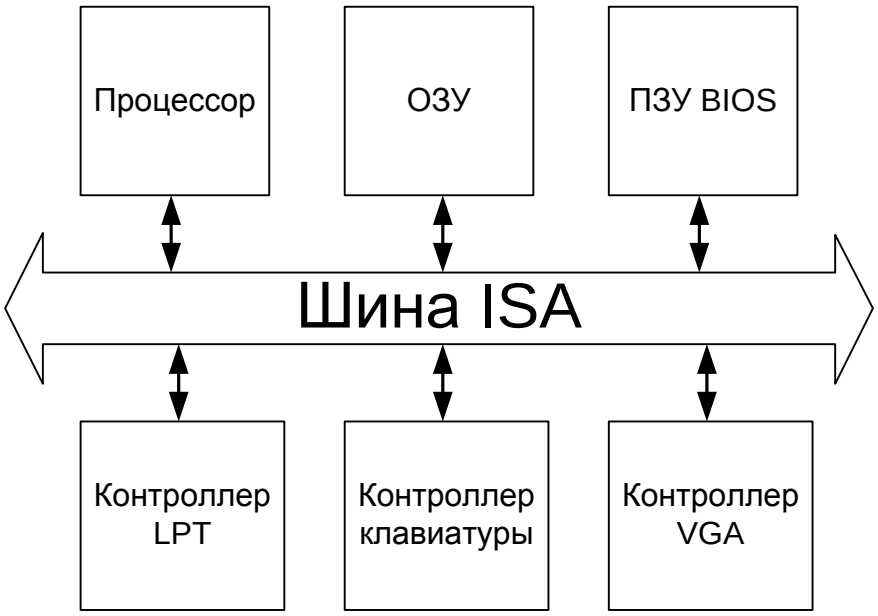
Выходы с третьим состоянием

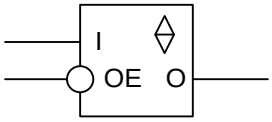
Давайте рассмотрим типовой компьютер начала девяностых. Нет, компьютер с шиной PCI устроен почти так же, только рассказ будет изобиловать лишними подробностями (что касается PCI Express, то там всё чуть иначе, но на сегодня всё равно нет микросхем, чтобы нам с вами что-то там в домашних условиях спаять, так что такой тип шины мы не рассматриваем). Итак. Что же было в том компьютере? Процессор, ОЗУ, ПЗУ с BIOSом. Ну, ещё порт LPT был. Контроллер клавиатуры и прочие контроллеры, втыкаемые в шину ISA. И все выше перечисленные устройства также подключались к шине ISA.



Мы уже кратко касались шины ISA и знаем, что там есть линии данных. И данные могут выходить из чего угодно: Из процессора, из ОЗУ, из ПЗУ, из каждого контроллера. А контроллеры, кроме того, могут втыкаться в слоты. Возникает вопрос: А где расположен тот мультиплексор, который спасает двухтактные выходы от выгорания? Мы же уже знаем, что выходы друг с другом соединять нельзя… Пришла пора признаться, что ~~детей приносит не аист~~ существует ещё один тип выходов: выходы с третьим состоянием.

У элемента, имеющего такой выход, есть дополнительный вход OE – Output Enable. Когда этот вход в активном состоянии, выход работает, как обычный двухтактный. Когда в пассивном – выход отключается. Его как будто бы нет. Он не оказывает никакого воздействия на шину, к которой подключен. Это третье состояние. Оно же Z-состояние. Оно же высокоимпедансное состояние. Оно же High Impedance State. Во сколько названий придумали. Вам может встретиться в литературе любое, так что помнить их необходимо.

Вот так обозначается элемент с Z-состоянием по ГОСТ. Ромбик как раз и показывает наличие Z-состояния у выхода. Вход OE у приведённого варианта - с активным нулём.



А вот его таблица истинности

I	OE	O
0	0	0
1	0	1
0	1	Z
1	1	Z

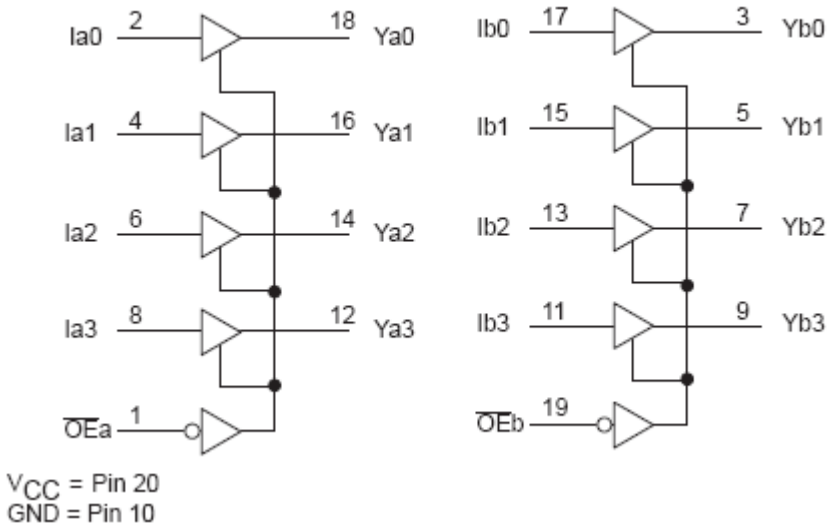
Ну, а дальше – тот, кто проектирует систему, может соединить сколько угодно таких выходов, но обязательно должен учитывать, что единовременно может быть открыт выход только одного устройства. Если откроются сразу два – они могут сжечь друг друга точно так же, как и два двухтактных выхода. У меня знакомый так несколько ПЛИС сжёг, нечаянно открыв внутри них несколько выходов. Сигналы даже наружу не вышли, где-то внутри буферы выходные сгорели, а ПЛИСки отправились на помойку.

Шинные формирователи

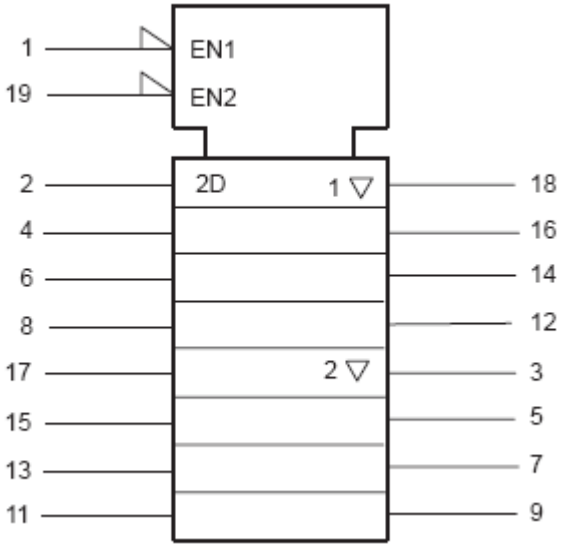
Существуют специальные микросхемы – шинные формирователи. Они содержат много (4 или 8) линий данных, управляемых одним сигналом OE. При работе с ПЛИС, мы сами не будем применять этот анахронизм (всё уже есть внутри ПЛИС), но в схемах от Заказчиков такие микросхемы встречались, поэтому уметь читать их в чужом материале – необходимо. А раз так, давайте с ними познакомимся.

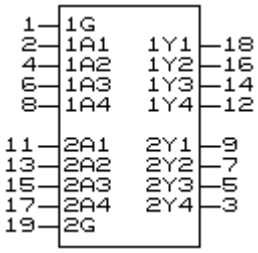
Первая популярная микросхема – К555АП5, она же 74LS244 (в русском аналоге вмсто 555 может быть что угодно, важно, что АП5, в иностранном – вместо LS может быть что угодно, важно, что 244).

Вот её структурная схема:



То есть, в одной микросхеме есть две однотипных структуры: четыре буфера с Z-состоянием. Получается, что с помощью этой микросхемы можно организовать или одну 8-разрядную шину в одну сторону, или одну четырёхразрядную шину в обе стороны. На реальных схемах эта микросхема будет выглядеть примерно так (каждый извращается, как может):





Ну, и вроде того.

Вторая микросхема – К555АП6, она же 74LS245. В отличие от предыдущей, у неё всегда 8 разрядов. Но зато имеется дополнительный вход T. который задаёт направление передачи данных.

Структурная схема такова:

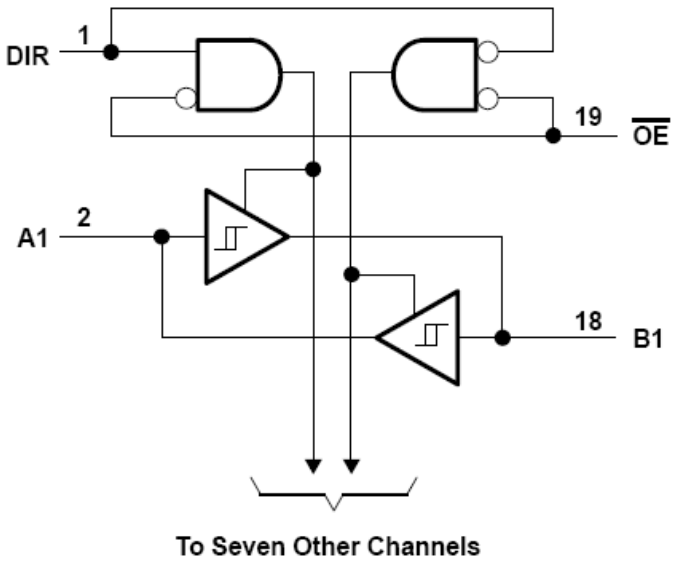
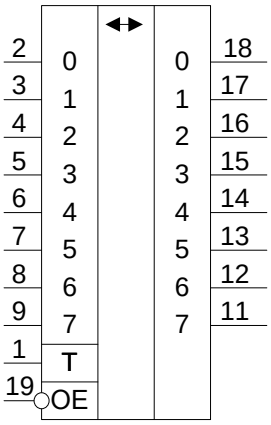


Таблица истинности выглядит так:

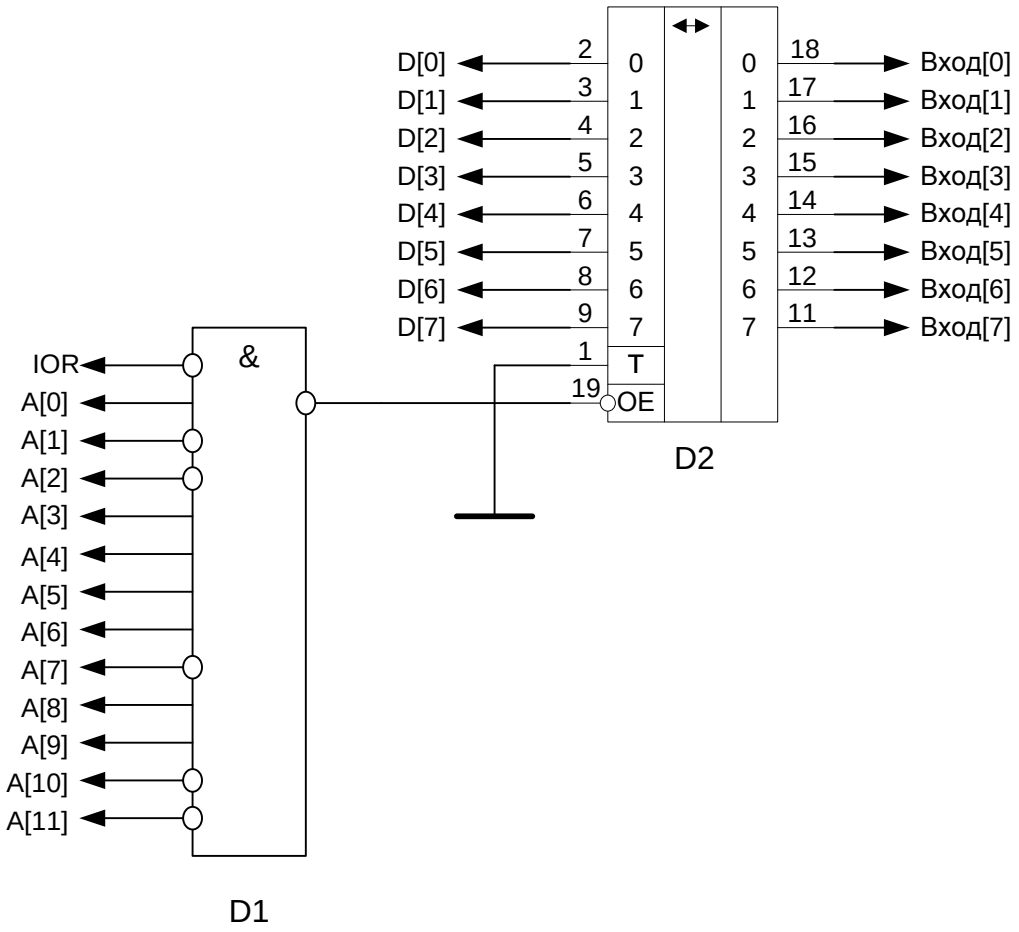
INPUTS		OPERATION
$\overline{OE}$	DIR	
L	L	B data to A bus
L	H	A data to B bus
H	X	Isolation

А на схемах этот шинный формирователь обозначается так:



Пример использования шинного формирователя

Мы уже рассматривали адаптер LPT порта для шины ISA. Там у нас был защёлкиватель выхода. А вход мы пока не сделали. Вот давайте сделаем вход для адреса 0x379 (берём калькулятор и выясняем, что в двоичном виде это 001101111001).



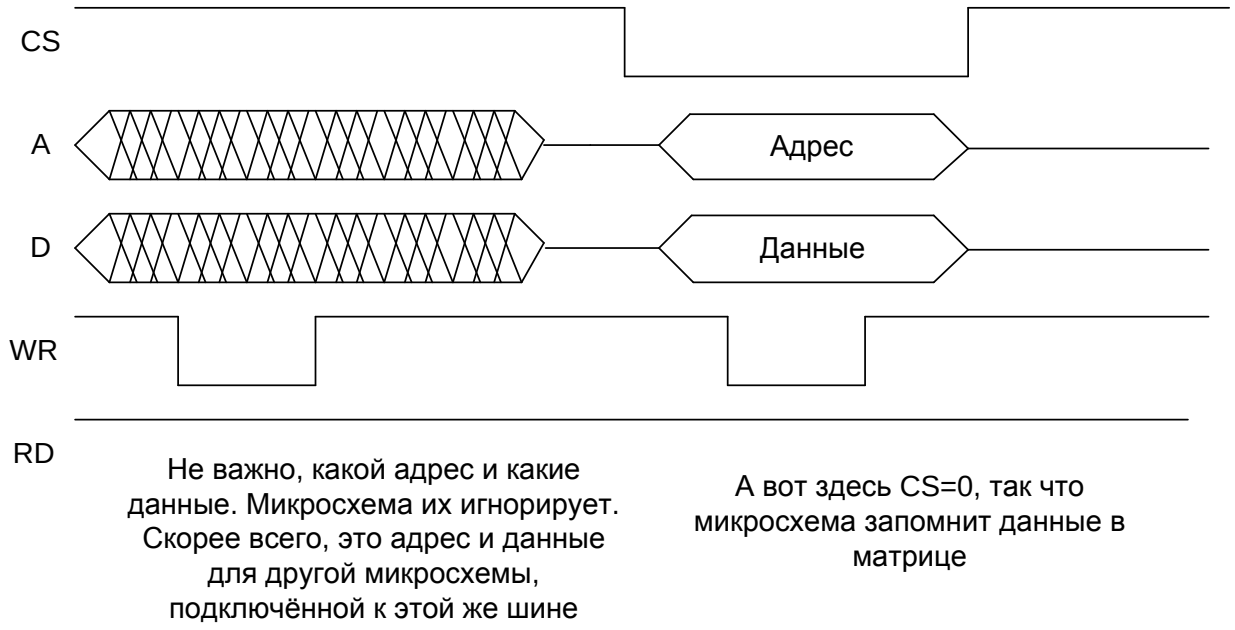
Итак. Элемент D2 – это шинный формирователь. Его вход T соединён с общим проводом, то есть, на нём логический ноль (Low). Согласно таблице истинности, в таком режиме шинник всегда пропускает сигнал справа налево (B->A). Значит справа подключаем входы порта LPT, слева – шину данных ISA. Когда нам открывать шинный формирователь? Когда адрес равен двоичному 001101111001, а сигнал IOR находится в нуле. Помним только, что у входа OE микросхемы К555АП6 активное значение – ноль. Учитываем это при изготовлении дешифратора адреса D1. Собственно, всё. Теперь, если надо, мы сможем сделать полноценный порт LPT для шины ISA, так как он представляет собой ни что иное, как два порта вывода (0x378 и 0x37A) и один порт ввода (0x379). И то и другое мы уже умеем делать. Мелочёвку (инверсные сигналы и прочее) мы также сможем подогнать под спецификацию порта.

Долгожданный пример использования дешифратора

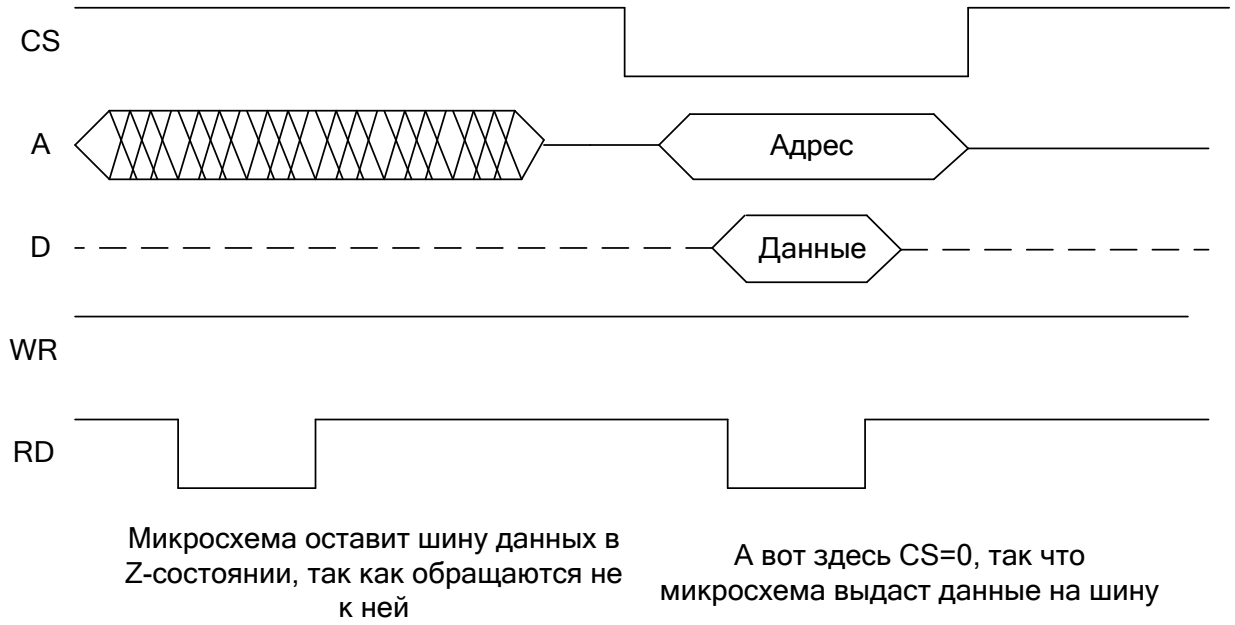
Внимательный читатель спросит: «А не забыл ли автор, что вообще-то речь шла про дешифраторы?». Нет, не забыл. Просто для того примера, который был задуман, надо было рассказать, что такое Z-состояние. Давайте вспомним видеокарты, у которых имелись панельки под установку дополнительной памяти. Как работает память? До недавних пор она работала примерно так же, как уже описывалось в шине ISA (сейчас – чуть сложнее, но в целом – так же, а лишние подробности затенят суть изложения, так что будем рассматривать старую память, тем более, что флэш-память до сих пор так работает). У неё был только один дополнительный сигнал – CS, что расшифровывается, как Chip Select. Давайте освежим в памяти эту шину.

Сигнал	Назначение
A[N..0]	Шина адреса. Адрес требуемой ячейки выставляется на этих линиях в двоичном коде
D[15..0]	Шина данных. Пусть она будет 16-разрядной, в нашем случае, это не столь важно
RD	Строб чтения. Чтобы считать данные из памяти, ставим адрес, затем роняем строб и снова поднимаем его. По срезу строба, микросхема выставит данные на шину, а по фронту, мы защёлкнем их в своём регистре
WR	Строб записи. Чтобы записать данные в микросхему, мы должны установить адрес и данные, а затем сбросить и снова поднять этот строб. По фронту микросхема защёлкнет данные в матрице памяти
\CS	Если этот сигнал равен 1, то микросхема находится в отключённом состоянии. Она не реагирует на стробы. Если сигнал равен 0, то микросхема активна, она реагирует на стробы.

Давайте представим это графически. Вот диаграмма записи в память



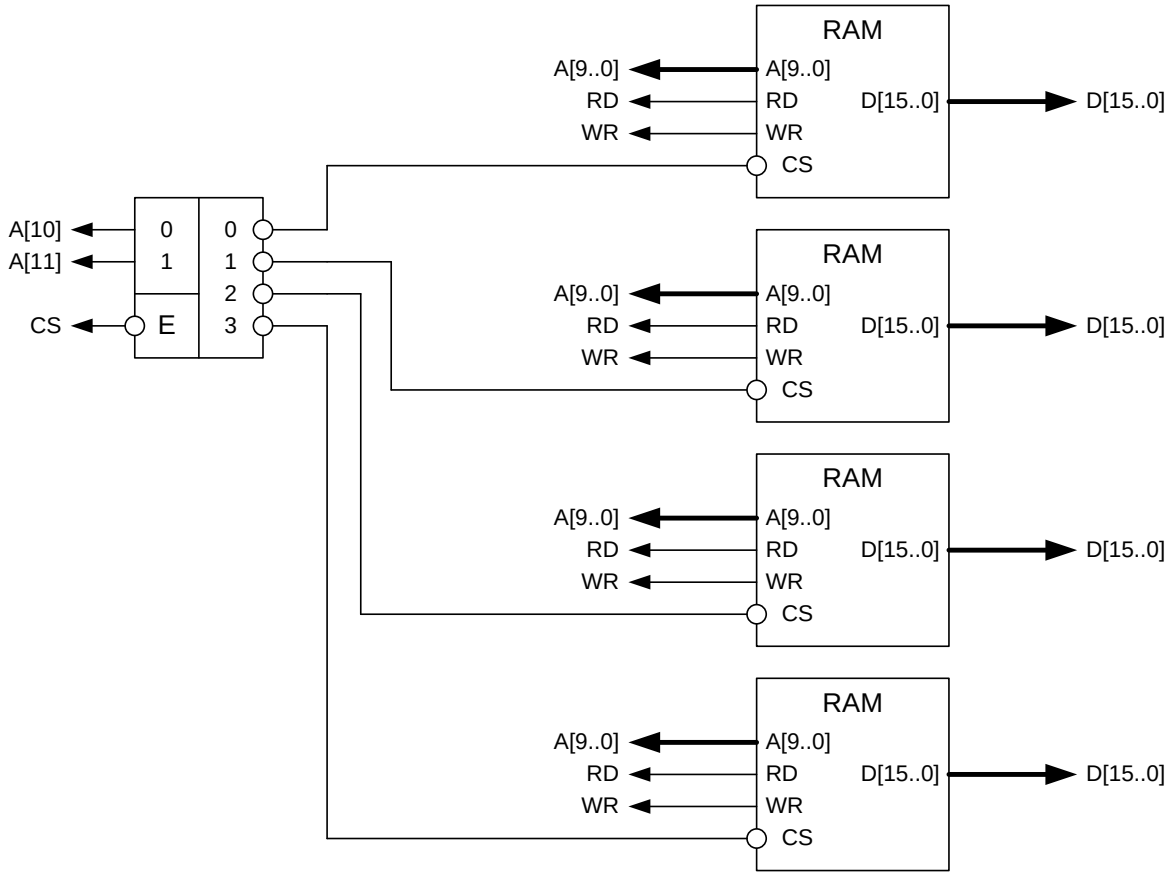
А вот – диаграмма чтения



Допустим, у нас объём одной микросхемы 1 килобайт (это чтобы рисунки не загромождать). То есть, адрес может быть от 0 до 1023 (десятичное). В двоичном виде, адрес 1023 равен 1111111111. И вот у нас на гипотетической видяшке припаяна одна микросхема и имеется ещё три посадочных места. Если адресуемся к первой микросхеме, то она лежит в адресах от 0 до 1023 (от 000000000000 до 001111111111), ко второй – от 1024 до 2047 (от 010000000000 до 011111111111), к третьей – от 2048 до 3071 (от 100000000000 до 101111111111), к четвёртой – от 3072 до 4095 (от 110000000000 до 111111111111). Запишем диапазоны в таблице

Микросхема	**От (двоичное)**	**До (двоичное)**
0	00 00 0000 0000	00 11 1111 1111
1	01 00 0000 0000	01 11 1111 1111
2	10 00 0000 0000	10 11 1111 1111
3	11 00 0000 0000	11 11 1111 1111

Что мы видим? А видим мы, что микросхема 0 обслуживает диапазон адресов 00XXXXXXXXXX, микросхема 1 – 01XXXXXXXXXX, микросхема 2 – 10XXXXXXXXXX, микросхема 3 – 11XXXXXXXXXX. Постойте! Но мы видим старое доброе 0 – 00, 1 – 01, 2 – 10, 3 – 11. Это же правило счёта в двоичной системе! Итого, в зависимости от двоичного кода 00, 01, 02, 03, входной сигнал CS надо пропустить или на микросхему 0, или на микросхему 1, или на микросхему 2, или на микросхему 3 (а остальные оставить не выбранными). Позвольте… Но именно этим занимается дешифратор-демультиплексор! Вот и воспользуемся им!

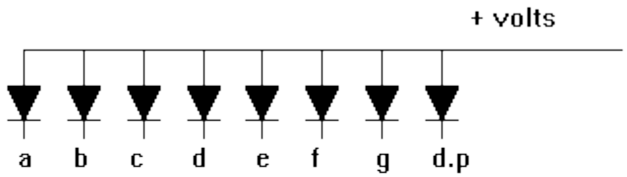


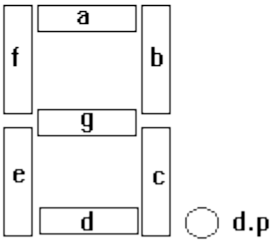
А теперь программно запишем 4 килобайта сигнатуры и тут же считаем. Если верно считался только килобайт (а по остальным адресам считалось не то, что мы записали), значит установлена одна микросхема. Если два килобайта – две микросхемы. Ну, и так далее. И уже затем, программно используем тот объём памяти, какой реально установлен. А помог нам в этой гибкости наш любимый дешифратор.

Дешифратор для семисегментного индикатора

В прошлом я приводил массу схем, где выходы шли «к дешифратору семисегментного индикатора». Это и часы, это и POST-индикатор. Что там ещё было, уже плохо помню. Пришла пора познакомиться с этим дядечкой поближе, чтобы понять, что дешифраторы бывают не только из двоичного кода в позиционный, но и из двоичного в любой другой.

Что такое семисегментный индикатор? С точки зрения схемотехника, это всего лишь восемь светодиодов, имеющих имена a, b, c, d, e, f, g, d.p (так принято). Располагаются они следующим образом:





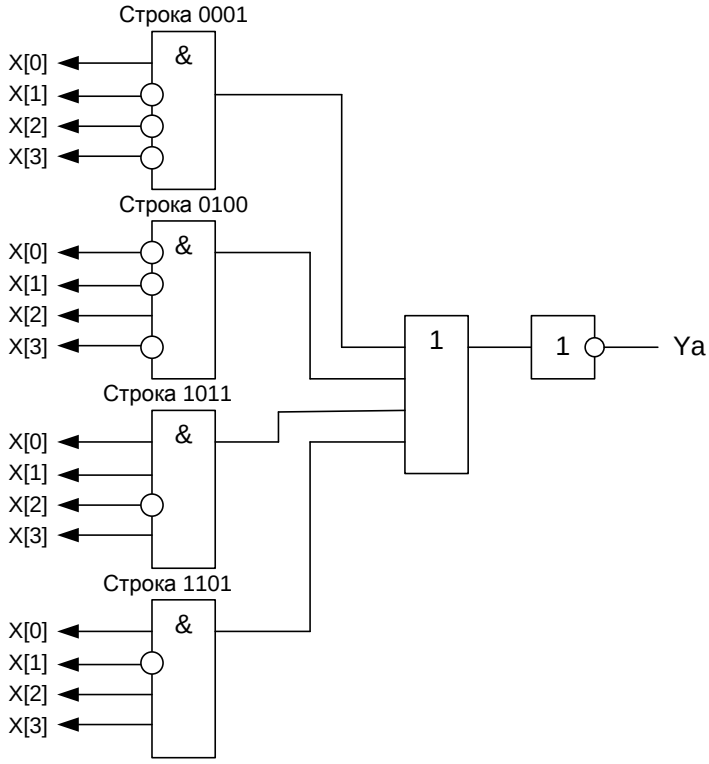
Если нам нужно зажечь цифру 0, мы зажигаем светодиоды a, b, c, d, e и f. Если цифру 1 – светодиоды b и c. Но давайте лучше это покажем в виде таблицы истинности. Сразу оговорюсь, что раньше использовались числа от 0 до 9. Это было в те стародавние времена, когда микросхемы были малой степени интеграции, и приходилось экономить на всём. Однако, четырьмя двоичными битами можно задать не 10, а 16 комбинаций. Теперь, после девятки идут символы A, B, C, D, E, F. Не пугайтесь, увидев их в таблице. Так принято.

X[3..0]	Символ	Ya	Yb	Yc	Yd	Ye	Yf	Yg	Yd.p.
0000	0	1	1	1	1	1	1	0	0
0001	1	0	1	1	0	0	0	0	0
0010	2	1	1	0	1	1	0	1	0
0011	3	1	1	1	1	0	0	1	0
0100	4	0	1	1	0	0	1	1	0
0101	5	1	0	1	1	0	1	1	0
0110	6	1	0	1	1	1	1	1	0
0111	7	1	1	1	0	0	0	0	0
1000	8	1	1	1	1	1	1	1	0
1001	9	1	1	1	1	0	1	1	0
1010	A	1	1	1	0	1	1	1	0
1011	b	0	0	1	1	1	1	1	0
1100	C	1	0	0	1	1	1	0	0
1101	d	0	1	1	1	1	0	1	0
1110	E	1	0	0	1	1	1	1	0

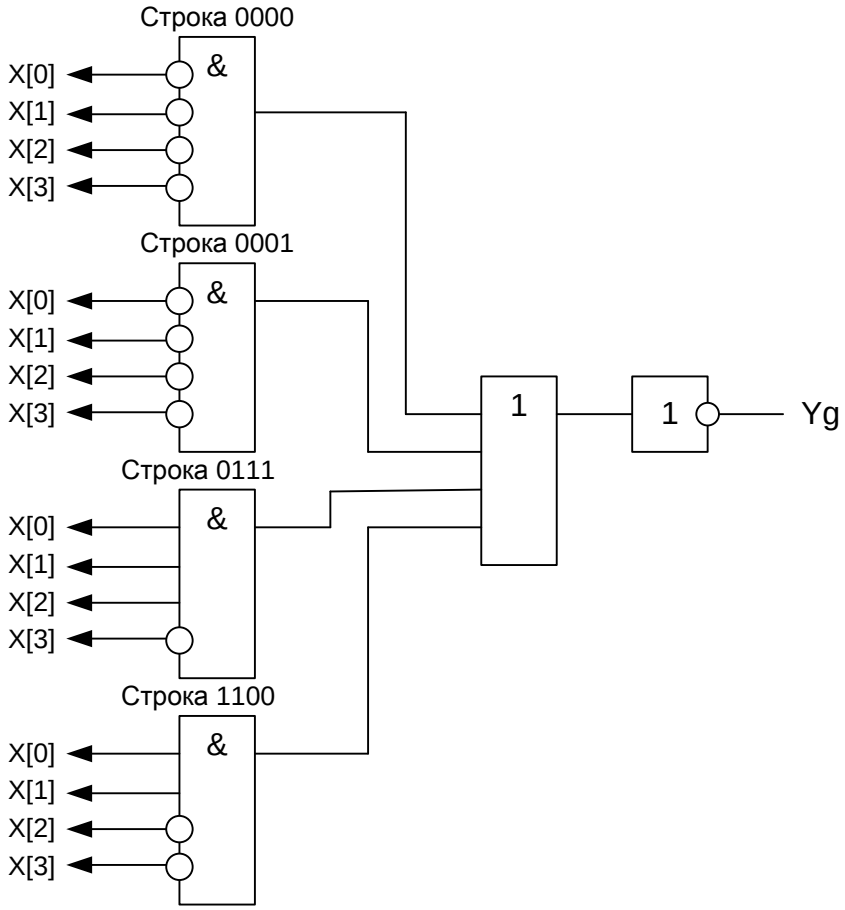
1111		**1**	0	0	0	**1**	**1**	**1**	0

Теперь нам надо построить функции для каждого выхода. Давайте построим для выходов Ya и Yg, а остальные оставим на самостоятельную работу.

Смотрим выход Ya. Там 12 единиц и 4 нуля. Поэтому проще учесть те строки, которые дают ноль, а результат – проинвертировать:



Выход Yg я выбрал не случайно. Там тоже нулей не очень много, поэтому рисовать не так сильно замучаюсь. К сожалению, строк где мало единиц, нет, так что неинвертированную схему не привести.



Вот так, потихоньку, и делается дешифратор для семисегментного индикатора. В жизни, такие дешифраторы берутся либо в магазине (готовая микросхема), либо в библиотеке макросов для ПЛИС. Так что мучаться каждый раз не придётся. Но чтобы понять, как оно работает – стоит разок всё это внимательно изучить, тем более, что из всего этого можно сделать один практический вывод: Если это реализовывать в базисе CPLD, то ёмкость схемы получается уже достаточно большая (используется большое количество элементов И и элементов ИЛИ). А вот если это делать в базисе FPGA, то там, как мы помним, независимо от сложности функции, она всегда определяется таблицей истинности, так что там всё зависит только от числа входов (а входов у нас здесь всего четыре). Таким образом, для реализации дешифратора на семисегментный индикатор, больше подходит FPGA. Подобные прикидки в уме всегда делать полезно, на случай, если нам не спустят сверху готовое железо, а подкинут задачку и предложат самостоятельно выбрать, на чём её решать. А смогли бы вы прикинуть сложность, не зная, как оно устроено внутри? Вряд ли. И мучались бы. А так – сразу прикинете и выберете лучшее.

Ну, на сегодня, всё.