 0$ (тобто $1+1 = 0$ і $0+0 = 0$), можна записати ще декілька систем рівнянь для визначення m_i^*:

$$\begin{aligned} m_0^{*2} &= r_2 + r_3 + r_4, \\ m_1^{*2} &= r_0 + r_2 + r_5, \end{aligned} \qquad \begin{aligned} m_0^{*3} &= r_1 + r_2 + r_5, \\ m_1^{*3} &= r_2 + r_3 + r_6, \end{aligned}$$

$$
\begin{aligned}
m_2^{*2} &= r_0 + r_3 + r_4, & m_2^{*3} &= r_0 + r_1 + r_5, \\
m_3^{*2} &= r_0 + r_2 + r_4; & m_3^{*3} &= r_1 + r_2 + r_6;
\end{aligned}
\qquad (1.30)
$$

$$
\begin{aligned}
m_0^{*4} &= r_1 + r_4 + r_6, & m_0^{*5} &= r_3 + r_5 + r_6, \\
m_1^{*4} &= r_0 + r_4 + r_6, & m_1^{*5} &= r_3 + r_4 + r_5, \\
m_2^{*4} &= r_4 + r_5 + r_6, & m_2^{*5} &= r_1 + r_3 + r_6, \\
m_3^{*4} &= r_0 + r_5 + r_6; & m_3^{*5} &= r_1 + r_4 + r_5.
\end{aligned}
$$

Таким чином, вийшло п'ять незалежних систем рівнянь для визначення одних і тих же компонент вектора $\boldsymbol{m^*}$, причому, вони будуть спільними (мати однакові рішення) лише за відсутності помилок в прийнятій послідовності $\boldsymbol{r}$, тобто при $\boldsymbol{r = U}$. Інакше рішення для m_i^*, що отримуються від різних систем, будуть різними.

Проте можна відмітити наступне: у виразах для m_i^* кожен з елементів прийнятої послідовності r_i присутній не більше двох разів (тобто не більше ніж в двох рівняннях з п'яти).

Якщо вважати, що в прийнятій послідовності можлива лише одиночна помилка (а з помилкою більшої кратності цей код не справляється), то помилковими будуть вирішення не більше ніж двох рівнянь з п'яти для кожного з елементів m_i^*, *останні три рівняння дадуть правильне рішення. Тоді правильна відповідь може бути отримана по "більшості голосів", або мажоритарно.*

1.2.10. Вага і відстань Хеммінга. Здатність код виявляти і виправляти помилки

Розглянемо, чим визначається здатність блочного коду виявляти і виправляти помилки, що виникли при передачі.

Нехай $U = (U_0, U_1, U_2, \ldots U_{n-1})$ - двійкова послідовність довжиною n.

Число одиниць (ненульових компонент) в цій послідовності називається вагою Хеммінга вектора ***U*** *і позначається* ***w(U).***

Наприклад, вага Хеммінга вектора ***U = (1001011)*** рівний чотирьом, для вектора ***U = (1111111)*** величина ***w(U)*** складе 7 і так далі.

Таким чином, чим більше одиниць в двійковій послідовності, тим більше її вага Хеммінга.

Далі, нехай ***U*** і ***V*** будуть двійковими послідовностями довжиною ***n***.

Число розрядів, в яких ці послідовності розрізняються, називається відстанню Хеммінга між ***U*** *і* ***V*** *і позначається* ***d(U, V).***

Наприклад, якщо ***U = (1001011)***, а ***V = (0100011),*** то ***d(U, V)= 3.***

Задавши лінійний код, тобто визначивши всі 2^k його кодових слів, можна обчислити відстань між всіма можливими парами кодових слів. Мінімальне з них називається мінімальною кодовою відстанню коду і позначається d_{min}.

Можна перевірити і переконатися, що мінімальна кодова відстань для того, що розглядається нами в прикладах (7,4) -коду рівно трьом: $d_{min(7,4)} = 3$. Для цього потрібно записати всі кодові слова (7,4) -коду Хеммінга (всього 16 слів), обчислити відстані між їх всіма парами і узяти найменше значення. Проте можна визначити d_{min} блочного коду і простішим способом.

Доведено, що відстань між нульовим кодовим словом і одним з кодових слів, що входять в породжуючи матрицю (рядки породжуючої матриці лінійної блочного коду, самі є кодовими словами, за визначенням) рівно d_{min}. Але відстань від будь-якого кодового слова до нульового дорівнює вазі Хеммінга цього слова. Тоді d_{min} рівно мінімальній вазі Хеммінга для всіх рядків породжуючої матриці коду .

Якщо при передачі кодового слова по каналу зв'язку в нім сталася одиночна помилка, то відстань Хеммінга між переданим словом U і прийнятим вектором r дорівнюватиме одиниці. Якщо при цьому одне кодове слово не перейшло в інше (а при $d_{min} > 1$ і при одиночній помилці це неможливо), то помилка буде виявлена при декодуванні.

У загальному випадку якщо блочний код має мінімальну відстань d_{min}, то він може виявляти будь-які поєднання помилок при їх числі, меншому або рівному

$d_{min} - 1$, оскільки жодне поєднання помилок при їх числі, меншому, ніж $d_{min} - 1$, не может перевести одно кодовое слово в друге.

Але помилки можуть мати кратність і більшу, ніж $d_{min} - 1$, і тоді вони залишаться невиявленими.

При цьому середню ймовірність помилки, що не виявляється, можна визначити таким чином.

Хай ймовірність помилки в каналі зв'язку рівна $P_{пом}$. Тоді ймовірність того, що при передачі послідовності довжини n в ній станеться одна помилка, рівна

$$P_1 = n\, P_{пом} \cdot (1 - P_{пом})^{n-1}, \qquad (1.36)$$

відповідно, ймовірність l-кратной помилки -

$$P_l = C_n^l\, P_{пом}^l \cdot (1 - P_{пом})^{n-l}, \qquad (1.37)$$

де C_n^l - число можливих комбінацій з n символів кодової послідовності по l помилок.

По каналу зв'язку передаються кодові слова з різними вагами Хеммінга. Покладемо, що a_i — число слів з вагою і в даному коді (всього слів в коді завдовжки n - $A = \sum_{i=0}^{n-1} d_i = 2^k$).

А тепер визначимо, що таке помилка, що не виявляється. Виявлення помилки виробляється шляхом обчислення синдрому прийнятої послідовності.

Якщо прийнята послідовність не є кодовим словом (тоді синдром не дорівнює нулю), то вважається, що помилка є. Якщо ж синдром дорівнює нулю, то вважаємо, що помилки немає (прийнята послідовність є кодовим словом). Але чи тим, яке передавалося? Або ж в результаті дії помилок передане кодове слово перейшло в інше кодове слово даного коду:

$$\boldsymbol{r = U + e = V,} \qquad (1.38)$$

тобто сума переданого кодового слова $\boldsymbol{U}$ і вектора помилки $\boldsymbol{e}$ дасть нове кодове слово $\boldsymbol{V}$? В цьому випадку, природно, помилка виявлена бути не може.

Але з визначення двійкового лінійного коду виходить, що якщо сума кодового слова і деякого вектора $\boldsymbol{e}$ є кодове слово, то вектор $\boldsymbol{e}$ також є кодовим словом. Отже, помилки, що не виявляються, виникатимуть тоді, коли поєднання помилок утворюватимуть кодові слова.

Ймовірність того, що вектор $\boldsymbol{e}$ збігається з кодовим словом, що має вагу $\boldsymbol{i}$, рівна

$$\boldsymbol{P_i = P_{пом}^{\,i} \cdot (1 - P_{пом})^{n-i}}. \qquad (1.39)$$

Тоді повна ймовірність виникнення помилки, що не виявляється

$$P(E) = \sum_{i=1}^{n} a_i \cdot P_{\text{ім}}^{i} \left(1 - P_{\text{ім}}\right)^{n-i}. \qquad (1.40)$$

Приклад: (7,4) –код, що ми розглядали, містить по сім кодових слів з вагами ***w = 3*** *і* ***w = 4*** *і одне кодове слово з вагою* ***w = 7****, тоді*

$$P(E)_{(7,4)} = 7 \cdot P_{\text{ім}}^{3} \left(1 - P_{\text{ім}}\right)^{4} + 7 \cdot P_{\text{ім}}^{4} \cdot \left(1 - P_{\text{ім}}\right)^{3} + P^{7} \qquad (1.41)$$

або, при $P_{пом} = 10^{-3}$, $P(E) \cong 7 \cdot 10^{-9}$.

Іншими словами, якщо по каналу передається інформація із швидкістю ***V = 1кбіт/с*** і в каналі в середньому кожну секунду відбуватиметься спотворення одного символу, то в середньому сім прийнятих слів на 10^{9} переданих проходитимуть через декодер без виявлення помилки (одна помилка, що не виявляється, за 270 годин).

Таким чином, використання навіть такого простого коду дозволяє на декілька порядків понизити ймовірність помилок, що не виявляються.

Тепер передбачимо, що лінійний блочний код використовується для виправлення помилок. Чим визначаються його можливості по виправленню?

Розглянемо приклад, наведений на рис. 1.9. Нехай ***U*** і ***V*** представляють пару кодових слів коди з кодовою відстанню ***d***, рівним мінімальному — d_{min} *для даного коду.*

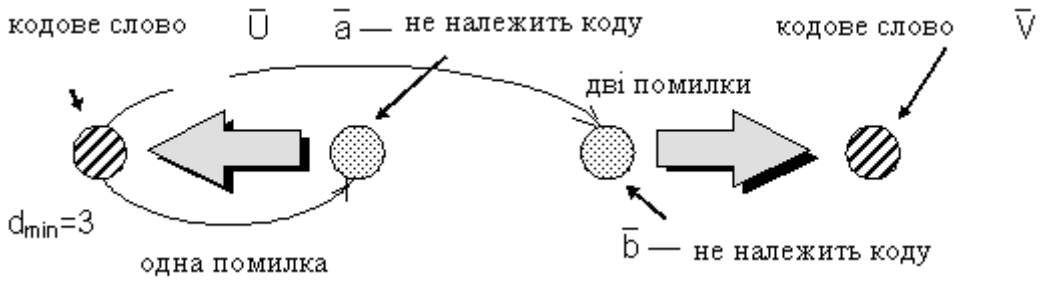


Рис. 1.9

Передбачимо, передано кодове слово ***U***, в каналі сталася одиночна помилка і був прийнятий вектор ***a*** (що не належить коду).

Якщо декодування відбувається оптимальним способом, тобто по методу максимальної правдоподібності, то в якості оцінки ***U**** потрібно вибрати найближче до ***a*** кодове слово.

Таким в даному випадку буде ***U***, отже, помилка буде усунена.

Уявимо тепер, що сталося дві помилки і був прийнятий вектор ***b***.

Тоді при декодуванні по максимуму правдоподібності як оцінка буде вибрано найближче до ***b*** кодове слово, і ним буде ***V***. Станеться помилка декодування.

Продовживши міркування для $d_{min} = 4$, $d_{min} = 5$ і т.д., неважко зробити висновок, що помилки будуть усунені, якщо їх кратність ***l*** не перевищує величини

$$l < INT((dmin - 1)/2), \qquad (1.41)$$

де ***INT (X)*** — ціла частина ***X***.

Так, використовуваний нами як приклад (***7,4***)-код має $d_{min} = 3$ і, отже, дозволяє виправляти лише одиночні помилки:

$$l = INT ((d_{min} - 1)/2) = INT((3-1)/2) = 1 . \qquad (1.42)$$

Таким чином, можливості лінійних блочних кодів по виявленню і виправленню помилок визначаються їх мінімальною кодовою відстанню. Чим більше d_{min}, тим більше число помилок в прийнятій послідовності можна виправити.

А тепер визначимо ймовірність того, що помилка, яка виникла в процесі передачі не буде все ж виправлена при декодуванні.

Нехай, як і раніше, ймовірність помилки в каналі буде рівна $P_{пом}$. Помилки, що виникають в різних позиціях коду, вважаємо незалежними.

Ймовірність того, що прийнятий вектор ***r*** матиме які-небудь (одиночні, двократні, трикратні і так далі) помилки, можна визначити як

$$P_{пом} = P_1 + P_2 + P_3 + ... + P_n , \qquad (1.43)$$

де P_1 — ймовірність того, що в ***r*** присутня одиночна помилка;
P_2 — ймовірність того, що помилка подвійна і т.д.;
P_n — ймовірність того, що всі **n** символів спотворені.

Визначимо ймовірність помилок заданої кратності:

$$P_1 = \textbf{Вер}\{\textit{помилка в 1-ій позиції АБО помилка в 2-ій позиції ..АБО в n-й позиції}\} = =$$

$$P_{пом}(1- P_{пом})^{n-1} + P_{пом}(1- P_{пом})^{n-1} + \ldots P_{пом}(1- P_{пом})^{n-1} = n \cdot P_{пом}(1- P_{пом})^{n-};$$

(1.44)

$$P_2 = \textbf{Вер}\{\textit{помилка в 1-ій I в 2-ій позиції АБО помилка в 2-ій I в 3-ій позиції}\ldots\} =$$

$$= P^2_{пом}(1- P_{пом})^{n-2} + \ldots P^2_{пом}(1- P_{пом})^{n-2} = C^n_2 P^2_{пом}(1- P_{пом})^{n-2}. \qquad (1.45)$$

Аналогічним чином

$$P_3 = C^n_3 P^3_{пом}(1- P_{пом})^{n-3} \quad \text{і т.д.} \qquad (1.46)$$

Декодер, як ми показали, виправляє всі помилки, кратність яких не перевищує

$$l \le INT\left[\frac{d_{min}-1}{2}\right], \qquad (1.47)$$

тобто всі помилки кратності $J \le l$ будуть виправлені.

Тоді помилки декодування - це помилки з кратністю, більшою кратності помилок l, що виправляються, і їх ймовірність

$$P(N) = \sum_{j=l-1}^{n} C_j^n \cdot P_{ім}^j \cdot (1 - P_{ім})^{n-j}. \qquad (1.48)$$

Для (7,4)-коду Хеммінга мінімальна відстань $d_{min} = 3$, $l = 1$. Отже, помилки кратності 2 і більше виправлені не будуть і

$$P(N)_{(7.4)} = \sum_{j=2}^{7} C_j^7 \cdot P_{ім}^j \cdot (1 - P_{ім})^{j-n}. \qquad (1.49)$$

Якщо $P_{ош} << 1$, можна вважати $(1 - P_{ош}) \approx 1$ і, крім того, $P^3_{пом} << P^2_{пом}$. Тоді

$$P(N)_{(7.4)} \approx C_2^7 \cdot P_{ім}^2 \approx 21P^2. \qquad (1.50)$$

Так, наприклад, при ймовірності помилки в каналі $P_{пом} = 10^{-3}$ ймовірність невиправлення помилки $P(N) \approx 2 \cdot 10^{-5}$, тобто при такій ймовірності помилок в каналі кодування (7.4)-кодом дозволяє понизити ймовірність помилок, що залишилися невиправленими, приблизно в п'ятдесят разів.

Якщо ж ймовірність помилки в каналі буде в сто разів менше $P_{пом} = 10^{-5}$, то ймовірність її невиправлення складе вже $P(N) \approx 2 \cdot 10^{-9}$, або *в 5000 раз менше!*

Таким чином, виграш від перешкодостійкого кодування (який можна визначити як відношення числа помилок в каналі до помилок, що залишилися невиправленими) істотно залежить від властивостей каналу зв'язку.

Якщо ймовірність помилок в каналі велика, тобто канал не дуже хороший, чекати великого ефекту від кодування не доводиться, якщо ж ймовірність помилок в каналі мала, то корегуюче кодування зменшує її в значно більшій мірі.

Іншими словами, перешкодостійке кодування істотно покращує властивості хороших каналів, в поганих же каналах воно великого ефекту не дає.