Программирование микросхем ПЛИС с помощью свободного программного инструментария (Yosys)

Залата Руслан Николаевич
Директор ООО «Фабмикро»,
г. Тюмень.
E-mail: rz@fabmicro.ru

ПЛИС про-ва «Xilinx» -->

Что такое ПЛИС ?

ПЛИС это такая **интегральная схема** (микросхема) которая содержит внутри себя некоторое количество (массив или матрицу) единообразных и достаточно примитивных ресурсов назначение которых не определено заранее, а задается конфигурацией загружаемой на стадии инициализации. Таким образом, **ПЛИС** это **Программируемая Логическая Интегральная Схема.** Англоязычный термин: **FPGA (Field Programmable Gateway Array)** — массив вентилей программируемых в «полевых» условиях.

Микросхемы ПЛИС используются для создания цифровых схем. Процесс их создания правильней называть «проектирование», а не «программирование», так как внутри микросхем ПЛИС нет заранее созданных программируемых устройств. Но так как процесс проектирования цифровой схемы связан с написанием кода на специальных формальных языках с последующим переводом (или синтезом) в двоичный формат, то в ходу плотно закрепился термин «программирование».

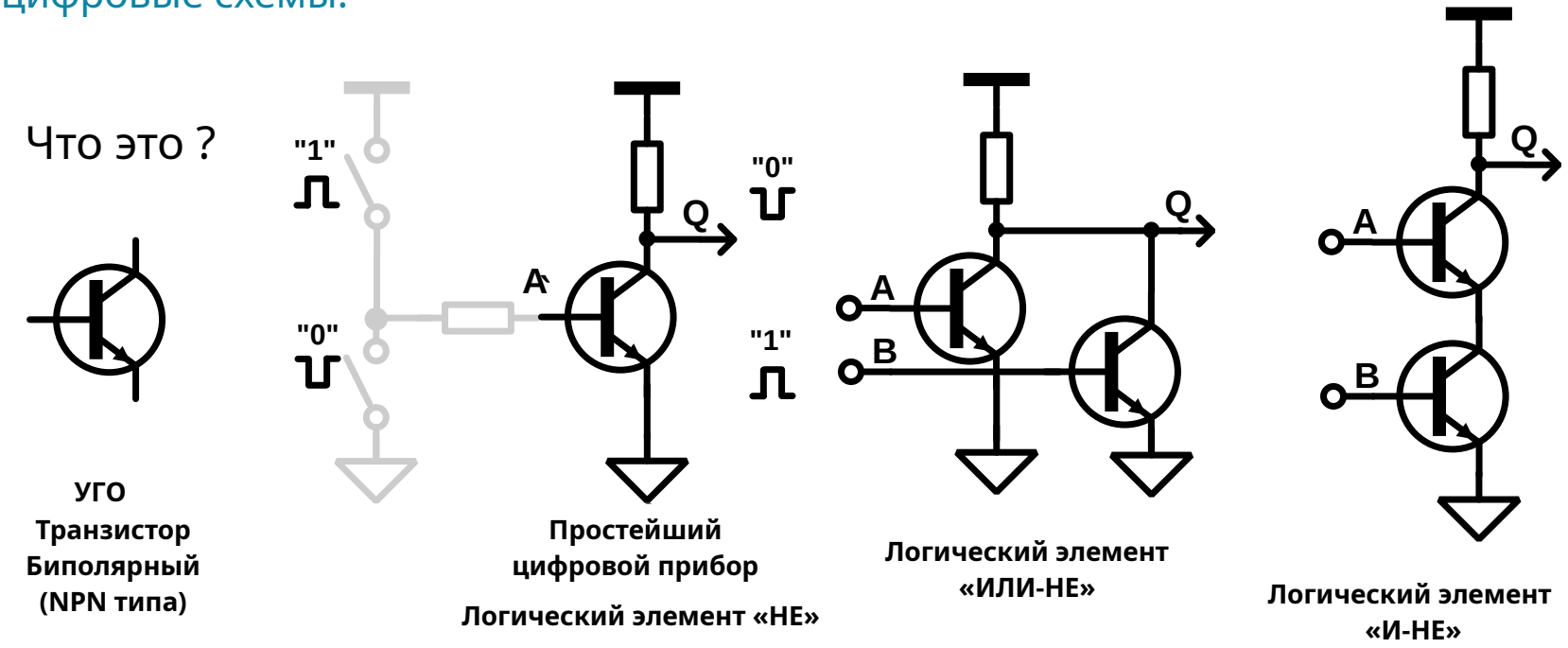
Далее будем **говорить** о **программировании** ПЛИС **подразумевая проектирование** цифровых схем и их синтеза для ПЛИС.

Часть 1

Устройство цифровых схем

Цифровые устройства и двоичная логика

Перед тем как говорить об устройстве микросхем ПЛИС и ресурсах, которые они предоставляют разработчику, давайте вспомним что из себя представляют цифровые схемы.



Резисторно-Транзисторная Логика — 1956-1965 годы.

Недостатки: а) потребляет большой ток, б) чувствительна к просадкам напряжения, в) небольшая нагрузочная способность, г) низкая частота переключения состояний

Другие виды цифровой транзисторной логики

Исторически развитие цифровой техники прошло в несколько этапов:

1. РТЛ - резисторно-транзисторная логика.

2. ДТЛ - диодно-транзисторная логика.

3. ТТЛ - транзисторно-транзисторная логика.

4. ТТЛШ - транзисторно-транзисторная логика с диодами Шотки.

5. ЭСЛ - эмиттерно-связанная логика — использует биполярные транзисторы с множеством эмиттеров и коллекторов**.** На конец 80-х частоты ЭСЛ достигали **2ГГц** (Википедия). Серия ЕС ЭВМ была постоена на ЭСЛ.

6. МОП (МДП) - логика на униполярных (полевых) транзисторах с каналами проводимости p- и n-типов.

7. КМОП (КМДП) - логика на основе полевых транзисторов с использованием дополняющих комплементарных транзисторов.

КМОП - комплементарная структура металл-оксид-полупроводник

Металл-оксид-Полупроводник (MOS) - это способ изготовления полевых транзисторов планарным фото-литографическим методом, когда на поверхности кремниевой пластины создаются «колодцы» из проводника **n-** или **p-** типа, поверх которого наносится тонкий слой изолятора (из поликристаллического кремния или его оксида), а на слой изоляции накладывается слой металла (алюминий) который выполняет функцию затвора у образовываемых таким образом полевых транзисторов.

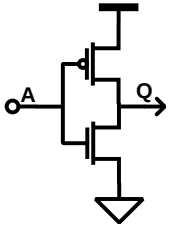
Комплиментарная структура МОП (**CMOS**) использует полевые транзисторы сразу двух типов: с **N-**каналом и **P-**каналом. При этом **исток** транзистора с P-каналом всегда имеет **связь с шиной питания**, а **исток** транзистор с N-каналом всегда связан с **шиной «земли»**. Логические схемы формируются из таких структур путем **связывания стоков с затворами** путем нанесения слоя металлизации. Такие структуры принято называть «**комплиментарной парой**»

Логический элемент «НЕ»

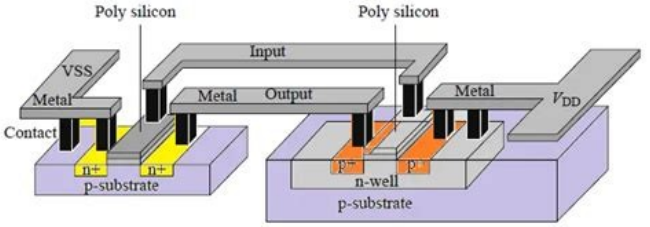
A	Q
0	1
1	0

Таблица истинности

УГО в дизайне цифровой схемы



Комплиментарная пара в схеме микросхемы



Произведенный фото-литографическим методом

Особенности цифровых схем на КМОП

1. Логические схемы построенные по КМОП технологии **в статическом состоянии** почти **не потребляют** электрический ток. Связано это с тем, что полевые транзисторы управляются напряжением, а не током как биполярные транзисторы в РТЛ. Через затворы полевых транзисторов в статическом состоянии не протекает ток.

Схемы на КМОП очень энергоэфективны.

2. Структура канал-затвор планарных полевых транзисторов образуют небольшие паразитные электрические ёмкости. При переключении состояния транзисторов происходит перетекание зарядов (затворы заряжаются и разряжаются), а следовательно в этот момент через затворы протекает ток. Чем чаще происходит переключение состояния, там больше времени транзисторы находятся в состоянии перемещения зарядов и тем больший ток потребляется схемой (вспомним наши смартфоны которые нагреваются как печки).

Одна из задач разработчика — отключать тактирование неиспользуемых участков схемы.

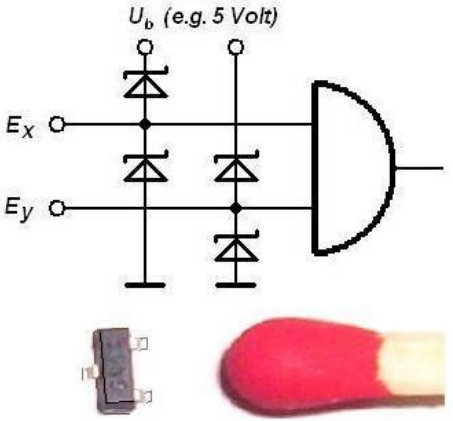
3. Паразитная емкость затворов определяет скорость реакции транзисторов, и, как следствие, ограничивает максимальную частоту переключений. **Чем меньше размер подзатворного пространства**, тем меньше паразитная ёмкость, **тем выше частота работы цифровой схемы**.

Увеличение тактовой частоты требует уменьшение размеров транзисторов.

Недостаток КМОП

Схемы на КМОП очень сильно подверженны влиянию статического электричества. Статические разряды образующиеся на теле человека легко выводят из строя комплиментарные пары в КМОП схемах. По этому все входные цепи КМОП схем требуется защищать от статики (ESD — Electro Static Discharge).

Схемы ESD-защиты устанавливаются по входам комплиментарных пар как во внутрь интегральных схем при их производстве, так и с помощью внешних компонентов.

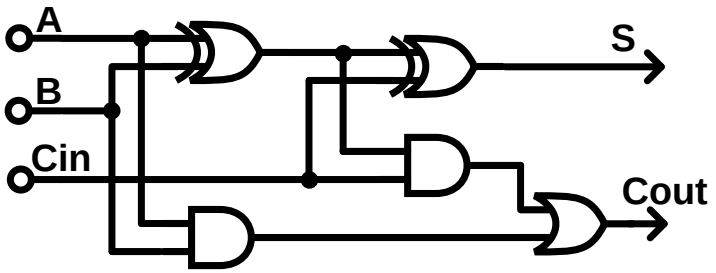


Компоненты цифровых схем

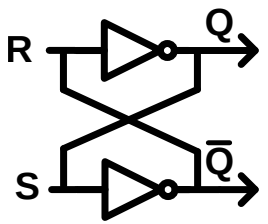
Цифровые схемы строятся из двух видов компонентов:

1. Элементы логики, такие как «НЕ», «И-НЕ», «ИЛИ-НЕ» и т. д. — из них складываются **комбинационные схемы** (combinational logic - **CL**). Комбинационные схемы занимаются преобразованием входного сигнала в выходной.

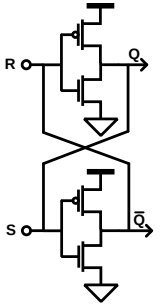
2. Элементы памяти (триггеры или регистры) — они сохраняют состояние обрабатываемых сигналов на какое-то время. Из регистров складываются **последовательностные** или **синхронные** схемы (sequencial logic - **SL**).



Полный одноразрядный двоичный сумматор (1-bit full adder)
Базовый блок для АЛУ



Простейшая бистабильная ячейка памяти (RS-триггер)



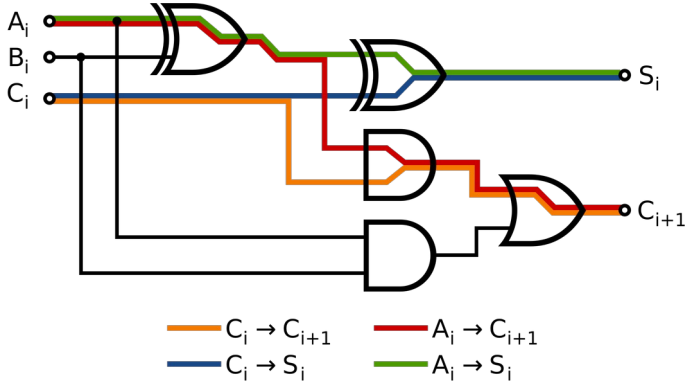
RS-триггер в КМОП исполнении

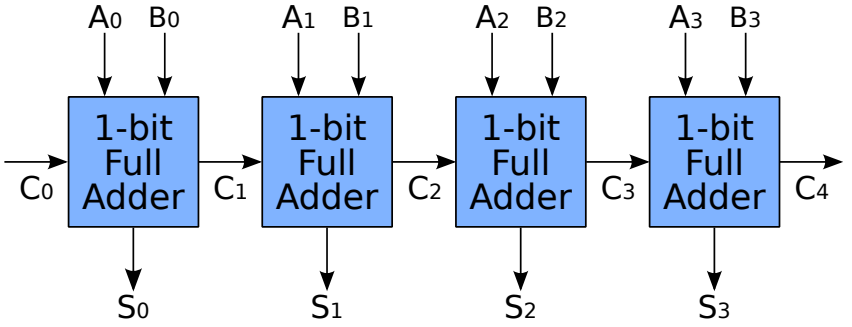
Физические свойства комбинационных схем

1. Комбинационные схемы работают **асинхронно**, т. е. входной сигнал может поступать в любой момент времени, а результат на выходе появляется «почти» сразу после изменения состояния входных линий.

2. Время проходящее от момента подачи входного сигнала до появления стабильного выходного сигнала называют **время задержки распространения** (**Tpd — propagation delay**).

3. Это время Tpd зависит: а) от физических свойств транзисторов из которых строятся логические элементы, и б) от максимальной длины цепочки элементов внутри схемы.





4. Красной линией обозначен **критический путь** — это максимально путь распространения сигнала. При каскадном соединении одноразрядных сумматоров критический путь увеличивается пропорционально числу разрядов, соответственно растет **Tpd**.

Задача разработчика цифровых схем состоит в том, чтобы минимизировать задержку распространения.

Логические свойства комбинационных схем

Комбинационные схемы подчиняются правилам Булевой алгебры: **коммутативность**, **идемпотентность**, **ассоциативность**, **дистрибутивность**, **поглощение,** законы **Де Морагана** и т.д. А это значит, что такие схемы можно преобразовывать без потери функциональности - часто схему с очень длинным критическим путем можно упростить.

Некоторые правила преобразования

Аксиомы:

1. $\bar{\bar{x}} = x,$
2. $x \vee \bar{x} = 1$
3. $x \vee 1 = 1$
4. $x \vee x = x$
5. $x \vee 0 = x$
6. $x \wedge \bar{x} = 0$
7. $x \wedge x = x$
8. $x \wedge 0 = 0$
9. $x \wedge 1 = x$

Дистрибутивность:

- $x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z),$
- $x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z),$
- $x \wedge (y \oplus z) = (x \wedge y) \oplus (x \wedge z).$

Законы де Моргана:

- $\overline{x \wedge y} = \bar{x} \vee \bar{y},$
- $\overline{x \vee y} = \bar{x} \wedge \bar{y}.$

Законы поглощения:

- $x \wedge (x \vee y) = x,$
- $x \vee (x \wedge y) = x.$

Обозначение логических операций:

¬ - отрицание (NOT), унарная;

∨ - дизъюнкция (OR), бинарная;

∧ - конъюнкция (AND), бинарная;

⊕ - сложения по модулю 2 (искл. ИЛИ, XOR), бинарная.

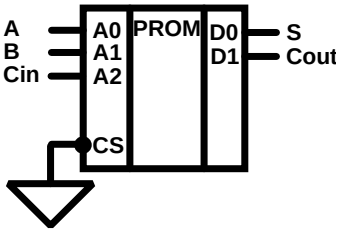
Таблицы истинности

1. Любую комбинационную схему можно представлять в виде таблицы истинности. Для представления схемы с **N** входами требуется не более 2^N строк. Такие таблицы принято называть «**look-up table**» (**LUT**).

2. Аппаратно, таблицы истинности реализуются в виде **блоков статической памяти** или **Программируемого ПЗУ**, где входами являются линии адреса (**A**x), а выходами — линии данных (**D**x).

Входы			Выходы	
Cin	A	B	Cout	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Таблица истинности для полного одноразрядного сумматора



Полный одноразрадный сумматор на ППЗУ (16 бит)

Как построить 4-х разрядный полный сумматор на ППЗУ ?

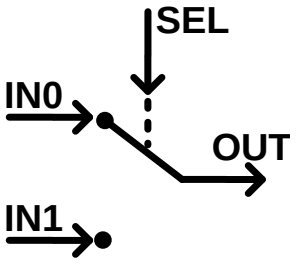
Из одной ППЗУ: 2^9 * 5 = 2560 бит — **быстро**.
Из четырех ППЗУ: 2^3 * 2 * 4 = 64 бит — в **4 раза** медленнее!

LUT на основе мультиплексоров

Еще один способ реализации таблиц истинности в цифровых схемах, основывается на использовании **логических мультиплексоров**.

Мультиплексор (MUX) - это электронно управляемый переключатель цифровых сигналов.

Мультиплексоры можно соединять каскадами и получать MUX большей разрядности: 4:1, 8:1 и т.д.



MUX 2:1

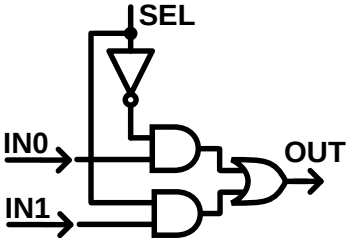
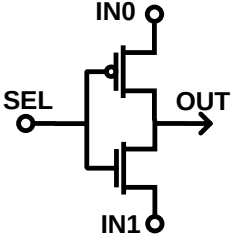
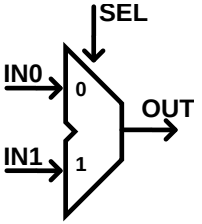


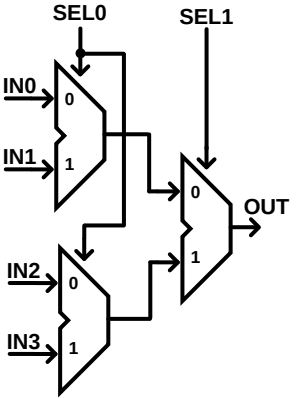
Схема MUX 2:1
на лог. элементах



MUX 2:1
на на полевых
транзисторах



MUX 2:1
на цифровых
схемах



MUX 4:1
из 3-х MUX 2:1

LUT на основе мультиплексоров

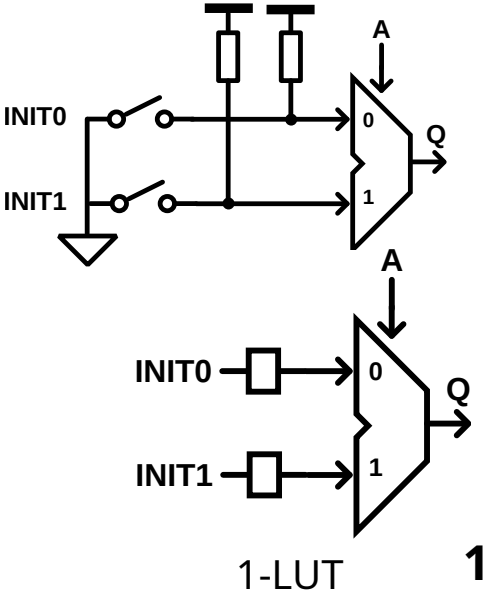
С помощью **MUX 2:1** можно легко сконструировать 1-входовую таблицу истинности. Для этого необходимо на входы MUX подать заранее **известную комбинацию**, а на управляющий вход **SEL** подавать входной сигнал **A**. При этом на выходе MUX мы будем получать сигнал **Q = f(A)** и эта функция однозначно определяется комбинацией сигналов на входах MUX **INIT0** и **INIT1**.

Из таблицы ниже видно, что при INIT0 = 1 и INIT1 = 0, данная схема работает как логическое «НЕ».

Заменим переключатели на ячейки памяти и получим программируемую таблицу истинности **1-LUT**.

Ячейки INIT0 и INIT1 называются **конфигурационными.**

INIT0	INIT1	A	Q	f(A)
0	0	0	0	Константа 0
0	0	1	0	Константа 0
1	0	0	1	«НЕ» (NOT)
1	0	1	0	«НЕ» (NOT)
0	1	0	0	«Повторение» (BUF)
0	1	1	1	«Повторение» (BUF)
1	1	0	1	Константа 1
1	1	1	1	Константа 1



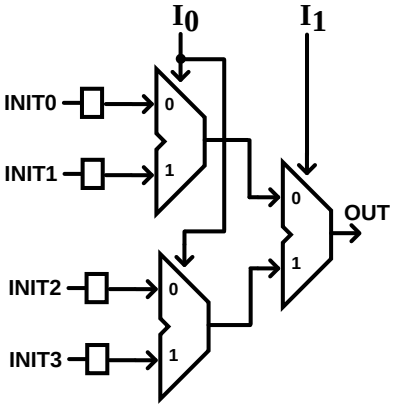
1-LUT

2-LUT, 3-LUT, 4-LUT

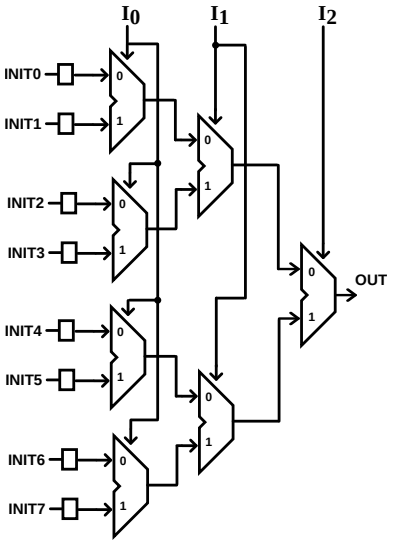
Мы знаем, что мультиплексоры можно соединять каскадом, а значит мы можем получить таблицы истинности с больш**и**м числом входов.

Соединив **две** схемы **1-LUT** с помощью **MUX 2:1** мы получим схему **2-LUT**.

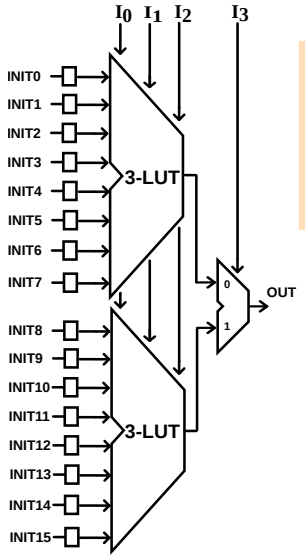
Соединив **две** схемы **2-LUT** все тем же **MUX 2:1** мы получим схему **3-LUT**.



2-LUT.
Содержит 4 конфигурационных ячейки памяти



3-LUT.
8 конфигурационных ячеек памяти



4-LUT из 2 x 3-LUT.
16 конфигурационных ячеек памяти

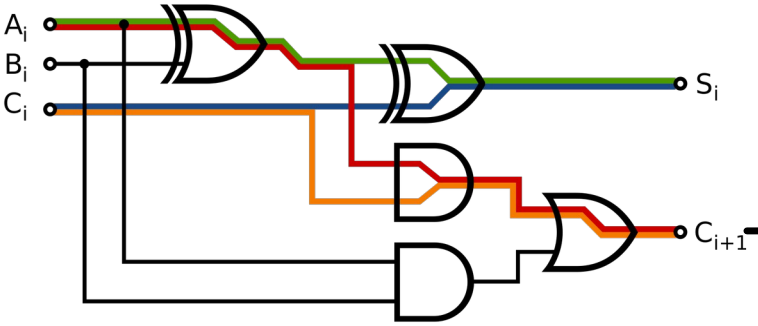
Схемы 2-, 3- и 4-LUT являются базовыми для создания устройств программируемой логики

Ячейки памяти с разрешающим сигналом (D-триггеры)

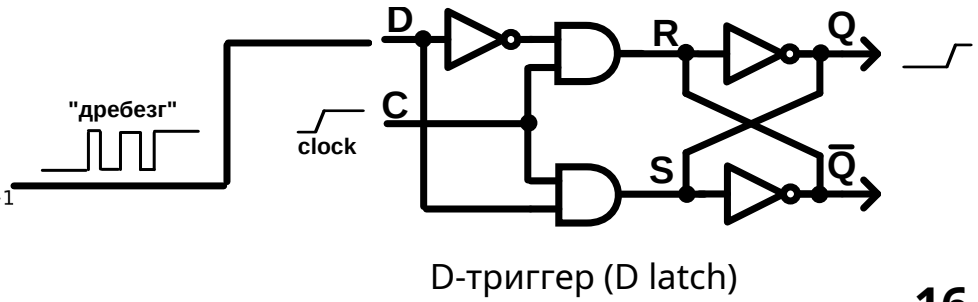
1. Комбинационные схемы подвержены эффекту «гонки» вызванного тем, что сигналы от разных входов перед тем как достичь выхода, проходят разные пути. В результате выход комбинационной схемы может несколько раз изменить свое состояние перед тем как установиться в **статические состояние** определяемое схемой. На выходах больших комбинационных схем образуется высокочастотный «дребезг» который мешает бесконечно наращивать сложность схемы. Возникает необходимость деления комбинационных схем на независимые участки, так чтобы один участок не влиял на работу другого.

2. В цифровых схемах возникает необходимость оградить ячейки памяти о того, чтобы их состояние изменялось в произвольные момент времени, то есть запись в ячейку должна происходить только по определенному **разрешающему** сигналу.

Для решения этих проблемы используются ячейки памяти с разрешающим сигналом. Запись в такую ячейку осуществляется либо по **фронту**, либо по **спаду** разрешающего сигнала который называют **«тактовый сигнал»**. Такая **синхронная** ячейка памяти с тактовым сигналом называется **D-триггер (D latch)**.



Комбинационная схема



D-триггер (D latch)

Синхронные ТТ-триггеры (DFF)

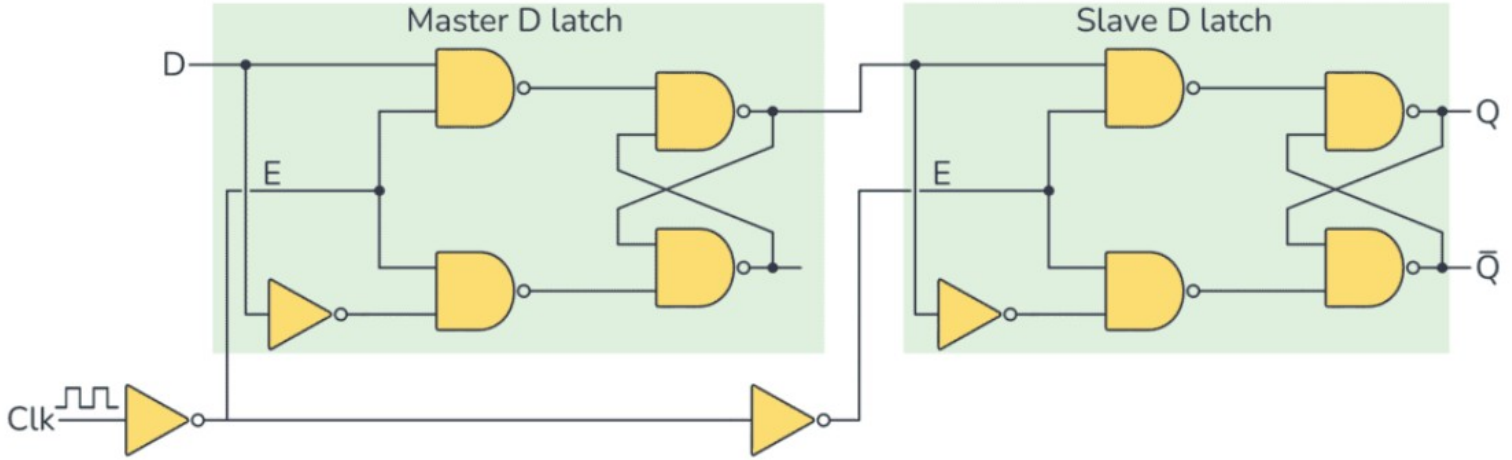
Один D-триггер не решает полностью проблему «дребезга», но её решают два последовательно включенных D-триггера с разным типом тактирования — **первый по фронту, второй по спаду** (или наоборот). Такая схема «тяни-толкая» называется ТТ-триггер или «D flip-flop» (DFF).

1. DFF может тактироваться как по фронту, так и по спаду.

2. DFF может быть с асинхронным сбросом (или установкой) и без таковых — **DFFAR.**

3. Данные подаваемые на вход **D** триггера DFF появятся на его выходе **Q** только в следующем такте!

4. Триггеры DFF являются базовыми блоками для устройств программируемой логики.



Синхронные (последовательностные) схемы

1. Любая схема содержащая DFF является синхронной и в ней присутствует хотя бы один тактовый сигнал.

2. Синхронные схемы могут иметь множество сигналов тактирования. Части одной схемы тактируемые одним и тем же сигналом называются **тактовым доменом**.

3. Схемы с **двумя и более тактовыми доменами** требуют определенного подхода при проектировании, особенно в точках передачи данных из одного тактового домена в другой.

4. Эта проблема называется **Clock Domain Crossing (CDC)**, для её решения применяют схемы **FIFO с кодом Грея.**

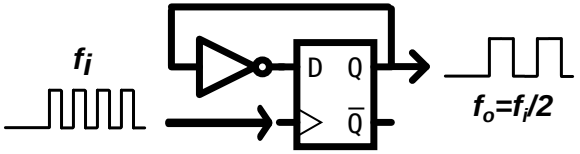
Конечные автоматы (КА)

Одним из видов **синхронных** схемы являются **конечные автоматы** (Finite State Machine - **FSM**). **КА** - это схемы которые содержат: **комбинационную схему**, **элемент памяти** и **обратную связь**.

КА принимает на свои входы данные, исходя из них изменяет своё состояние которое сохраняет в элементе памяти, и формирует сигналы на выходах исходя их текущего состояния (принцип: **принял**, **обработал**, **сохранил** и **передал дальше**). В процессе работы, КА могут проходить огромное количество состояний. В цифровой технике слово «**машина**» является синонимом «**конечному автомату**».

Любая задача обработки данных может быть сведена к созданию конечного автомата. Современные микропроцессоры, системы-на-кристалле и нейровычислители это очень **сложные машины (или КА) с гигантским числом состояний**.

Существует два вида автоматов КА: **автомат Мура** и **автомат Мили**. У Мура выходные сигналы зависят только от текущего состояния атомата. У Мили — от текущего состояния и состояния входных сигналов.



Простейший автомат:
«делитель частоты на 2»

Конвейеры

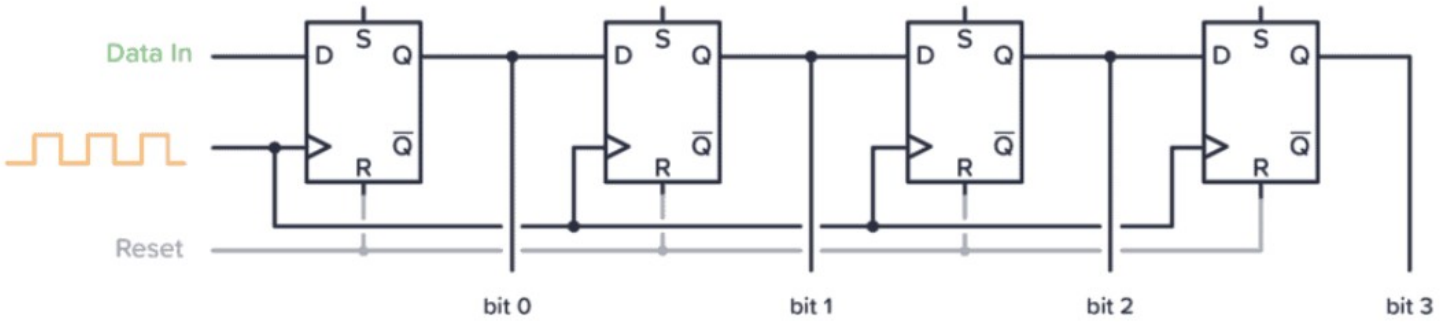
Большие схемы обработки данных разбивают на **стадии** с помощью DFF триггеров. Такой подход называется **конвейеризация**.

Конвейеризация позволяет уменьшить длину критического пути в комбинационных схемах, а следовательно — уменьшить задержку распространения и увеличить частоту тактирования.

Помимо этого, конвейеризация позволяет нагружать «работой» простаивающие части схемы, тем самым увеличивая пропускную способность и производительность всей схемы.

Примером простейшего конвейера может служить сдвиговый регистр который **за каждый импульс** тактового сигнала **обрабатывает** и **передает по конвейеру,** от стадии к стадии, один бит данных.

Важно, что все стадии конвейера тактируются от одного и того же источника тактовых импульсов!



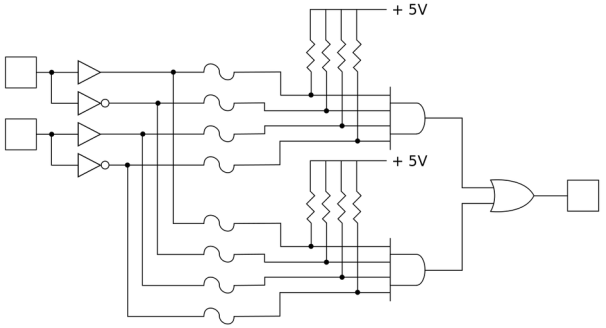
Часть 2

Устройство микросхем программируемой логики

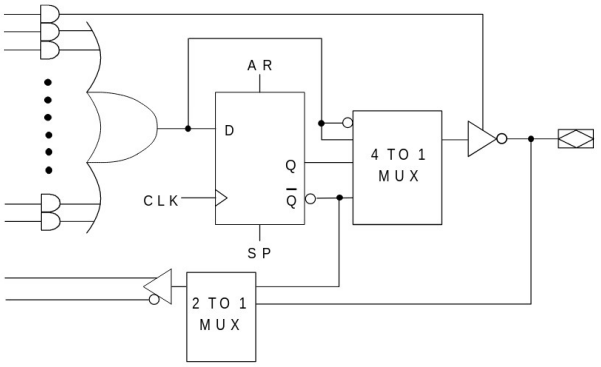
Микросхемы PAL/GAL

Первые микросхемы программируемой логики (PAL - Programmable Logic Array) появились в конце 1970-х и прошли серьезную эволюцию. Известные в своё время микросхемы PAL16R4 от MMI имели в составе небольшой массив из «таблиц истинности» близких по структуре к 2-LUT. Связи в них программировались путем электрического пережигания «фьюзов», по образу и подобию **программируемых ПЗУ**.

В середине 1980-х фирма Lattice выпустила изделие GAL22V10 содержащее десять более сложных программируемых структур которые получили название «макроячейка». В них к каждому выходу LUT добавили по одному DFF триггеру, а также ввели в схему два MUX: один для обхода DFF, второй — для организации обратной связи. Программировались они также, как и PAL.



Элемент структуры микросхем MMI PAL16R4.
2-LUT подобная структура.



Элемент структуры микросхем GAL22V10.
Макроячейка с DFF и обратной связью.

Микросхемы PLD и CPLD

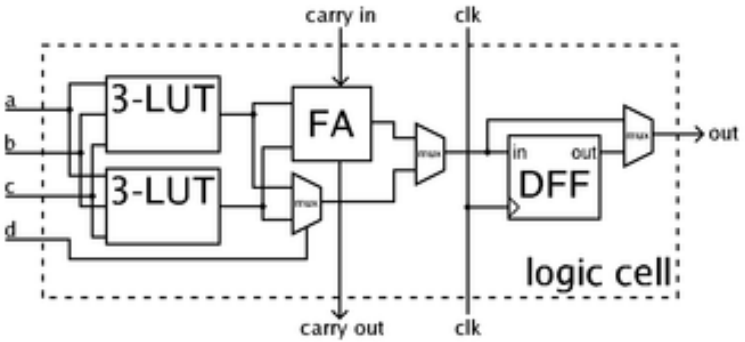
Прогресс не стоял на месте и в 1990-х появляются микросхемы PLD и CPLD (от анг. **Complex Programmable Logic Device**). Основное изменение по отношению к PAL/GAL - это введение в структуру **полного одноразрядного сумматора** (Full Adder) и переход на использование уже зарекомендовавших себя программируемых таблиц (2-LUT, 3-LUT, 4-LUT).

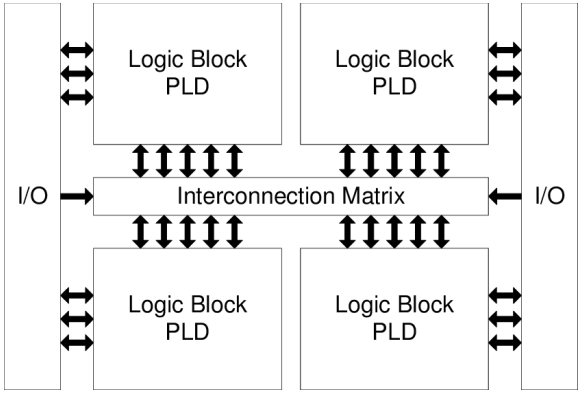
Такой элемент структуры получил название «логическая ячейка» (или «логический блок» - у разных производителей названия различаются в виду патентных проблем).

Программируются логические ячейки все тем же способом что и PAL/GAL — через программно перепрограммируемые «фьюзы». Конфигурационная информация располагается прямо в конфигурационных ячейках внутри «логических ячеек».

В микросхемах CPLD к рубежу 2000 года могло быть размещено несколько сотен логических ячеек.

Долгое время русскоязычный термин **ПЛИС** плотно **ассоциировался** именно с микросхемами **CPLD**.





Микросхемы FPGA

В 2000-х годах появляются микросхемы FPGA (Field Programmable Logic Array). Основное их отличие от CLPD состоит в том, что конфигурационные ячейки заменены с «фьюзов» на элементы статической памяти.

Такое решение позволило радикально увеличить плотность компоновки, так как размер статической ячейки памяти на порядок меньше размера «фьюза». Число логических блоков современных FPGA достигает **9 млн шт**.

Помимо этого, в структуру FPGA начали вводиться более крупные блоки обработки данных, как-то **многоразрядные АЛУ**, блоки **DSP**, блоки **сериализации (SERDES)**, блоки **PLL** и т.д.

Микросхемы **FPGA** при включении **требуют выполнения процедуры инициализации** — то есть загрузки данных в конфигурационные ячейки. Информация о конфигурации FPGA как правило содержится на внешнем носителе — на микросхеме Flash памяти, или может быть загружена отдельным микроконтроллером.

Таким образом **цифровой дизайн** (разработка цифровых схем) окончательно превратился в **программный продукт**, который можно распространять, **продавать** и **лицензировать**.

Дизайн «на продажу» получил название «IP-блок» от анг. Intellectual Property. Существует целая многомиллиардная индустрия оборота IP-блоков.

На данный момент русскоязычный термин ПЛИС подразумевает именно микросхемы FPGA.

Гибридные микросхемы

Приблизительно с 2015 года в состав FPGA начали вводить высопроизводительные вычислительные ядра на архитектуре ARM Cortex, блоки интерфейсов PCIe, GigabitEthernet, интерфейсы динамической памяти DDR4/DDR5.

С 2020-х в FPGA стали появляться блоки нейровычислителей.

Современные FPGA это **удивительно сложные устройства** нашпигованные различной аппаратурой которую можно программным путем соединять между собой для решения сложных задач. Всё чаще в обороте появляется термин «**гибридные микросхемы**». В современных ПЛИС от FPGA уже мало что осталось.

AMD/Xilinx Virtex UltraScale+ VU19P. 2020г.

- 9M logic cells
- PCI Express® Gen3 x16 / Gen4 x8
- Enhanced routing, logic, and clocking
- 2,072 GPIOs
- Up to 8x DDR4 x72 at 1.5 Tb/s bandwidth 80 high-speed transceivers with 2.3Tb/s I/O bandwidth
- Quad-core ARM® Cortex-A53
- Dual-core ARM® Cortex-R5 real-time processing units,
- ARM® Mali-400 MP2 GPU, integrated H.264/H.265 video codec

Области применения микросхем программируемой логики

1. Первоначально микросхемы PAL использовались как «клей» (glue logic) для стыковки различных несовместимых между собой устройств и изготовления дешифраторов.

2. С появлениям GAL их начали применять как обособленные устройство для простых автоматов — управления освещением, контроль температуры, простейшая автоматизация технологических процессов.

3. Появление ПЛИС открыло для программируемой логики тему цифровой обработки сигналов (DSP): радиолокация, цифровая радиосвязь, приборостроение (в том числе медицина), обработка видео и аудио.

4. В 1987 году Стивом Кассельманом была предложена идея полностью «синтезируемой» вычислительной машины. То есть машины которая изготавливается из некоторого объемы микросхем программируемой логики, а архитектура машины полностью определяется «программно». В 1992 году идея успешно реализована. Его ЭВМ содержала 600 000 программируемых логических элементов.

5. С тех пор программируемая логика широко используется для симуляции разрабатываемых микропроцессоров с целью проверки (верификации) их работоспособности перед запуском производства. Очевидно, что ошибку в разрабатываемой схеме легко выявить и исправить на ПЛИС, и невозможно исправить когда микросхема изготовлена и выпущена на рынок.

6. С 2010-х годов идет бурное развитие синтезируемых вычислительных ядер для ПЛИС, появляется открытая система команд RISC-V. Студенты различных зарубежных ВУЗов массово проектируют микропроцессоры на базе RISC-V с индивидуальными микроархитектурными решениями и синтезируют их в ПЛИС.

7. На ПЛИС легко проверить самые смелые идеи в области обработки данных с минимальными финансовыми затратами.

ПЛИС используются там, где требуется высокопроизводительная параллельная обработка специализированных данных с малой задержкой, а изготовление отдельной специализированной микросхемы является экономически не целесообразным.

Производители микросхем программируемой логики

На заре появления PAL/GAL, а это 1980-90 годы, насчитывалось более сотни производителей микросхемы программируемой логики. Каждый производитель интегральных схем считал своим долгом отметиться выпуском изделия в этой области. К 2000-му году их число значительно поубавилось, а с 2010-х годов из известных осталось менее десятка. Произошла консолидация рынка.

На данный момент более 80% рынка делят между собой два производителя из США: **Xilinx** — принадлежит компании **AMD**, и **Alter** — ныне подразделение компании **Intel**. Изделия этих компаний на две головы превосходят изделия конкурентов, как по число логических ячеек, так и по насыщенности **специализированными IP-блоками**.

Среди небольших, но известных (и доступных) производителей:

1. Lattice Semiconductors из США. Известен своими классическими ПЛИС серии iCE40 и ECP5 — до 85 тыс. логических ячеек.

2. Gowin Semiconductors из КНР. Известен очень дешевыми ПЛИС серии LittleBee: GW1NR — до 8K LUT и GW2A — до 55K LUT.

3. Efinix, Inc. КНР-США. Серия ПЛИС емкостью 4K-500K LUT.

Изделия всех трех (Lettice, Gown и Effinix) можно программировать с помощью открытых (open source) тулов под общим названием YOSYS.

Часть 3

Открытый инструментария для программирования микросхем ПЛИС на базе СПО

Что такое СПО ?

СПО — свободное программное обеспечение (Free/Open Source Software):

- защищено **открытыми** (публичными) **лицензиями** (GPL, BSD, MIT);
- распространяется **в исходных кодах;**
- может быть **модифицировано пользователем по его усмотрению** без получения каких либо предварительных разрешений от правообладателя;
- **не требует** обязательной **оплаты** за использование;
- часто **разрабатывается и поддерживается сообществом** (неформальным объединением разработчиков), и как правили исходит из академической среды.

Пример СПО: Операционная система **GNU/Linux** — разработка начата финским студентом Линусом Торвальдсом в 1991 году. Переросла в пандемию вселенского масштаба. Linux — везде!

Почему важно использовать СПО

1. Традиционно тема «ПЛИСостроения» очень сильно огорожена различными юридическими аспектами: интеллектуальной собственностью, патентами и договорами о неразглашении (NDA). Реальное устройство микросхем ПЛИС никому кроме производителя не известно.

2. Программное обеспечение для работы с ПЛИС выпускается конкретным производителем под конкретную серию микросхем. ПО от разных производителей не совместимо друг с другом. У разных производителей очень сильно отличается «work-flow» (подход к разработке). Отличаются комплекты библиотек которые тоже «закрыты». Часто техническая документация на библиотеки выдается «под расписку» (NDA).

3. Микросхемы ПЛИС разных производителей плохо совместимы между собой, а общая аура и туман нагнетаемый производителями делает перенос дизайна с ПЛИС одного производителя на ПЛИС другого — мало реализуемым. Особенно если в дизайне используются проприетарные IP-блоки, а они используются в любом более-менее сложном дизайне.

Производители умышленно привязывают пользователя к своим изделиям и своим средам разработки.

4. Программное обеспечение стоит серьезных денег. Имеются ограниченные академические лицензии, но эти лицензии больше не доступны в России. Раздобыть лицензию на Vivado или Quartus очень не просто.

Yosys Open Source Synthesis Suite (YOSYS)

Засилье «проприетарщины» в отрасле продолжалось до 2013 года, пока **студент** из Венского Политехнического Университета по имени Клиффорд Вулф не решил **проанализировать** содержимое **«битстрима»** статистическими методами. Битстрим - это поток конфиграционных битов, который загружается в микросхему ПЛИС в процессе её инициализации.

Для своего исследования Вулф выбрал микросхему Lattice iCE40. В кратчайшие сроки ему удалось узнать всю «поднаготную» устройства этой микросхемы. Он разработал комплект утилит позволяющий переводить схемы написанные на HDL языке Verilog в битсрим для этой ПЛИС и загружать их. О результатах своего исследование Вулф опубликовал научную статью:

> *Wol13] Wolf, Clifford: Design and Implementation of the Yosys Open SYnthesis Suite. 2013. – Bachelor Thesis, Vienna University of Technology.*

С этими разработками Вулф выступил на нескольких конференциях и к нему начали примыкать исследователи из других европейских университетов и лабораторий. Оказалось, что этой же темой интересуются многие. Позже Вулф и сотоварищи создали отдельную компанию **YosysHQ GmbH** и целенаправлено занимаются разработкой открытого (open source) инструментария для синтеза, формальной верификации и симуляции цифровых схем.

На данный момент инструментарий YOSYS поддерживает большое количество микросхем ПЛИС, в том числе от именитых производителей (Xilinx, Altera, Lattice, Gowin) и является единственным кросс-платформенным инструментарием разработки для ПЛИС.

Традиционно YOSYS популярен в академической среде, среди исследователей, небольших компаний и простых любителей электроники.

Языки описания аппаратуры (HDL) /1

Разработка цифровой схемы это процесс создания и отладки конечного автомата (КА).

К началу 1980х сложность цифровых схем достигла такого уровня, что создавать их традиционными начертательным методом стало невозможно:
- это титанический труд связанный с большим количеством ошибок;
- огромное число состояний КА не укладывается в головах разработчиков и не позволяет предвидеть все возможные проблемы, всегда будут оставаться «темные пятна» непроверенных состояний;
- трассировка микросхемы занимает огромное количество времени и также связана с ошибками человека.

Трассировщицы
микросхем за работой -->

Языки описания аппаратуры (HDL) /2

Первая проблема решается путем компьютеризации процесса разработки и представления цифровых схем в виде графов из связанных списков. Такой граф называется **нетлист** (от анг. netlist).

Вторая проблем - проблема верификации, потребовала создания специальных языков описания аппаратуры, к тексту которых можно применять математический аппарат с целью выполнения автоматического доказательства. Такие языки называют Hardware Description Languages (HDL).

Третья проблема — решается автоматическим синтезом фотошаблонов по HDL описанию.

Было опробовано множество различных концепций и форматов HDL. В результате концепция описания цифровых схем на уровне регистровых передач (**Register-Transfer Level — RTL**) оказалась наиболее жизнеспособной.

На данным момент в ходу у разработчиков два HDL языка:

- VHDL — на базе синтаксиса Ada, разработанный в недрах DARPA (Министерство обороны США);
- Verilog (SystemVerilog) — Си-подобный язык, вышел из коммерческой среды.

Verilog на данный момент доминирует. Подавляющее большинство дизайнов разрабатывается на нём. YOSYS поддерживает оба этих HDL языка.

Процесс разработки цифровых схем /1

Вкратце, последовательность действий RTL разработчика выглядит следующим образом:

Этап 0. Составление спецификации и эталонной модели. Исходя из ТЗ создается эталонная модель на ЯП высокого уровня. Модель проверяется, симулируется. По результатам симуляции строятся временны**е** диаграммы.

Этап 1. Выбор ПЛИС и средств разработки. По модели оцениваются ресурсы, по ним — выбирают подходящую серию ПЛИС, приобретают отладочную плату.

Этап 2. Составление плана распиновки микросхемы. Подготавливают конфигурационные файлы в которых описывают какие выводы ПЛИС за что отвечают.

Этап 3. Написание кода на HDL. Разработчик пишет код цифровой схемы на языках HDL — традиционно на Verilog (SystemVerilog) или VHDL. Нетрадиционно — на Chisel, SpinalHDL или nMigen/ AmaranthHDL.

Этап 4. Верификация и симуляция. HDL код отлаживается, верифицируется и симулируется (реализуется по-тактово). В процессе верификации используется «эталонная модель».

В процессе верификации и симуляции добиваются, чтобы результаты которые дает симуляция совпадали с результатами полученными от «эталонной модели». Процесс повторяют итеративно.

Процесс разработки цифровых схем /2

После того как схема описанная на HDL готова и прошла верификацию, приступают к синтезу. Процесс этот состоит из последовательности запуска различных утилит и хорошо автоматизирован.

Этап 5. Синтез. Из полученного кода на HDL, средствами выбранного инструментария производится синтез netlist-а.

Этап 6. Оптимизация и STA. Производится оптимизация netlist-а — из него удаляются лишние блоки, типовые структуры комбинируются и замещаются на упрощенные/эквивалентные или на специальные IP-блоки специфичные для выбранной ПЛИС. Производится анализ задержек и прогнозируемых частотных характеристик схемы (Static Time Analysis). Схема должна удовлетворять требования STA.

Этап 7. Расстановка и трассировка. Оптимизированный netlist поступает на вход следующей утилиты которая выполняет процесс расстановки блоков по местам и трассировку цепей, процесс «place and route».

Этап 8. Генерация битстрима. Завершающим этапом синтеза является генерация «битстрима» — последовательности бит которую можно загрузить в микросхему ПЛИС.

Этап 9. Тестирование на реальной аппаратуре. Битстрим загружают в ПЛИС и испытывают изделие на стенде — подают входные тестовые сигналы и анализируют реакцию схемы.

Концептуально, работа RTL дизайнера мало чем отличается от работы программиста.

Синтез для ПЛИС с помощью YOSYS

Ниже представлена последовательность запуска утилит из состава открытого инструментария для выполнения синтеза битстрима на примере платы «Карно» содежащей ПЛИС Lattice ECP5.

Этап 1. Синтез:

```
$ yosys -p "verilog_defaults -add -I,$(INC))" $(READ_VERILOG) -p "synth_ecp5
     -json $(NAME).json -top top"
```

Этап 2. Размещение:

```
$ nextpnr-ecp5 --package CABGA256 --25k --json $(NAME).json --textcfg $
(NAME)_out.config --lpf board_specific.lpf
```

Этап 3. Формирования битстрима:

```
$ ecppack --compress --input $(NAME)_out.config --bit $(NAME).bin
```

Этап 4. Загрузка битстрима в ПЛИС:

```
$ openFPGALoader -f $(NAME).bin
```

HDL языки Verilog и SystemVerilog

Как упоминалось выше, HDL язык Verilog на данный момент является самым распространенным среди RTL дизайнеров.

Verilog разработан в компании Gateway Design Automation в 1985 как средство верификации цифровых схем. В 1990 году компания Cadene приобрела компанию GDA и сделала язык Verilog открытым. Cadence — крупнейшая в США компания разработчик САПР для электроники и микроэлектроники.

Verilog прошел много редакций. В 2002 году на его основе был создан язык отвечающий современным требованиям который назвали SystemVerilog.

Языки Verilog и SystemVerilog соотносятся друг с другом примерно также, как ЯП «C» и «C++». SystemVerilog является объектноориентированной версией языка Verilog, с расширенным синтаксисом и включает в себя Verilog как подмножество.

Текущая версия стандарта IEEE: 1800-2009 SystemVerilog.

Verilog и SystemVerilog позволяют разработчику:

- описывать схемы в виде простого и понятного текста;
- синтезировать схемы в netlist для ПЛИС или для ASIC (для разработки микросхем);
- проводить верификацию схемы;
- проводить по-тактовую симуляцию схемы.

Основные концепции HDL языка Verilog

Язык Verilog оперирует очень небольшим количеством сущностей:

- **wire** — провод или связь по которой сигнал может распространятся между участками схемы, провод может иметь размерность в виде числа бит образуя шину;
- **reg** — регистр или ячейка памяти, синтезируется в триггер DFF, как и провод, может иметь размерность в виде числа бит;
- **module** — участок цифровой схемы со строго специфицированным интерфейсом (входными или выходными сигналами);
- **aways @()** - процедурный блок описывающий последовательностную схему;
- оператор **assign** — позволяет связать сигналы между собой или назначить сигналу какое-то значение;
- элементы комбинационной схемы представляются операторами: «&», «|», «~» - логические операции «И», «ИЛИ», «НЕ», «<<», «>>» - битовый сдвиг влево и вправо, «+», «-» - сложение и вычитание, и т. д.
- «**<=**» - оператор блокирующего присваивания, передает данные в регистр;
- «**=**» - оператор неблокирующего присваивания, передает данные по проводам.
- операторы **if then else** — выполняют сравнение сигналов.

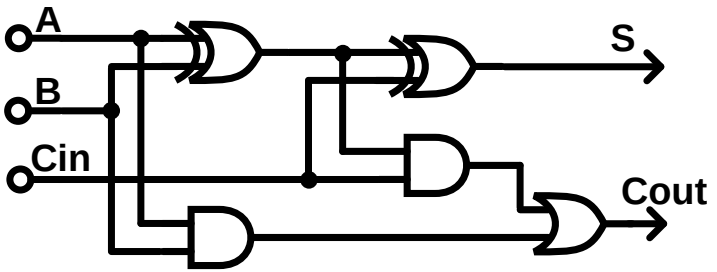
Все эти конструкции — синтезируемые, то есть могут быть представлены в виде реальной аппаратуры или её связей.

В языке Verilog также имеются несинтезируемые конструкции, они начинаются с символ #.

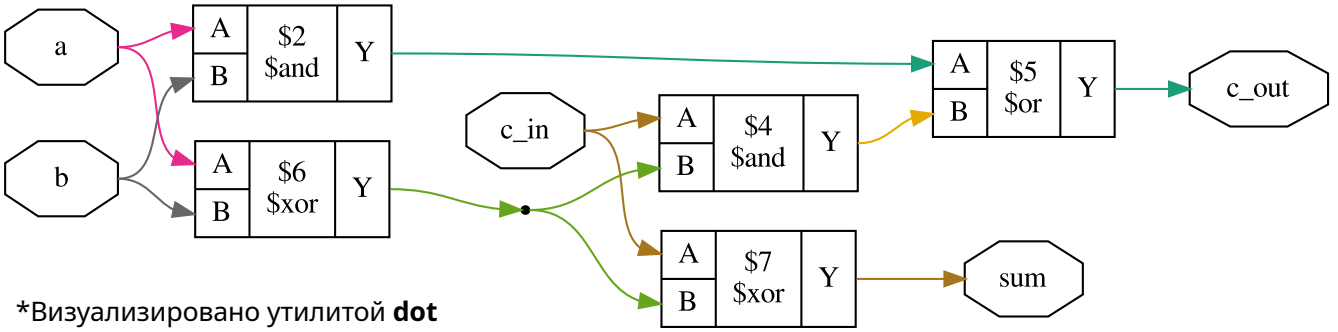
Пример комбинационной схемы на Verilog

Ниже приведена схема на 1-но разрядного полного сумматора:

```
module fulladder1 (
        input  wire       c_in,
        input  wire       a,
        input  wire       b,
        output wire       sum,
        output wire       c_out,
);
        assign c_out = (a & b) | (c_in & (a ^ b));
        assign sum = (c_in ^ (a ^ b));
endmodule
```



После выполнения синтеза мы получим следующий netlist (граф):



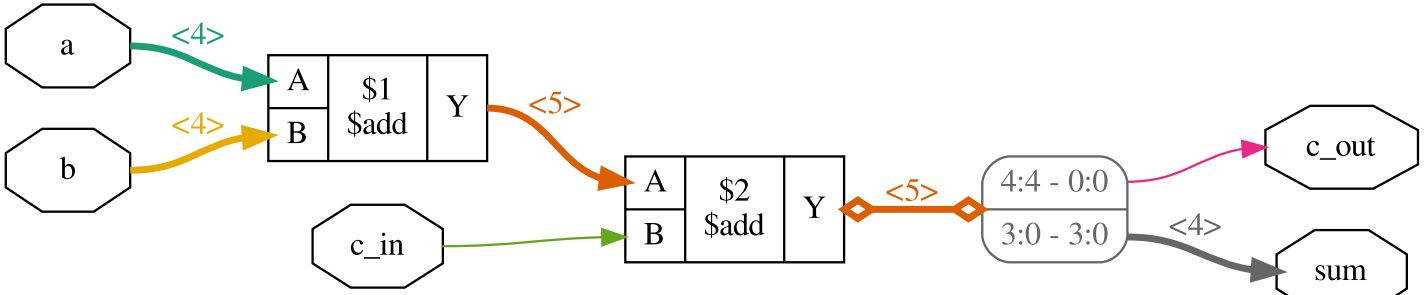
*Визуализировано утилитой **dot**

Пример еще одной комбинационной схемы на Verilog

Ниже приведена схема для 4-х разрядного полного сумматора с использованием двух блоков АЛУ:

```
module fulladder4 (
        input  wire       c_in,
        input  wire [3:0] a,
        input  wire [3:0] b,
        output wire [3:0] sum,
        output wire       c_out,
);
        assign {c_out, sum} = a + b + c_in;
endmodule
```

После выполнения синтеза мы получим следующий netlist (граф):



*Визуализировано утилитой **dot**

Пример синхронной (последовательностной) схемы на Verilog

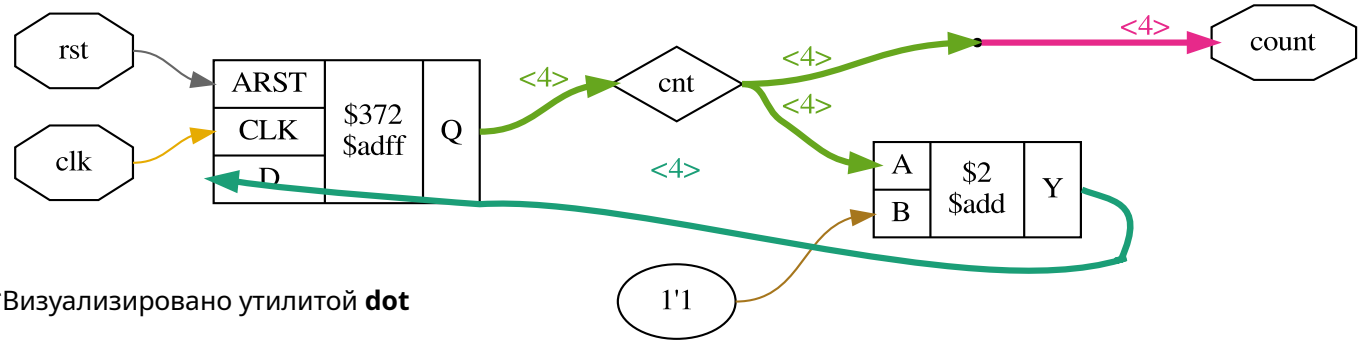
Ниже приведена схема 4-х разрядного счетчика с асинхронным сбросом:

```
module counter4 (
        input  wire       rst,
        inout  wire       clk,
        output wire [3:0] count,
);
        reg     [3:0] cnt;

        assign  count = cnt;

        always @(posedge clk or posedge rst)
        begin
                if (rst)
                        cnt <= '0;
                else
                        cnt <= cnt + 'b1;
        end

endmodule
```



*Визуализировано утилитой **dot**

Спасибо за внимание!

Ссылки:

Репозиторий платы ПИР СЦХ «Карно» (Karnix)
от ООО «Фабмикро» (открытый OSHW проект)
https://github.com/Fabmicro-LLC/Karnix_ASB-254

Разработка цифровой аппаратуры нетрадиционным методом: Yosys, SpinalHDL, VexRiscv

О себе:

В 2001 году окончил **ТюмГНГУ** по специальности **АСОиУ**, учился в аспирантуре.

Профессиональную деятельность начал достаточно рано, в 1992 году (первую ЭВМ увидел в 1987 году). Долгое время занимался развертыванием и администрированием телекоммуникационных сетей, разработкой сетевого ПО в т.ч. биллингов, мониторинга, систем для передачи голоса и видео через IP (VoIP)), разработкой мобильных приложений в до-андроидную эпоху. В 90-х годах — сисоп крупного узла сети FidoNet и BBS.

С 2010 года занимаюсь разработкой и опытным производством промышленной электроники. Накопил богатый опыт в таких сферах как:

- разработка схемотехники приборов на базе различных популярных микроконтроллеров: STM32 и ATmega, на различных китайских СнК на базе архитектуры ARM, с недавних пор работаю с отечественными МК на архитектуре RISC-V;
- трассировка многослойных печатных плат по топовым топологическим нормам;
- программирование микроконтроллеров и СнК, от «голого железа» до разработки драйверов для ОС Linux;
- программирование микросхем ПЛИС (или «синтез цифровых схем»).

Сфера интересов: сети и телекоммуникации, устройство современных микропроцессоров и операционных систем, открытая микропроцессорная архитектура **RISC-V**, ретро-компьютеры.