Логические выражения хоть и называются “логическими”, но на работе с ними строится вся вычислительная мощь современных компьютеров. В данной главе мы занимаемся т.н. ”параллельными” схемами, которые должны успеть сработать в течении одного такта. Если брать аналогию с языками программирования, они соответствуют вычислению выражений в императивных языках или примитивно-рекурсивным функциям в языках функциональных.

Логические функции.

Логические или булевы функции (названы так в честь английского математика Джорджа Буля (George Boole, 1815-1864)) работают с элементами булева множества {0, 1}. Элементы этого множества часто рассматривают как {ложь, истина}, хотя с точки зрения математики это просто символы без какой-либо смысловой нагрузки.

Строго говоря, существуют и логики высших порядков, например троичная - над множеством {0, 1, 2} и т.д..Если довести мысль до предела, получим бесконечное множество целых чисел {..., -2, -1, 0, 1, 2, …} и операции над ними. Но это уже совсем другая история.

Итак, логическая функция принимает на вход N аргументов (из множества {0,1}) и выдаёт результат из того же множества. Математически это записывается так:
F: $E_2^n \rightarrow E_2$, где $E_2 = \{0,1\}$ - булево множество.

Функции с N = 0, т.е. без аргументов, являются константами и их очевидно только две

True → 1
False → 0

Функций с одним аргументом - четыре.

- F(0) → 0, F(1) → 0: всегда возвращает 0 (тождественный ноль)
- F(0) → 1, F(1) → 1: всегда возвращает 1 (тождественная единица)
- F(0) → 0, F(1) → 1: повторитель аргумента, буфер
- F(0) → 1, F(1) → 0: логическое отрицание, инвертор
  (обозначается как ¬x, !x, $\bar{x}$, NOT(x))

Функций от двух аргументов - 16. Число это вычисляется так:
для функции от n аргументов множество всех вариантов значений аргументов образует n-мерный единичный ку.б. Число вершин n-мерного куба равно 2^n в случае n=0 это 1, n=1 → 2, n=2 → 4, n=3 → 8, … .
Для каждой из вершин значение функции может принимать два значения.
Итого, общее число функций - 2^{2^n} . Не будем приводить все, рассмотрим только самые важные.

Для функции $F(x_0, x_1)$:

x_0 =	0	1	0	1	
x_1 =	0	0	1	1	
	0	0	0	1	логическое И, конъюнкция, обозначается как x ∧ y = x · y = xy = x & y = x AND y = x И y = AND(x, y)
	0	1	1	1	логическое ИЛИ, дизъюнкция, … x ∨ y = x + y = x OR y = x ИЛИ y = OR(x, y)
	1	1	1	0	штрих Шеффера, NAND, 2И-НЕ, НЕ-И, … x | y = x ↑ y = x NAND y = x НЕ-И y = NAND(x, y)
	1	0	0	0	стрелка Пирса, NOR, НЕ-ИЛИ, 2ИЛИ-НЕ, … x ↓ y = x † y = x NOR y = x НЕ-ИЛИ y = NOR(x,y)
	0	1	1	0	исключающее или, XOR, сложение по модулю 2, … x <> y = x NE y = NE(x, y) = x ⊕ y = x XOR y = XOR(x,y)
					…

Всего, вспомним, функций 16. Функций от трёх аргументов 256, логично предположить, что комбинируя одни функции, можно получать другие.
Например, $NAND(x_0, x_1) = NOT(AND(x_0, x_1)) = AND(NOT(x_0),NOT(x_1))$.
Следовательно, существует минимальный набор (или наборы) функций, с помощью которых можно получить все остальные. В математике такие наборы называют "функционально полными". Каковы условия образования таких наборов функций?

Вопрос этот не так уж и тривиален. Критерий был сформулирован американским математиком Эмилем Постом (Emil Leon Post) в 1941 г. и назван в его честь. Окончательно доказать истинность критерия удалось лишь в 1980-х гг.

Критерий Поста

Пост ввёл классы т.н. "предполных" функций, всего их пять.

1) функции, сохраняющие ноль: F (0,0,…,0) = 0.
2) функции, сохраняющие единицу: F (1,1,…,1) = 1.
3) самодвойственные функции: $f(x_1, x_2, \ldots, x_n) = \overline{f(\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n)}$
4) монотонные функции: $f(\beta_1, \beta_2, \ldots, \beta_n) \geq f(\alpha_1, \alpha_2, \ldots, \alpha_n)$ при том, что для любого i $\beta_i \geq \alpha_i$
5) линейные функции: $f(x_1, x_2, \ldots, x_n) = \alpha_0 \oplus \alpha_1 x_1 \oplus \alpha_2 x_2 \oplus \ldots \oplus \alpha_n x_n$ при том, что a_i принадлежат множеству {0, 1}

Собственно формулировка критерия Поста такова - набор функций полон, если с их помощью можно реализовать пять функций - не сохраняющую ноль, не сохраняющую единицу, несамодвойственную, немонотонную и нелинейную.

Примеры функционально полных наборов:

- наборов булевых функций, состоящих из единственной функции два - это штрих Шеффера (NAND) и стрелка Пирса (NOR).
  Для них обеих справедливо F(0,0) = 1 и F(1,1) = 0, т.е. они не сохраняют ноль и единицу,
  обе несамодвойственны: NAND(1,0) != NOT(NAND(0, 1)), NOR(1,0) != NOR(0,1),
  немонотонны: NAND(1,1) < NAND(0,0),
  а также нелинейны, что дотошный читатель может проверить самостоятельно.

  Тот факт, что с помощью одних только элементов NAND и NOR можно собрать любую логическую функцию, активно использовался на заре компьютеростроения. Например, советский бортовой компьютер “Гном” (1965 г.) для самолётов АН-22, был полностью построен на микросборках “Квант”, каждая из которых содержала по четыре NOR элемента, соединённых в разных вариациях.

- в качестве примера минимально полного набора из двух функций (всего их 8) можно привести {¬, ∧}, этого достаточно, чтобы вывести NAND, аналогично, набора {¬, ∨} хватит для NOR.
  NAND(x,y) = NOT (AND (x,y) = AND (NOT (x), NOT (y))
  NOR(x,y) = NOT (OR (x,y)) = OR (NOT (x), NOT (y))

- самый известный набор из трёх функций ({¬, ∧, ∨} т.е. НЕ,И,ИЛИ) функционально полон, но не является минимальным. Он хорошо согласуется с привычной нам бытовой логикой и повсеместно используется в языках программирования. Но может быть упрощен до {¬, ∧} или {¬, ∨}.
  Вот как с его помощью вычисляется исключающее или (⊕, XOR):
  XOR(x,y) = OR (AND (NOT (x), y), AND (x, NOT (y)))
  = AND (OR (NOT (x), NOT (y)), OR (x, y)),
  Также популярен минимальный набор {∧, ⊕, 1}, который образует т.н.алгебру Жегалкина, об этом чуть позже.

- Минимальные наборы из четырёх функций существуют, но на практике не используются.

- Функционально полные наборы из пяти и более булевых функций всегда могут быть упрощены как минимум до четырёх функций.

Булева алгебра.

Выбор функционально полного набора булевых (т.е. для значений {0, 1}) функций определяет аксиоматику соответствующей булевой алгебры. Обычно под булевой алгеброй понимают именно избыточную алгебру с набором функций {¬, ∧, ∨}. Или, что то же самое, {¬, •, +}

Система аксиом для неё такова:

- если B != 1 то B = 0 и наоборот, если B != 0 то B = 1
- ¬0 = 1 и ¬1 = 0 свойства отрицания
- 0 • 0 = 0 И (∧, •, аналог умножения)
- 1 • 1 = 1 …
- 1 • 0 = 0 • 1 = 0 …
- 0 + 0 = 0 ИЛИ (∨, +, аналог сложения)
- 1 + 1 = 1 …
- 1 + 0 = 0 + 1 = 1 …

Теперь можно приниматься за теоремы (утверждения, выведенные из аксиом).
Теоремы с одним переменным:

- A • 1 = A идентичность
- A • 0 = 0 нулевой элемент
- A • A = A идемпотентность (равносильность)
- A + 1 = 1
- A + 0 = A идентичность
- A + A = A идемпотентность
- ¬¬A = A инволюция, снятие двойного отрицания
- A + ¬A = 1 дополнительность
- A • ¬A = 0 …

Теоремы с несколькими переменными:

- A • B = B • A коммутативность
  A + B = B + A …
- (A • B) • C = A • (B • C) ассоциативность
  (A + B) + C = A + (B + C) …
- (B • C) + (B • D) = B • (C + D) дистрибутивность
  (B + C) • (B + D) =B + (C • D) …
- B • (B + C) = B поглощение
  B + (B • C) = B …
- (B • C) + (B • ¬C) = B склеивание
  (B + C) • (B + ¬C) = B …
- (B • C) + (¬B • D) + (C • D) = B • C + ¬B • D согласованность
  (B + C) • (¬B + D) • (C + D) = (B + C) • (¬B +D) …

- ¬(A + B) = ¬A • ¬B — теорема де-Моргана
  ¬(A • B) = ¬A + ¬B — …

Обратим внимание, что теоремы идут парами, это называется двойственность теорем и получается она заменой операторов • на + и наоборот, а также 0 на 1 и наоборот. Это следует из симметричности аксиом к такого рода заменам.

Теперь, имея набор основных теорем, мы можем трансформировать логические выражения, например, стараясь из упростить.

Таблицы истинности.

С таблицами истинности мы уже имели дело, когда разбирались с возможными булевыми функциями двух аргументов. В сущности это перечисление всех вариантов значений аргументов и значений результата функции для них. Записывается таблица истинности в виде таблицы с колонками аргументами и одной колонкой - результатом. Число строк, очевидно, 2^{Narg}, где Narg - число аргументов функции.

Например, для функции ¬C + A • ¬B • C таблица истинности выглядит так:

A	B	C	¬C + A • ¬B • C
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Карты Карно

В сущности, значения таблицы истинности расположены в вершинах единичного куба размерности Narg, но если, скажем, трёхмерный куб можно легко представить, некоторым даже удаётся обращаться в уме с четырёхмерными кубами, всё что выше - за пределами человеческих визуальных способностей.

Карты Карно служат именно для этого - для упрощения работы с N-мерными кубами. Придуманы Морисом Карно (Maurice Karnaugh) из Bell Labs в 1953 г.

Для того же выражения ¬C + A • ¬B • C карта Карно выглядит так:

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

Для функции с четырьмя аргументами ¬C + A • ¬B • C + D карта Карно такова:

AB\CD	0 0	0 1	1 1	1 0
0 0	1	1	1	0
0 1	1	1	1	0
1 1	1	1	1	0
1 0	1	1	1	1

А для функции с пятью аргументами ¬C + A • ¬B • C + D • E, так:

C	0				1			
AB\DE	0 0	0 1	1 1	1 0	0 0	0 1	1 1	1 0
0 0	1	1	1	1	0	0	1	0
0 1	1	1	1	1	0	0	1	0
1 1	1	1	1	1	0	0	1	0
1 0	1	1	1	1	1	1	1	1

Карты Карно не используют для функций более чем с пятью - шестью аргументами, это становится неудобно.

Нормальные (канонические) формы логических функций.

Применяя к логическому выражению разные теоремы, мы получаем эквивалентные представления этого выражения. Например, выражение
¬A • ¬B • ¬C + A • ¬B • ¬C + A • ¬B • C == ¬B • ¬C + A • ¬B • C
но как это быстро определить, как понять, что разные выражения суть одно и то же?

Один из способов - сравнить их таблицы истинности, для эквивалентных выражений они должны быть идентичны.

Второй вариант - приведение выражений к т.н. нормальным или каноническим формам. Оба выражения приводят к канонической форме и если эти формы совпали, значит мы имеем дело с разными представлениями одного и того же выражения. На

практике применяют три вида таких форм. Теоретически, существуют и другие, но никакого практического смысла в них нет.

Дизъюнктивная нормальная форма (ДНФ)

ДНФ - это представление выражения в виде дизъюнкции конъюнкций литералов или их инверсий. Т.е. в ДНФ на верхнем уровне находится цепочка операций ИЛИ, которые имеют дело с цепочками из выражений И, которые работают с литералами (переменными и константами) и/или их отрицаниями.

Т.н. совершенная ДНФ (СДНФ) напрямую получается из таблицы истинности. Совершенной она называется из-за того, что в каждом И под-выражении участвуют все аргументы.

Построим её для выражения ¬B • ¬C + A • ¬B • C

A	B	C	¬B • ¬C + A • ¬B • C
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

Далее запишем для каждой строки с результатом 1 выражение, которое эту единицу порождает: (¬A • ¬B • ¬C) + (A • ¬B • ¬C) + (A • ¬B • C).
Если у функций идентичные таблица истинности, то и СДНФ будет одна.
Но получилось довольно громоздко.

На практике применяется т.н. минимизированная ДНФ или просто ДНФ. Ее получают из карт Карно. Это простой визуальный способ, которым удобно пользоваться в случае небольшого количества входов логической функции. Существуют и алгоритмические способы, более пригодные для автоматических преобразований, но мы сейчас не будем углубляться в это. Просто построим карту Карно.

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

В этой карте ищем прямоугольные последовательности единиц, причем стороны прямоугольника должны быть длиной в степень 2 (т.е. 1, 2, 4, 8). Прямоугольники эти могут пересекаться, ведь каждый из них порождает И выражение, которые будут объединены через ИЛИ. В данной карте таких прямоугольников два.

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

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

Теперь выясняем чему соответствуют эти прямоугольники и записываем результат: ДНФ = (¬B • ¬C) + (A • ¬B).

Внимательный читатель обратит внимание, что порядок следования B C необычен, это сделано для того, чтобы диапазон длины 2, например: 0,0,1 …0,1,1 соответствовал (в данном случае) выражению C.

Собственно в этом и заключается “изюминка” метода Карно - просто нарисовать N-мерный куб в виде двумерной таблицы умели и до него. Порядок в картах Карно задаётся т.н. кодом Грея, когда обходятся все вершины N-мерного куба так, чтобы на каждом шаге происходил переход только по одному ребру.

Конъюнктивная нормальная форма (КНФ).

КНФ - это представление выражения в виде конъюнкции дизъюнкций литералов или их инверсий. То же, что и ДНФ, но операции И и ИЛИ меняются местами.

Совершенная КНФ (СКНФ) строится аналогично СДНФ из таблицы истинности. Но в этот раз мы выписываем строки, где функция обращается в ноль. На том же примере находим 5 таких строк: {001→0, 010→0, 011→0, 110→0, 111→0}.

Если в случае ДНФ для каждой строки мы выписывали И выражение, то сейчас это будет ИЛИ выражение, причем значения аргументов инвертируются.
Например, 001→0 превратится в (A + B + ¬C). Если бы это был единственный 0 в значениях функции, то действительно, любое из A=1 или B=1 или C = 0, привело бы к 1 в результате. Если нулевых значений в результатах функции несколько, достаточно перечислить их через И. Итоговая СКНФ будет:

(A + B + ¬C) • (A + ¬B + C) • (A + ¬B + ¬C) • (¬A + ¬B + C) • (¬A + ¬B + ¬C)

Минимизированная КНФ или просто КНФ строится аналогично ДНФ. Но строятся максимальные прямоугольники (со сторонами в степень двойки), заполненные нулями.

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

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

Итоговое КНФ равно ¬B • (A + ¬C), что после раскрытия скобок превращается в (¬B • ¬C) + (A • ¬B) или в ДНФ.

Алгебраическая нормальная форма (полиномы Жегалкина).

В отличие от ДНФ и КНФ, полиномы Жегалкина построены не на наборе функций {¬, •, + }, а на {1, ⊕, • }, где ⊕ - это исключающее ИЛИ (XOR), 1 - тождественная единица - функция, всегда возвращающая единицу.

Предложен такой набор был Иваном Жегалкиным (МГУ) в 1927 г. Булева алгебра при работе с таким набором так и называется - алгебра Жегалкина.

Полином Жегалкина может быть получен из СДНФ формы функции (один из вариантов). Алгоритм основан на том факте, что только один из элементов дизъюнкции при любых значениях аргументов может принимать значение 1. Это следует из того, что СДНФ строится на основе таблицы истинности, раз одна строка дала единицу, значит все остальные строки, потенциально способные дать единицу, нулевые.

А следовательно, оператор ИЛИ в СДНФ может быть заменён на XOR, ведь

1 ⊕ 0 → 1
1 ⊕ 0 ⊕ 0 → 1
1 ⊕ 0 ⊕ 0 ⊕ 0 → 1
…

Берём СДНФ знакомой нам функции и меняем + на ⊕.
(¬A • ¬B • ¬C) + (A • ¬B • ¬C) + (A • ¬B • C) →
(¬A • ¬B • ¬C) ⊕ (A • ¬B • ¬C) ⊕ (A • ¬B • C)

Дальнейший шаг - надо избавиться от инверсий,
для этого воспользуемся тем, что ¬A = A ⊕ 1

(¬A • ¬B • ¬C) ⊕ (A • ¬B • ¬C) ⊕ (A • ¬B • C) →
((A⊕1)•(B⊕1)•(C⊕1)) ⊕ (A•(B⊕1)•(C⊕1)) ⊕ (A•(B⊕1)•C)

Далее раскрываем скобки, И (умножение) имеет больший приоритет чем сложение по модулю. Вспомним, A • 1 = A.

((A⊕1)•(B⊕1)•(C⊕1)) ⊕ (A•(B⊕1)•(C⊕1)) ⊕ (A•(B⊕1)•C) →
(A•B•C⊕A•B⊕A•C⊕A⊕B•C⊕B⊕C⊕1) ⊕ (A•B•C⊕A•B⊕A•C⊕A) ⊕ (A•C⊕A•B•C)

Теперь упростим выражение, пользуясь тем соображением, что A ⊕ B ⊕ B = A,
в данном случае B можно просто сократить. Вычеркиваем все парные выражения.
(A•B•C⊕A•B⊕A•C⊕A⊕B•C⊕B⊕C⊕1) ⊕ (A•B•C⊕A•B⊕A•C⊕A) ⊕ (A•C⊕A•B•C) →
B•C ⊕ B ⊕ C ⊕ 1 ⊕ A•C ⊕ A•B•C

Вот и получился полином Жегалкина для исходного выражения ¬B • ¬C + A • ¬B • C.

Полиномы Жегалкина не получили распространения в электронике по двум причинам:
- они громоздкие и, в отличие от СДНФ, не упрощаются
- операция исключительного ИЛИ довольно сложна для реализации с помощью полупроводников в отличие от привычных всем И,ИЛИ,НЕ,И-НЕ,ИЛИ-НЕ

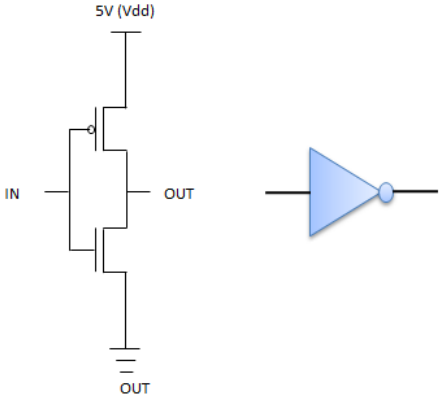
Схемотехника.

В электронике логические выражения в конце концов превращаются в электронные схемы. При использовании языков описания аппаратуры (HDL) разработчику даже не обязательно знать во что именно. Но понимание возможности и алгоритмы компиляторов HDL точно не помешает.

Базовые элементы.

Рассмотрим реализацию основных логических элементов в технологии КМОП (CMOS) т.к. она и ее варианты доминируют на данный момент. Подробнее об этом в главе 3.6.Микросхемы, синтез, топология.

- **Инвертор, NOT**. Функция с одним элементов, обращает логическую единицу в ноль и наоборот. В случае CMOS состоит из одного pMOS (обозначен пузырьком, размыкает цепь при высоком напряжении на базе) и одного nMOS (замыкает цепь при высоком напряжении на базе) транзисторов.

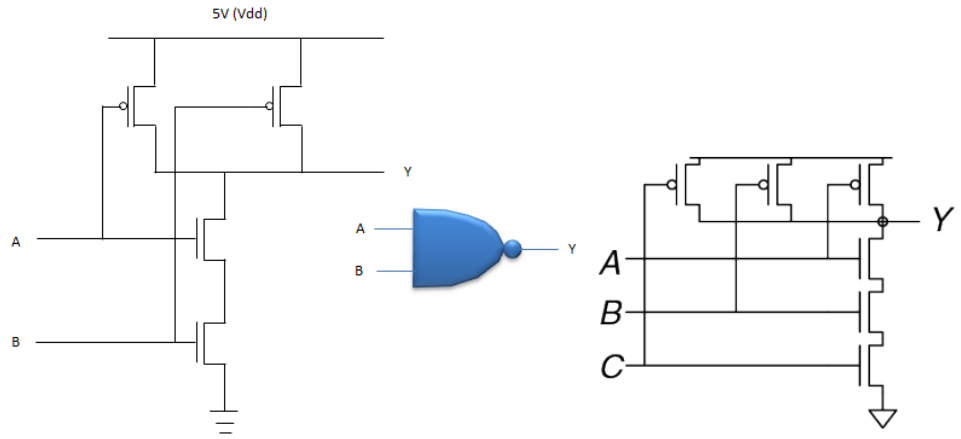


Фиг.4.4.1 Инвертор, схема в CMOS и обозначение на схеме

- **Буфер**, просто копирует аргумент в результат. Несмотря на кажущуюся тривиальность, это важный логический элемент. Поскольку на его прохождение требуется время, он используется для выравнивания времени прохождения сигналов в схеме. В CMOS буфер обычно устроен как два последовательно включенных инвертора.

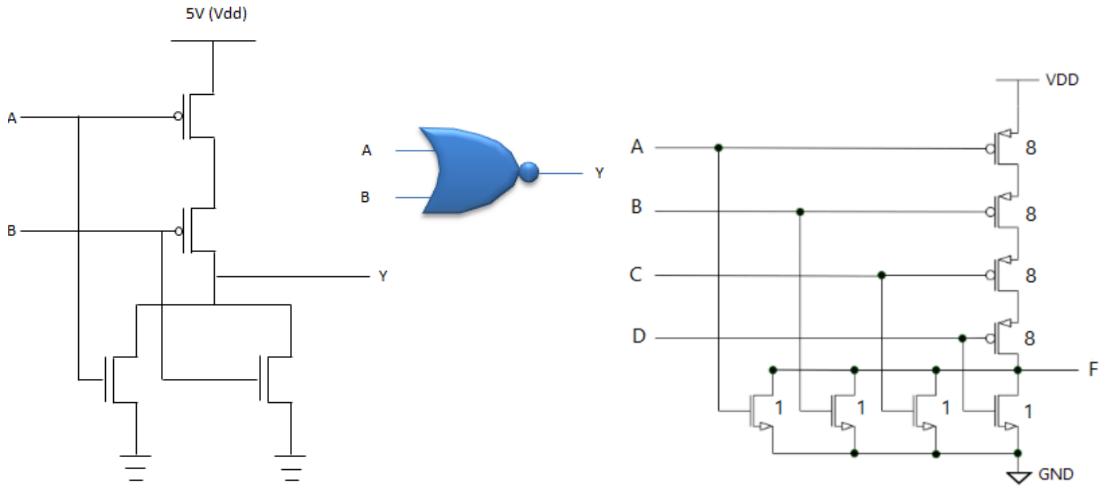
  Кроме того, существуют т.н. тристабильные или управляемые буферы в которых есть еще один управляющий вход. Если на этом входе высокий сигнал, буфер копирует аргумент в результат, если нет, размыкает выходную цепь. Это полезно, когда есть несколько логических схем, выдающих результат в одни и те же линии (шина), но в каждый конкретный момент результат может выдавать только одна схема.

- **И-НЕ, NAND**



Фиг.4.4.2 NAND(2), обозначение и схема, справа NAND3
Разумеется, это только один из возможных вариантов. Здесь и далее.

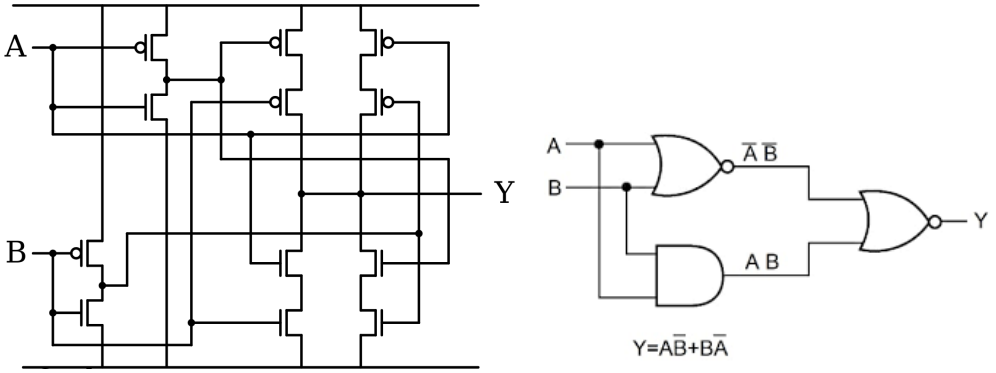
- **ИЛИ-НЕ, NOR**



Фиг.4.4.3 NOR(2), обозначение, схема, справа NOR4

- **И (AND), ИЛИ (OR)**. Когда требуется именно подобный элемент, на выход добавляют инвертор. Но иногда в рамках CMOS проще подобрать следующий элемент с отрицанием на входе.

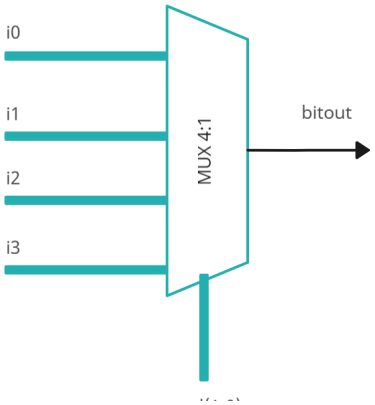
- **Исключающее ИЛИ, XOR**.



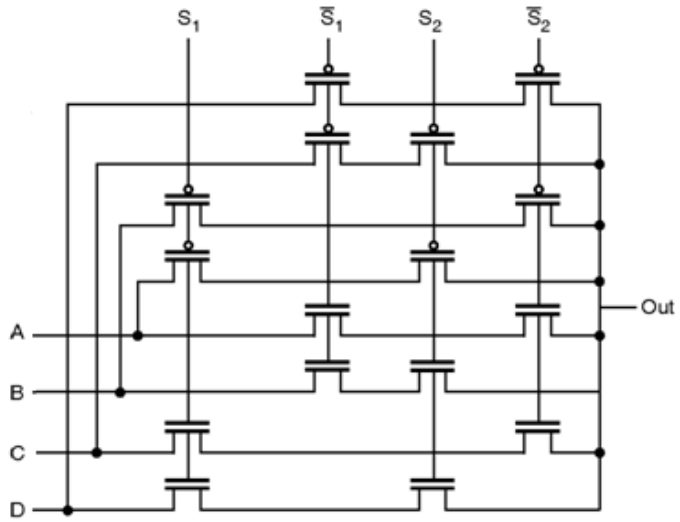
Фиг.4.4.4 CMOS схема, фактически реализация схемы справа

Неудивительно, что этой операции стараются избегать.

- **Мультиплексор, MUX**. На вход мультиплексору поступают два вида переменных. Во первых, это данные в количестве 2^n штук, во-вторых - код из n разрядов, в которых закодировано, какое именно из данных попадает в выходной сигнал.



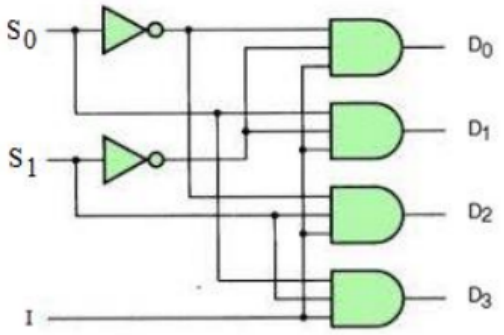
Фиг.4.4.5 Обозначение мультиплексора на схемах



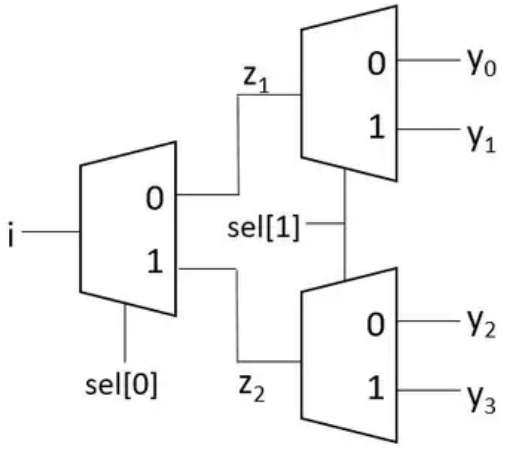
Фиг.4.4.6 возможная схема MUX4

На Фиг.4.4.6 видим мультиплексор на 4 элемента с двумя управляющими входам S1 и S2. В случае S1=0 и S2=0 в Out попадает сигнал A, (0,1) => B, (1,0) => C, (1,1) => D.

- **Дешифратор, (Decoder)** выполняет функцию, обратную мультиплексору. На вход поступает сигнал и двоичный код выхода (n разрядов), сигнал поступает в один из 2^n выходов.



Фиг.4.4.7 схема дешифратора 1x4 на элементах AND3

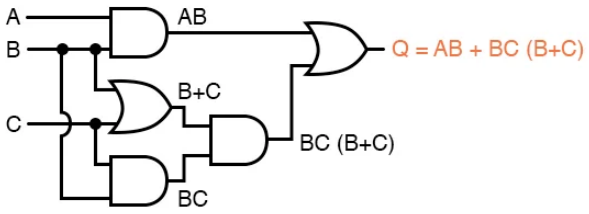


Фиг.4.4.8 дешифратор 1x4, сделанный из дешифраторов 1x2

Произвольные логические схемы.

Произвольную логическую функцию можно реализовать разными способами. Компилятор HDL выберет самый подходящий, выполнив при этом, если необходимо, эквивалентные преобразования. Но, предположим, под рукой нет компилятора HDL.

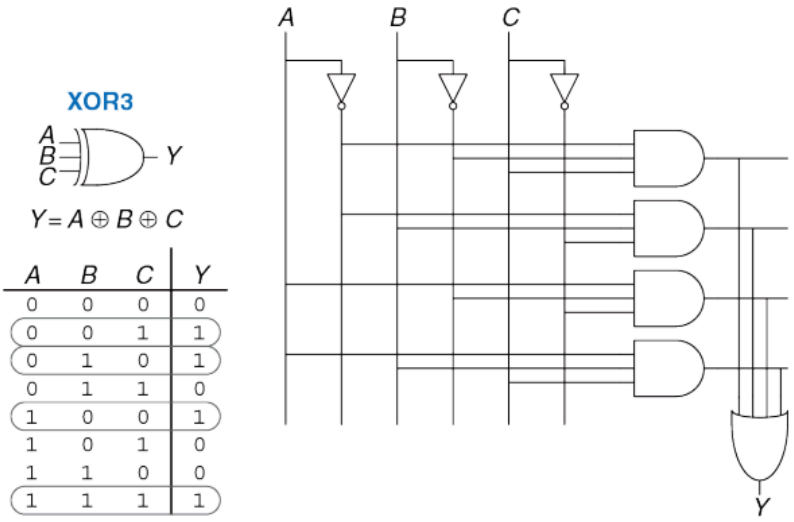
- Самый простой способ - переложить функцию на логические элементы как есть. Т.е. если исходная функция была A • B + B • C • (B + C), вот прямо так и записываем (см. Фиг.4.4.9). Хотя, выражение можно было свести к B • (A + C)



Фиг.4.4.9 пример логического выражения

- Предварительно упрощаем выражение. Например, приведя его к сокращенной ДНФ. В большинстве случаев это отлично работает. Но есть и выражения, которые не очень любят преобразовываться в ДНФ, Например, выражение вида (A1 + B1) • (A2 + B2) • … (An + Bn) породит дизъюнкцию из 2^n элементов.

- Более надёжный вариант, приведение к СДНФ. Это очень популярная и эффективная техника, она называется двухуровневая логика (конъюнкции, потом дизъюнкция), на этом приёме работают программируемые логические матрицы (CPLD).



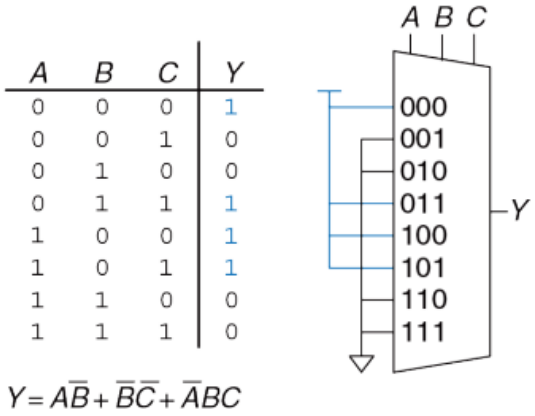
Фиг.4.4.10 Трёхвходовый XOR, представленный в виде СДНФ (отсюда)

- Итеративное упрощение. Пробуем разные варианты, например, не только ДНФ, но и КНФ, находим и вычисляем отдельно общие подвыражения, если они есть

…

- Строим и запоминаем в ПЗУ матрицу истинности выражения вместе с результатом. После чего вычисление результата сводится к его чтению из ПЗУ по адресу, закодированному в виде набора аргументов. Это также весьма популярный метод, используется в логических матрицах FPGA типа.

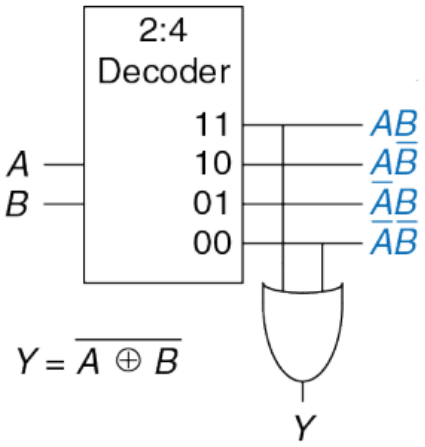
- Вычисление с помощью мультиплексора, почему нет.



A	B	C	Y
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

Фиг.4.4.11 таблица истинности функции и её запоминание в мультиплексоре.

- Вычисление с помощью дешифратора.



Фиг.4.4.12 Использование дешифратора на примере NOT(XOR(A,B))

- …