14

Лекция 8–9 декабря

Схемы и верификация

Константин Чухарев

Логические схемы

> *Что я не могу создать, я не понимаю.*
>
> ***— Ричард Фейнман***

Вентили

Вентиль — физический исполнитель булевой операции.

Вентиль	Функция	Смысл
NOT	$\overline{x}$	инвертирует
AND	$x \wedge y$	оба входа — 1
OR	$x \vee y$	хотя бы один — 1
XOR	$x \oplus y$	входы различны
NAND	$\overline{x \wedge y}$	отрицание AND

Через один штрих Шеффера $\uparrow$ выражается всё: $\neg x = x \uparrow x$, а $\wedge$ и $\vee$ — по де Моргану.

Любая схема собирается из вентилей одного типа.

Схема

Определение 1 (Логическая схема)

Схема — ориентированный ациклический граф: истоки — входы, внутренние узлы — вентили, стоки — выходы.

Выходы одних вентилей подаются на входы других — схема вычисляет композицию булевых функций.

Ацикличность — комбинационная логика: результат зависит только от текущих входов.

Цикл в схеме сделал бы значение неопределённым: выход ждал бы сам себя.

Размер и глубина

Определение 2 (Размер и глубина)

Размер схемы — число вентилей: площадь, стоимость.

Глубина — длина самого длинного пути от входа к выходу: задержка.

Размер и глубина — базовый инженерный компромисс.

Схема с малой глубиной собирается из большего числа вентилей, а компактная работает медленнее.

Пример: схема для XOR

Пример

$x \oplus y = (x \vee y) \wedge \overline{x \wedge y}$.

Вентили: OR, AND, NOT, AND — четыре, размер 4.

Самый длинный путь: $x, y \to$ AND $\to$ NOT $\to$ AND — глубина 3.

Задержку задаёт критический путь — не сумма всех вентилей.

Параллельные ветви считаются одновременно.

Семейства схем

Одна схема имеет фиксированное число входов: схема для 4-битного сложения не сложит 8-битные числа.

Определение 3 (Семейство схем)

Семейство схем — последовательность $C_1, C_2, \ldots$, где C_n имеет n входов.

Семейство **равномерно**, если существует алгоритм, строящий C_n по n.

Чётность: C_n — цепочка из $n - 1$ XOR-вентилей.

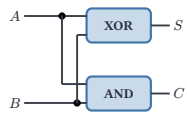
Равномерные полиномиальные семейства — схемный взгляд на эффективные вычисления.

Полусумматор

Определение 4 (Полусумматор (*half adder*))

Складывает два бита A и B:

$$S = A \oplus B, \quad C = A \wedge B.$$



Полусумматор: таблица

A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

«Полу-» — честно: столбик при сложении складывает три бита, а эта схема — только два.

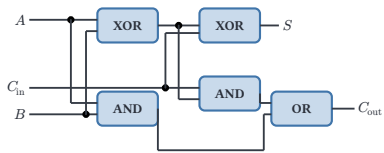
Входящего переноса у полусумматора нет.

Полный сумматор

Определение 5 (Полный сумматор (*full adder*))

Складывает три бита — A, B и входящий перенос C_{in}:

$$S = A \oplus B \oplus C_{\text{in}}, \quad C_{\text{out}} = (A \wedge B) \vee (C_{\text{in}} \wedge (A \oplus B)).$$



Перенос — функция большинства

Перенос $C_{\text{out}} = 1$ тогда и только тогда, когда хотя бы два из трёх входов равны 1.

Перенос — это уже знакомая функция большинства.

Формулы полусумматора и полного сумматора — минимизированные ДНФ.

Минимизация из прошлой лекции — это то, из чего собирается арифметика процессора.

Каскад сумматоров

Определение 6 (Ripple-carry)

Цепочка из n полных сумматоров: перенос каждого разряда подаётся на вход следующего.

Размер $O(n)$, глубина $O(n)$.

Пример

$0111 + 0001$: перенос идёт по всем четырём разрядам.

Результат 1000 — каждый разряд ждал соседа.

Глубина каскада

Глубина каскада ограничивает тактовую частоту: такт не короче критического пути.

Опережающий перенос (*carry-lookahead*) считает переносы группами и снижает глубину до $O(\log n)$ — ценой роста размера.

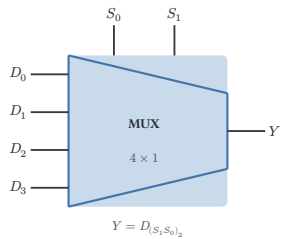
Вычитание тоже каскад: $A - B = A + \overline{B} + 1$ в дополнительном коде.

Вся арифметика процессора — вариации одного сумматора.

АЛУ

Арифметико-логическое устройство — вычислительное ядро процессора.

Однобитный срез АЛУ: параллельные блоки AND, OR, сложения, вычитания — и мультиплексор, выбирающий результат.



Мультиплексор «из 2^k в 1» выдаёт один из 2^k входов по k адресным битам.

n-битное АЛУ — n таких срезов с цепочкой переносов.

Код Грея

Определение 7 (Код Грея (*Gray code*))

Упорядочение всех 2^n двоичных строк длины n, где соседние строки — включая последнюю и первую — отличаются ровно в одном бите.

Пример

G_3: $000, 001, 011, 010, 110, 111, 101, 100$.

Каждая соседняя пара — один бит разницы.

Код цикличен: последняя и первая строки тоже соседи.

Зачем код Грея?

При переключении нескольких битов сразу возникают ложные промежуточные состояния.

Код Грея меняет один бит за шаг — энкодеры угла и позиции читаются безошибочно.

Код Грея уже работал в карте Карно: соседние клетки — соседние строки.

Построение кода Грея

Теорема 1 (Рекурсивное построение)

$G_1 = [0, 1]$.

$$G_{n+1} = 0G_n, 1G_n^R,$$

где $0G_n$ — слова G_n с приписанным слева нулём, а $1G_n^R$ — слова G_n в обратном порядке с приписанной слева единицей.

Пример

$G_2 = [00, 01, 11, 10]$.

Стык половин отличается только первым битом — обращение второй половины и даёт цикличность.

Преобразование в двоичный

Теорема 2 (Двоичный и код Грея)

В код Грея: $g_i = b_i \oplus b_{i+1}$, старший бит сохраняется.

Обратно: $b_i = g_i \oplus b_{i+1}$ — накопительный $\oplus$ от старшего бита.

Пример

$b = 1011$: $g = 1110$ — старший бит тот же, дальше $\oplus$ соседей.

Оба преобразования — цепочка XOR-вентилей глубины $O(n)$.

Верификация через SAT

> *Тестирование программы может убедительно продемонстрировать наличие ошибок, но безнадёжно несостоятельно для демонстрации их отсутствия.*
>
> **— Эдсгер Дейкстра**

Задача верификации

Схема спроектирована, затем оптимизирована — и нужно доказать, что поведение не изменилось.

Перебор всех входов невозможен: у n входов 2^n наборов.

Обе схемы переводят в КНФ: на каждый вентиль заводится переменная его выхода и дизъюнкты, задающие таблицу вентиля (преобразование Цейтина).

Размер растёт линейно по числу вентилей.

Митер

Теорема 3 (Эквивалентность через SAT)

Схемы C_1 и C_2 эквивалентны тогда и только тогда, когда их **митер** (*miter*) невыполним.

Митер строится так:

- входы C_1 и C_2 соединяются попарно.
- на каждой паре выходов ставится XOR.
- все выходы XOR объединяются OR.

Митер выдаёт 1 ровно на тех входах, где схемы различаются.

Митер в действии

Пример

Сравним $f_1(x, y) = x \vee y$ и $f_2(x, y) = x \oplus y$.

На входе $(1, 1)$: $f_1 = 1$, $f_2 = 0$ — XOR выходов даёт 1, митер срабатывает.

На входе $(1, 0)$ обе функции выдают 1 — расхождения нет.

SAT — решатель предъявляет контрпример: вход, разводящий схемы.

UNSAT — доказательство эквивалентности на всех входах сразу.

Единичное распространение

Определение 8 (Unit propagation)

Если в дизъюнкте все литералы, кроме одного, ложны, оставшийся обязан быть истинным.

Пример (Каскад без ветвлений)

$(x) \wedge (\overline{x} \vee y) \wedge (\overline{y} \vee z) \wedge \overline{z}$.

Единичный дизъюнкт (x) даёт $x = 1$ — дизъюнкт $(\overline{x} \vee y)$ становится единичным: $y = 1$.

Затем $z = 1$ — и последний дизъюнкт ложен.

Формула невыполнима, противоречие найдено без единого ветвления.

CDCL

Современные решатели — CDCL (*Conflict-Driven Clause Learning*): DPLL плюс обучение на конфликтах.

- решение: эвристика VSIDS выбирает переменную — приоритет у частых участников недавних конфликтов.
- анализ конфликта: по истории присваиваний извлекается выученный дизъюнкт.
- нехронологический возврат и периодические перезапуски.

DPLL при конфликте просто отступает.

CDCL запоминает его причину — и не ходит туда второй раз.

Выученный дизъюнкт

Пример (Конфликт и резолюция)

Пусть решения дали $x_4 = 1$ и $x_5 = 1$.

Дизъюнкты $(\overline{x_4} \vee \overline{x_5} \vee x_6)$ и $(\overline{x_4} \vee \overline{x_5} \vee \overline{x_6})$ ложны независимо от x_6 — конфликт.

Резолюция двух дизъюнктов по x_6 даёт **выученный дизъюнкт**:

$$\overline{x_4} \vee \overline{x_5}.$$

Комбинация $x_4 = x_5 = 1$ запрещена до конца поиска.

Выученные дизъюнкты — шаги резолюции.

Каждый отсекает целое подпространство перебора, а не одну ветвь.

Сертификат невыполнимости

Ответ UNSAT тоже предъявляет доказательство: выученные дизъюнкты — шаги резолюции.

Решатель сохраняет цепочку резолюций, приведшую к пустому дизъюнкту.

Независимая проверяющая программа воспроизводит её за доли секунды.

Верить решателю не нужно: доказательство проверяется механически.

Для митера невыполнимость — это и есть доказательство эквивалентности схем.

SAT и NP

Определение 9 (NP-полнота)

NP — задачи, чей ответ «да» проверяется за полиномиальное время по предъявленному сертификату.

Задача **NP-полна**, если она лежит в NP и к ней полиномиально сводится любая задача из NP.

Для SAT сертификат — выполняющий набор: подставить и проверить каждый дизъюнкт.

Проверить легко, найти — перебор.

Теорема Кука—Левина

Теорема 4 (Теорема Кука—Левина)

SAT NP-полна.

Без доказательства: конструкция кодирует таблицу вычисления машины Тьюринга в КНФ.

Формула выполнима тогда и только тогда, когда машина принимает вход — любая NP-задача превращается в SAT.

Следствие 1

Если $\mathrm{P} \neq \mathrm{NP}$, полиномиального алгоритма для SAT не существует.

Вопрос о полиномиальном алгоритме для SAT равносилен вопросу $\mathrm{P} = \mathrm{NP}$.

Практика против худшего случая

NP-полнота — свойство худшего случая.

Индустриальные формулы структурны, и CDCL решает формулы с миллионами переменных.

SAT — рабочий инструмент верификации, планирования и криптоанализа.

Эквивалентность схем с миллионами вентилей — стандартный шаг потока проектирования процессоров.

Теорема Кука обещала трудность, практика нашла структуру.

Лекция 14: что мы поняли?

- Схема — булева функция в железе: размер — площадь, глубина — задержка.
- Полный сумматор: сумма — $\oplus$ трёх битов, перенос — функция большинства.
- Каскад тянет перенос через все разряды: глубина $O(n)$, опережающий перенос снижает её до $O(\log n)$.
- Код Грея меняет один бит за шаг — энкодеры без ложных состояний и карта Карно живут на нём.
- Митер сводит эквивалентность схем к SAT: UNSAT — эквивалентны, SAT — контрпример.
- SAT NP-полна, но CDCL на структурных формулах решает миллионы переменных.

Вопросы для самопроверки

1. Постройте схему для $\overline{x \wedge y} \vee z$ и посчитайте её размер и глубину.
2. Запишите формулы полусумматора и полного сумматора.
3. Почему глубина каскада сумматоров ограничивает тактовую частоту?
4. Постройте G_4 из G_3 рекурсивно.
5. Переведите 0110 из кода Грея в двоичный.
6. Что выдаст SAT-решатель для митера схем $x \vee y$ и $x \oplus y$?
7. Почему выученный дизъюнкт отсекает больше, чем простой откат?

Чтение к следующей паре

- m11 — «Логические вентили и схемы», «Сумматоры», «Код Грея».
- m14 — «CDCL-солверы», «SAT и NP».
- Флеш-опросник — в начале лекции 15.