

Конспект к экзамену по билетам
(архаичные ЭВМ)
1-й семестр

Латыпов Владимир (конспектор)

t.me/donRumata03, github.com/donRumata03, donrumata03@gmail.com

Скаков Павел Сергеевич (лектор)

t.me/pavelxs

25 января 2022 г.

Содержание

1	Введение	3
2	Названия билетов (ровно как в оригинале)	3
3	О чём говорить при каждом из билетов?	4
3.1	Элементная база вычислительной системы: логические элементы, триггеры.	4
3.1.1	МЕМ	4
3.1.2	JK-триггер	4
3.1.3	Физические основы работы логических элементов	4
3.1.4	Дребезг контактов	6
3.2	Оперативная память: статическая/динамическая, организация.	6
3.3	Оперативная память: характеристики, типы динамической памяти. NUMA.	7
4	Кэш-память.	8

1. Введение

Максимально сжатый материал: если читатель не знаком с курсом, возможно, стоит сначала изучить конспект Тимофея на Overleaf.

2. Названия билетов (ровно как в оригинале)

1. Устройство памяти

- 1.1. Элементная база вычислительной системы: логические элементы, триггеры.
- 1.2. Оперативная память: статическая/динамическая, организация.
- 1.3. Оперативная память: характеристики, типы динамической памяти. NUMA.
- 1.4. Кэш-память.
- 1.5. Протоколы когерентности кэш-памяти.
- 1.6. Носители информации: магнитные, оптические и на основе флеш-памяти. RAID.

2. Архитектура процессорных систем

- 2.1. Архитектура фон Неймана и её альтернативы.
- 2.2. Архитектура набора команд (ISA) и микроархитектура.
- 2.3. Конвейерная архитектура. Конвейер MIPS.
- 2.4. Проблемы конвейера (hazards) и пути их решения.
- 2.5. Суперскалярная и VLIW архитектуры. Спекулятивное исполнение. Уязвимости классов Spectre и Meltdown.
- 2.6. Многоядерные/многопроцессорные системы, одновременная многопоточность (SMT/HT).

Нужно ответить на два вопроса: по одному из каждой части. Пользоваться ничем нельзя, отвечать сразу.

3. О чём говорить при каждом из билетов?

3.1. Элементная база вычислительной системы: логические элементы, триггеры.

Европейские и американские обозначения логических элементов

Полусумматоры и сумматоры (медиана + XOR или два полусумматора + OR)

RS-триггер, каноническая и наиболее эффективная версия — через два ИЛИ-НЕ.

Проблемы высоких частот и малого размера, «В Ethernet соотношение сигнал/шум — как разговаривать рядом с турбиной самолёта»

⇒ Синхронная версия RS-триггера, D-триггер через «не» на входе.

3.1.1. МЕМ

Декодер 3to8 (для каждого из 8 выходов AND от трёх, возможно, инвертированных входов) ⇒ мультиплексор ($3 + 8 \rightarrow 1$) и демультимплексор ($3 + 1 \rightarrow 8$).

Из этого можно сделать модуль памяти «Мем» на 8 бит:

Входы: 3 адресных бита, R/W , C (clock), D (запись, если выбран режим W).

Выход: «Q» — если выбран режим R (не обязательно только в этом случае) — на нём значение, соответствующее биту по адресу $A_0A_1A_2$.

3.1.2. JK-триггер

JK-триггер (Входы J , K , синхронный, \Leftrightarrow умеет инвертировать состояние при двух единицах). Крафтится из двух RS-триггеров с тактированием в противофазе, на вход первого кроме оригинальных входов J , K через «AND» даётся то, что запомнил второй (то есть что было на первом до начала такта). В нормальной версии результат (Q и \tilde{Q}) берётся из выходов первого триггера.

3.1.3. Физические основы работы логических элементов

Нас интересуют в основном транзисторы. По многим причинам победила $CMOS$ -логика, конструирующая только транзисторы (причём

полевые) для конструкции элементов. Это позволяет минимизировать потребляемую энергию, так как ток течёт только при переключении. В отличие от полевых транзисторов (где он нужен через базу для поддержания открытого состояния), а также *NMOS* логики, где он течёт по резистору в некоторых случаях (когда транзистор открыт).

В случае *CMOS* логики расход энергии растёт при увеличении частоты почти линейно по понятным причинам.

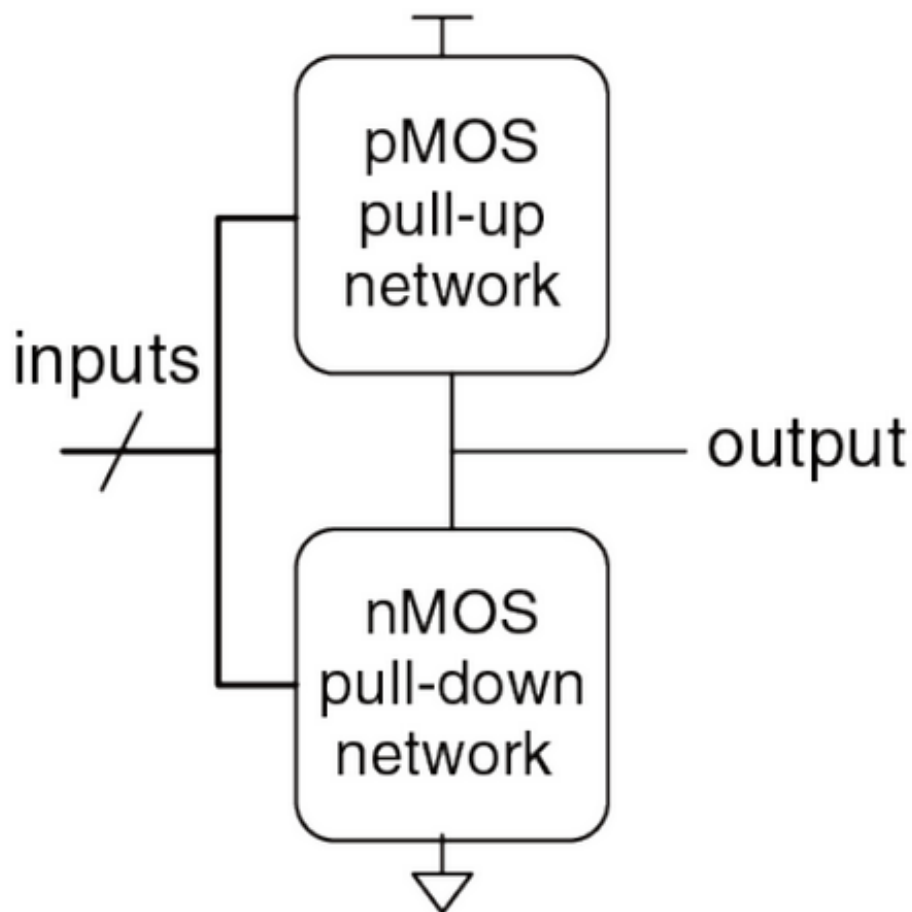


Рис. 1: Общий принцип построения схем в CMOS логике

В качестве элементарных частиц используем *NOT*, *NAND*, *NOR* (нельзя сделать *AND* и *OR* эффективнее, чем соответствующий *NAND/NOR* ◦ *NOT*)

Fun Fact: можно сделать *NOR* и *NAND* на много входов эффективнее,

чем просто внешне композировать (например, для трёх — 6 вместо 8-и).

3.1.4. Дребезг контактов

Кнопки из реальной жизни при нажатии не сразу устанавливаются в новое состояние, а сначала колеблется. Это называется *Contact bounce* (дребезг контактов). Причём чем старше и некачественнее контакты, тем дольше будет происходить *bouncing*.

Есть несколько подходов к борьбе с ним. Бывает, программно, бывает аппаратно. Причём кроме вопросов реализации возникают ещё и концептуальные. Проще всего реагировать на нажатие с запозданием: по истечении времени с начала или с последнего изменения. Другой вариант — реагировать как только произошло изменение после затишья, но после этого игнорировать дребезг, пока не установится. Однако, если у нас не просто бинарная кнопка с одним контактом «нажат/не нажат», а имеющая положения «не нажата, положение 1 и положение 2» и мы хотим на выход подавать бинарный сигнал в виде последнего «кас'анного» контакта, то всё гораздо очевиднее и у нас есть ультимативное решение через триггер и, возможно, транзистор. Не забыть про подтягивающие резисторы.

3.2. Оперативная память: статическая/динамическая, организация.

Память располагают в 2D решётчатой структуре.

Количество проводов $\propto cols + rows \Rightarrow O(\sqrt{memory_size})$

Row отвечает за выбор ряда. Если он ноль, ячейка вообще не работает.

Если подаётся *true*, ячейка открывается. Можно считывать информацию с соответствующих *Col* и \overline{Col} . Также можно записывать на эти входы.

Статическая память строится как

	Статическая память	Динамическая память
Скорость	быстрая	медленная
Количество транзисторов	Много (≈ 6)	Мало (обычно один + конденсатор)
Надёжность	Наличествует	Постоянно дегенерирует \Rightarrow надо регенерировать
Примеры	Регистры, кэш	Оперативная память, видеопамять

Параметры, по которым оценивается память:

Объём,

скорость доступа («latency»),

скорость передачи («throughput», пропускная способность)

Уязвимость Row-Hammer: если очень долго обращаться к ячейкам, соседним с данной, можно установить её в ноль.

3.3. Оперативная память: характеристики, типы динамической памяти. NUMA.

Адресуемся в порядке: старшие биты адреса — номер строки, младшие — номер столбца.

Типичные интерфейсные выводы оперативки:

.

Порядок работы с памятью:

1. Процессор сообщает о таймингах
2. Всё время идет синхронизация
3. Выбор строки \Rightarrow модуль динамической памяти заносит строку в статическую
4. Работа со строкой: запрос адреса
5. После передачи данных и выбора следующей строки происходит «закрытие» старой — в динамические ячейки вновь записываются данные из статической строки.
- 6.

Что какие тайминги означают?

Аббревиатуры:

- RAS — Row Access Strobe — сигнал выборки строки
- CAS — Column Access Strobe — —" — столбца
- CL — CAS Latency — с получения адреса столбца до ответа
- t_{RCD} — RAS to CAS Delay —
- t_{RAS} —

Производители обычно указывают либо 4 тайминга:

$$CL - t_{RCD} - t_{RP} - t_{RAS}$$

Либо только CL .

Причём измеряется это в количестве тактов при какой-то заданной частоте.

Существуют двух- и более портовые ячейки памяти. Есть тот же бит, но много интерфейсных транзисторов. На каждый — новые « Col_k », « Row_k »

На практике используется редко, так как сложно изготавливать.

Бывают ещё несколько матриц, процессор работает с ними отдельно. Другой вариант — многогранговая память, когда процессор видит одну память, но на самом деле их несколько. Но так большая нагрузка на модуль памяти (тот PU , который внутри планки оперативки).

4. Кэш-память.

Это такой большой костыль, возникший из-за того, что процессоры стали быстрее памяти. Находится близко к вычислительным ядрам, на кристалле процессора. Делает то, о чём говорит его название: кэширует запросы к памяти. Имеет меньший размер, чем оперативка, но работает быстрее.