

# Языки описания схем

mk.cs.msu.ru → Лекционные курсы → Языки описания схем

## Блок К6

Кое-что ещё:

SPI для произвольного числа устройств

Состояние высокого импеданса

Лектор:

**Подымов Владислав Васильевич**

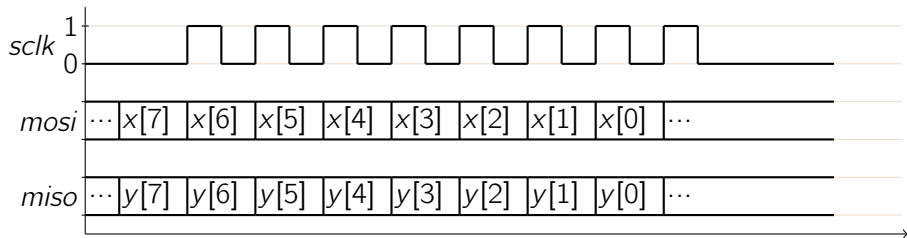
E-mail:

**valdus@yandex.ru**

ВМК МГУ, 2023/2024, осенний семестр

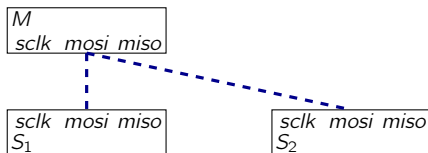
# SPI: общее описание

Пересылка пары сообщений по протоколу SPI:



SPI в общем случае позволяет организовать обмен данными между одним ведущим и произвольным заранее заданным числом ведомых

Для примера представим себе, что хочется передавать данные между ведущим ( $M$ ) и двумя ведомыми ( $S_1$ ,  $S_2$ ):

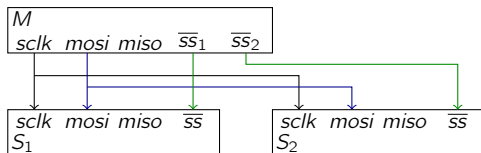


## SPI: sclk, ss, mosi

Передать тактовый сигнал от  $M$  к  $S_1$  и  $S_2$  — это просто

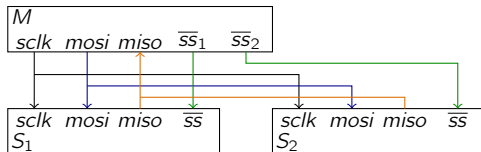
Передать данные от  $M$  к  $S_1$  и  $S_2$  — это чуть сложнее, но тоже достаточно просто:

- ▶ Направим данные от  $M$  на соответствующие входы  $S_1$  и  $S_2$
- ▶ Соединим  $M$  с каждым  $S_i$  дополнительным проводом
  - ▶  $ss$  = slave select, сигнал выбора ведомого:
    - 1  $\Rightarrow$   $M$  обменивается данными с  $S_i$ ;
    - 0  $\Rightarrow$  обмена нет
  - ▶  $\overline{ss}$  — отрицание сигнала  $ss$
  - ▶ иногда вместо  $ss$  используется название  $cs$ : chip select, сигнал выбора схемы



## SPI: miso

Попробуем соединить все порты *miso* настолько же «в лоб», как и порты *mosi*

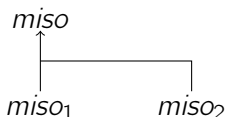


Значения в *sclk*,  $\overline{ss}$  и *mosi* выставляются **одним** устройством: M

Возможность выставить значение в проводе *miso* нужно уметь предоставлять **каждому** ведомому устройству

Если попытаться реализовать передатчики ведомых устройств, используя **только** знания, полученные из всех предыдущих лекций, то возникнет непреодолимая трудность

# SPI: miso



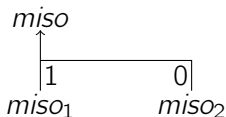
## Непреодолимая трудность

*miso* в ведомом устройстве — **выход** схемы

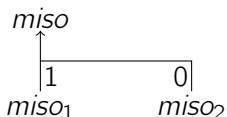
Согласно предыдущим лекциям, в каждый момент времени на этом выходе выставлено одно из значений 0, 1

Предположим (в качестве примера), что

- ▶ если  $ss = 0$ , то в соответствующем ведомом устройстве  $miso = 0$
- ▶  $M$  обменивается данными с  $S_1$
- ▶  $S_1$  передаёт значение 1



# SPI: miso

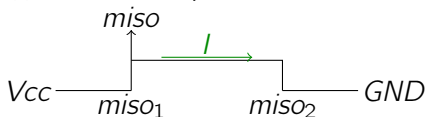


## Непреодолимая трудность

Вспоминаем физику:

- ▶ «в точке  $x$  выставлено значение 0» = «точка  $x$  соединена проводником с источником постоянного напряжения GND»
- ▶ «в точке  $x$  выставлено значение 1» = «точка  $x$  соединена проводником с источником постоянного напряжения  $V_{CC}$ »

Результат такого соединения — короткое замыкание:



Чтобы преодолеть эту трудность, достаточно научиться **изолировать** значение в заданной точке схемы (на выходах  $miso_k$ ) от источников напряжений GND и  $V_{CC}$

# Состояние высокого импеданса (z-состояние)

$Z$  — состояние высокого импеданса, оно же высокоимпедансное, оно же высокоомное, оно же плавающее (floating), оно же просто z-состояние

Это третье «логическое» значение, используемое в схемах наряду с 0 и 1

Точный смысл этого значения в точке  $x$ :

сопротивление  $R$  между  $x$  и остальными точкам схемы очень большое

Насколько большое:

ток  $I$  между  $x$  и остальными точками схемы пренебрежимо мал

С поправкой на небольшую погрешность, это означает, что  $R = \infty$  и  $I = 0$  — то есть точка  $x$  изолирована от остальных, и в частности, от источников напряжений GND и Vcc

Попробуем придумать какой-нибудь «логический» элемент, способный «выдавать» значение  $Z$

# Состояние высокого импеданса (z-состояние)

Спустимся сверху вниз по уровням абстракции описания схемы:

1. Схема — это описание способа реакции одних сигналов на изменение других сигналов (**Verilog**)
2. Схема — это набор регистров и система булевых функций, описывающих взаимосвязь значений на входах и выходах схемы и регистров (**RTL**)
3. Схема — это набор триггеров и логических вентилей (**последовательные схемы** и **комбинационные схемы с обратной связью**)
4. Схема — это набор **КМОП-транзисторов** (**КМОП-схемы**)

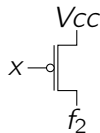
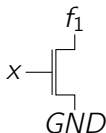
Из всех перечисленных уровней абстракции напряжения GND и Vcc упоминаются явно только на уровне **КМОП-схем**

Научимся изолировать точки схемы от GND и Vcc на этом уровне, и последовательно встроим то, чему научились, в остальные уровни



# Z-состояние: КМОП и вентили

Начнём с простейших КМОП-схем:



Им соответствуют функции с такими таблицами «значений»:

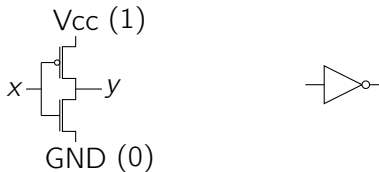
$x$	$f_1$
0	$\mathcal{Z}$
1	0

$x$	$f_2$
0	1
1	$\mathcal{Z}$

## Z-состояние: КМОП и вентили

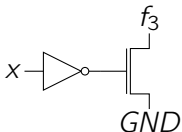
**Напоминание:** неконстантная булева функция с самой простой КМОП-реализацией — это отрицание:

x	$y = \bar{x}$
0	1
1	0

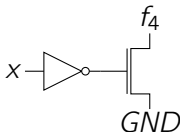


Попробуем реализовать такие функции, используя простейшие КМОП-схемы и отрицание:

x	$f_3$
0	0
1	Z



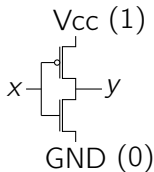
x	$f_4$
0	Z
1	1



# Z-состояние: КМОП и вентили

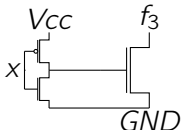
**Напоминание:** неконстантная булева функция с самой простой КМОП-реализацией — это отрицание:

$x$	$y = \bar{x}$
0	1
1	0

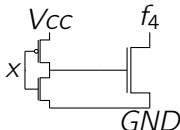


Попробуем реализовать такие функции, используя простейшие КМОП-схемы и отрицание:

$x$	$f_3$
0	0
1	$\mathcal{Z}$



$x$	$f_4$
0	$\mathcal{Z}$
1	1



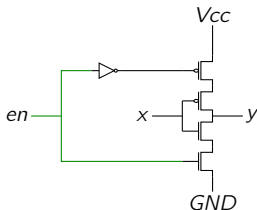
## Z-состояние: КМОП и вентиля

Рассмотрим функцию посложнее:

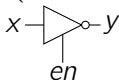
отрицание с дополнительным входом, управляющим *изоляцией* выхода

$y = f_5(x, en)$		$en$	
		0	1
$x$	0	Z	1
	1	Z	0

КМОП-реализация:



Это **управляемый инвертор**, он же **инвертор с третьим состоянием** (tri-state inverter)



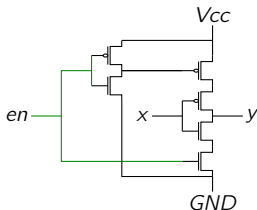
## Z-состояние: КМОП и вентиля

Рассмотрим функцию посложнее:

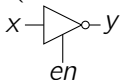
отрицание с дополнительным входом, управляющим *изоляцией* выхода

$y = f_5(x, en)$		$en$	
		0	1
$x$	0	$Z$	1
	1	$Z$	0

КМОП-реализация:



Это **управляемый инвертор**, он же **инвертор с третьим состоянием** (tri-state inverter)

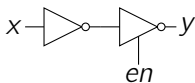


## Z-состояние: КМОП и вентили

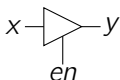
Попробуем аналогично расширить не отрицание, а тождественную функцию:

$y = f_6(x, en)$		$en$	
		0	1
$x$	0	$Z$	0
	1	$Z$	1

Эту функцию можно реализовать на уровне вентилей, не «спускаясь» на уровень КМОП-транзисторов:



Это **управляемый буфер**, он же **буфер с третьим состоянием** (tri-state buffer)

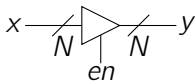


# Z-состояние: RTL

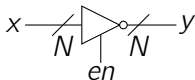
Вспоминаем, что такое RTL:

- ▶ Последовательная схема = триггеры + вентили
- ▶ RTL-схема = регистры + булевы функции
- ▶ Регистр = набор триггеров + входные шины + выходные шины

Управляемый буфер ширины  $N$  в зависимости от значения  $en$  либо изолирует выход  $y$  ширины  $N$ , либо направляет на него вход  $x$  ширины  $N$



Управляемый инвертор ширины  $N$  в зависимости от значения  $en$  либо изолирует выход  $y$  ширины  $N$ , либо направляет в каждый разряд этого выхода отрицание соответствующего разряда на входе  $x$  ширины  $N$



# Z-состояние: Verilog

## Программная семантика

$Z$  — такое же логическое значение, как и 0, 1,  $X$

При использовании в качестве аргументов операций  $Z$  в большинстве случаев идентично значению  $X$

`casez` — команда, аналогичная `casex` и использующая  $Z$  вместо  $X$  в качестве «любого» значения



# Z-состояние: Verilog

## Аппаратная семантика

### Разрешено:

- ▶ Как и для  $\mathcal{X}$ :  
**явно** использовать значение  $\mathcal{Z}$  в правых частях присваиваний
  - ▶ Если значение  $\mathcal{Z}$  используется неявно или комбинируется с другими значениями, то оно заменяется на  $\mathcal{X}$  со всеми соответствующими разрешениями и запретами
- ▶ Использовать  $\mathcal{Z}$  в команде `casez`  
так же, как  $\mathcal{X}$  разрешено использовать в `casex`
- ▶ Присваивать заданной точке значение **из нескольких процессов**, если *на уровне компиляции* достоверно известно, что в каждый момент времени значение, отличное от  $\mathcal{Z}$ , задано не более чем одним процессом
  - ▶ Если каким-либо процессом задаётся значение, отличное от  $\mathcal{Z}$ , то значения  $\mathcal{Z}$ , задаваемые остальными процессами, игнорируются
  - ▶ Иначе итоговое значение в точке —  $\mathcal{Z}$

### Запрещено всё остальное

# Z-состояние: Verilog

## Примеры

```
wire [3:0] i, o;  
wire en;  
assign o = en ? i : 4'bz;
```

```
wire [3:0] i;  
wire en;  
reg [3:0] o;  
always @*  
    if(en) o = i;  
    else o = 4'bz;
```

Это поддерживаемые реализации управляемого буфера ширины 4

# Z-состояние: Verilog

## Примеры

```
wire [1:0] en;  
wire a, b, c;  
assign c = (en == 2'd1) ? a : 1'bz;  
assign c = (en == 2'd2) ? b : 1'bz;
```

Это поддерживаемый код, задающий значение в точке c согласно следующей таблице:

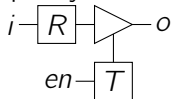
		<i>en</i>			
		00	01	10	11
<i>a b</i>	00	Z	0	0	Z
	01	Z	0	1	Z
	10	Z	1	0	Z
	11	Z	1	1	Z

# Z-состояние: Verilog

## Примеры

```
wire [3:0] i;  
wire en;  
reg [3:0] o;  
always @(posedge clk)  
    if(en) o <= i;  
    else o <= 4'bz;
```

Это поддерживаемый код, который, согласно написанному в стандарте, может иметь следующую аппаратную семантику:



- ▶  $R$  — параллельный регистр, запоминающий значение  $i$  (если  $en = 0$ , то поведение  $R$  произвольно)
- ▶  $T$  — D-триггер, запоминающий значение  $en$
- ▶ если в  $T$  сохранено значение 1, то  $o$  совпадает с выходом  $R$ , а иначе выход  $o$  изолирован

# Z-состояние: Verilog

## Примеры

```
reg a, b;  
always @* begin  
    a = 1'bz;  
    b = a;  
end
```

Это неподдерживаемый код:

значение  $Z$  в присваивании для  $b$  используется неявно

# Z-состояние: Verilog

## Примеры

```
wire a, b, c;  
assign a = 1'bz;  
assign c = a && b;
```

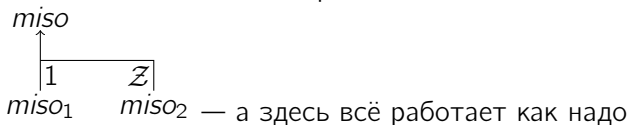
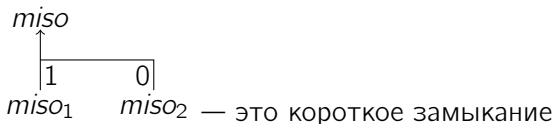
Это неподдерживаемый код:  
значение  $Z$  в присваивании для  $c$

- ▶ используется неявно и
- ▶ комбинируется со значением  $b$

## SPI: miso

Теперь можно легко преодолеть **трудность**, связанную с выставлением значения в *miso* из несколькими точеч:

- ▶ Если  $M$  обменивается данными с  $S$ ,  
то  $S$  выставляет в *miso* значения 0, 1 согласно протоколу
- ▶ Если  $M$  не обменивается данными с  $S$ ,  
то  $S$  выставляет в *miso* значение  $\mathcal{Z}$  — **изолирует** от себя этот порт



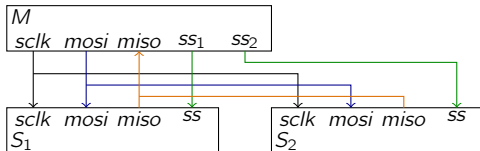
Такое соединение (с динамическим подключением и изоляцией устройств на концах) обычно называется **шиной**<sup>1</sup> и **магистралью**

<sup>1</sup> Я уже упоминал, что у термина «шина» много значений

# SPI: соединение портов

## Итоги

Все порты в реализации SPI для многих устройств можно соединить «в лоб»:



*(давайте для простоты забудем про то, что для выбора ведомого устройства используется порт  $\overline{ss}$ , а не  $ss$ )*

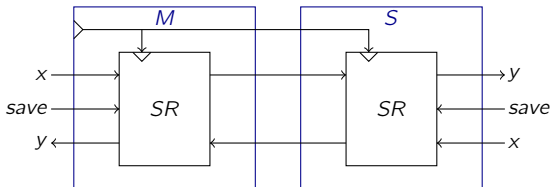
Если  $ss = 1$ , то выход *miso* соединён со схемой соответствующего ведомого устройства  $S_i$ , и  $S_i$  передаёт разряды сообщения согласно протоколу SPI для двух устройств

Если  $ss = 0$ , то выход *miso* изолирован от источников напряжения ведомого устройства  $S_i$ , и  $S_i$  не передаёт разряды сообщения



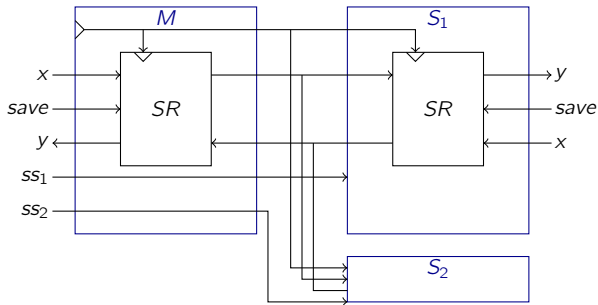
## SPI: реализация

Вспомним, как выглядела реализация приёмника и передатчика по протоколу SPI для двух устройств:



Адаптируем эту реализацию к общему случаю  
(вернее, к рассматриваемому упрощённому случаю  
с двумя ведомыми устройствами  
с очевидным обобщением на произвольное число ведомых устройств)

## SPI: реализация



Будем полагать, что сигналы  $ss_1$  и  $ss_2$  поступают на входы  $M$

Тогда можно легко модифицировать схемы так:

- ▶ добавим в сериализатор  $SR$  ведомого устройства вход  $en$ :
  - ▶  $en = 1 \Rightarrow$  сериализатор работает как обычно
  - ▶  $en = 0 \Rightarrow$  сериализатор не сдвигает хранящееся число
- ▶ добавим в ведомое устройство **управляемый буфер**, подходящим образом изолирующий и подключающий выход  $miso$

# SPI: реализация

Итог:

