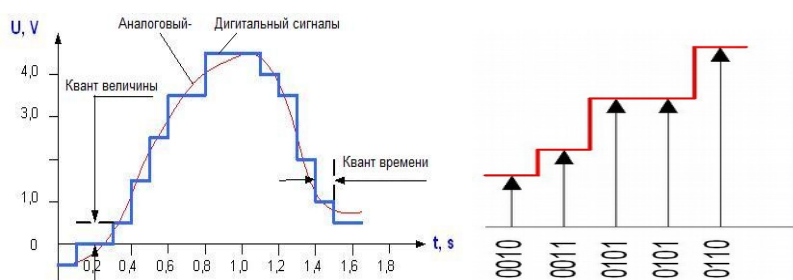


Л.14 Представление информации в цифровых системах. Системы счисления. Двоичная природа цифровых сигналов. Основные логические функции, схемотехническая реализация на дискретных элементах.

Сигнал — материальное воплощение сообщения, носитель информации при передаче, переработке и хранении информации. Понятие *сигнал* позволяет абстрагироваться от конкретной физической величины тока, напряжения, амплитуды акустической волны и рассматривать вне физического контекста явления, связанные кодированием информации и извлечением её из сигналов.

Сигналы можно разделить на две класса:

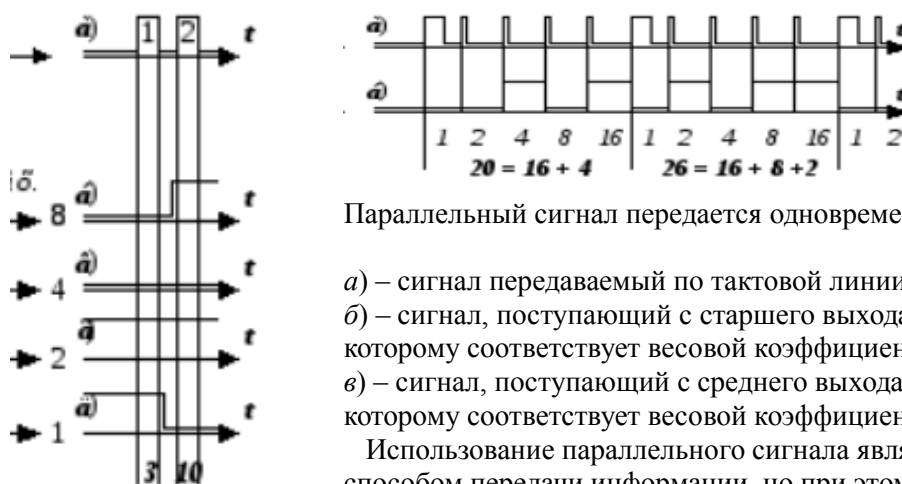
- аналоговые, непрерывные, изменяющиеся плавно;
- дискретные, изменяющиеся скачками обычно в определенные моменты времени. **Дискретные (цифровые) электрические сигналы** — имеют два или более допустимых состояния, например, уровни напряжения 0В и 5В кодируют две цифры — «0» и «1». Дискретные сигналы обычно изменяются только в дискретные моменты времени, интервал T между которыми называется **тактом**.



Самый распространенный тип дискретного сигнала имеет два значения: **ВЫСОКИЙ** (HIGH, TRUE, 1) и **НИЗКИЙ** (LOW, FALSE, 0) уровни. Такой сигнал называют двоичным (цифровой сигнал может иметь не только два уровня). Двоичный цифровой сигнал в природе не встречается, он создается человеком для удобства работы с информацией.

Информационным параметром может быть а) **количество импульсов N** за время между двумя определенными метками времени, причем ни длительность импульсов, ни их расположение при этом не имеют значения (сигнал набора номера в телефонных аппаратах с импульсным набором);

б) **место в группе** временной последовательности нулей и единиц внутри с определенным числом тактов дискретного времени. Первая позиция обычно соответствует самому младшему разряду, имеет самый малый вес, а все последующие позиции (номера дискретного времени внутри группы) имеют последовательно нарастающие веса вплоть до последней позиции с максимальным весом, соответствующей старшему разряду (байт, все веса, разряды кода являются степенями двойки).



Параллельный сигнал передается одновременно по нескольким линиям.

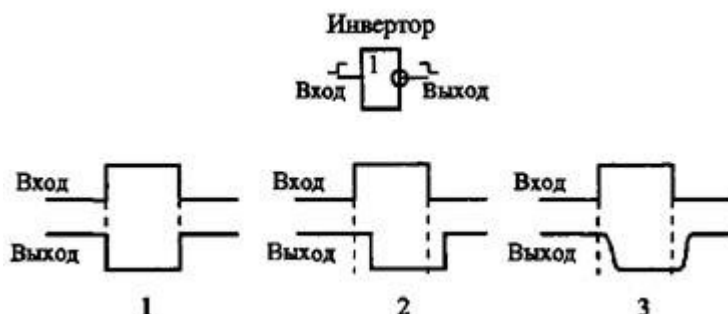
- а) — сигнал передаваемый по тактовой линии;
- б) — сигнал, поступающий с старшего выхода цифрового устройства CU , которому соответствует весовой коэффициент 8;
- в) — сигнал, поступающий с среднего выхода цифрового устройства CU , которому соответствует весовой коэффициент 4 и т.д.

Использование параллельного сигнала является самым быстрым способом передачи информации, но при этом требуются самые большие аппаратные затраты. Самым медленным является применение число — импульсного сигнала. Последовательный позиционный двоичный сигнал является промежуточным.

В подавляющем большинстве случаев разработчики **цифровых схем** используют три модели, три уровня представления о работе цифровых устройств.

1. Логическая модель.
2. Модель с временными задержками.
3. Электрическая модель.

На рис. 1 показано, как будет выглядеть выходной сигнал инвертора при использовании трех уровней его представления.



В основе формирования синхроимпульсов лежит частота задающего генератора, обеспечивающая получение требуемой формы и фазы синхроимпульса..

Рис. 1. Три уровня представления цифровых устройств.

Логическая модель иначе называется синхронным моделированием: сигналы определяются в момент прихода фронта синхроимпульса. В промежутке между синхросигналами входные сигналы не меняются, а переходной процесс в схеме завершается в течение некоторого Δt , меньшего периода повторения тактирующих импульсов. Но возникает вопрос – как часто могут приходить синхроимпульсы? Надо найти самый «медленный» логический элемент и установить такую частоту, что бы к моменту прихода следующего синхроимпульса вычисления закончились.

Представление информации в цифровых системах.

В различные исторические периоды развития человечества для подсчетов и вычислений использовались различные способы представления чисел.

ДВЕНАДЦАТЕРИЧНАЯ английская система мер: 1 фут = 12 дюймов, 1 шиллинг = 12 пенсов

ШЕСТИДЕСЯТЕРИЧНАЯ система Вавилона: 1 час = 60 минут; $1' = 60''$; $1^\circ = 60'$;

ДЕСЯТИЧНАЯ система – арабская, (возникла в Индии), позиционная:

$12 = \text{"Две-на-дцать"}$, $12 = 1 * 10^1 + 2 * 10^0 = 12$, положение чисел 1 и 2 определяется степенью числа 10. Аналогично: $342 = 3 * 10^2 + 4 * 10^1 + 2 * 10^0 = 300 + 40 + 2$.

ДВОИЧНАЯ СИСТЕМА СЧИСЛЕНИЯ - основанием является число 2. Для записи чисел используют всего две цифры: 0 и 1. $12 = 1 * 2^3 + 1 * 2^2 + 0 * 2^1 + 0 * 2^0$. $1100_2 = 12_{10}$

При написании программ на языках низкого уровня или в кодах МП и при обработке данных широко используются еще две системы счисления.

ВОСЬМЕРИЧНАЯ система в качестве основания использует число 8 и, соответственно, 8 цифр от 0 до 7. Перевод из десятичной системы в восьмеричную осуществляется по тому же правилу, что и в случае с двоичной системой. Например: число $453_{10} = 111.000.101_2 = 705_8$

ШЕСТИНАДЦАТЕРИЧНАЯ система в качестве основания использует число 16 и, соответственно, цифры от 0 до 9 и первые 6 букв латинского алфавита A, B, C, D, E, F.

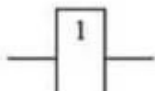

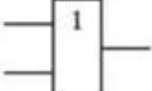

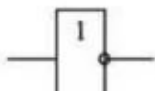

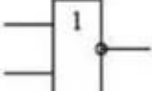



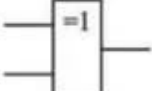



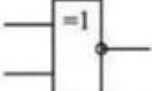

Запись операндов: десятичная (15, 154), шестнадцатеричная (префикс 0x или \$, например, 0x0f, \$0f, 0x9a, \$9a), восьмеричная (префикс – нуль, например, 017, 0232) и двоичная (префикс 0b, например, 0b00001111, 0b10011010).

Элементы алгебры логики, основные теоремы булевой алгебры и логические функции

В настоящее время математический аппарат алгебры логики является основой проектирования цифровых устройств. (Джордж Буль (1815-1864 г.) - булева алгебра).

Булева алгебра оперирует двоичными переменными, которые условно обозначаются как 0 и 1 (изначально «ложь» и «истина»). В ее основе лежит понятие *переключательной, булевой или логической функции* вида $f(x_1, x_2, \dots)$ относительно аргументов x_1, x_2, \dots , которая, как и ее аргументы, может принимать только два значения 0 или 1 (изначально «ложь» или «истина»). Логическая функция может быть задана словесно, алгебраическим выражением или таблицей, которая называется **таблицей истинности** или **таблицей соответствия**. Физически логические элементы могут быть выполнены *электромеханическими (на электромагнитных реле), электронными (на*

диодах и транзисторах), пневматическими, гидравлическими, оптическими и др.

ГОСТ	ANSI	ГОСТ	ANSI
 Буфер	 BUF	 ИЛИ	 OR
 Инвертор	 INV	 ИЛИ-НЕ	 NOR
 И	 AND	 Исключающее ИЛИ	 XOR
 И-НЕ	 NAND	 Исключающее ИЛИ-НЕ	 XNOR

Логический элемент изображают в виде прямоугольника. Слева **входы**, на которые подаются сигналы, соответствующие аргументам логической функции. Справа **выход**, с которого снимается сигнал, соответствующий логической функции. Логические элементы реализуются в виде интегральных микросхем, когда в одном корпусе выполняется сразу несколько логических элементов. Простые логические элементы ИЛИ, И, НЕ могут быть реализованы на дискретных элементах. **Большие схемы**, такие как микропроцессоры, могут быть очень сложными, поэтому рассмотрение схемы как «черного ящика» с тщательно определенными интерфейсом и функцией есть применение **принципов абстракции и модульности**. Построение схемы из более мелких элементов является применением **иерархического подхода** к разработке.

Три функции, **инверсию (НЕ)**, **дизъюнкцию (ИЛИ)** и **конъюнкцию (И)** часто называют основными, так как они составляют функционально полную систему, с помощью которой можно наиболее просто выразить любую другую логическую функцию. Число аргументов однозначно определяет число различных функций от этих аргументов. При числе аргументов равном n , число их различных сочетаний равно 2^n , а число функций – 4^n . Все логические функции для двух переменных, реализуемые в виде логических электронных элементов, приведены в таблице 1.

Аргументы Функция

x1	x2	ИЛИ	И	ИЛИ-НЕ	И-НЕ	Неравнозначность	Равнозначность
0	0	0	0	1	1	0	1
0	1	1	0	0	1	1	0
1	0	1	0	0	1	1	0
1	1	1	1	0	0	0	1

Словесное описание приведенных выше функций выглядит так.

Функция ИЛИ равна 1 при равенстве любого аргумента 1.

Функция И равна 1 при равенстве всех аргументов 1.

Функция ИЛИ-НЕ равна 1 при равенстве всех аргументов 0.

Функция И-НЕ равна 1 при равенстве любого аргумента 0.

Функция ИСКЛЮЧАЮЩЕЕ ИЛИ (при двух аргументах) равна 1 при неравных (неравнозначных) аргументах.

Функция РАВНОЗНАЧНОСТЬ (при двух аргументах) равна 1 при равных (равнозначных) аргументах.

Как и в алгебре чисел, в алгебре логики существуют теоремы, знание которых значительно облегчает действия с логическими переменными:

Коммутативный закон:

$$x_1 x_2 = x_2 x_1 \qquad x_1 + x_2 = x_2 + x_1$$

Ассоциативный закон:

$$x_1 (x_2 x_3) = (x_1 x_2) x_3 \qquad x_1 + (x_2 + x_3) = (x_1 + x_2) + x_3$$

Дистрибутивный закон:

$$x_1 (x_2 + x_3) = x_1 x_2 + x_1 x_3 \qquad x_1 + x_2 x_3 = (x_1 + x_2)(x_1 + x_3)$$

Правило склеивания:

$$x_1 (x_1 + x_2) = x_1 \qquad x_1 + x_1 x_2 = x_1$$

Правило повторения:

$$x x = x \qquad x + x = x$$

Правило отрицания:

$$x \bar{x} = 0 \qquad x + \bar{x} = 1$$

Правило двойного отрицания:

$$\overline{(\bar{x})} = x$$

Теорема де Моргана:

$$\overline{x_1 x_2} = \bar{x}_1 + \bar{x}_2 \qquad \overline{x_1 + x_2} = \bar{x}_1 \bar{x}_2$$

Операции с нулем и единицей:

$$x \cdot 1 = x \qquad x + 0 = x$$

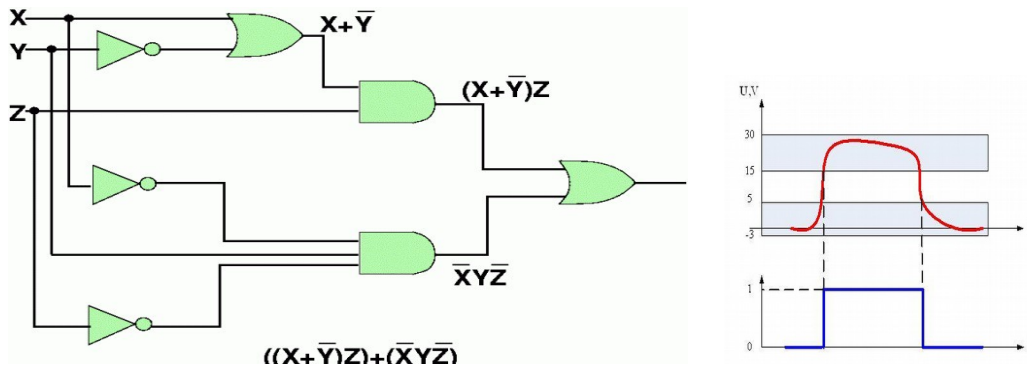
$$x \cdot 0 = 0 \qquad x + 1 = 1$$

$$\bar{0} = 1 \qquad \bar{1} = 0$$

Аксиомы: свойства констант 0 и 1:	$1+A=1$ $0*A=0$ $0+A=A$ $1*A=A$
идемпотентность:	$A+A=A$ $A*A=A$
Закон исключения третьего:	$A + \neg A = 1$
Закон непротиворечивости:	$A * \neg A = 0$
Закон отрицания:	$\neg(\neg A) = A$
Законы коммутативности:	$A+B=B+A$ $A*B=B*A$
Законы ассоциативности:	$A+B+C=A+(B+C)$ $A*B*C=A*(B*C)$
Законы дистрибутивности:	$A*(B+C)=A*B+A*C$ $A+(B*C)=(A+B)*(A+C)$
Законы де Моргана:	$\neg(A+B) = \neg A * \neg B$ $\neg(A*B) = \neg A + \neg B$
Законы поглощения:	$A+A*B=A$ $A*(A+B)=A$

Математическая логика подразумевает **упрощение** заданных уравнений. Так же, как и в алгебре, необходимо сначала максимально облегчить условие (избавиться от сложных вводимых и операций

с ними), а затем приступить к поиску верного ответа. Для упрощения можно преобразовать все производные операции в простые, раскрыть все скобки (или наоборот, вынести за скобки, чтобы сократить этот элемент). Следующим действием должно стать применение свойств булевой алгебры на практике (поглощение, свойства нуля и единицы и т. д.)



Логическая формула с наименьшим числом логических связей называется **минимальной**. Так получаем **минимальную дизъюнктивную нормальную форму** (МДНФ) и, соответственно, - МКНФ. Процесс отыскания минимальной формы называется **минимизацией логической функции** или просто минимизацией.

Минимизировать функции можно тремя методами:

- 1) расчетным путем, используя законы алгебры логики,
- 2) графическим путем (метод карт Карно или диаграмм Вейча), используя специальные карты,
- 3) расчетно-графическим путем (метод Квайна и его модификации).

Схемотехническая реализация на дискретных элементах.

С помощью простейших транзисторных каскадов, работающих в ключевом режиме (включен/выключен), можно реализовать основные функции алгебры логики и основные математические действия (сложение, вычитание, умножение, деление) для чисел в двоичном представлении. А также схемы хранения (памяти) для двухуровневых (цифровых) значений и их передачи на значительные расстояния.

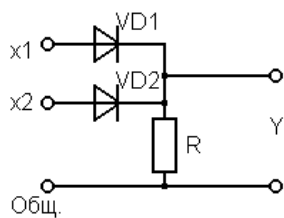
Семейства цифровых интегральных схем

В зависимости от типа применяемых элементов и особенностей схемотехники различают следующие семейства ЦИС: **ТЛНС**— транзисторные логические ИС с непосредственной (гальванической) связью; **РТЛ** — резисторно-транзисторные логические ИС; **РЕТЛ** резисторно-емкостные логические ИС; **ДТЛ** —диодно-транзисторные логические ИС; **ТТЛ** — транзисторно-транзисторные логические ИС; **И²Л**—интегральные инжекционные логические схемы; **ЭСЛ**—эмиттерно-связанные логические ИС; **МДП** —логические схемы на основе МДП транзисторов; **КМДП** — логические схемы на основе комплементарных МДП транзисторов.

Существуют наборы логических функций, называемых полными. К полному набору относятся функции ИЛИ-НЕ и И-НЕ. Это означает, что имея достаточное количество логических элементов, например, И-НЕ, можно построить любую цифровую схему

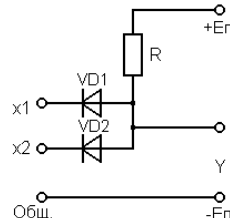
Схема ИЛИ: - если входные напряжения меньше напряжения открытия диода ($\sim 0,7$ В), то диоды закрыты. Нагрузка схемы R отключена от входных сигналов и, следовательно, напряжение на выходе равно 0. Если же напряжения на входах, не выходя за пределы уровня логического 0, больше напряжения открытия диодов, то диоды открыты. Напряжение на выходе схемы (резисторе R), определяется из 2 закона Кирхгофа – ($U_{вх} - U_{VD}$): $U_{ВЫХ} = U_{ВХ} - U_{VD}$ меньше входного, т.е. соответствует уровню логического нуля. Если считать диоды идеальными, т.е. $U_{VD} = 0$, то и в этом случае напряжение на выходе не выходит за пределы уровня логического 0.

Если на вход x_2 подается напряжение 5 В, диод VD_2 открыт, напряжение на выходе $U_{ВЫХ} = 5 - 0,7 = 4,3$ В, а это есть уровень логической единицы. Диод VD_1 закрыт, так как напряжение на его катоде (4,3 В) больше анодного ($U_{X1} < 1$ В).



A	B	$A \vee B$
1	1	1
1	0	1
0	1	1
0	0	0

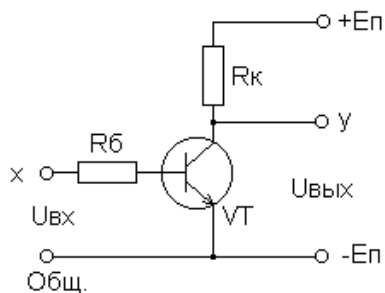
Рис. 2. Схема логического элемента ИЛИ.



A	B	$A \wedge B$
1	1	1
1	0	0
0	1	0
0	0	0

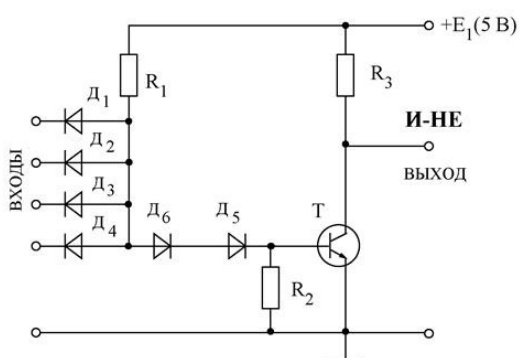
Рис. 3. Схема логического элемента И

Элемент И (рис. 3) также состоит из диодов и одного резистора, но требует источника питания. Для 1 строчки таблицы истинности оба диода закрыты и напряжение на выходе $= E_n$ (уровень логической единицы). Для второй и третьей строчек таблицы истинности на один из входов подан уровень логической 1. Диод, на катод которого подан 0, будет открыт, а диод, связанный своим катодом с 1, будет закрыт. На выходе – уровень логического 0. Для 4-й строки оба диода открыты. $U_{ВЫХ} = U_{ВХ} + U_{VD}$. Напряжение на выходе будет равно напряжению на открытом диоде, т.е. 0,7 В, это уровень логического 0.



Для реализации элемента НЕ используется схема транзисторного ключа (рис. 4).

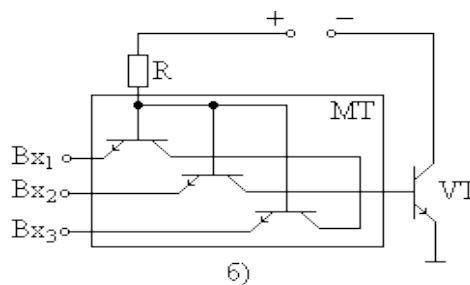
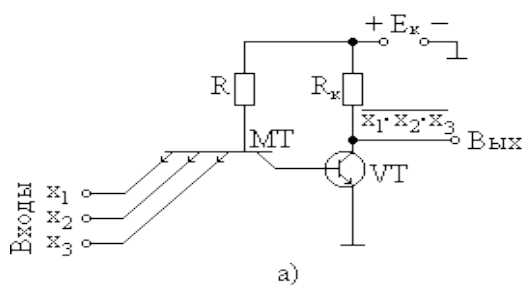
Рис. 4. Схема логического элемента НЕ



В базовых элементах семейства ДТЛ отказались от резисторов и конденсаторов в цепях без транзисторов и используют вместо них диоды.

Операция И осуществляется диодной частью схемы ($D_1 - D_4$, R_1), а транзисторный каскад с общим эмиттером служит инвертором. Введение в схему диодов VD_5 , VD_6 способствует увеличению перепада логических уровней и помехоустойчивости элемента ($U_{ном} \gg 0,5$ В). Для того чтобы работа диодов VD_5 , VD_6 не зависела от состояния транзистора VT, в схеме предусмотрено сопротивление R_2 .

В семействе ТТЛ удалось избежать основного недостатка элементов ДТЛ—большого количества диодов. В базовом элементе семейства ТТЛ (рис. 3.2, б) функции диодов (операцию И) выполняет входной многоэмиттерный транзистор VT_1 , а транзистор T_2 служит в качестве инвертора.

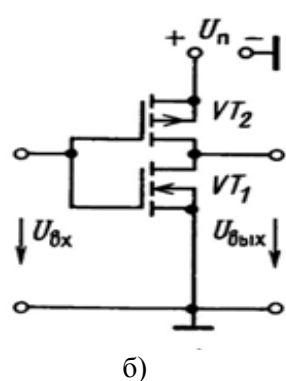
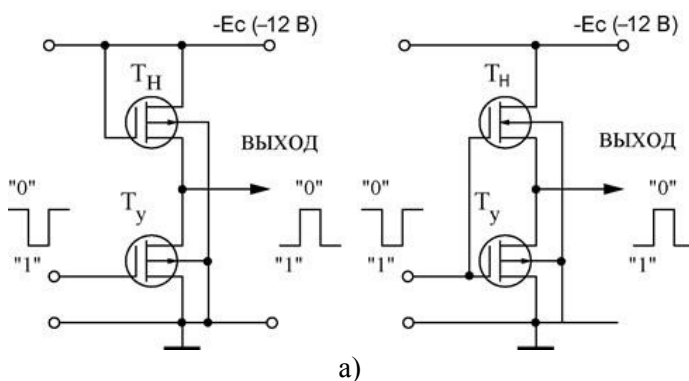


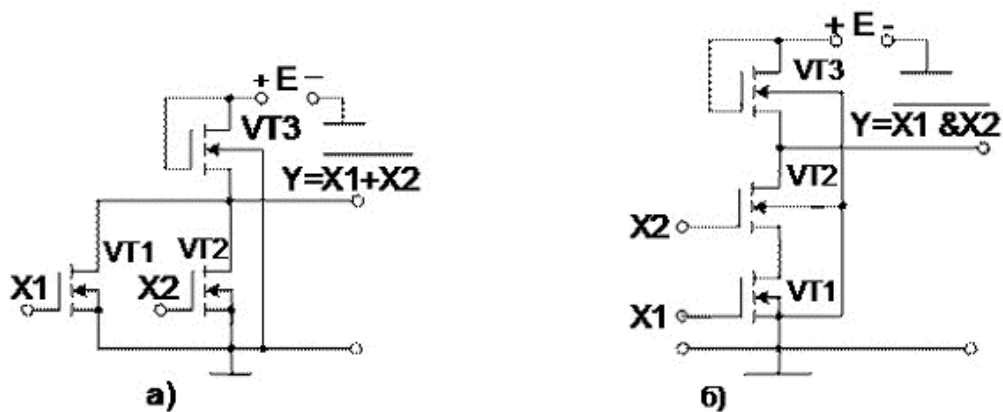
Тип операции	Обозначение элемента		Таблица истинности	Пример технической реализации															
	буквенное	графическое																	
НЕ	ЛН	$X \rightarrow Y$	<table><tr><td>X</td><td>Y</td></tr><tr><td>1</td><td>0</td></tr><tr><td>0</td><td>1</td></tr></table>	X	Y	1	0	0	1										
X	Y																		
1	0																		
0	1																		
или	ЛЛ	$X_1, X_2 \rightarrow Y$	<table><tr><td>X₁</td><td>X₂</td><td>Y</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>0</td><td>0</td><td>0</td></tr></table>	X ₁	X ₂	Y	1	0	1	1	1	1	0	1	1	0	0	0	
X ₁	X ₂	Y																	
1	0	1																	
1	1	1																	
0	1	1																	
0	0	0																	
и	ЛИ	$X_1, X_2 \rightarrow Y$	<table><tr><td>X₁</td><td>X₂</td><td>Y</td></tr><tr><td>1</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td></tr></table>	X ₁	X ₂	Y	1	1	1	1	0	0	0	1	0	0	0	0	
X ₁	X ₂	Y																	
1	1	1																	
1	0	0																	
0	1	0																	
0	0	0																	
или-НЕ	ЛЕ	$X_1, X_2 \rightarrow Y$	<table><tr><td>X₁</td><td>X₂</td><td>Y</td></tr><tr><td>1</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>1</td></tr></table>	X ₁	X ₂	Y	1	1	0	1	0	0	0	1	0	0	0	1	
X ₁	X ₂	Y																	
1	1	0																	
1	0	0																	
0	1	0																	
0	0	1																	
и-НЕ	ЛА	$X_1, X_2 \rightarrow Y$	<table><tr><td>X₁</td><td>X₂</td><td>Y</td></tr><tr><td>1</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>0</td><td>0</td><td>1</td></tr></table>	X ₁	X ₂	Y	1	1	0	1	0	1	0	1	1	0	0	1	
X ₁	X ₂	Y																	
1	1	0																	
1	0	1																	
0	1	1																	
0	0	1																	

Логические элементы на полевых транзисторах.

Преимуществами логических элементов на полевых МОП (МДП) - транзисторах являются: малая мощность, потребляемая входной цепью, в результате чего соответственно возрастает коэффициент разветвления по выходу $K_p \gg 10-20$; простота технологического процесса изготовления, сравнительно низкая стоимость, малая потребляемая мощность, большая степень интеграции элементов в кристалле микросхем. Однако, по быстродействию лучшие логические элементы на МОП транзисторах уступают схемам на биполярных транзисторах. Это обусловлено тем, что у них имеются сравнительно большие входные емкости, на перезарядку которых затрачивается определенное время, поскольку входные и выходные сопротивления велики.

а) Нижний транзистор выполняет функцию ключа, а верхний — функцию нелинейного резистора нагрузки (при наличии тока его сопротивление R , при уменьшении тока сопротивление $\rightarrow 0$, передвижение по выходной характеристике). б) комплементарное включение ($U_{вх} = 1$ открыт нижний транзистор и закрыт верхний, при $U_{вх} = 0$ — наоборот).





*) Схемы И-ИЛИ на ключах с динамической нагрузкой

Рис.13. а). Схема логического элемента НЕ на однотипных МОП-транзисторах; б) схема логического элемента НЕ на комплементарных МОП-транзисторах

Р-канальные элементы работают в режиме отрицательной логики, n-канальные — в режиме положительной логики. При подаче на входы отпирающего напряжения ($U_{вх} = U_{вх}^1$) транзисторы VT_1 открываются и на выходе напряжение принимает значение, соответствующее логическому 0 ($U_{вых} = U_{вых}^0$).

Существенно лучшими показателями обладает инвертор КМДП логики (рис. 13-б), у которого нагрузочный транзистор VT_2 включается и выключается в противофазе с транзистором VT_1 . Для упрощения схемы управления удобно использовать транзисторы VT_1 и VT_2 с разными типами проводимости каналов. Это позволяет соединить их затворы между собой и управлять однополярными импульсами. Если на входе низкий уровень напряжения $U_{вх} = 0$, то открыт р-канальный транзистор VT_2 , а n-канальный VT_1 закрыт. Выходное напряжение соответствует логической 1 ($U_{вых} = U_n$). Если на входе появляется высокий уровень напряжения ($U_{вх} = U_n$), то транзистор VT_1 открывается, а VT_2 закрывается. Выходное напряжение близко к нулю ($U_{вых} = 0$).

КМДП элементы допускают работу с напряжениями питания, изменяющимися в широких пределах (3 ... 15 В). В статическом режиме потребление тока данным элементом равно нулю. Лишь в момент переключения потребляется ток, определяемый в основном процессами перезаряда емкости нагрузки. Мощность, потребляемая элементом, растет пропорционально частоте переключения. При напряжении $U_n = 5$ В элементы КМДП обладают совместимостью с элементами ТТЛ логики, превосходя последние по помехоустойчивости ($U_{пом} \gg U_n/3$).

Передаточной характеристикой КМОП инвертора называется зависимость выходного напряжения логического элемента от напряжения на его входе. На передаточной характеристике можно выделить несколько областей (рис. 16 а,б):

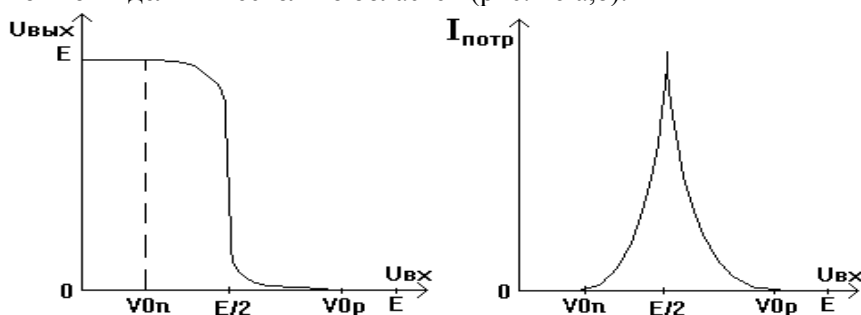


Рис.16. Зависимость выходного напряжения (а) и потребляемого тока (б) КМОП-инвертора от входного напряжения.

1. Входное напряжение меньше порогового напряжения V_{0n} , n-канальный транзистор закрыт. Напряжение затвор-исток р-канального транзистора больше его порогового напряжения V_{0p} (по модулю), поэтому он полностью открыт. Напряжение на выходе равно напряжению питания, потребляемый ток равен нулю.
2. Входное напряжение больше порогового напряжения V_{0n} п-канального транзистора, но меньше половины напряжения питания. N-канальный транзистор начинает открываться, а Р-канальный транзистор начинает закрываться.

3. В области значения входного напряжения равного $E/2$ передаточная характеристика идет практически вертикально. При входном напряжении равном $E/2$ оба транзистора открыты в одинаковой степени, потребляемый ток максимален.
4. Входное напряжение больше $E/2$ и меньше $(E-V_{op})$. Картина аналогична (симметрична) участку 2, но теперь n- и p-канальные транзисторы меняются ролями. Напряжение на выходе уменьшается.
5. Входное напряжение больше $(E-V_{op})$ и меньше напряжения питания. N-канальный транзистор полностью открыт, p-канальный – закрыт, его напряжение затвор-исток меньше порогового по модулю. Напряжение на выходе равно 0. При этом потребляемый ток равен нулю, т.к. p-канальный транзистор закрыт.

Отличительной чертой КМОП-схем является то, что на первом и пятом участках схема тока не потребляет (т. к. закрыт, соответственно, n-канальный или p-канальный транзистор). В этом состоянии потребляемая схемой мощность обусловлена только токами утечки через обратно смещенные переходы сток-подложка, исток-подложка. При комнатной температуре эти токи очень малы. Максимум потребляемого тока наблюдается в точке входного напряжения, близком $E/2$

Схемы элементов ИЛИ - НЕ, И - НЕ в КМОП – логиках получают путем последовательного соединения группы транзисторов одного типа (ИЛИ) и параллельного соединения группы транзисторов другого типа, представляющие собой элементы И. Если нагрузочные транзисторы по выходу соединены параллельно, а каналы входных транзисторов соединены последовательно, получился логический элемент, выполняющий операцию И-НЕ (рис. 14-б).

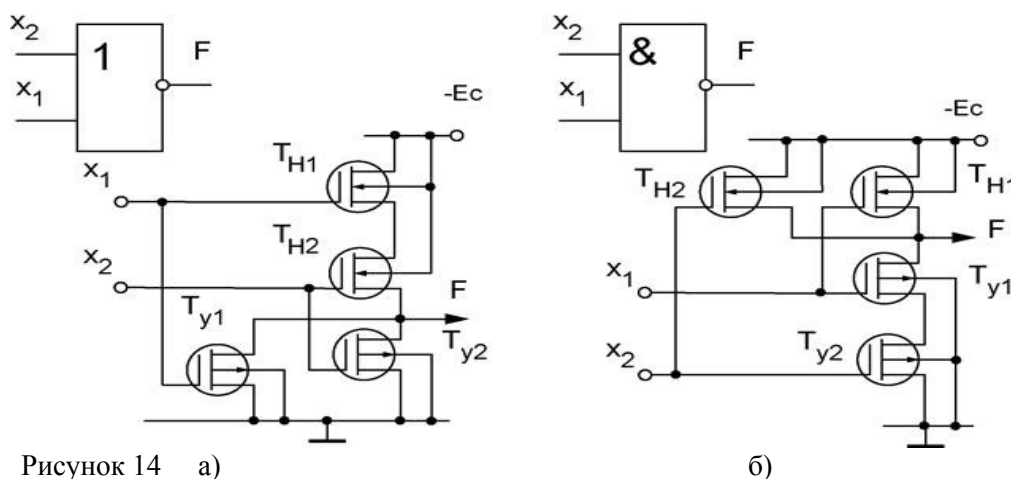


Рисунок 14 а)

б)

Сравнение параметров логических элементов основных семейств

Тип элемента	Напряжение питания	Средняя потребляемость, Р, мВт	Среднее время задержки $t_{з.ср}$, нс	Работа переключения $A_{п}$, пДЖ	Нагрузочная способность К раз
РТЛ	3	5	25	125	4
ДТЛ	5	9	25	225	7
ТТЛ	5	10	10	100	10
ТТЛШ	5	2	10	20	10
ЭСЛ	5	40	0,75	30	10
p-МДП	+4, -12	0,5	100	50	20
n-МДП	+12, ±5	0,5	30	15	20
КМДП	+3 до 15	$(0,2...0,3) \cdot 10^{-3}$	90...30	0,05	50
И ² Л	1	$(1...10) \cdot 10^{-3}$	1000...10	1	3

Наименьшее время задержки имеют элементы с малым перепадом логических уровней и повышенным энергопотреблением. Сравнение логических элементов по этому показателю позволяет сделать вывод, что наиболее перспективными семействами логических элементов являются И²Л и КМДП.

По характеру связи между входными и выходными переменными с учетом изменения этих связей по тактам работы различают **комбинационные устройства** и **последовательные (цифровые автоматы)**. В *комбинационных устройствах* совокупность выходных сигналов в каждый такт работы однозначно определяется входными сигналами, имеющимися в этот момент на его входах. *Цифровые автоматы* содержат устройства памяти (триггеры) и используются, если выходные сигналы определяются не только входными, но и предыдущими сигналами.

Схемное описание цифровых устройств было широко распространено в прошлом веке.

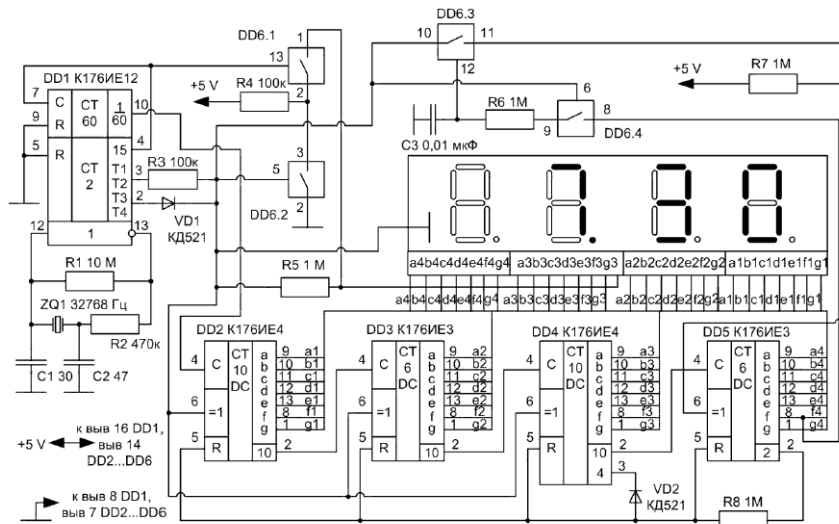


Рис. 3.1. Электрическая схема электронных часов на ЖКИ

В настоящее время в среде разработки ЭВМ используется мало, основную роль при разработке цифровых устройств играют **языки описания аппаратуры. HDL** (от *hardware description language*) — специализированные компьютерные языки, используемый для описания структуры и поведения цифровых логических электронных схем. Они внешне похожи на такие языки программирования, как Си или Паскаль,

написанные на них программы также состоят из выражений, операторов, управляющих структур. Важнейшим отличием между обычными языками программирования и языками HDL является явное включение концепции времени в языки описания аппаратуры и параллельности выполнения операторов. Языки описания аппаратуры являются неотъемлемой частью САПР специализированных интегральных схем (ASICs), микропроцессоров, микроконтроллеров и программируемых логических интегральных схем (ПЛИС).

Два основных языка описания аппаратуры (Hardware Description Language, HDL) – **SystemVerilog** и **VHDL**. В промышленности склоняются к SystemVerilog, но много компаний еще используют VHDL, поэтому многим разработчикам нужно владеть обоими языками. По сравнению с SystemVerilog, VHDL более многословный и громоздкий.

В 1990-е годы разработчики цифровых схем обнаружили, что их производительность труда резко возрастала, если они работали на более высоком уровне абстракции, определяя только логическую функцию и предоставляя создание оптимизированных логических элементов **системе автоматического проектирования (САПР)**. На обоих языках можно полностью описать любую логическую систему, но у каждого языка есть свои особенности. Лучше использовать язык, который уже распространен в вашей организации или тот, которого требуют ваши клиенты. Большинство САПР сейчас позволяют смешивать языки, поэтому разные модули могут быть написаны на разных языках. **SystemVerilog** и **VHDL** построены на похожих принципах, но их синтаксис весьма различается. В лекции будет представлено две колонки для сравнения, в одной SystemVerilog, а в другой – VHDL и будут даны предварительные объяснения и комментарии для сравнения описания аппаратуры в схематическом виде и в форме HDL-модели.

Блок цифровой аппаратуры, имеющий входы и выходы, называется **модулем**. Логический элемент «И», «ИЛИ», схема приоритетов и пр. являются примерами модулей цифровой аппаратуры. Есть два общепринятых типа описания функциональности модуля – **поведенческий и структурный**. Поведенческая модель описывает, что модуль делает. Структурная модель

описывает то, как построен модуль из простых элементов, с применением принципа иерархии. **Поведенческий уровень** описания объектов проекта является базовым, поскольку любые схемы, по крайней мере на самом нижнем уровне, представлены элементами, реализация которых выполнена на уровне поведения.

Код на SystemVerilog и VHDL из примера показывает поведенческое описание модуля, который рассчитывает булеву функцию $Y = A^{-} B^{-} C^{-} + AB^{-} C^{-} + AB^{-} C$. На обоих языках модуль назван sillyfunction ['sili] и имеет 3 входа, a, b и c и один выход y, и, как и следовало ожидать, следует принципу модульности.

SystemVerilog

```
module sillyfunction (input logic a, b, c, output logic y);  
  assign y = ~a & ~b & ~c | a & ~b & ~c | a & ~b & c;  
endmodule
```

Модуль на SystemVerilog начинается с имени модуля и списка входов и выходов. Оператор **assign** (присвоить) описывает комбинационную логику. Тильда (~) означает НЕ, амперсанд (&) – И, а вертикальная черта (|) – ИЛИ.

В Verilog существуют два класса типов данных: для **моделирования аппаратуры** и стандартные **арифметические типы данных** для моделирования алгоритмов, скопированные из языка Си. Для моделирования сигналов в большинстве модулей на Verilog используются – **wire** и **reg**. Из названия может показаться, что **wire** моделирует провод, а **reg** – регистр, но, это не всегда так. Наличие двух типов это скорее баг в дизайне языка, в **SystemVerilog** – современной версии Verilog, есть универсальный тип **logic**, который может использоваться во всех случаях.

VHDL

```
library IEEE;  
use IEEE.STD_LOGIC_1164.all;  
entity sillyfunction is  
  port(a, b, c: in STD_LOGIC;  
        y: out STD_LOGIC);  
end;  
architecture synth of sillyfunction is  
  begin  
    y <= (not a and not b and not c) or  
         (a and not b and not c) or (a and not b and c);  
  end;
```

Код на VHDL состоит из трех частей: объявления используемых библиотек и внешних объектов (library, use), объявления интерфейса объекта (entity) и его внутренней структуры (architecture). Конструкция для объявления используемых внешних объектов рассматривается отдельно. В объявлении интерфейса указывается имя модуля и перечисляются его входы и выходы. Блок architecture определяет, что модуль делает.

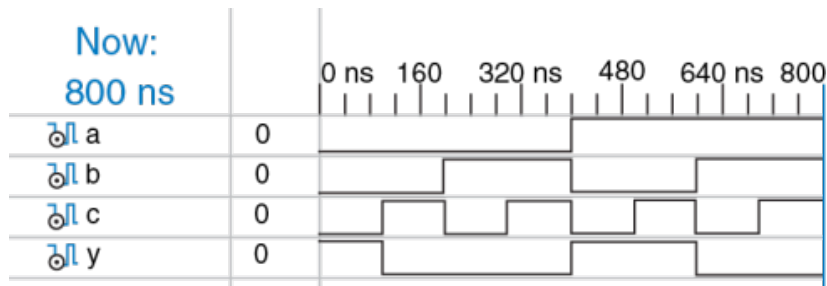
У сигналов в VHDL, в том числе входов и выходов, должен быть указан тип. Цифровые

сигналы стоит объявлять как STD_LOGIC. Сигналы этого типа принимают значения '0' или '1', а также плавающее и неопределенное значения. Тип STD_LOGIC определен в библиотеке IEEE.STD_LOGIC_1164, поэтому библиотеку объявлять обязательно. VHDL не определяет соотношение приоритетов операций AND и OR, поэтому при записи логических выражений нужно всегда использовать скобки.

Кроме описания аппаратуры и алгоритма обработки, две **основные цели HDL – логическая симуляция и синтез устройства** (по конкретной технологии). Во время симуляции на входы модуля подаются некоторые воздействия и проверяются выходы, чтобы убедиться, что модуль функционирует корректно. Во время синтеза текстовое описание модуля преобразуется в конструкцию на конкретной элементной базе. Логическая симуляция необходима для тестирования системы до того, как она будет выпущена.

** Ошибки в цифровой аппаратуре называют багами. Ясно, что устранение багов в цифровой системе очень важно. Тестирование системы в лаборатории весьма трудоёмко, так как наблюдать можно только сигналы, подключенные к контактам чипа, а то, что происходит внутри чипа, напрямую наблюдать невозможно. Исправление ошибок уже после того, как система была выпущена, может быть очень дорого и занимает несколько месяцев. Печально известный баг в команде деления с плавающей точкой (FDIV) в процессоре Pentium (Intel) стоил им 475 миллионов долларов.*

Рис. показывает графики симуляции сигналов предыдущего модуля sillyfunction, демонстрирующие, что модуль работает корректно. (Симуляция была проведена в программе



ModelSim, она имеет студенческую версию с возможностью бесплатной симуляции до 10 тыс. строк кода).

Рис. Графики сигналов: Y есть лог. 1, когда a, b, c есть 000, 100, или 101, как и указано в

логическом выражении.

Логический синтез преобразует код на HDL в нетлист, описывающий цифровую аппаратуру (т.е. логические элементы и соединяющие их проводники). Логический синтезатор может выполнять оптимизацию для сокращения количества необходимых элементов. Нетлист может быть текстовым файлом или нарисован в виде схемы, чтобы было легче визуализировать систему. **Рис. 4.2** показывает результаты синтеза модуля sillyfunction. Три трехвходовых элемента И упрощены в 2 двухвходовых элемента И, используя булеву алгебру. Описание схем на HDL напоминает программный код, однако надо помнить, что код предназначен для описания аппаратуры. **Битовые операторы** манипулируют однобитовыми сигналами или многоразрядными шинами.

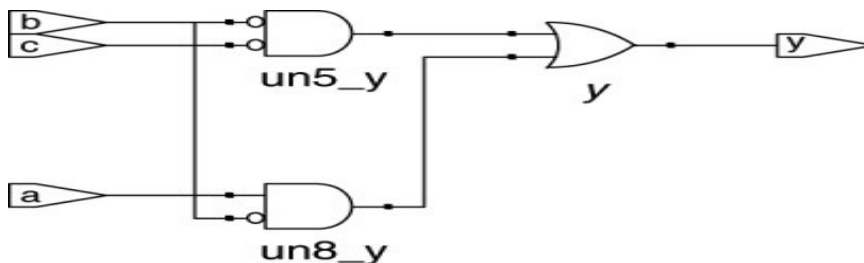


Рис. 4.2 Схема sillyfunction

ИНВЕРТОРЫ

SystemVerilog

```
module inv(input logic [3:0] a, output logic [3:0] y);
  assign y = ~a;
endmodule
```

a[3:0] представляет собой 4-битную шину. Биты, от старшего к младшему, записываются так: a[3], a[2], a[1] и a[0]. Такой порядок битов называется **little-endian**, т.к. младший бит имеет наименьший битовый номер. Мы могли бы назвать шину a[4:1], и тогда a[4] был бы старшим. Или мы могли бы написать a[0:3], и тогда порядок битов от старшего к младшему был бы следующим: a[0], a[1], a[2] и a[3]. Такой порядок битов называется **big-endian**.

VHDL

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity inv is
  port(a: in STD_LOGIC_VECTOR(3 downto 0);
        y: out STD_LOGIC_VECTOR(3 downto 0));
end;
architecture synth of inv is
  begin y <= not a;
end;
```

В VHDL для определения шин типа STD_LOGIC используется STD_LOGIC_VECTOR. STD_LOGIC_VECTOR (3 downto 0) представляет собой 4-битную шину. Биты от старшего к младшему: a(3), a(2), a(1) и a(0), little-endian, младший бит имеет наименьший битовый номер. Порядок следования разрядов шины является чисто условным. Будем постоянно использовать

порядок битов слева направо от старшего к младшему, $[N - 1:0]$ на языке SystemVerilog и $(N - 1 \text{ downto } 0)$ на языке VHDL для N -разрядной шины.

После каждого примера кода приводится схема, созданная из кода SystemVerilog инструментом синтеза Synplify Premier. Рис. 4.3 показывает, что модуль inv синтезируется в виде блока из 4 инверторов, обозначенных символом инвертора с надписью $y[3:0]$. Блок инверторов соединен с четырехбитными входной и выходной шинами. Подобная же аппаратная реализация получается из синтезированного VHDL-кода.

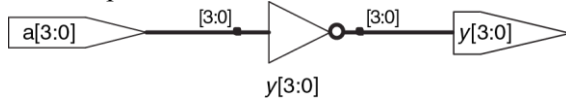


Рис. 4.3 Синтезированная схема модуля inv

ЛОГИЧЕСКИЕ ЭЛЕМЕНТЫ

SystemVerilog

```
module gates (input logic [3:0] a, b,
              output logic [3:0] y1, y2, y3, y4, y5);
  /*five different two-input logic gates acting on 4-bit busses */
  assign y1 = a & b; // AND
  assign y2 = a | b; // OR
  assign y3 = a ^ b; // XOR
  assign y4 = ~(a & b); // NAND
  assign y5 = ~(a | b); // NOR
endmodule
```

Символы \sim , \wedge и $|$ – это примеры операторов в языке SystemVerilog, тогда как a , b и $y1$ являются операндами. Комбинация операторов и операндов, таких как $a \& b$ или $\sim(a | b)$ называется выражением. Полная команда, такая как `assign y4 = $\sim(a \& b)$;` называется оператором.

Assign out = in1 op in2; называется оператором непрерывного присваивания. Он заканчивается точкой с запятой. Когда в операторе непрерывного присваивания входные значения справа от знака « $=$ » меняются, результат слева от знака « $=$ » вычисляется заново. Непрерывное присваивание описывает комбинационную логику.

Комментарии в языке SystemVerilog схожи с комментариями языков C или Java. Комментарии, начинающиеся с « $/*$ », могут занимать несколько строк, до следующего знака « $*/$ ». Комментарии, начинающиеся с « $//$ », продолжаются до конца строки. SystemVerilog чувствителен к регистру символов (прописным и строчным буквам). $y1$ и $Y1$ в SystemVerilog – это разные сигналы, однако, использование множества сигналов, отличающихся только регистром символов, вносит путаницу.

VHDL

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity gates is
  port(a, b: in STD_LOGIC_VECTOR (3 downto 0);
        y1, y2, y3, y4, y5: out STD_LOGIC_VECTOR (3 downto 0));
end;
architecture synth of gates is
begin
```

```
  -- five different two-input logic gates acting on 4-bit busses
  y1 <= a and b; y2 <= a or b; y3 <= a xor b; y4 <= a nand b; y5 <= a nor b; end;
```

НЕ, исключаящее ИЛИ и ИЛИ – это примеры операторов в языке VHDL, тогда как a , b и $y1$ являются операндами. Комбинация операторов и операндов, таких как $a \text{ and } b$ или $a \text{ nor } b$, называется выражением. Полная команда, например, `y4 = a nand b;` называется оператором. **out= in1 op in2;** называется оператором одновременного присваивания сигнала. Операторы присваивания в VHDL заканчиваются точкой с запятой. Когда в операторе одновременного присваивания сигнала входные значения справа от знака « $=$ » изменяются, результат слева от знака « $=$ » вычисляется заново. Таким образом, оператор одновременного присваивания сигнала описывает комбинационную логику.

Комментарии, начинающиеся с « $/*$ », могут занимать несколько строк до следующего знака « $*/$ ». Комментарии, начинающиеся с « $--$ », продолжаются до конца строки. VHDL не чувствителен к

регистру символов. В VHDL $y1$ и $Y1$ – это один и тот же сигнал. Однако, другие программы, открывающие ваш файл, могут оказаться чувствительны к регистру символов, что приводит к неприятным ошибкам, если вы смешиваете прописные и строчные буквы.

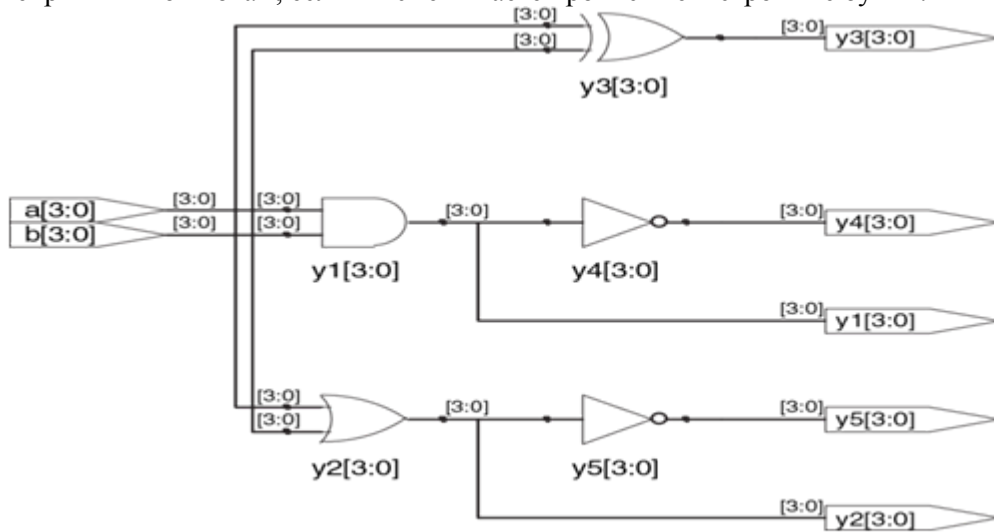


Рис. 4.4 Синтезированная схема модуля gates

Операторы сокращения соответствуют многовходовым элементам, работающим на одной шине.

Пример 4.4 описывает восьмивходовый вентиль И с входами a_7, a_6, \dots, a_0 . Аналогичные операторы сокращения существуют для вентилях ИЛИ, исключающее ИЛИ, И-НЕ, ИЛИ-НЕ и исключающее ИЛИ с инверсией. Многовходовый вентиль исключающее ИЛИ осуществляет функцию контроля четности, возвращая значение ИСТИНА, если нечетное количество входов имеют состояние ИСТИНА.

Пример 4.4 ВОСЬМИВХОДОВЫЙ ВЕНТИЛЬ «И»

SystemVerilog

```
module and8(input logic [7:0] a, output logic y);
assign y = &a; // &a is much easier to write than assign y = a[7] & a[6] & a[5] & a[4] & a[3] & a[2]
& a[1] & a[0];
endmodule
```

VHDL

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity and8 is
port(a: in STD_LOGIC_VECTOR (7 downto 0);
y: out STD_LOGIC);
end;
architecture synth of and8 is
begin
y <= and a;
-- and a is easier to write than y <= a(7) and a(6) and a(5) and a(4) and a(3) and a(2) and a(1) and
a(0);
end;
```

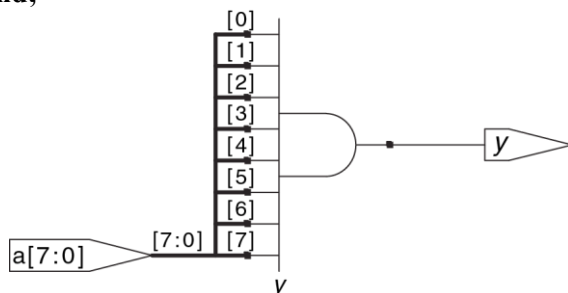


Рис.4.5 Синтезированная схема модуля and8

Тип **reg** может хранить значение и используется в процедурных блоках, срабатывающая по определенному событию - фронту тактового сигнала или начало симуляции. В процедурных блоках могут использоваться Си-подобные управляющие конструкции: *if... else; for ; do... while; case (аналог switch)*.

ПРИОРИТЕТ ОПЕРАТОРОВ

SystemVerilog

Операция	Значение
~	Побитовое отрицание (НЕ)
*, /, %	Умножение, деление, остаток
+, -	Сложение, вычитание
<<, >>	Сдвиг влево/вправо (лог. – без знака)
<<<, >>>	Арифметический сдвиг влево/вправо (не изменяет знак)
<, <=, >, >=	Сравнение на больше-меньше
Операция	Значение
==, !=	Сравнение на равенство
&, ~&	И, И-НЕ
^, ~^	Исключающее ИЛИ, исключающее ИЛИ-НЕ
, ~	ИЛИ, ИЛИ-НЕ
?:	Условный оператор

Система приоритета операторов для SystemVerilog подобна системам, принятым в других языках программирования. В частности, И имеет приоритет над ИЛИ. Можно пользоваться приоритетом операторов, чтобы исключить использование круглых скобок. `assign cout = g |p & cin;`

VHDL

Операция	Значение
not	НЕ
*, /, mod, rem	Умножение, деление, модуль, остаток
+, -	Сложение, вычитание
rol, ror, srl, sll	Циклический сдвиг влево/вправо Логический сдвиг влево/вправо
<, <=, >, >=	Сравнение на больше-меньше
=, /=	Сравнение на равенство
and, or, nand, nor, xor, xnor	Логические операции

В VHDL умножение имеет приоритет над сложением. Однако, в отличие от SystemVerilog, здесь все логические операторы (and, or и т.д.) имеют одинаковый приоритет. Поэтому скобки необходимы; в противном случае `cout = g or p and cin` будет интерпретироваться слева направо как `cout = (g or p) and cin`.

Числа указываются в двоичной, восьмеричной, десятичной или шестнадцатеричной системе счисления (основания 2, 8, 10 и 16 соответственно). Размер, т.е. количество бит, может быть также указан, свободные разряды заполняются нулями.

ЧИСЛА SystemVerilog

Запись	Число бит	Основание	Значение	Представление
3'b101	3	2	5	101
'b11	?	2	3	0000011
8'b11	8	2	3	00000011
8'b1010_1011	8	2	171	10101011
3'd6	3	10	6	110
6'o42	6	8	34	100010
8'hAB	8	16	171	10101011
42	?	10	42	000101010

Формат для объявления констант – **N'Bvalue**, где N – размер в битах, B – буква, указывающая на основание и value – значение. Например, 9'h25 определяет 9-битное число со значением $25_{16} = 37_{10} = 000100101_2$. SystemVerilog поддерживает 'b для основания 2, 'o – для основания 8, 'd – для основания 10 и 'h – для основания 16. Если основание опущено, то по умолчанию оно

равно 10. Если не указан размер, то предполагается, что число имеет столько же бит, сколько и выражение, в котором оно используется. Недостающие старшие разряды дополняются нулями автоматически до полного размера. Например, если w – 6-битная шина, то assign w = 'b11 присваивает w значение 000011. Лучшей практикой является явное указание размера. Исключением является то, что '0 и '1 служат конструкциями SystemVerilog для заполнения шины нулями или единицами соответственно.

VHDL

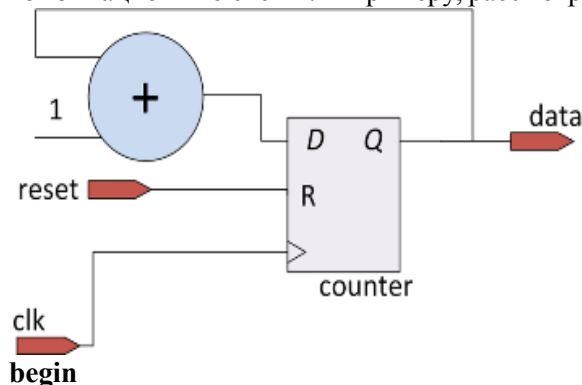
Запись	Число бит	Основание	Значение	Представление
3B"101"	3	2	5	101
B"11"	2	2	3	11
8B"11"	8	2	3	00000011
8B"1010_1011"	8	2	171	10101011
3D"6"	3	10	6	110
6O"42"	6	8	34	100010
8X"AB"	8	16	171	10101011
"101"	3	2	5	101
B"101"	3	2	5	101
X"AB"	8	16	171	10101011

В VHDL числа STD_LOGIC записываются в бинарном коде и заключаются в одинарные кавычки: '0' and '1' указывают на логические уровни 0 и 1. Формат объявления констант типа STD_LOGIC_VECTOR следующий: NB"value", где N – размер в битах, B – буква, указывающая на основание и value – значение. Например, 9X"25" определяет 9-битное число со значением $25_{16} = 37_{10} = 000100101_2$. VHDL 2008

поддерживает B для основания 2, O – для основания 8, D – для основания 10 и X – для основания 16.

Если основание опущено, то по умолчанию оно равно 10. Если размер не указан, то предполагается, что число имеет размер, соответствующий числу битов значения. По состоянию на октябрь 2011 SynplifyPremier от Synopsys не поддерживает указание размера. others = '0' и others = '1' – конструкции VHDL с заполнением всех битов нулями или единицами соответственно.

Процедурные блоки могут моделировать как синхронные схемы (схемы с памятью), так и комбинационные схемы. К примеру, рассмотрим схему двоичного счетчика:



```
module counter
(
  input clk, reset,
  output [7:0] data
);
  reg [7:0] counter;
  assign data = counter;
  always @(posedge clk) // процедурный блок
    сбрасывается по положительному фронту clk
```

```
begin
  if (reset) counter = 0;
  else
```

```

counter = counter + 1;
end
endmodule

```

После названия модуля описываются сигналы (...), необходимые для реализации аппаратуры, потом аппаратура и алгоритм после begin до end.

Строка **always @(posedge clk)** называется списком чувствительности. Она определяет события, по которым выполняется процедурный блок - по каждому положительному фронту (positive edge) сигнала синхронизации. Таким образом, блок моделирует логику работы счетчика, сигнал counter будет просинтезирован как регистр.

Процедурные блоки могут моделировать и комбинационные схемы. К примеру, следующий код просинтезируется в комбинационный сумматор:

```

wire [7:0] a,b;
reg [7:0] res;

```

```

always @*
begin
res = a+b;
end

```

Здесь список чувствительности **always @*** означает, что процедурный блок выполняется каждый раз, когда изменяются значения сигналов в правой части операций присваивания - когда изменяются сигналы a и b. Эквивалентном этой строчке, будет следующий список чувствительности: - **always @(a or b)** // блок срабатывает каждый раз, когда меняется a или b,

значения всех изменяемых в блоке сигналов, должны полностью определяться сигналами в списке чувствительности.

Процесс в VHDL определяет независимую повторяющуюся последовательность операторов и представляет поведение некоторой части проекта. Для определения процесса используется оператор **process**, который состоит из объявлений (после ключевого слова process) и операторной части, которая начинается после слова **begin**. В объявлении процесса допускается создавать переменные, в то время как объявлять сигналы в этой части нельзя.

Предложения в операторной части **process выполняются строго последовательно** (в отличие от параллельного назначения потоковой формы описания и конкретизации компонентов в структурном стиле) и включают, подобно языкам высокого уровня, **операторы присваивания, условные операторы и операторы циклов.**

Оператор "<=>" здесь используется как **оператор назначения сигнала** (в отличие от потоковой формы, где он используется как оператор параллельного присваивания) и также выполняется последовательно.

- вариант А -- процесс запускается при изменении хотя бы одного из сигналов X1, X2

```

PROCESS (X1, X2)

```

```

BEGIN

```

```

Y <= not X1 and X2;

```

```

Z <= Y or X3;

```

```

END PROCESS;

```

- вариант Б — с помощью оператора WAIT ON задержки процесса, который может располагаться в любом месте операторной части процесса. WAIT ON <список сигналов> UNTIL <условие> FOR <время>;

```

PROCESS (X1, X2)

```

```

BEGIN

```

```

Y <= not X1 and X2;

```

```

Z <= Y or X3;

```

```

WAIT ON X1, X2;

```

```

END PROCESS;

```

```

PROCESS (X1, X2)

```

```

BEGIN

```

```

WAIT ON X1, X2;

```

```

Y <= not X1 and X2;

```

```

Z <= Y or X3;

```

```

END PROCESS;

```

--или

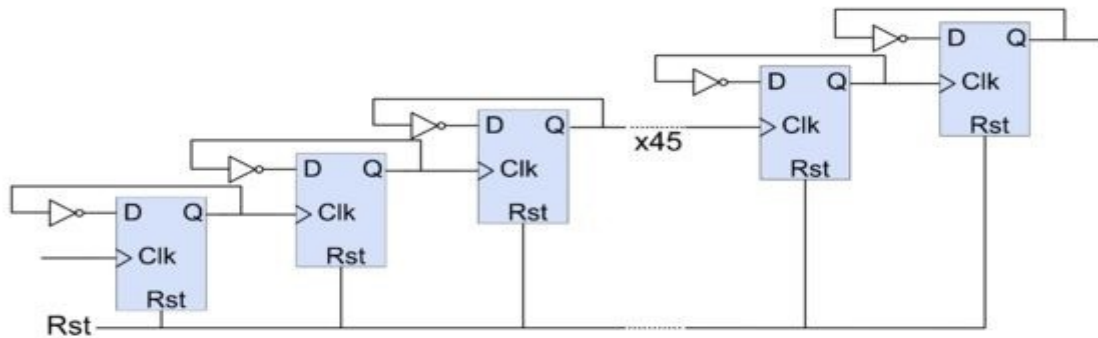
Например, для задания условий срабатывания схемы по фронту или срезу сигнала C с задержкой на t ns можно записать следующим образом: WAIT ON C UNTIL C='0' for t ns;

Процессы в VHDL могут быть **вложенными**, а также пассивными — в которых (явно или опосредовано) не производится назначение сигнала ни в одном из его операторов.

Циклы в Verilog. Что делать, если потребуется создать много (например, 50) D-триггеров, которые должны быть соединены между собой в соответствии со схемой делителя частоты? Нужно сделать 50 экземпляров этого модуля и соединить входы тактового сигнала каждого модуля с выходом предыдущего, подвести сигнал Сброс (Reset) ко всем модулям и вывод D через инверсию соединить с выходом Q.

Вручную писать столько экземпляров (50 штук по 6 строк в каждом – 300 строк!) – не практично. Поэтому в Verilog для генерации большого числа одинаковых модулей можно

воспользоваться циклом счетчика, который является чем-то вроде **цикла for**. Тактовый вход первого триггера будет соединен со входом модуля верхнего уровня, а выход последнего триггера с выходом верхнего уровня (светодиодом). Подключение остальных D-триггеров друг к другу является шаблонным.



Все входы триггеров, кроме первого должны зависеть от выхода предыдущего, поэтому, все эти сигналы можно объединить в одну шину, что равносильно созданию массива в C/C++. D-вход каждого триггера соединен с выходом этого триггера, имеется и общий сброс. Инвертирование сигнала, идущего на D-вход сделаем позже. Поскольку шина просто переносит сигнал от источника к выходу, то это будет **wire**.

```
wire in[49:0] //50-битная шина для входов триггеров
wire out[49:0] //50-битная шина для выходов триггеров
```

Экземпляр для первого триггера

```
dff dff0(
  .clk(clk),
  .rst(rst),
  .D(in[0]),
  .Q(out[0])
);
```

Сосредоточимся на создании цикла. Хотя он похож на цикл в C/C++, все же у него имеются некоторые отличия. Сначала необходимо **создать переменную счетчика**. В Verilog такой тип переменной называется «genvar» и используется для объявления переменной - **genvar y**;

Цикл начинается с ключевого слова «generate» и заканчивается «endgenerate». Объявление переменных с помощью genvar должно производиться вне тела цикла.

```
genvar y;
generate
...
endgenerate
```

После ключевого слова «generate» начинается область цикла. Скобки, как в C/C++, не нужны, но тело цикла должно быть заключено между ключевыми словами «begin» и «end». Verilog не поддерживает специальные операции вроде инкремента $y++$, поэтому $y = y + 1$. Считаем, что первый (нулевой) D-триггер уже создан, поэтому диапазон будет от 1 до 49 (<50). При создании цикла после «begin» ставится двоеточие и указывается имя начинаемого процесса. Пусть это будет «dff_generation»:

```
genvar y;
generate
for(y = 1; y < 50; y = y + 1 )
begin : dff_generation
...
end
endgenerate
```

Создание модели экземпляра D-триггера. В качестве подводимых к модулю сигналов будет использоваться переменная, объявленная как «genvar».

```
genvar y;
generate
```

```

for(y = 1; y < 50; y = y + 1 )
begin : dff_generation
//имя экземпляра не имеет значения в дальнейшем
dff dff_insts (
.clk(out[y-1]), //clk триггера под номером «у» является входом выхода «у-1»
.rst(rst), //общий сброс
.D(in[y]), //D-вход
.Q(out[y]) //выход
);
end
endgenerate

```

Не хватает инверсии на входе порта «.D». Ее можно применить ко всей шине, а не к отдельным сигналам:

```

assign in = ~in;

```

Вся современная схемотехника разделяется на две большие области: **аналоговую и цифровую**. Аналоговая схемотехника характеризуется максимальным быстродействием, малым потреблением энергии и малой стабильностью параметров. Цифровая схемотехника обладает прекрасной повторяемостью параметров. В результате в ряде устройств потребление цифровых модулей оказалось сравнимым и даже меньше потребления аналоговых схем, реализующих те же функции. Основные направления развития цифровых микросхем в настоящее время приведены на рисунке.

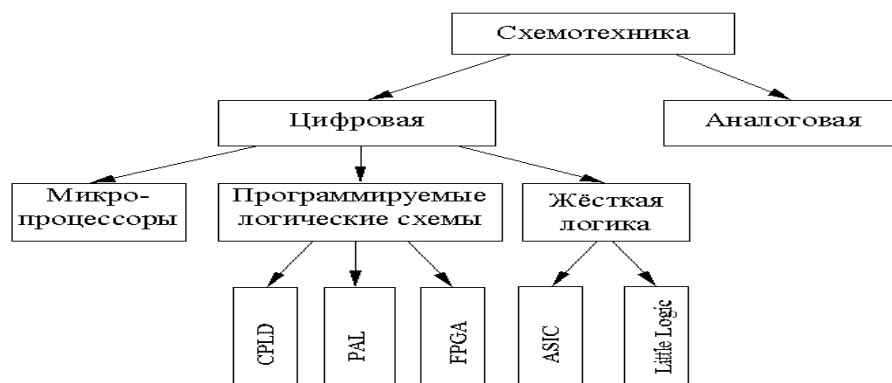


Рис. Классификация видов цифровых микросхем

Аппаратная и аппаратно-программная реализация на ПЛИС.

Программируемая логическая интегральная схема (**ПЛИС**, programmable logic device, **PLD**) — электронный компонент, используемый для создания цифровых интегральных схем, **архитектура и логика работы** которого не определяется при изготовлении, а **задаётся посредством программирования** ([проектирования](#)).

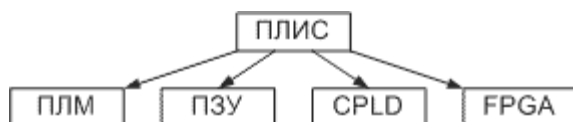


Рис. Классификация программируемых логических интегральных схем (ПЛИС)

Программируемые логические матрицы (**ПЛМ**) реализовали хорошо известные принципы создания цифровой комбинационной схемы по таблице истинности, обычно с относительно малым количеством сигналов на выходе и большим количеством входных сигналов, либо хорошо минимизирующиеся логические функции. **ПЗУ** применялись для создания комбинационных схем с малым количеством входов. При росте количества входов сложность внутреннего устройства ПЗУ и его цена резко возрастали (по квадратичному закону).

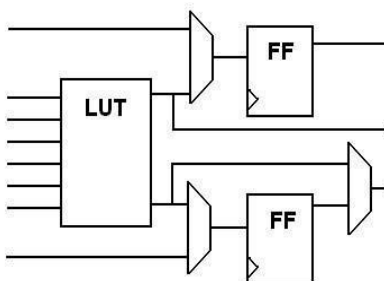
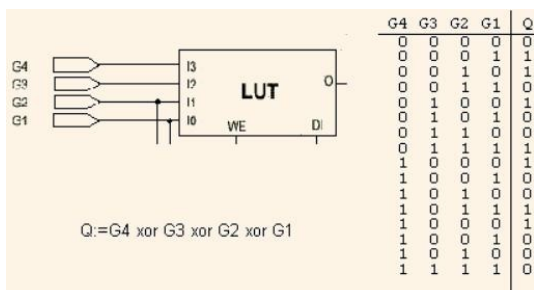
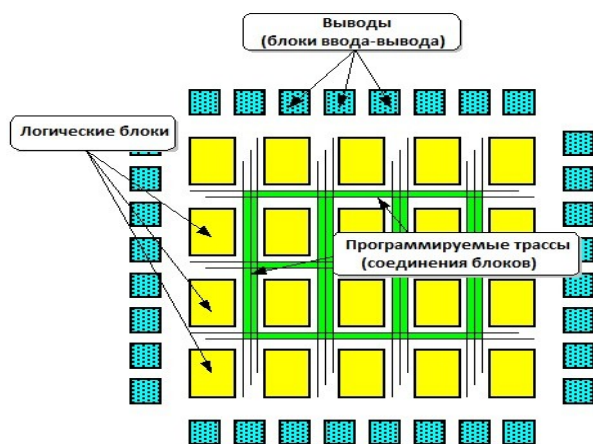
CPLD попроще (Complex Programmed Logic Device) и **FPGA** подороже (Field Programmed Gate Array) отличаются возможностями загрузки конфигурации, строением блоков, количеством триггеров и дополнительными "непростыми" IP-блоками - умножители, блоковая память,

интерфейсные схемы (Ethernet, PCI-express...) и даже процессорные ядра PowerPC. Серия Zinq от Xilinx содержит ядра ARM.

Логика работы ПЛИС определяется не на фабрике изготовителем микросхемы, а путем дополнительного программирования (в полевых условиях, field-programmable) с помощью специальных средств: **программаторов и программного обеспечения**.

В ПЛИС есть **проводные трассы и магистрали, переключатели, входы, выходы, логические блоки, ячейки, блоки памяти, процессоры**. Трасса — проводник электричества между блоками, трасса для тактирования привязана к определенным ножкам, через которые проводят тактовую частоту. **Переключатели** — отдельные полевые транзисторы, осуществляющие соединение блоков, трасс, ячеек и пр. при подаче сигнала управления (определяются прошивкой). **Блоки** — отдельные места в плате, состоящие из ячеек, служат для запоминания информации, умножения, сложения и логических операций над сигналами. **Ячейки** — группы от нескольких единиц до нескольких десятков транзисторов, выполняющих функции хранения и переключения. **Выводы** (ножки микросхемы) — через них происходит обмен ПЛИС с окружающим миром. Есть ножки специального назначения, а так же ножки, назначение которых устанавливаются пользователем в программе. И их гораздо больше, чем у микроконтроллера. **Тактовый генератор** с возможностью перестройки частоты вырабатывает тактовые импульсы, на которых основывается большая часть работы ПЛИС.

Для программирования используются программатор и **отладочная среда (IDE)**, позволяющие задать желаемую структуру цифрового устройства в виде принципиальной электрической схемы или программы на специальных языках описания аппаратуры: Verilog, VHDL, AHDL. **Компилятор** конкретной среды разработки (**IDE**), зная внутреннее устройство ПЛИС, пытается разместить требуемую схему по имеющимся конфигурируемым логическим блокам и соединить эти блоки с помощью имеющихся программируемых электронных связей.



LUT исключаяющее ИЛИ на четыре входа элемента (LE)

Обобщенная структура логического

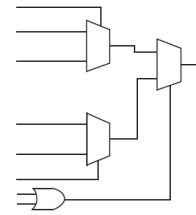
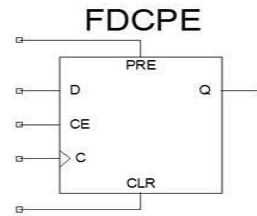
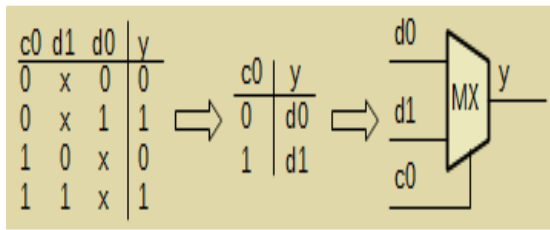
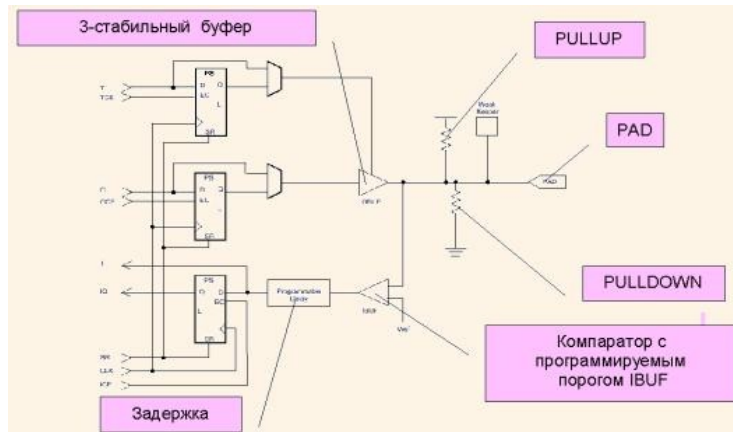


Таблица истинности и обозначение мультиплексора 2:1 Триггер Logic Module однократного программирования

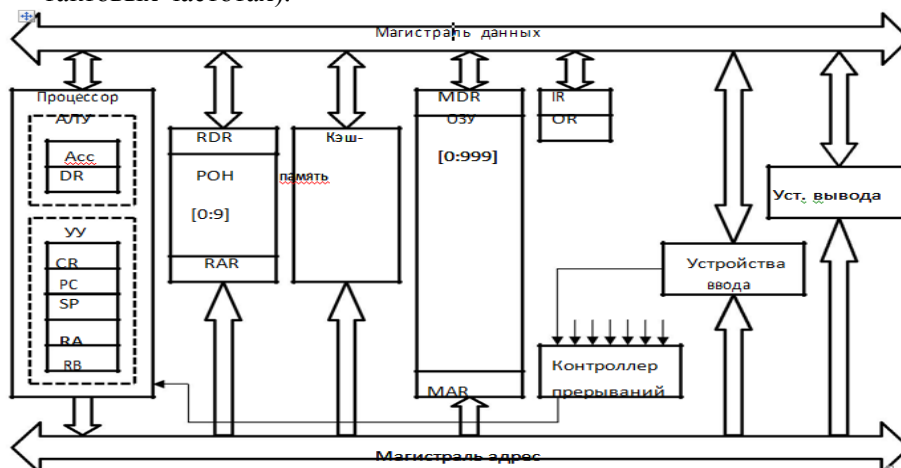


а) Согласование выхода с открытым коллектором (стоком) для систем с различными уровнями логики (ТТЛ, КМОП, шины PCI и др.)

К **аппаратным блокам**, размещаемым в FPGA, относятся:

- блоки синхронной статической двупортовой памяти BRAM;
- блоки цифровой обработки сигналов «умножение с накоплением» XtremeDSP;
- формирователи тактовых сигналов MMCM (Multi-Mode Clock Managers) и PLL (Phase-Locked Loop);
- скоростные последовательные приемопередатчики (MGT- Multi-GigabitTransceivers);
- контроллеры Ethernet MAC (Virtex-4, 5, 6);
- контроллеры PCI Express endpoint;
- процессорные ядра PowerPC (Virtex-II Pro, Virtex-4 FX, Virtex-5 FXT), ARM (Zynq-7000).

Отличие ПЛИС и микроконтроллера. Контроллер заточен под выполнение длинных цепочек команд, их циклического повторения, переключения с одной цепочки на другую и т.д., а ПЛИС - под выполнение большого количества простых логических операций сразу (и даже на разных тактовых частотах).



Архитектура микроконтроллера устоявшаяся, облегчающая процесс программирования, связи между блоками постоянные (а не перепрошиваемые). При прошивке изменяется только постоянная память, на которую опирается вся работа МК.

ПЛИС прошивается на уровне железа (архитектуры). Сигналы проходят через сложные цепочки ПЗУ, мультиплексоров, триггеров. Чтобы реализовать программу на **ПЛИС**, нужно отследить каждый сигнал по каждому проводнику, записать некоторые сигналы в ячейки памяти или отправить на запуск алгоритма (программы на языке **Verilog, VHDL**), позаботиться, чтобы набор сигналов вышел на определенную выходную ножку, которая является адресной.

Время на программирование условного робота на МК и ПЛИС будет отличаться в разы, однако робот, работающий на ПЛИС, будет гораздо шустрее, точнее и проворнее.

Особенности использования ПЛИС с архитектурой FPGA для цифровой обработки сигналов.

ПЛИС давно используются для построения высокопроизводительных систем ЦОС. Наиболее эффективны для реализации в ПЛИС методы и алгоритмы, использующие **параллельную обработку** потоков данных. Некоторые сигнальные процессоры (DSP) также допускают выполнение двух или четырех операций "умножение с накоплением" одновременно, тем не менее, при расчете фильтров высокого порядка общая скорость их вычислений существенно снижается. FPGA с большим числом блоков вычислений вполне может обеспечить **однотактное исполнение всех операций**, используя параллельную реализацию. За счет архитектуры ПЛИС выигрывает в быстродействии и более широких возможностях обработки, МК выигрывает в простоте написания алгоритмов.

Основные преимущества этого направления:

- 1) конфигурируемая матрица ПЛИС позволяет создавать параллельные устройства (рис. 1);

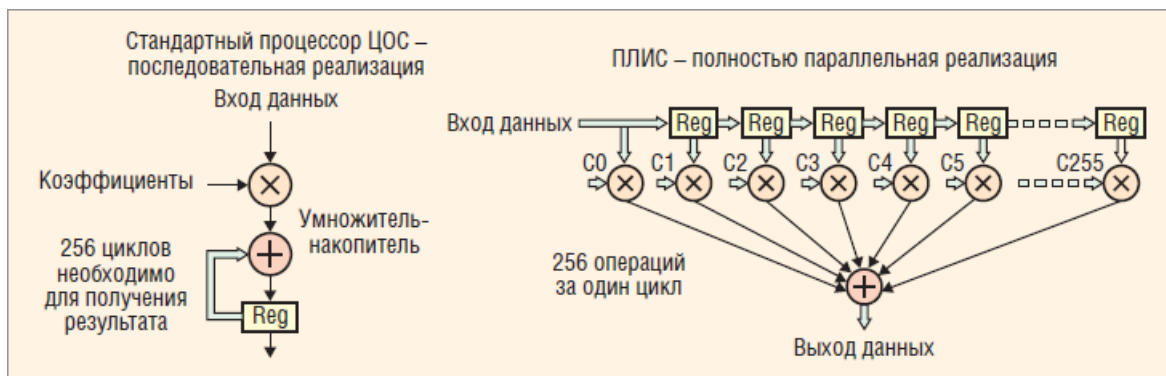


Рис. 1. Параллельная и последовательная реализации

- 2) возможность выбора между алгоритмами и ресурсами (рис. 2);

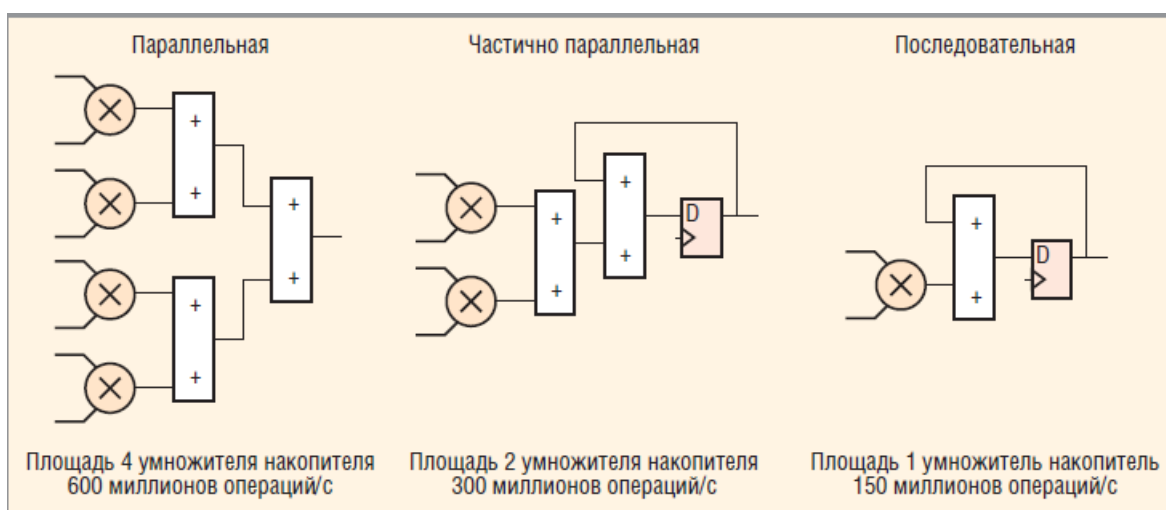


Рис. 2. Возможность выбора между быстродействием и ресурсами

- 3) низкая стоимость современных приборов. Так, например, кристалл XC6SLX150T (семейство Spartan 6) содержит:

- 180 умножителей 18x18,
- 256 блоков памяти по 18 Кбит,

- 184 000 логических ячеек (LUT) и столько же триггеров,
- четыре контроллера памяти,
- восемь блоков трансиверов со скоростью до 3,125 Гбит/с.
- обеспечивает производительность более 30 млрд операций умножения - сложения в секунду и продаётся за 180 долл. США.

4) автоматизация - реализация проекта с верхнего уровня (MATLAB) до вентильного представления (рис. 3) в рамках **модельно-ориентированного программирования**. Вместо физических прототипов при традиционном проектировании программируемых систем, в модельно-ориентированном применяются **исполняемые модели**, которые используются на всех этапах разработки, вплоть до автоматической генерации кода и его верификации. С помощью HDL Coder и HDL Verifier можно осуществлять моделирование, симуляцию и анализ ваших алгоритмов в MATLAB и Simulink, генерацию оптимизированного под конечное устройство, либо независимого HDL-кода (для конфигурирования конкретной ПЛИС в среде производителя из MATLAB и Simulink), верификацию разработки на соответствие спецификациям системного уровня.

Таблица 1. Сравнение классических методов проектирования ПЛИС с модельно-ориентированным

Традиционные методы	Модельно-ориентированный
Текстовый ввод препятствует быстрой реализации по заданным требованиям	Графический ввод на основе готовых параметрических библиотек
Моделирование неполное и дорогое	Быстрое моделирование сразу во временной и частотной области
Длительные итерации для выбора разрядности с фиксированной точкой	Единая среда моделирования проектов с плавающей и фиксированной точкой
Ручной ввод кода вносит дополнительные ошибки из-за человеческого фактора	Автоматическая генерация VHDL-описания и тестов
При традиционном тестировании ошибки выявляются слишком поздно	Быстрый возврат к верхнему уровню проекта в случае изменения спецификации

Модельно-ориентированный подход, реализованный впервые в пакете **System Generator** (Xilinx), имеет преимущества на каждом этапе проектирования. Поэтому, другие компании (**Mathworks**) пытаются выйти

на рынок с новыми продуктами – **HDL Coder**.

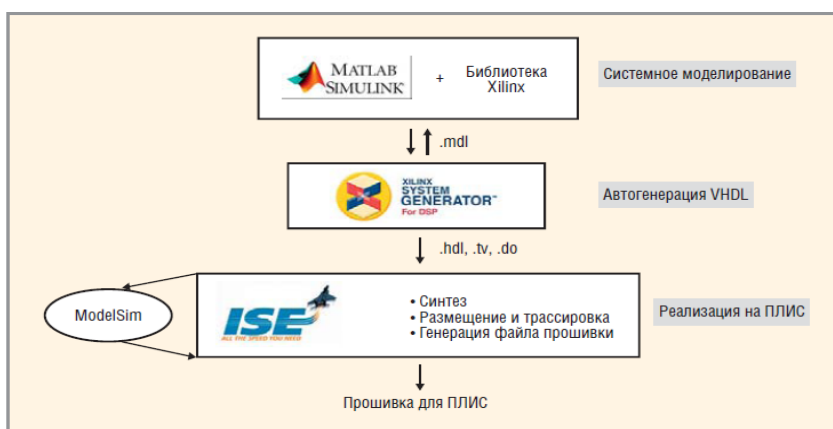


Рис. 3. Методология проектирования устройств ЦОС

Пакет Matlab стал популярным инструментом разработки устройств ЦОС. Библиотеки **DSP system toolbox** и **Communication toolbox**, содержат тысячи готовых элементов для ЦОС и телекоммуникаций. Одних только примеров модуляторов, демодуляторов, беспроводных каналов передачи – сотни. Это значительно ускоряет разработку проекта на

высоком уровне. Кроме того, **библиотечные блоки могут быть извлечены в виде исходных текстов на языке Си**.

Список доступных IP блоков:

- фильтры КИХ, фильтры Хогенауэра (CIC), преобразование Фурье (длиной до 65 536 точек, в реальном времени или оптимизированные по ресурсам), ОЗУ, двухпортовое ОЗУ, FIFO, ПЗУ, синтезатор частоты, умножитель, комплексный умножитель, делитель, арктангенс, натуральный логарифм, квадратный корень из суммы квадратов, микроконтроллер PicoBlaze, блок для собственных функций на сокращенном m-языке; - **блоки с плавающей запятой**: преобразователь Фурье, умножитель, делитель, натуральный логарифм, квадратный корень; лицензионные (платные) блоки – перемежитель-деперемежитель, сверточный кодер- декодер, кодер-декодер Риди-Соломона, декодер Витерби.

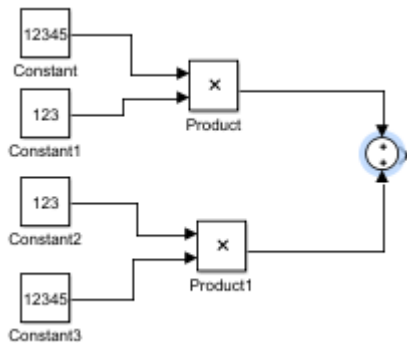
ModelSim - среда для симуляции и отладки программ на HDL языках, изучения языка VHDL. Она подходит для разработки описаний алгоритмов работы цифровых устройств, при этом она может быть подключена к системам проектирования на ПЛИС и использована как отладчик,

вместо встроенных симуляторов. Бесплатную студенческую версию ModelSim PE можно скачать с сайта www.model.com.

ПЛИС не оправдывают свое применение в случае повторения широко распространенных процессорных архитектур или однопоточных вычислений. Преимущества ПЛИС в системах ЦОС проявляются только при реализации **массово-параллельных вычислительных архитектур**, где наиболее полно используется высокая суммарная пропускная способность памяти FPGA, блоков цифровой обработки сигналов и скоростных последовательных приемопередатчиков.

Высокоуровневое программирование. «Два умножителя и сумматор» - решение задачи хорошо подходит в качестве теста. Функция принимает 4 параметра, перемножает первый со вторым, третий с четвертым и складывает результаты умножения.

HDL coder от Mathworks



m-скрипт выглядит примерно следующим образом:

```
function [out] = TwoMultAdd(a,b,c,d)
    out = (a*b)+(c*d);
end
```

Во- первых, нужно задать тип и диапазон входных значений. В FPGA нет привычных char, int, float, double. Разрядность числа может быть любой, логично выбирать ее, исходя из диапазона входных значений, который планируется использовать. MATLAB проводит проверку типов переменных, их значений и сам подбирает правильные разрядности для шин и регистров, — это

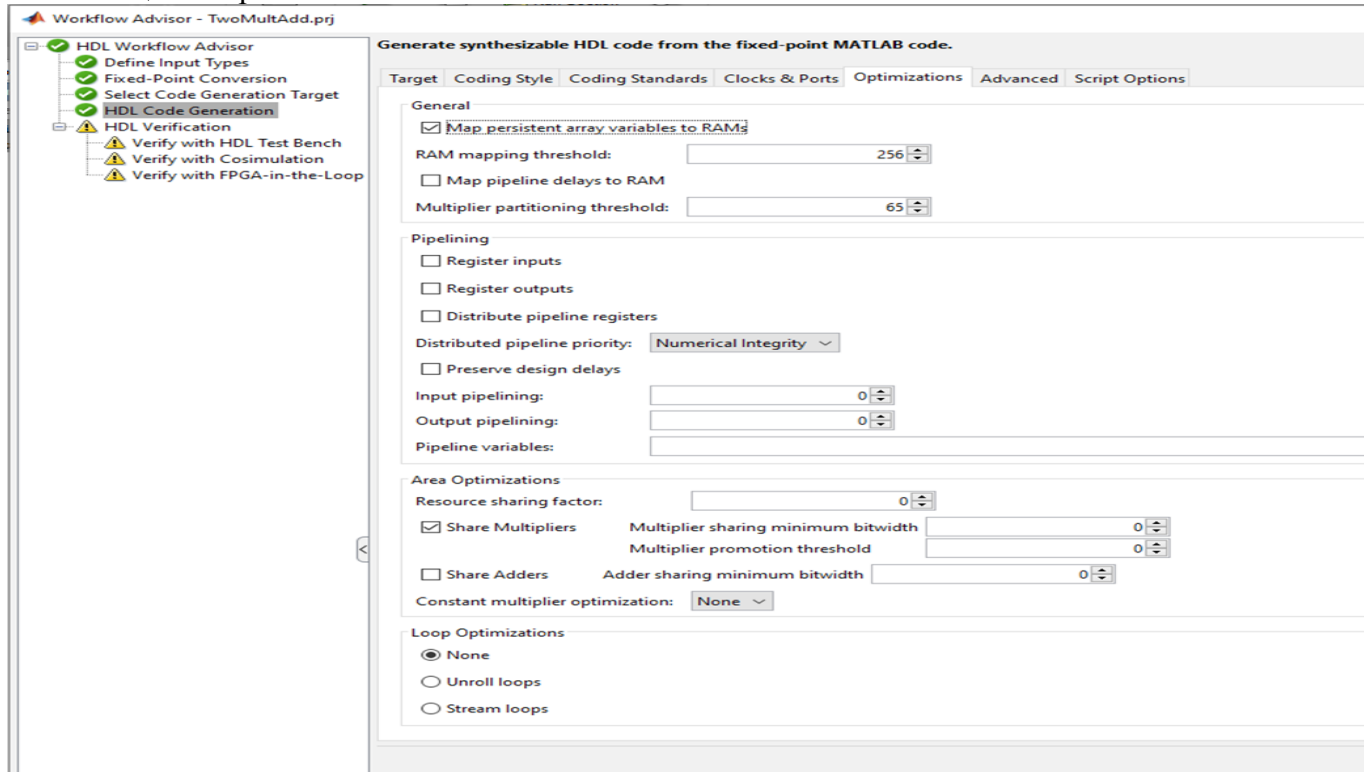
действительно удобно.

Fixed-Point Designer предоставляет доступ к типам данных и инструментам для проектирования алгоритмов, работающих **в арифметике с фиксированной точкой**, в виде кода MATLAB, моделей Simulink и диаграмм Stateflow. Инструмент автоматически подбирает такие характеристики переменных, заданных в арифметике с фиксированной точкой, как **длина слова и метод округления**. Все это можно также задать вручную. Симуляция работы системы позволяет наблюдать влияние, вносимое в систему, ограниченностью диапазона и точности. Инструмент поддерживает работу с длиной слова от 1 до 128 бит. Fixed-Point Designer позволяет переводить алгоритмы из арифметики с плавающей точкой в арифметику с фиксированной точкой – рис. 1

Variable	Type	Sim Min	Sim Max	Static Min	Static Max	Whole Num...	Proposed T
Input							
a	double			0	255	Yes	numericty
b	double			0	255	Yes	numericty
c	double			0	255	Yes	numericty
d	double			0	255	Yes	numericty
Output							
out	double			0	131071	Yes	numericty

В HDL Code Generation есть несколько вкладок, где можно выбрать язык, в который

произойдет конвертация (Verilog или VHDL); стиль кода; названия сигналов, наличие оптимизации – Optimization.



Нажимаем кнопку Run и получаем следующий код (verilog): автоматическая генерация кода сумматора-умножителя

``timescale 1 ns / 1 ps /* `timescale 1ns/ps` - это директива, которая определяет единицу модельного времени (первый параметр, 1 нс) и точность модельного времени (второй параметр, 1 пс). Для FPGA эта директива не важна, поскольку в ней нельзя задавать задержки вентилей или манипулировать с временными параметрами, но при симуляции схемы эта директива важна*/.

```
module TwoMultAdd_fixpt
    (
        a,
        b,
        c,
        d,
        out);

    input  [7:0] a; // ufix8
    input  [7:0] b; // ufix8
    input  [7:0] c; // ufix8
    input  [7:0] d; // ufix8
    output [16:0] out; // ufix17

    wire [15:0] TwoMultAdd_fixpt_mul_temp; // ufix16
    wire [16:0] TwoMultAdd_fixpt_2; // ufix17
    wire [15:0] TwoMultAdd_fixpt_mul_temp_1; // ufix16
    wire [16:0] TwoMultAdd_fixpt_3; // ufix17

    //HDL code generation from MATLAB function: TwoMultAdd_fixpt //%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    //
    %
    //          Generated by MATLAB 9.2 and Fixed-Point Designer 5.4
    % //%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %

    assign TwoMultAdd_fixpt_mul_temp = a * b;
    assign TwoMultAdd_fixpt_2 = {1'b0, TwoMultAdd_fixpt_mul_temp};
    assign TwoMultAdd_fixpt_mul_temp_1 = c * d;
```

```

assign TwoMultAdd_fixpt_3 = {1'b0, TwoMultAdd_fixpt_mul_temp_1};
assign out = TwoMultAdd_fixpt_2 + TwoMultAdd_fixpt_3;

endmodule // TwoMultAdd_fixpt

```

MATLAB понимает, что запись всего выражения в одну строку на Verilog — плохая практика. Создает отдельные [wire](#) для умножителя и сумматора, придаться особо не к чему. Так получилось, что отсутствует описание регистров, потому, что мы не просили об этом HDL-coder, а все поля в настройках оставили в значениях по умолчанию. Вот что из такого кода синтезирует Quartus.

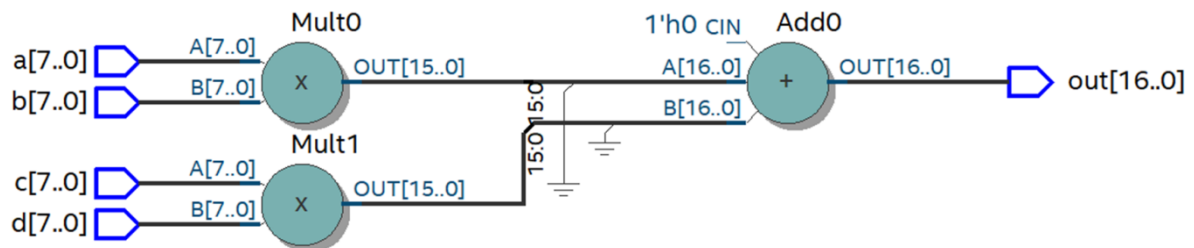
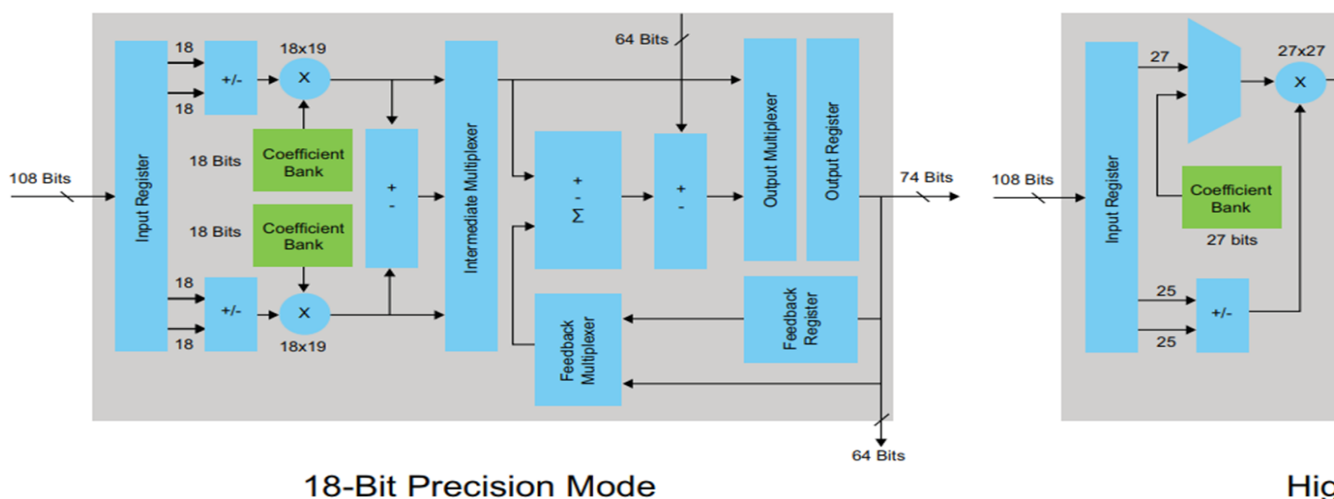


Рисунок 3.

Для синтеза выбрано семейство FPGA Cyclone V (Intel FPGA, Altera), где для реализации арифметических действий используются специальные DSP блоки со встроенными сумматорами и умножителями. Выглядит DSP блок на рис. 4.



DSP блок имеет **входные и выходные регистры**. Как поступать с входным и выходным регистром - решать разработчику, исходя из требований к **задержке** (latency) и необходимой **максимальной частоте**. В схеме принято решение использовать только выходной регистр. Для того, чтобы этот регистр был описан в коде, генерируемом HDL-coder'ом, во вкладке Options в HDL coder необходимо поставить галочку напротив Register output и повторно запустить генерацию. Получается следующий код:

```

`timescale 1 ns / 1 ns // Повтор автоматической генерации кода с применением
выходного регистра

module TwoMultAdd_fixpt
    (clk,
     reset,
     clke_ena_i,
     a,
     b,
     c,

```

```

        d,
        clke_ena_o,
        out);

input    clk;
input    reset;
input    clke_ena_i;
input    [7:0] a; // ufix8
input    [7:0] b; // ufix8
input    [7:0] c; // ufix8
input    [7:0] d; // ufix8
output   clke_ena_o;
output   [16:0] out; // ufix17

wire enb;
wire [16:0] out_1; // ufix17
wire [15:0] TwoMultAdd_fixpt_mul_temp; // ufix16
wire [16:0] TwoMultAdd_fixpt_2; // ufix17
wire [15:0] TwoMultAdd_fixpt_mul_temp_1; // ufix16
wire [16:0] TwoMultAdd_fixpt_3; // ufix17
reg [16:0] out_2; // ufix17

//HDL code generation from MATLAB function: TwoMultAdd_fixpt
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
//
%
//          Generated by MATLAB 9.2 and Fixed-Point Designer 5.4
%
//
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
assign TwoMultAdd_fixpt_mul_temp = a * b;
assign TwoMultAdd_fixpt_2 = {1'b0, TwoMultAdd_fixpt_mul_temp};
assign TwoMultAdd_fixpt_mul_temp_1 = c * d;
assign TwoMultAdd_fixpt_3 = {1'b0, TwoMultAdd_fixpt_mul_temp_1};
assign out_1 = TwoMultAdd_fixpt_2 + TwoMultAdd_fixpt_3;

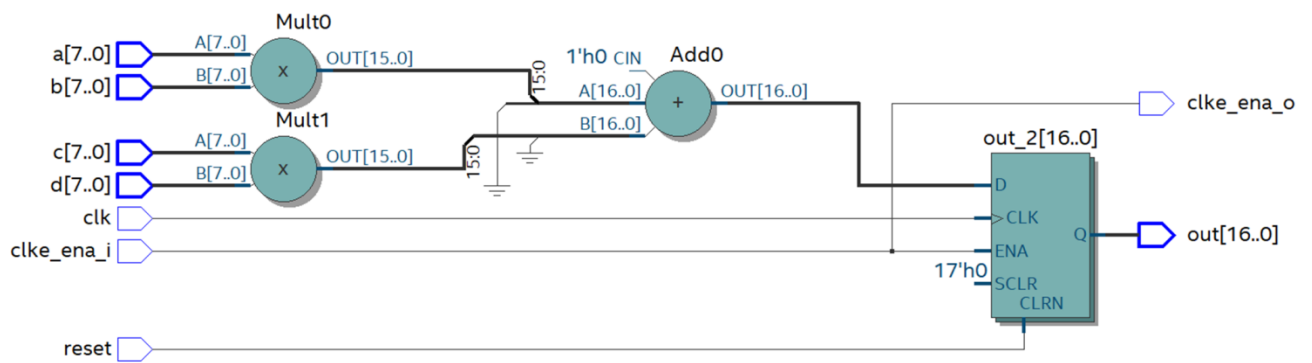
assign enb = clke_ena_i;

always @(posedge clk or posedge reset)
begin : out_reg_process
    if (reset == 1'b1) begin
        out_2 <= 17'b000000000000000000;
    end
    else begin
        if (enb) begin
            out_2 <= out_1;
        end
    end
end

assign clke_ena_o = clke_ena_i;
assign out = out_2;
endmodule // TwoMultAdd_fixpt

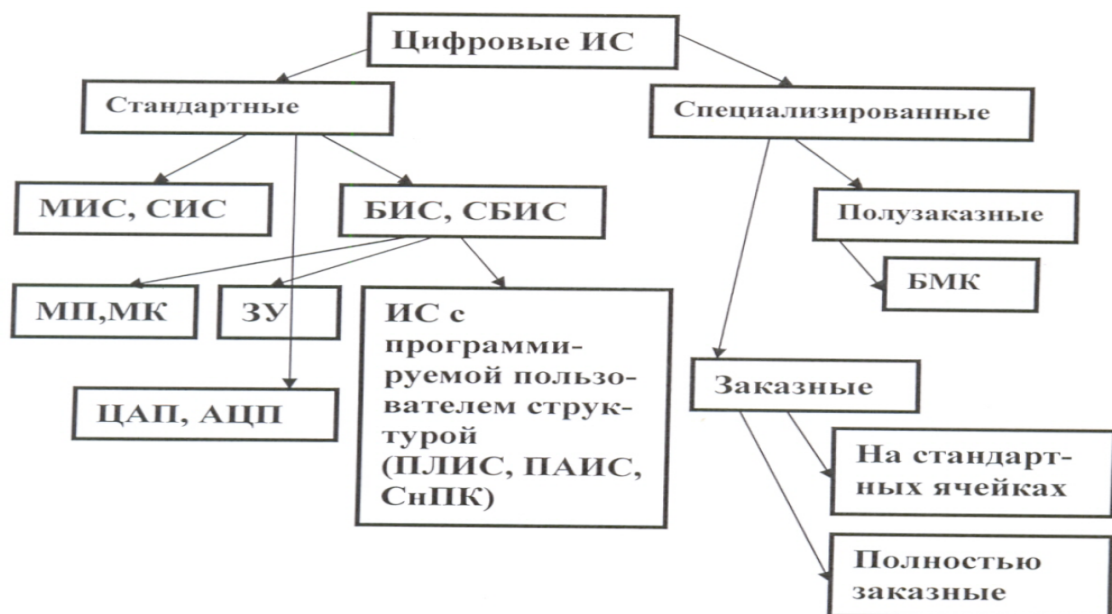
```

В коде появился **always-блок**, который является описанием выходного регистра. Для работы always-блока появились также входы модуля clk (тактовая частота) и reset (сброс). Выход сумматора защелкивается в триггере, описанном в always. Также есть пара сигналов разрешения ena. Посмотрим на схему, которую теперь синтезирует Quartus.



Family	Cyclone V
Device	5CEBA2F17A7
Timing Models	Final
Logic utilization (in ALMs)	27 / 9,430 (< 1 %)
Total registers	0
Total pins	0 / 128 (0 %)
Total virtual pins	53
Total block memory bits	0 / 1,802,240 (0 %)
Total DSP Blocks	1 / 25 (4 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0 / 4 (0 %)
Total DLLs	0 / 4 (0 %)

Рисунок . Таблица использованных ресурсов.



По характеру связи между входными и выходными переменными с учетом изменения этих связей по тактам работы различают **комбинационные устройства** и **последовательные (цифровые автоматы)**.

В *комбинационных устройствах* совокупность выходных сигналов в каждый такт работы однозначно определяется входными сигналами, имеющимися в этот момент на его входах.

Цифровые автоматы используются, если выходные сигналы определяются не только входными, но и предыдущими сигналами.

В идеальном случае напряжение на выходе микросхем должно быть равным напряжению питания или общего провода схемы. В реальных схемах так на полностью открытом транзисторе есть падение

напряжения. В результате на выходе цифровой микросхемы напряжение всегда будет меньше напряжения питания и больше потенциала общего провода.

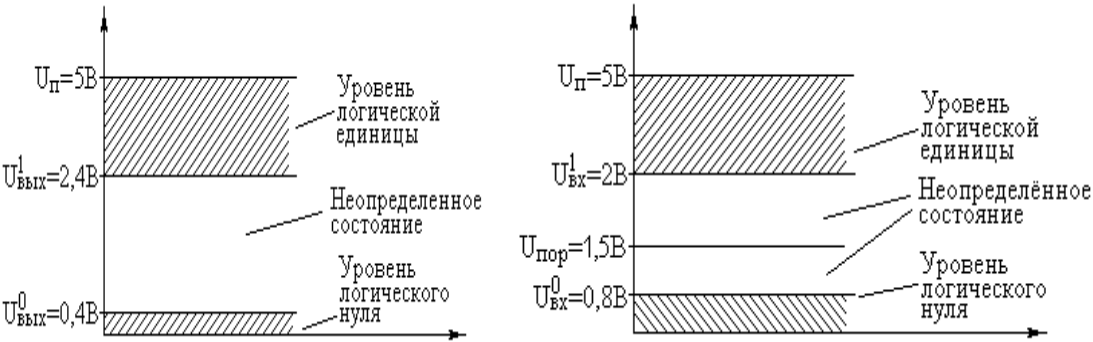
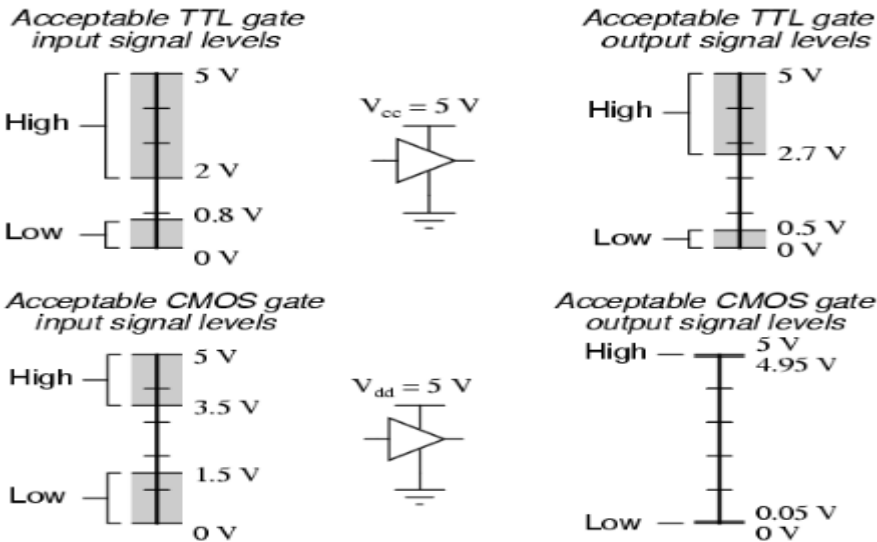


Рисунок 7 а) Уровни логических сигналов на выходе цифровых ТТЛ микросхем б) микросхем Texas Instruments.

Поэтому договорились напряжение, меньшее заданного уровня (уровень логического нуля) считать нулём, а напряжение, большее заданного уровня (уровень логической единицы), считать единицей. Если же напряжение на выходе микросхемы будет больше уровня логического нуля, но меньше уровня логической единицы, то такое состояние микросхемы будем называть неопределённым. На рисунке 7 приведены допустимые уровни выходных логических сигналов для ТТЛ микросхем. Микросхемы могут работать при воздействии неблагоприятных факторов, таких как пониженная температура, старение микросхем, воздействие радиации. Поэтому производители микросхем гарантируют срабатывание микросхем с некоторым запасом.



Исторически просто выполняется сумматор, а "вычитатор" влечет за собой дополнительные усилия. Поэтому операция вычитания была заменена операцией сложения, где вычитаемое представлялось в **дополнительном или обратном** коде.

Прямой, обратный и дополнительный код - это модели представления целых чисел, как положительных, так и отрицательных. Примеры записи некоторых чисел во всех трех восьмиразрядных кодах показаны в таблице ниже.

Числ о	Прямой код	Обратный код	Дополнительный код
0	00000000	00000000	00000000
1	00000001	00000001	00000001
-1	10000001	11111110	11111111
5	00000101	00000101	00000101
-5	10000101	11111010	11111011
8	00001000	00001000	00001000

-8	10001000	11110111	11111000
120	01111000	01111000	01111000
-120	11111000	10000111	10001000
127	01111111	01111111	01111111
-127	11111111	10000000	10000001

Во всех трех кодах старший разряд указывает на знак числа и он равен единице, если число отрицательное и нулю в противном случае. Различие между кодами наблюдается в способах представления модуля. Для положительного числа модуль во всех трех кодах представляется одинаково - это просто естественная запись двоичного числа. **Для отрицательных чисел, в обратном коде это просто поразрядная инверсия прямого кода, а в дополнительном - к обратному коду, как к числу, прибавляется единица.**

Сложение и вычитание чисел в обратном и дополнительном кодах выполняется с использованием обычного правила арифметического сложения многоразрядных чисел. При поразрядном сложении чисел разряды, изображающие знаки чисел рассматриваются как равноправные разряды двоичного числа, которые складываются друг с другом и с единицей переноса из предыдущего разряда числа по обычным правилам арифметики. Различия же обратного и дополнительного кодов связаны с тем, что делается с единицей переноса из старшего разряда (изображающего знак числа).

При сложении чисел в дополнительном коде единица переноса из старшего разряда игнорируется (теряется), а в обратном коде эту единицу надо прибавить к младшему разряду результата.

Пример: Сложить числа +12 и -5. А) В обратном коде

Десятичная форма	Двоичная форма	Прямой код	Обратный код
+12	+1100	00001100	00001100
-5	-101	10000101	11111010

Выполним сложение:

	+	0	0	0	0	1	1	0	0
		1	1	1	1	1	0	1	0
+	1	0	0	0	0	0	1	1	0
									1
		0	0	0	0	0	1	1	1

Единица из старшего разряда переехала в младший разряд. Результат в обратном коде - 00000111. Знаковый разряд равен 0, результат положительный, запись кода числа совпадает с записью прямого кода. Теперь восстановим алгебраическую запись результата. Он равен +111 (незначащие нули отброшены), или в десятичной форме +7. Проверка (+12-5=+7) показывает, что результат верный.

б) В дополнительном коде

Десятичная форма	Двоичная форма	Прямой код	Обратный код	Дополнительный код
+12	+1100	00001100	00001100	00001100
-5	-101	10000101	11111010	11111011

Выполним сложение в дополнительном коде:

	+	0	0	0	0	1	1	0	0
		1	1	1	1	1	0	1	1
	1	0	0	0	0	0	1	1	1
		0	0	0	0	0	1	1	1

Старший разряд теряется. Результат в дополнительном коде - 00000111. Знаковый разряд равен 0, результат положительный, равный 7₁₀

Исторически первым было семейство ЦИС типа ТЛНС. Нагрузками логических элементов являются входные цепи аналогичных элементов. Серьезный недостаток ТЛНС - неравномерное распределение тока между базами нагрузочных транзисторов. Такая неравномерность связана и с различием входных характеристик транзисторов и с различием

коллекторных токов насыщенных транзисторов. Токи насыщения существенно зависят от числа транзисторов базового элемента, находящихся в открытом состоянии. При подключении нескольких нагрузок к базовому элементу снижается логический перепад выходных уровней и, следовательно, допустимое значение статических помех.

В семействе **РТЛ** для выравнивания входных характеристик транзисторов в цепях баз включены резисторы с сопротивлениями несколько сот Ом. При этом возросли уровень логической 1, логический перепад уровней и допустимое напряжение статической помехи, но снизилось быстродействие. Сопротивления в цепи базы и входные емкости образуют RC-цепочки, из-за которых возрастает длительность фронта выходного импульса.

Для того чтобы избежать указанного недостатка в семействе **РЕТЛ**, резисторы в цепях базы шунтированы конденсаторами небольшой емкости. В момент переключения предыдущего элемента эти конденсаторы на некоторое время шунтируют резисторы и обеспечивают повышенные значения базовых токов, тем самым снижая длительность фронта.

Резисторы и особенно конденсаторы занимают большую площадь. Поэтому элементы семейства РТЛ и РЕТЛ оказались неперспективными и в настоящее время используются редко.

Элементы ТТЛ с диодами Шоттки получили название ТТЛШ. Они имеют примерно в 3 раза меньшее среднее время задержки сигналов.

Для повышения быстродействия между базой и коллектором VT_2 включают **диод Шоттки**. Как известно, максимальная частота переключения транзистора ограничивается в основном временем рассасывания накопленных зарядов. Для повышения максимальной частоты переключения необходимо предотвратить насыщение транзистора и этим исключить накопление заряда. Включение диода Шоттки параллельно переходу «база — коллектор» транзистора приводит к возникновению ООС по напряжению и препятствует снижению напряжения между коллектором и эмиттером ниже 0,3 В. Рис. 10.

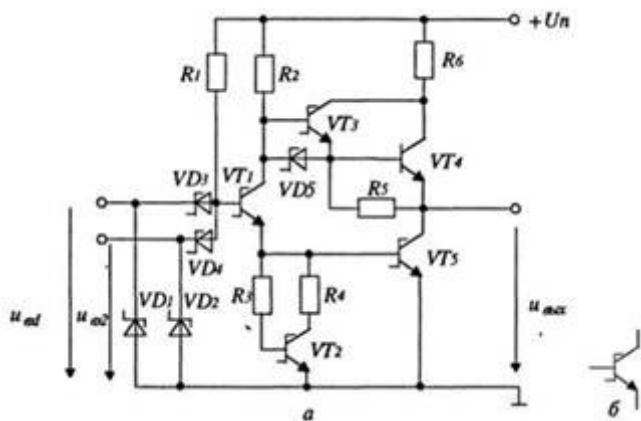


Рис.10 Элементы ТТЛШ.

Элементы ТТЛ с тремя выходными состояниями имеют дополнительный вход V — вход разрешения (рисунки 11,а), которым можно отключить схему от нагрузки.

При подаче на этот вход напряжения U^0 транзистор VT_5 открыт и насыщен, а транзисторы VT_6 и VT_7 закрыты и поэтому не влияют на работу логического элемента. В зависимости от комбинации сигналов на информационных входах на выходе ЛЭ может быть сигнал с уровнем «лог. 0» или «лог. 1». При подаче на вход V напряжения с уровнем «лог. 1» транзистор VT_5 закрывается, а транзисторы VT_6 и VT_7 открываются, напряжение на базе транзистора VT_3 уменьшается до уровня $U_{БЭ,нас} + U_\Delta$, транзисторы VT_2, VT_3, VT_4 закрываются и ЛЭ переходит в высокоимпеданное (третье) состояние, то есть отключается от нагрузки.

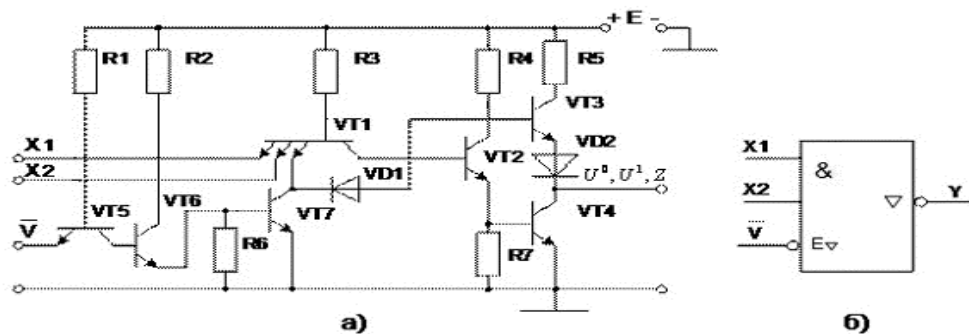


Рис.11.

Особенность ЭСЛ (рис. 12) в том, что схема логического элемента строится на основе интегральных дифференциальных усилителей, транзисторы T_1 , T_2 , T_3 которые могут переключать ток и при этом никогда не попадают в режим насыщения из-за наличия в коллекторных и эмиттерных цепях резисторов $R_1 \dots R_6$, ограничивающих этот ток. Этим устраняется этап рассасывания избыточных зарядов, поэтому элементы типа ЭСЛ – самые быстродействующие, их быстродействие достигает субнаносекундного диапазона (доли нсек), но потребляют значительную мощность от источника.

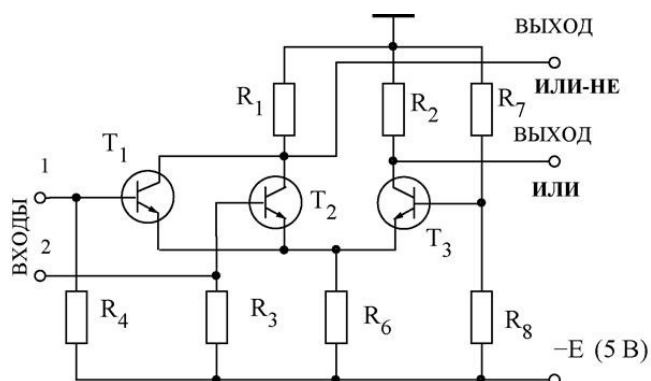


Рис. 12

Если на один из входов или оба входа подать напряжение «1», то соответствующий или оба транзистора вместе откроются. Ток, протекающий через них, создаст падение напряжения на резисторе R_6 , напряжение $U_{бэ}$ транзистора T_3 падает и напряжение на коллекторе T_3 повышается – «1» (схема ИЛИ). Если напряжение на обоих входах низкое и не достигает порогового значения транзисторы T_1 и T_2 закрыты, а транзистор T_3 открыт высоким напряжением на его переходе $U_{бэ}$, электрический потенциал коллектора T_3 падает – «0». Инверсный выход с коллекторов транзисторов T_1 и T_2 реализует функцию ИЛИ–НЕ.

Как известно, МДП транзисторы чувствительны к наведенным электростатическим нарядам. Чтобы избежать пробоя МДП транзисторов, в современных ИС предусматривается встроенная защита с помощью диодов.

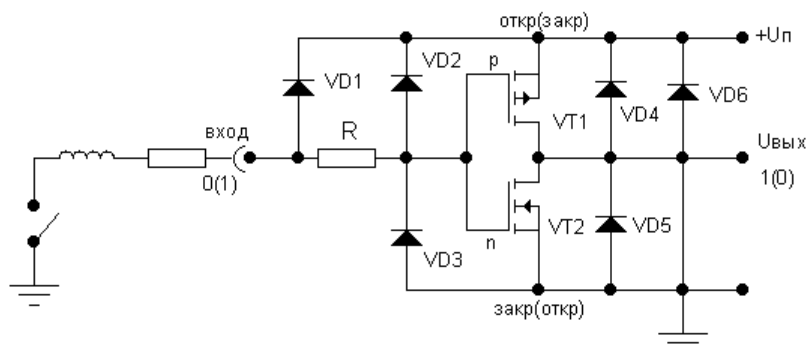


Рис.15 Защитные элементы в реальной схеме инвертора КМОП

Элементы полупроводниковых запоминающих устройств

Цифровые полупроводниковые ИС памяти используются в *оперативных (ОЗУ) и постоянных (ПЗУ) запоминающих устройствах*. ПЗУ хранят информацию при отключении источника питания, тогда как в ОЗУ она теряется. *Статические ОЗУ* памяти могут хранить информацию в течение длительного времени, а *динамические ОЗУ* — ограниченное время. Статические ОЗУ обладают максимальным быстродействием, а динамические ОЗУ обеспечивают максимальную информационную емкость и минимальную потребляемую мощность. Большая часть БИС памяти создаются на МДП-транзисторах, а ИС памяти — на биполярных транзисторах, которые обладают повышенным быстродействием, но меньшей информационной емкостью.

Элемент памяти – элемент, для которого можно определить состояние «0» или «1» и обладающий способностью сохранять состояние во времени при отсутствии внешних воздействий.

Бистабильная ячейка представляет собой **симметричный триггер**, содержащий два инвертора с перекрестными обратными связями; выход первого инвертора соединен со входом второго, а выход второго — со входом первого.

В данной схеме параметры левой и правой части идентичны, то есть $R_{b1} = R_{b2}$, $R_{k1} = R_{k2}$, $R_1 = R_2$, $C_1 = C_2$, транзисторы VT1 и VT2 имеют одинаковые параметры. Хотя триггер и называется симметричным, в реальных схемах никогда не удаётся добиться идентичности параметров транзистора, поэтому при подключении триггера к источнику питания один из его транзисторов окажется открытым (состояние насыщения), а другой транзистор будет в закрытом состоянии (состояние отсечки). В данном состоянии триггер может находиться сколько угодно долго (пока присутствует напряжение питания).

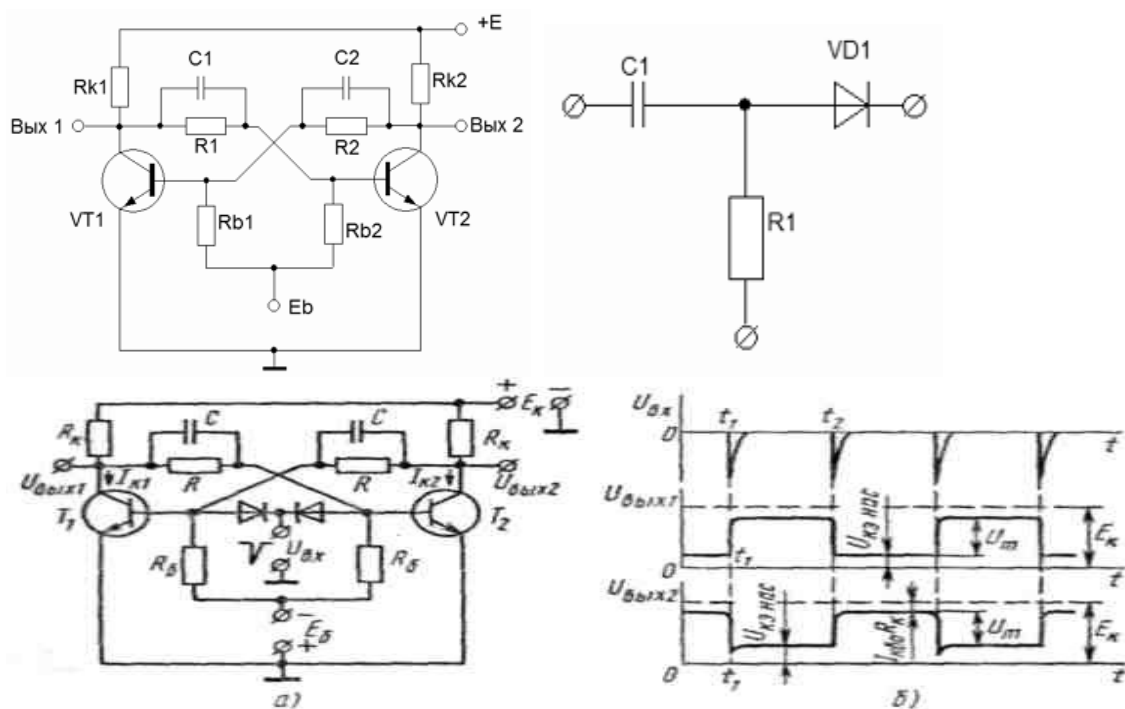
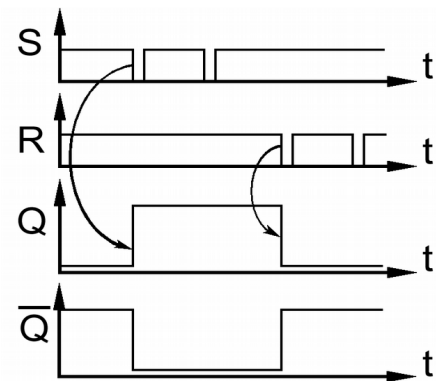
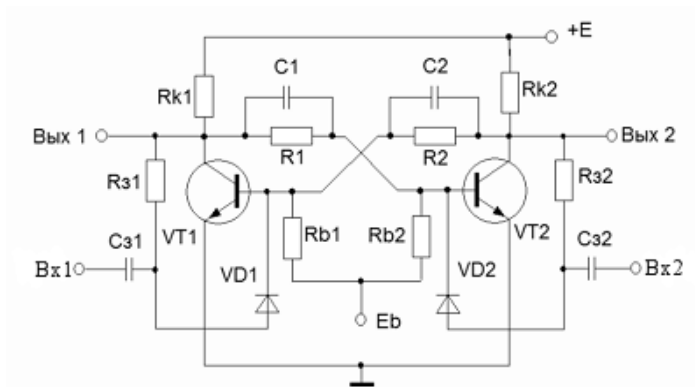


Рис. 6.16

а) Схема симметричного триггера с независимым смещением б) Схема запуска в) Симметричный триггер с счетным входом

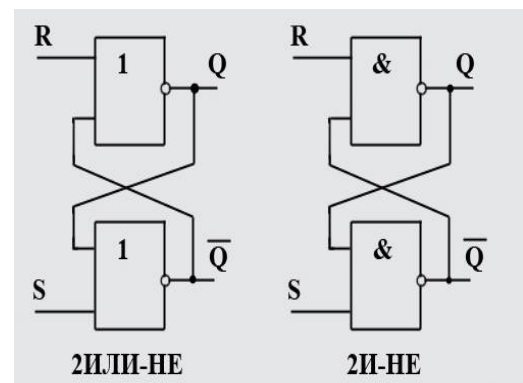
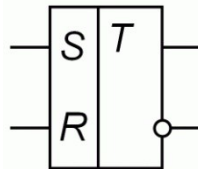
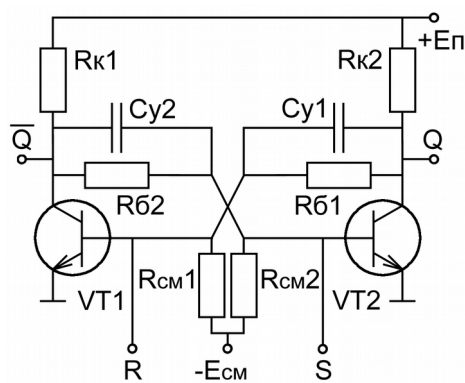
Поскольку на базу VT2 поступает дополнительное напряжение смещения E_b , поэтому на базе VT2 поддерживается напряжение меньше, чем необходимо для открытия данного транзистора. Таким образом, за счёт дополнительного источника смещения E_b схема триггера находится в устойчивом состоянии, а на выходах триггера поддерживаются парафазные напряжения.

Для того чтобы на выходах симметричного триггера изменились напряжения необходимо подать на триггер **внешний управляющий (запускающий) импульс** напряжения или тока. В этом случае триггер переходит из одного устойчивого состояния в другое, транзисторы в схеме изменяют своё состояние: открытый транзистор – закрывается, а закрытый – открывается.



Симметричный триггер с независимым смещением и разделным запуском.

Симметричный триггер с разделным запуском называется **RS-триггером**, он имеет два входа и два выхода. Входы, на которые подают управляющие импульсы, называются установочными и обозначают R и S, выходы триггера обозначают Q и \bar{Q} .



На функциональной схеме изображен RS-триггер асинхронного типа на транзисторах, элементах И-НЕ и элементах ИЛИ-НЕ.

Мультивибратор представляет собой генератор колебаний почти прямоугольной формы на основе двухкаскадного усилителя с положительной обратной связью (ПОС), в котором выход каждого каскада соединен с входом другого. Колебания представляют собой смену квазиустойчивых состояний, в которых каждый транзистор попеременно находится в открытом состоянии, характеризующимся напряжением на базе $U_b > 0,7 \text{ В}$, напряжением на коллекторе $U_k = (0,1 - 0,2) \text{ В}$ и током коллектора $I_k = V_k / R_k$, и закрытом состоянии, характеризующимся напряжением на базе $U_b < 0,6 \text{ В}$, напряжением на коллекторе $U_k = V_k$, токе коллектора $I_k = 0$. Фаза перехода очень короткая относительно длительности нахождения в состояниях благодаря глубокой положительной обратной связи, охватывающей два каскада усиления. ПОС существует только тогда, когда оба транзистора открыты.

Переход транзисторов из одного состояния в другое определяют времязадающие цепочки $R_{б1} C1$ и $R_{б2} C2$ и соотношение напряжений V_b и V_k . Открытие (закрытие) одного транзистора передается на базу другого с некоторой задержкой, а положительная обратная связь формирует короткие фронты.

Математические модели мультивибратора отличаются от реальных необходимостью введения разбаланса в плечах, что бы колебания возникли, в редакторе начальных условий.

Мультивибраторы применяются в устройствах автоматики и измерительной техники, радиотехнических игрушках в качестве задающих генераторов и формирователей импульсов,

делителей частоты, бесконтактных переключателей тока и др.

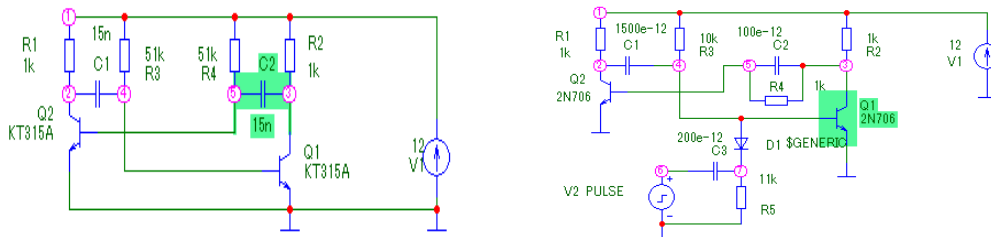
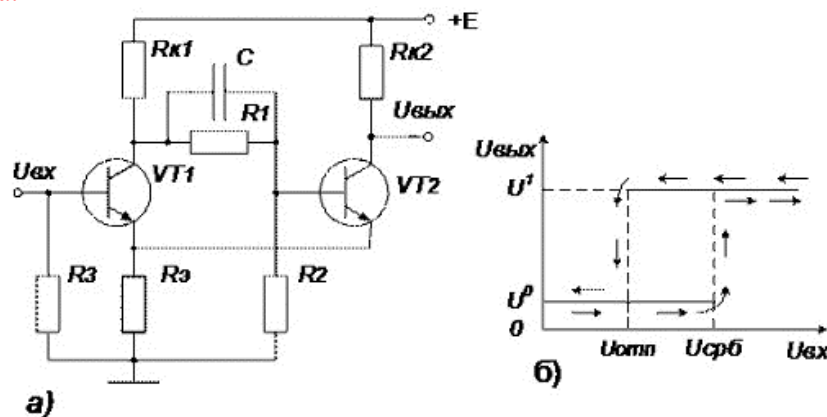


Рис. Мультивибраторы в лабораторных работах.

Несимметричный триггер (триггер Шмита) имеет два устойчивых состояния, однако, в отличие от симметричного триггера, нахождение его в том или ином устойчивом состоянии зависит от величины входного сигнала.

Несимметричный триггер на дискретных элементах состоит из двух транзисторов, в эмиттерную цепь которых включён резистор R_3 (Рис.). При таком включении напряжение на базе транзистора VT1 зависит от значения коллекторного тока I_{K2} транзистора VT2. В свою очередь, базовая цепь VT2 через делитель $R1/R2$ соединена с коллекторной цепью транзистора VT1. Эти цепи создают замкнутую петлю положительной обратной связи, которая, как и в симметричном триггере, обеспечивает быстрое переключение триггера Шмита из одного устойчивого состояния в другое, когда оба транзистора работают в активном режиме. Параметры схемы несимметричного триггера рассчитываются таким образом, чтобы при уменьшении входного напряжения транзистор VT2 открывался и триггер переходил в исходное устойчивое состояние при напряжении отпускания $U_{BX} = U_{отп} < U_{срб}$. При таком условии амплитудная передаточная характеристика имеет петлю гистерезиса.



Вариант реализации **триггера** на элементах «стрелка пирса», «штрих шеффера».



В ОЗУ используется достаточно много типов запоминающих ячеек.

Статические ОЗУ. (СОЗУ). Элементарной ячейкой статического ОЗУ с произвольной выборкой является триггер на транзисторах T_1 - T_4 (рис.) с ключами T_5 - T_8 для доступа к шине данных. Причем T_1 - T_2 - это нагрузки, а T_3 - T_4 - нормально закрытые элементы.

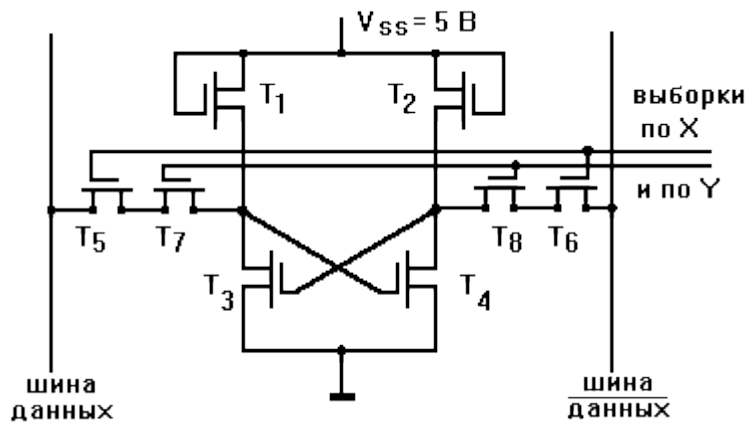


Рис. Ячейка статического ОЗУ. Количество транзисторов (6 или 8) на ячейку зависит от логической организации памяти микропроцессорной системы.

Память на КМДП транзисторах (без использования дополнительных элементов) можно построить лишь на триггерах, каждый из которых состоит из двух КМДП инверторов. Выход первого инвертора, состоящего из р-канального транзистора $Tr1$ и н-канального транзистора $Tn1$, присоединен ко входу второго инвертора, состоящего из транзисторов $Tr2$ и $Tn2$. А выход второго инвертора присоединен ко входу первого. Благодаря такой перекрестной обратной связи и получается бистабильная схема триггера.

Когда на затворах транзисторов первого инвертора потенциал низкий, транзистор $Tr1$ открыт, транзистор $Tn1$ закрыт, и на выходе этого инвертора устанавливается высокий потенциал, близкий к потенциалу источника питания U_n . Поскольку этот выход электрически соединен с затворами транзисторов второго инвертора, то транзистор $Tn2$ открыт, а транзистор $Tr2$ закрыт. На выходе второго инвертора устанавливается низкий потенциал, который и поддерживает первое стабильное состояние триггера. Когда на затворах первого инвертора потенциал высокий, транзистор $Tr1$ закрыт, транзистор $Tn1$ открыт, и на выходе этого инвертора устанавливается низкий потенциал, близкий к потенциалу "земли". Поэтому транзистор $Tn2$ закрыт, а транзистор $Tr2$ открыт. На выходе второго инвертора устанавливается высокий потенциал, который и поддерживает альтернативное стабильное состояние триггера. Триггер – это классический бистабильный элемент, который сохраняет 1 бит информации.

Оперативные ЗУ состоят из **накопителя и схем управления**. Данные, которые необходимо запомнить, хранятся в накопителе. Схемы управления включают усилители, разного рода ключи, коммутаторы, дешифраторы и т. д. Накопитель состоит из элементов памяти в основном на базе бистабильных ячеек, каждая из которых хранит один бит информации, соответствующей хранению логических 0 и 1. Для того, чтобы каждый триггер, из которых состоит память, имел индивидуальный адресный доступ, в состав ячейки вводят также 2 ключевых МДП транзистора – $T5$ и $T6$.

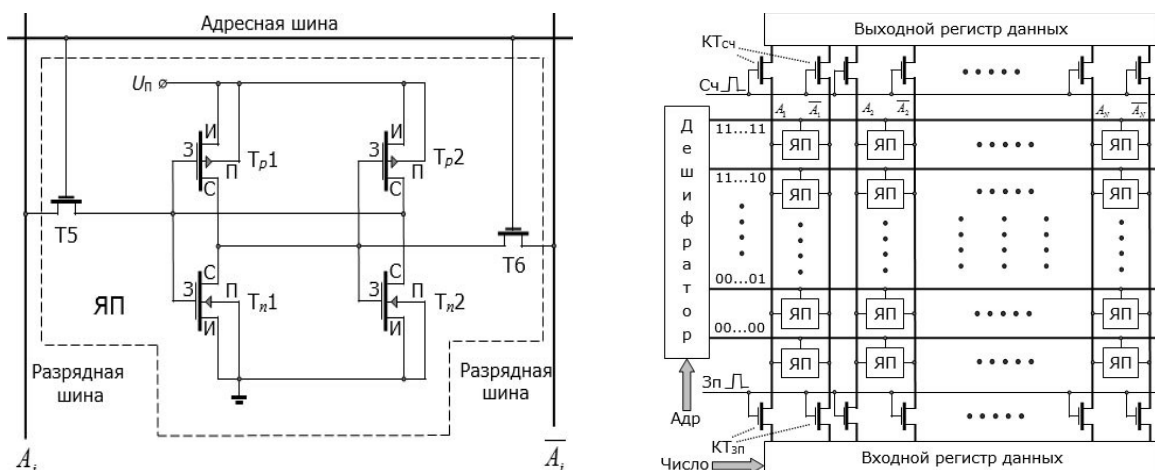


Рис. Принципиальная электрическая схема ячейки памяти (ЯП) "статического" ОЗУ на триггере из КМДП транзисторов и подключения ее к адресной и разрядным шинам

Статическая ячейка на МДП-транзисторах с р-каналами представляет собой триггер (транзисторы VT_1 — VT_4) с управляющими ключами VT_5 и VT_6 , соединенных с шинами столбцов Y' и Y'' (рис. - а). При отсутствии выборки напряжение на шине X близко к нулю, транзисторы VT_5 и VT_6 закрыты, триггер отключен от шин столбца и элемент памяти хранит ранее записанную информацию.

При записи информации на одну из шин столбца подают напряжение 0, а на другую — напряжение 1, после этого на адресную шину X поступает положительный импульс с амплитудой, близкой к напряжению источника питания $E_{\text{пит}}$, который открывает транзисторы VT_5 и VT_6 и в точках A и B устанавливаются такие же напряжения, что и на шинах Y' , Y'' , и триггер находится в необходимом состоянии (взводится).

В режиме считывания при поступлении на шину X импульса выборки VT_5 и VT_6 отпираются и на шинах столбца устанавливаются напряжения, соответствующие состоянию триггера (0 на одной из шин и E на другой), которые воспринимаются усилителем считывания. Таким образом, импульс на адресной шине в обоих режимах играет роль тактового импульса.

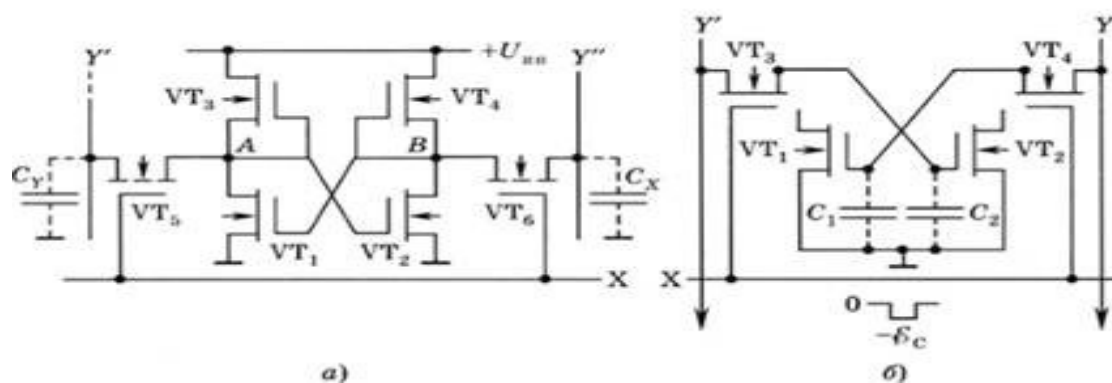


Рис. а) статическая

б) динамическая

ячейки

Динамические ОЗУ (ДОЗУ). В отличие от статических ЗУ, которые хранят информацию пока включено питание, в динамических ЗУ необходима постоянная регенерация информации, однако при этом для хранения одного бита в ДОЗУ нужны всего 1-2 транзистора и накопительный конденсатор (рис.). Такие схемы более компактны.

На рис. - б изображена **запоминающая ячейка динамического типа**, в которой информация сохраняется с помощью конденсаторов C_1 и C_2 , окруженных транзисторами. Алгоритм записи и считывания аналогичен предыдущему случаю. При записи на шины Y' и Y'' поданы соответственно уровни 0 и $1 = E$. Уровень 0 через ключ VT_4 поступает на затвор VT_1 , который будет открыт. На затвор VT_2 подается уровень 0 и он будет закрыт. На емкостях C_1 и C_2 напряжения будут иметь значения соответственно $U_{C1} = 1$ с, $U_{C2} = 0$. Остаточный ток запертого VT_2 мал, и конденсатор C_s будет разряжаться очень медленно. Следовательно, U_{C1} и U_{C2} будут сохраняться длительное время.

Для поддержания напряжения на емкости постоянным при ее неизбежном разряде при считывании, осуществляют **регенерацию**, т. е. периодически производят запись того же кода. Динамические запоминающие ячейки из-за отсутствия источника питания в режиме хранения не потребляют мощности, поэтому они экономичнее статических.

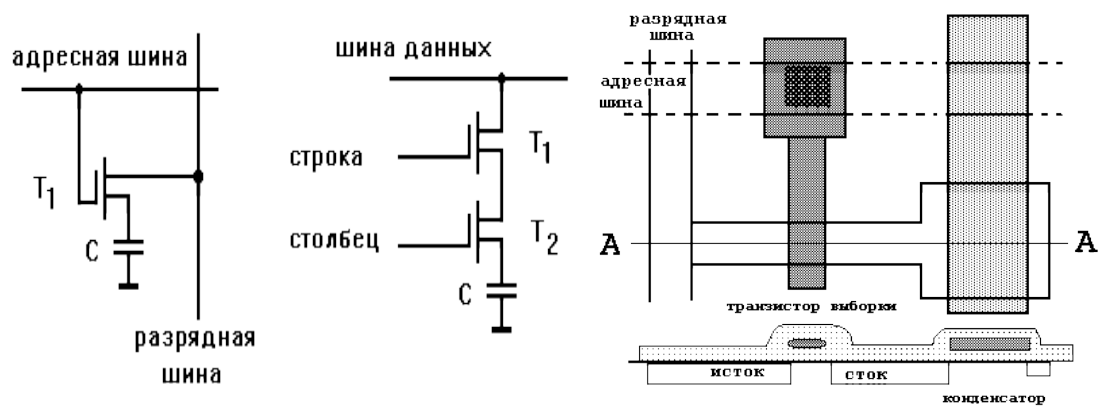


Рис. Запоминающая ячейка динамического ОЗУ. Конструкция ячейки ДОЗУ (разрез схемы по линии А-А).

Запоминающие ячейки на МДП(МОП)-транзисторах экономичнее и компактнее по сравнению с ячейками на биполярных транзисторах. Однако, последние обладают лучшим быстродействием, чем МДП-ячейки. В англоязычных источниках оперативную КМДП память с произвольным доступом обычно называют SRAM (Static Random Access Memory).

<https://www.intuit.ru/studies/courses/12180/1173/lecture/19628>