Пам’ять цифрових систем і скінченні автомати

Комп’ютерна логіка 2026–2027 навчальний рік

Мета лекції

Сформувати цілісне розуміння організації пам’яті цифрових систем, правил адресації та обміну даними, а також навчитися описувати керувальну поведінку за допомогою скінченних автоматів.

1. Організація цифрової пам’яті

Пам’ять у цифровій системі

Цифрова система потребує зберігання не лише поточного стану, а й великих масивів даних. Пам’ять дозволяє звернутися до конкретного слова за його номером — адресою (address).

Типова взаємодія містить три групи сигналів: адресні лінії, лінії даних і керувальні лінії. Адреса відповідає на запитання «де», дані — «що», а керування — «яку операцію виконати».

- Процесор і цифрові вузли обробляють дані
- Пам’ять зберігає команди, операнди та результати
- Адреса вибирає конкретне слово
- Керувальні сигнали визначають читання або запис

Основні параметри пам’яті

Позначення N×M означає N слів по M бітів. Наприклад, пам’ять 1024×8 містить 1024 адресовані слова, кожне з яких має вісім бітів.

Місткість у бітах дорівнює N·M. Для 1024×8 це 8192 біти, тобто 1024 байти або 1 KiB. Час доступу показує, скільки потрібно чекати від стабільної адреси й активного читання до появи коректних даних.

- Глибина (depth) — кількість адресованих слів
- Ширина слова (word width) — кількість бітів в одному слові
- Місткість (capacity) — загальна кількість бітів
- Час доступу (access time) — затримка отримання даних

Параметр	Позначення	Зміст
Глибина	N	Кількість адресованих слів
Ширина	M	Кількість бітів у слові
Місткість	N·M	Загальна кількість бітів
Час доступу	tAA	Затримка від адреси до коректних даних

Комірка, слово і адреса

Фізична комірка пам’яті зберігає один біт. Комірки одного рядка часто утворюють слово, яке читається або записується паралельно.

Якщо адресна шина має n ліній, вона може сформувати 2^n різних двійкових кодів. Отже, без додаткового банкування така пам'ять може адресувати не більше 2^n слів.

- Комірка (cell) зберігає один біт
- Слово (word) містить групу бітів, що читаються разом
- Адреса (address) є номером слова
- Розрядність адреси визначає кількість доступних слів

Розрахунок місткості

Приклад потрібно виконувати послідовно. Спочатку переводимо кількість слів у степінь двійки: $1024=2^{10}$. Звідси отримуємо десять адресних входів A9...A0.

Далі множимо кількість слів на ширину одного слова: 1024·8=8192 біти. Оскільки один байт містить вісім бітів, маємо 1024 байти. KiB означає кібібайт і відповідає 1024 байтам.

- $1K = 1024 = 2^{10}$ слів
- Кількість адресних ліній: $\log_2 1024 = 10$
- Місткість: 1024·8 = 8192 біт
- 8192 біт / 8 = 1024 байти = 1 KiB

Крок	Розрахунок	Результат
Кількість слів	$1K=1024=2^{10}$	10 адресних ліній
Місткість у бітах	1024·8	8192 біт
Місткість у байтах	8192/8	1024 байти = 1 KiB

Інтерфейс мікросхеми пам'яті

Адресна шина (address bus) передає номер слова. Шина даних (data bus) переносить значення. У RAM вона зазвичай двонапрямлена: під час читання пам'ять керує шиною, а під час запису — зовнішній пристрій.

Надриска в $\overline{CS}$, $\overline{OE}$ і $\overline{WE}$ означає активний низький рівень (active low). Сигнал виконує свою функцію при нулі. Якщо $\overline{CS}$=1, мікросхема не вибрана й зазвичай не реагує на інші входи.

- A0...An−1 — адресна шина
- D0...Dm−1 — шина даних
- $\overline{CS}$ — вибір мікросхеми
- $\overline{OE}$ — дозвіл виходу
- $\overline{WE}$ — дозвіл запису

Декодування адреси

Декодер адреси (address decoder) реалізує принцип «один із багатьох». Для кожного двійкового коду адреси активується один рядок або одна група комірок.

Наприклад, чотири адресні біти A3...A0 вибирають один із 16 рядків. Код 0101_2 дорівнює 5_{10}, тому активується рядок із номером 5.

- n адресних бітів задають до 2^n комбінацій
- Декодер перетворює код на один активний вихід
- Вибраний рядок підключається до ліній даних
- Інші рядки залишаються неактивними

2. Цикли читання і запису

Цикл читання

Читання (read) не змінює збережений вміст. Зовнішній контролер подає адресу, вибирає мікросхему та дозволяє вихід. Пам’ять потребує певного часу для декодування адреси й поширення сигналу через вихідні каскади.

Дані не можна вважати коректними відразу після зміни адреси. Потрібно витримати час доступу tAA або інший параметр із документації. Після деактивації $\overline{OE}$ виходи часто переходять у високоімпедансний стан Z.

- Встановити стабільну адресу
- Активувати $\overline{CS}$
- Активувати $\overline{OE}$ і залишити $\overline{WE}$ неактивним
- Дочекатися часу доступу
- Зчитати стабільне слово з шини даних

Часова діаграма читання

На часовій діаграмі адреса повинна стабілізуватися до моменту, коли контролер очікує коректні дані. Заштрихована або позначена область означає, що значення на шині ще не гарантується.

Якщо контролер зчитує раніше завершення tAA, він може отримати попереднє слово або проміжний код. Саме тому швидкість пам’яті має відповідати періоду системної шини.

- Адреса змінюється на A
- $\overline{CS}$ та $\overline{OE}$ переходять у нуль
- Протягом tAA дані ще невизначені
- Після tAA на шині з’являється DATA[A]

Цикл запису

Запис (write) змінює вибране слово. Контролер, а не пам’ять, керує шиною даних. Адреса визначає місце запису, а значення D задає новий вміст.

Документація визначає ширину імпульсу запису, час підготовки даних до завершення запису та час утримання після нього. Порушення цих інтервалів може призвести до неправильного слова або відсутності запису.

- Встановити адресу слова
- Подати дані на шину
- Активувати $\overline{CS}$

- Сформувати активний імпульс $\overline{WE}$
- Зберегти адресу й дані протягом потрібних інтервалів

Асинхронна і синхронна пам’ять

Асинхронна пам’ять (asynchronous memory) починає операцію після зміни адреси або сигналів керування. Її часову поведінку описують абсолютні затримки в наносекундах.

Синхронна пам’ять (synchronous memory) працює від тактового сигналу. Адреса й команда фіксуються на активному фронті, а дані з’являються через визначену кількість тактів. Це полегшує інтеграцію у конвеєрні системи.

- Асинхронна пам’ять не потребує Clock
- Затримку визначають адреса та керувальні сигнали
- Синхронна пам’ять приймає команди на фронті Clock
- Результат синхронної операції часто має фіксовану кількість тактів затримки

3. Оперативна пам’ять RAM

Оперативна пам’ять RAM

Оперативна пам’ять із довільним доступом (Random Access Memory, RAM) дозволяє звертатися до будь-якої адреси приблизно за однаковий час. Слово random означає довільний порядок адрес, а не випадковість даних.

RAM використовується для змінних, буферів, стеків і проміжних результатів. Енергозалежність (volatility) означає втрату вмісту після вимкнення живлення.

- RAM — Random Access Memory
- Будь-яке слово доступне без послідовного перегляду попередніх
- Підтримує багаторазове читання та запис
- Зазвичай є енергозалежною пам’яттю

Статична пам’ять SRAM

Статична оперативна пам’ять (Static Random Access Memory, SRAM) зберігає біт у стабільному стані комірки зі зворотним зв’язком. Вона не потребує періодичного оновлення, але залишається енергозалежною.

SRAM швидка, тому її застосовують у кешах процесорів, невеликих буферах і вбудованих блоках пам’яті. Недолік — велика кількість транзисторів на один біт.

- SRAM — Static Random Access Memory
- Комірка використовує перехресно з’єднані транзистори
- Дані зберігаються, доки подається живлення
- Висока швидкість, більша площа та вища ціна за біт

Динамічна пам’ять DRAM

Динамічна оперативна пам’ять (Dynamic Random Access Memory, DRAM) використовує конденсатор як елемент зберігання. Через витік заряду інформацію потрібно регулярно відновлювати — виконувати регенерацію (refresh).

Проста комірка дає велику щільність, тому DRAM використовують як основну пам’ять комп’ютера. Читання й керування складніші, ніж у SRAM, а затримка вища.

- DRAM — Dynamic Random Access Memory
- Біт подається зарядом конденсатора
- Заряд поступово витікає
- Контролер періодично виконує refresh
- Висока щільність і нижча ціна за біт

Порівняння SRAM і DRAM

SRAM і DRAM вирішують різні задачі. SRAM забезпечує низьку затримку, але займає більше площі кристала. DRAM дозволяє реалізувати значно більшу місткість за прийнятною ціною.

У типовій системі ці технології доповнюють одна одну: SRAM утворює кеші та локальні буфери, DRAM — основну оперативну пам’ять.

- SRAM: швидкість, проста логіка доступу, велика комірка
- DRAM: висока щільність, refresh, складніший контролер
- Обидві технології енергозалежні
- Вибір визначається місткістю, швидкістю, енергією та вартістю

Ознака	SRAM	DRAM
Комірка	Транзисторна структура зі зворотним зв’язком	Конденсатор і транзистор
Refresh	Не потрібний	Потрібний
Швидкість	Вища	Нижча
Щільність	Нижча	Вища
Типове застосування	Кеші, буфери	Основна пам’ять

4. Постійна пам’ять ROM

Постійна пам’ять ROM

Постійна пам’ять (Read-Only Memory, ROM) належить до енергонезалежної пам’яті (non-volatile memory). Її вміст зберігається без живлення.

Назва read-only історично підкреслює основний режим використання. Сучасні різновиди допускають перепрограмування, але запис виконується повільніше, рідше або спеціальним способом.

- ROM — Read-Only Memory
- Вміст зберігається після вимкнення живлення
- У нормальному режимі система переважно читає дані
- Застосування: таблиці, мікрокод, початкове завантаження

Різновиди постійної пам’яті

Маскова ROM (Mask ROM) отримує вміст на виробництві. Програмована ROM (Programmable ROM, PROM) допускає одноразове програмування. Стирання EPROM виконується ультрафіолетовим світлом.

Електрично стирана програмована ROM (Electrically Erasable Programmable ROM, EEPROM) дозволяє змінювати дані електрично. Flash-пам’ять є різновидом EEPROM із блоковим стиранням, що підвищує щільність і швидкість масових операцій.

- Mask ROM програмується під час виробництва
- PROM програмується користувачем один раз
- EPROM стирається ультрафіолетом
- EEPROM стирається електрично
- Flash стирає дані блоками

Тип	Програмування	Стирання
Mask ROM	На виробництві	Не передбачене
PROM	Один раз	Не передбачене
EPROM	Електричне програмування	Ультрафіолет
EEPROM	Електричне	Електричне, побайтово або сторінками
Flash	Електричне	Електричне, блоками

RAM і ROM

Основна відмінність полягає не лише в можливості запису, а в типовому сценарії. RAM призначена для частих оперативних змін. ROM та її перепрограмовані різновиди зберігають прошивку, конфігурацію або постійні таблиці.

Flash не варто трактувати як повну заміну RAM: запис і стирання повільніші, стирання відбувається блоками, а кількість циклів обмежена.

- RAM швидко читає та записує робочі дані
- ROM зберігає дані без живлення
- RAM зазвичай має практично необмежену кількість робочих записів

- Flash має обмежений ресурс стирання та запису

Ознака	RAM	ROM / Flash
Основне призначення	Робочі дані	Постійні дані та прошивка
Збереження без живлення	Зазвичай ні	Так
Запис	Частий і швидкий	Рідший, за спеціальною процедурою
Ресурс запису	Практично не обмежує нормальну роботу	Для Flash обмежений

5. Керування, розширення та карта пам’яті

Активні низьким рівнем сигнали

Надриска є частиною назви сигналу. Вона вказує, що логічний нуль означає активну команду. Наприклад, $\overline{WE}$=0 запускає запис, а $\overline{WE}$=1 забороняє його.

Високоімпедансний стан Z дозволяє кільком мікросхемам підключатися до спільної шини. У кожний момент керувати лініями повинна лише одна мікросхема, інакше виникає конфлікт виходів.

- $\overline{CS}$=0 вибирає мікросхему
- $\overline{OE}$=0 дозволяє вихідні драйвери
- $\overline{WE}$=0 дозволяє запис
- Неактивний вихід часто переходить у стан Z

Збільшення ширини слова

Щоб збільшити ширину слова, кілька мікросхем працюють одночасно з однаковою адресою. Перша передає молодші біти, друга — старші.

Дві мікросхеми 1024×8 дають 1024 слова по 16 бітів. Кількість адрес залишається 1024, тому адресна шина все ще має десять ліній. Загальна місткість подвоюється до 16 Kibit.

- Адресні входи мікросхем з’єднуються разом
- Керувальні входи отримують спільні сигнали
- Кожна мікросхема формує свою частину шини даних
- Дві пам’яті 1K×8 утворюють 1K×16

Збільшення кількості слів

Для збільшення глибини мікросхеми утворюють банки. Лінії A9...A0 вибирають слово всередині кожної 1K×8 пам’яті, а додаткова лінія A10 визначає, який банк активний.

У результаті доступно 2048 адрес, тому потрібні одинадцять адресних ліній. Ширина слова залишається вісім бітів.

- Молодші адресні лінії подаються на всі мікросхеми
- Старший адресний біт керує декодером вибору банку
- У кожний момент активний лише один $\overline{CS}$
- Дві пам’яті 1K×8 утворюють 2K×8

Карта пам’яті

Карта пам’яті (memory map) показує, який пристрій відповідає на кожний діапазон адрес. У системі можуть бути RAM, ROM і периферійні регістри, підключені до однієї адресної шини.

Якщо діапазони перекриваються, два пристрої можуть одночасно активувати виходи. Якщо між діапазонами є невикористані адреси, звернення до них не повинно випадково вибирати будь-який пристрій.

- Кожний пристрій отримує неперекривний діапазон адрес
- Старші біти адреси формують Chip Select
- Молодші біти вибирають внутрішню комірку
- Карта спрощує аналіз звернень системи

Приклад пам’яті 16×8

Пам’ять 16×8 має 16 слів, тому $\log_2 16=4$ адресні входи. Кожне слово має вісім бітів, отже шина даних D7…D0 є восьмирозрядною.

Для адреси 0101_2 декодер вибирає рядок 5. Якщо виконується читання, на шині з’являється вміст цього рядка. Якщо виконується запис, значення із шини замінює попередній вміст рядка.

- Адреси: 0000_2…1111_2
- A3A2A1A0=0101_2 вибирає слово 5
- Під час читання D7…D0 отримують Memory[5]
- Під час запису D7…D0 замінюють Memory[5]

A3A2A1A0	Десяткова адреса	Вибране слово
0000	0	Memory[0]
0101	5	Memory[5]
1010	10	Memory[10]
1111	15	Memory[15]

Типові помилки під час роботи з пам’яттю

Під час розрахунків потрібно явно записувати одиниці. 8 Kibit і 1 KiB описують однакову місткість, але різні одиниці. Позначення 1K у контексті глибини цифрової пам’яті часто означає 1024.

Друга група помилок виникає в керуванні. Активний нуль читається навпаки, а двонапрямлена шина потребує дисципліни: під час читання її веде пам’ять, під час запису — контролер.

- Плутання бітів і байтів
- Хибне трактування 1K як 1000 у двійковій організації
- Ігнорування надриски в $\overline{CS}$, $\overline{OE}$, $\overline{WE}$
- Одночасне керування шиною двома пристроями
- Зчитування даних до завершення часу доступу

6. Основи скінченних автоматів

Скінченний автомат FSM

Скінченний автомат (Finite State Machine, FSM) — математична й апаратна модель системи, яка в кожний момент перебуває в одному зі скінченної кількості станів.

Стан стискає важливу історію роботи. Наприклад, контролеру дверного замка не потрібно пам'ятати всі натискання від початку часу, але потрібно знати, скільки правильних символів коду вже введено.

- FSM — Finite State Machine
- Поточний стан зберігає історію, потрібну для наступного рішення
- Входи впливають на переходи
- Виходи залежать від стану або від стану та входів

Структура FSM

Структурно FSM має регістр стану, логіку наступного стану та логіку виходів. Поточний стан Q і входи X надходять у комбінаційну логіку, яка обчислює Q^+.

На активному фронті Clock регістр приймає Q^+. Reset переводить систему у визначений початковий стан. Такий поділ важливий для подальшого опису автоматів у SystemVerilog.

- Регістр стану зберігає Q
- Логіка наступного стану обчислює $Q^+=F(Q,X)$
- Логіка виходів формує Y
- Clock переносить Q^+ у регістр
- Reset задає початковий стан

Автомат Moore

В автоматі Moore (Moore machine) вихід залежить тільки від поточного стану. Якщо входи змінюються між тактами, вихід не змінюється, доки автомат не перейде в інший стан.

Такий підхід часто дає стабільні синхронні виходи й спрощує аналіз. Недоліком може бути додатковий стан або додатковий такт реакції.

- Наступний стан: $Q^+=F(Q,X)$
- Вихід: $Y=G(Q)$
- Вихід зазвичай змінюється після зміни стану
- Виходи зручно вказувати всередині станів

Автомат Mealy

В автоматі Mealy (Mealy machine) вихід залежить від поточного стану та входів. Тому зміна входу може змінити вихід ще до наступного фронту Clock.

Mealy FSM іноді потребує менше станів і швидше реагує, але комбінаційний шлях від входу до виходу може передавати короткі імпульси. У критичних системах вихід часто додатково реєструють.

- Наступний стан: $Q^{+}=F(Q,X)$
- Вихід: $Y=G(Q,X)$
- Вихід може реагувати в межах поточного такту
- Виходи позначають на переходах

Moore і Mealy

Вибір моделі залежить від вимог до реакції та стабільності виходу. Moore природно підходить для сигналів, які повинні змінюватися лише на тактових межах.

Mealy зручний, коли реакція на вхід має з'явитися в тому самому такті. Однак потрібно перевірити, чи асинхронні зміни входу не створять небажані імпульси.

- Moore: вихід стабільний між змінами стану
- Mealy: можлива швидша реакція
- Moore може потребувати більше станів
- Mealy вимагає уважного аналізу комбінаційного шляху

Ознака	Moore	Mealy
Вихід	Y=G(Q)	Y=G(Q,X)
Реакція	Після зміни стану	Може виникнути в межах такту
Позначення виходу	Усередині стану	На переході
Ризик коротких імпульсів	Нижчий	Вищий для нерегістрового виходу

Позначення діаграми станів

Діаграма станів (state diagram) є графічним описом поведінки. Кожна вершина — стан, а спрямоване ребро — можливий перехід.

Умови мають бути повними й несуперечливими. Для кожного стану та кожної допустимої комбінації входів повинен існувати однозначний наступний стан.

- Стан подається колом або заокругленим блоком
- Початкова стрілка вказує стан після Reset
- Петля означає збереження стану
- Moore-вихід записують у стані
- Mealy-вихід записують на переході як умова/вихід

Таблиця переходів

Таблиця переходів (state transition table) є формальним аналогом діаграми. Вона особливо корисна перед синтезом рівнянь або написанням коду.

Таблиця повинна охоплювати всі комбінації, які реально можуть виникнути. Якщо певні умови об'єднуються символом «будь-яке значення», це потрібно явно позначити.

- Один рядок описує конкретний стан та умову входу
- Q — поточний стан
- X — входи
- Q^+ — наступний стан
- Y — виходи

7. Проєктування та реалізація FSM

Алгоритм проєктування FSM

Не варто починати з логічних рівнянь. Спочатку потрібно зрозуміти поведінку: що система повинна пам'ятати, які події змінюють режим і які виходи мають бути активними.

Після побудови діаграми її переносять у таблицю, кодують стани й синтезують логіку. Останній етап — перевірка нормальних переходів, Reset і виходу з невикористаних кодів.

- Сформулювати входи, виходи та поведінку
- Виділити мінімально необхідні стани
- Побудувати діаграму і таблицю переходів
- Вибрати кодування станів
- Реалізувати регістр, наступний стан і виходи
- Перевірити Reset та невикористані коди

Приклад FSM світлофора

Розглянемо спрощений синхронний світлофор із трьома станами. Вхід T (timeout) стає одиницею, коли таймер завершив інтервал поточного кольору.

Якщо T=0, автомат залишається в поточному стані. Якщо T=1, переходить S0→S1, S1→S2, S2→S0. Вихід залежить лише від стану, тому це модель Moore.

- S0: червоний сигнал
- S1: зелений сигнал
- S2: жовтий сигнал
- Вхід T означає завершення інтервалу
- За T=0 поточний стан зберігається

Діаграма станів світлофора

Початкова стрілка веде в S0, тому після Reset горить червоний. У кожному стані є два випадки: T=0 дає петлю, T=1 переводить у наступний стан.

Вихід кольору записується всередині стану. Він змінюється після того, як регістр стану прийме новий код на фронті Clock.

- Reset переводить автомат у S0
- S0 формує RED=1
- S1 формує GREEN=1
- S2 формує YELLOW=1
- Переходи відбуваються на активному фронті Clock

Таблиця переходів світлофора

У таблиці для кожного стану потрібно розглянути T=0 і T=1. Виходи RED, GREEN і YELLOW залежать тільки від поточного стану.

Завдяки повній таблиці можна перевірити відсутність суперечностей і безпосередньо перейти до логічного або програмного опису автомата.

- S0, T=0: S0
- S0, T=1: S1
- S1, T=0: S1
- S1, T=1: S2
- S2, T=0: S2
- S2, T=1: S0

Поточний стан	T	Наступний стан	RED	GREEN	YELLOW
S0	0	S0	1	0	0
S0	1	S1	1	0	0
S1	0	S1	0	1	0
S1	1	S2	0	1	0
S2	0	S2	0	0	1
S2	1	S0	0	0	1

Кодування станів

Для N станів двійкове кодування потребує ceil($\log_2 N$) тригерів. Три стани можна закодувати двома бітами: 00, 01 і 10.

Одногаряче кодування (one-hot encoding) використовує три біти: 001, 010, 100. Воно потребує більше тригерів, але часто спрощує логіку переходів у FPGA.

- Двійкове кодування: S0=00, S1=01, S2=10
- Код 11 залишається невикористаним
- One-hot кодування використовує окремий біт на стан

- Вибір кодування впливає на кількість тригерів і складність логіки

Кодування	S0	S1	S2	Кількість тригерів
Двійкове	00	01	10	2
One-hot	001	010	100	3

Невикористані коди стану

Невикористаний код не можна просто ігнорувати. У фізичній системі помилка, неповний Reset або одиничний збій може встановити регістр у такий стан.

Самовідновний автомат (self-recovering FSM) повертається до безпечного стану. Для світлофора логічно обрати S0 із червоним сигналом.

- Для кодування 00, 01, 10 код 11 не належить нормальному циклу
- Перешкода або помилка може перевести регістр у 11
- Безпечний варіант задає 11→S0
- Виходи для незаконного стану мають бути безпечними

Структурна реалізація FSM

Апаратна структура чітко розділяє пам'ять і комбінаційну логіку. Регістр стану складається з D-тригерів. На їхні входи D надходить код Q^+.

У Moore FSM вихідний блок читає лише Q. У Mealy FSM він також отримує X. Ця структурна відмінність пояснює різну часову поведінку моделей.

- Блок F(Q,X) формує наступний стан
- D-тригери записують Q^+ на фронті Clock
- Блок G формує виходи
- У Moore G отримує Q
- У Mealy G отримує Q та X

8. Приклади й часовий аналіз FSM

Покроковий перехід світлофора

Нехай після Reset Q=S0. Поки T=0, логіка формує Q^+=S0, тому кожний фронт зберігає стан.

Коли T стає одиницею до активного фронту й виконує setup time, логіка формує Q^+=S1. На фронті регістр оновлюється, а після затримки clock-to-Q і вихідної логіки активується GREEN.

- Початково Reset встановлює S0
- T=0: S0 зберігається
- Перед фронтом T стає 1
- На фронті регістр приймає S1
- GREEN активується після затримки регістра та вихідної логіки

Приклад Mealy: детектор 01

Автомат повинен виявити послідовність 01 у потоці бітів. Стан S1 означає, що попередній прийнятий біт був нулем.

Коли в S1 надходить X=1, потрібна послідовність завершена, тому Mealy-вихід Y одразу стає одиницею на цьому переході. Для Moore-реалізації знадобився б окремий стан виявлення.

- S0: значущого попереднього нуля немає
- S1: попередній біт дорівнював 0
- У S1 при X=1 формується Y=1
- Після виявлення автомат повертається до S0

Reset і початковий стан FSM

FSM не може коректно виконувати алгоритм, якщо початковий стан невідомий. Reset встановлює визначений код регістра стану.

Для систем керування початковий стан обирають за принципом безпеки. Асинхронний Reset швидко активується, але його деактивацію біля фронту Clock потрібно організовувати обережно.

- Reset задає відомий код Q
- Початковий стан має бути безпечним
- Пріоритет Reset визначає специфікація
- Зняття асинхронного Reset потрібно синхронізувати

Повнота умов і пріоритети

Неповний опис залишає ситуації без визначеного наступного стану. У програмному описі це може створити небажану засувку, а в таблиці — приховану неоднозначність.

Корисна практика: спочатку задати безпечні значення за замовчуванням, а потім перевизначати їх у конкретних умовах. Якщо кілька умов можуть виконуватися одночасно, пріоритет має бути явним.

- Для кожного стану перевіряються всі значення входів
- Умови переходів не повинні суперечити одна одній
- За потреби задається явний пріоритет
- Типове безпечне значення Q^+ дорівнює Q або початковому стану

Часові обмеження FSM

Критичний шлях FSM проходить від виходу регістра стану через логіку наступного стану до входу того самого або іншого тригера.

Умова періоду має вигляд $T_{CLK} \geq t_{CQ} + t_{logic} + t_{setup} +$ запас. Mealy-вихід може мати додатковий комбінаційний шлях безпосередньо від входу, тому його стабільність також потрібно аналізувати.

- Q змінюється після затримки clock-to-Q
- Комбінаційна логіка обчислює Q^+
- Q^+ має стабілізуватися до setup time наступного фронту
- Період Clock має покривати весь критичний шлях

Типові помилки під час проєктування FSM

Надлишкові стани ускладнюють схему, але об’єднання станів із різною майбутньою поведінкою робить автомат неправильним. Кожний стан повинен мати чітке смислове призначення.

Після побудови таблиці потрібно перевірити всі рядки, виходи та переходи. Окремо тестують Reset, невикористані коди й граничні моменти зміни входів.

- Надмірна або недостатня кількість станів
- Відсутній перехід для частини вхідних комбінацій
- Змішування Moore- і Mealy-виходів
- Ігнорування невикористаних кодів
- Відсутність визначеного початкового стану

9. Підсумок теорії

Для пам’яті головними є організація N×M, кількість адресних ліній, напрямок шини даних і часові правила операцій. Технології SRAM, DRAM і ROM-пам’яті мають різні властивості та сфери застосування.

FSM формалізує керування через стани. Повний процес проєктування включає специфікацію, діаграму, таблицю переходів, кодування, структурну реалізацію та перевірку.

10. Узагальнений приклад

Пам’ять 2K×16. Глибина $2048=2^{11}$, тому потрібні 11 адресних ліній. Ширина шини даних становить 16 бітів. Загальна місткість $2048 \cdot 16=32768$ бітів=4096 байтів=4 КіБ.

Світлофор Moore. Після Reset автомат перебуває у S0. За T=0 стан зберігається. Послідовні одиниці T переводять автомат S0→S1→S2→S0. Вихід кольору залежить лише від стану.

11. Терміни та скорочення

Термін	Англійська назва	Зміст
Оперативна пам’ять	Random Access Memory (RAM)	Пам’ять для частого читання і запису
Статична RAM	Static RAM (SRAM)	Швидка пам’ять без refresh
Динамічна RAM	Dynamic RAM (DRAM)	Щільна пам’ять із refresh
Постійна пам’ять	Read-Only Memory (ROM)	Енергонезалежне зберігання
Скінченний автомат	Finite State Machine (FSM)	Модель керування зі станами
Вибір мікросхеми	Chip Select ($\overline{CS}$)	Активує конкретну пам’ять
Дозвіл виходу	Output Enable ($\overline{OE}$)	Дозволяє драйвери даних
Дозвіл запису	Write Enable ($\overline{WE}$)	Запускає запис

Термін	Англійська назва	Зміст
Високоімпедансний стан	High-impedance state (Z)	Від’єднання виходу від шини
Одногаряче кодування	One-hot encoding	Один активний біт на стан

12. Питання для самоперевірки

- Що означає організація пам’яті 1K×8 і скільки адресних ліній вона потребує?
- Чим відрізняються цикли читання і запису?
- Навіщо потрібен високоімпедансний стан Z?
- Чим SRAM відрізняється від DRAM?
- Як збільшити ширину слова та кількість слів?
- З яких блоків складається FSM?
- Чим Moore FSM відрізняється від Mealy FSM?
- Навіщо визначати перехід із невикористаного коду стану?
- Як визначається критичний шлях FSM?