
ЛЕКЦИЯ №1 - ОСНОВНЫЕ ТЕНДЕНЦИИ ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ. ПРОБЛЕМЫ ПРОЕКТИРОВАНИЯ

Современная вычислительная техника развивается, сталкиваясь с рядом тенденций и проблем, связанных с совершенствованием аппаратных компонентов. Основные направления включают в себя уменьшение размеров транзисторов (технологический процесс), увеличение плотности упаковки, повышение производительности и снижение энергопотребления. Существуют также новые проблемы проектирования, вызванные миниатюризацией и высокими частотами работы, такие как тепловыделение и трудности в обеспечении синхронности.

ТЕХНОЛОГИЯ ПРОИЗВОДСТВА ИНТЕГРАЛЬНЫХ СХЕМ (ИС) И ТЕХНОЛОГИЧЕСКИЙ ПРОЦЕСС

Производство ИС осуществляется через технологический процесс, который включает этапы фотолитографии, травления, допирования и напыления различных материалов для создания транзисторов и других компонентов на кристалле кремния. Характеристики технологического процесса измеряются в нанометрах (например, 130 нм –> 90 нм, 28 нм –> 16 нм), что отражает размер транзисторов и плотность размещения компонентов на кристалле. Каждый новый шаг уменьшения технологических норм увеличивает плотность транзисторов, но также требует усовершенствования в отводе тепла и управлении энергией.

ПОНЯТИЕ ТЕХНОЛОГИЧЕСКОГО СДВИГА

Технологический сдвиг описывает изменения в процессе производства ИС, которые оказывают значительное влияние на проектирование. Сдвиг обычно включает переход на новые нормы литографии, использование новых материалов и оптимизацию архитектур ИС для повышения производительности и эффективности энергопотребления. Например, переход от 130 нм к 16 нм привел к увеличению плотности и производительности, но также повысил сложность проектирования и управления энергией из-за возросших проблем с тепловыделением и целостностью сигналов.

ПОНЯТИЕ POWER-DELAY PRODUCT (PDP)

Power-Delay Product (PDP) — это показатель, который оценивает баланс между потребляемой мощностью и задержкой прохождения сигнала через элемент. PDP используется для оценки эффективности устройств в контексте потребляемой мощности и производительности. Низкое значение PDP свидетельствует об энергоэффективности элемента при высокой скорости работы, что важно для мобильных и портативных устройств.

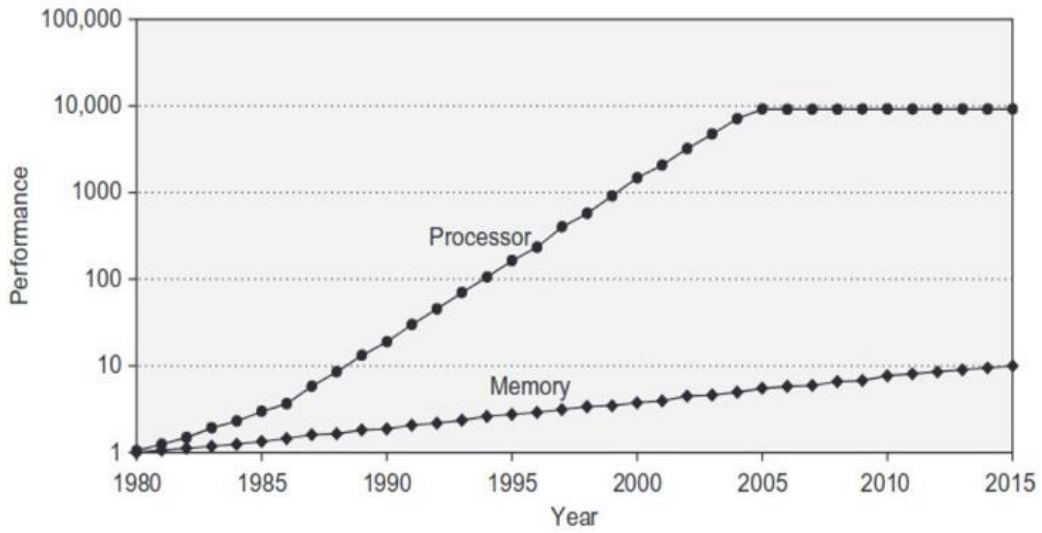
СОВРЕМЕННЫЕ ТЕНДЕНЦИИ И ПРОБЛЕМЫ

1. **Темный кремний**: Уменьшение размеров транзисторов и повышение их плотности приводят к увеличению выделения тепла. Однако при значительном выделении тепла невозможно использовать все транзисторы на кристалле одновременно, так как это может вызвать перегрев. Необходимость оставаться в определенных рамках энергопотребления привела к появлению ограничения называемого Utilization Wall, согласно которому с каждым новым техпроцессом и в отсутствие радикальных технологических изменений, доля площади кристалла,

задействованной в активной работе (буквально, где могут переключаться транзисторы) убывает экспоненциально. В итоге, часть кристалла остается неиспользуемой — это явление называют “темным кремнием”.

2. **GALS (Globally Asynchronous, Locally Synchronous):** Эта архитектура предусматривает асинхронное взаимодействие между локальными синхронными блоками. GALS позволяет снизить проблемы синхронизации между модулями системы, работающими на разных частотах, что облегчает интеграцию сложных схем и снижает энергопотребление.

3. **Стена памяти**: Ограничения скорости обмена данными между центральным процессором и памятью называют “стеной памяти”. По мере роста производительности процессоров, скорость доступа к памяти отстает, что приводит к простоям процессора в ожидании данных из памяти.



СИНХРОННЫЙ СТИЛЬ ПРОЕКТИРОВАНИЯ

Синхронный стиль проектирования подразумевает использование одного тактового сигнала для управления синхронизацией всех элементов системы.

Характерные особенности:

- Использование одного фронта (или спада) тактового сигнала для всех триггеров.
- Применение D-триггеров (не защелки).
- Размещение регистров на выходах блоков.
- Управление сигналами «разрешения счета», вместо управления тактовым сигналом непосредственно.
- Синхронизация асинхронных сигналов для обеспечения стабильной работы.

При проектировании важно избегать типичных ошибок, таких как использование тактовых сигналов, сформированных с помощью логических вентилей, что приводит к задержкам и нарушению синхронности в цепи.

ЛЕКЦИЯ №2 - ОСНОВНЫЕ ТЕНДЕНЦИИ ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ. ПРОБЛЕМЫ ПРОЕКТИРОВАНИЯ

Аппаратные платформы проектирования варьируются в зависимости от требований проекта и типа микросхемы, которую нужно создать. Эти платформы определяют, как будет выполнено моделирование, синтез и оптимизация схемы. Они различаются по уровню интеграции, гибкости, стоимости и временным затратам на разработку.

МАРШРУТ ПРОЕКТИРОВАНИЯ И УРОВНИ ОПИСАНИЯ

Процесс проектирования микросхемы включает несколько уровней описания и этапов:

Уровни проектирования:

- Системный
- RTL
- Топологический

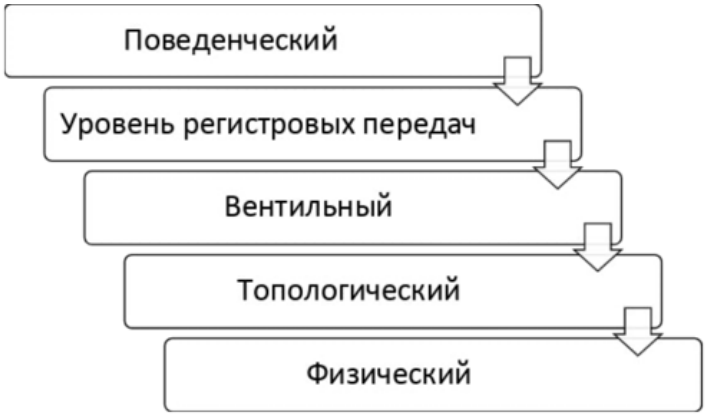
1. **Системный уровень:**
    - Фокусируется на общей архитектуре системы, ее функциях и требованиях
    - Определяются ключевые модули и их взаимодействие.
    - Используются высокоуровневые модели для оценки функциональности и выбора архитектуры.

2. **RTL уровень:**
    - Описывает работу устройства на уровне регистров и логики передачи данных.
    - Создается модель на языках описания аппаратуры (HDL), таких как Verilog или VHDL.
    - RTL-модель проверяется симуляцией и готовится для последующего синтеза.

3. **Топологический уровень:**
    - Преобразует логическое описание в физическую структуру на кристалле
    - Включает размещение компонентов, трассировку соединений, создание тактового дерева.
    - Адаптируется к требованиям производства для максимальной надежности и производительности.

Маршрут проектирования:

- Поведенческий
- Уровень регистровых передач
- Вентильный
- Топологический
- Физический



1. **Поведенческий уровень:** определяет общую функциональность и логику системы без детализации аппаратной реализации, создается на высокоуровневых языках
2. **Уровень регистровых передач (RTL)**: описывает передачу данных между регистрами на каждом такте, используя языки HDL (например, Verilog, VHDL), и формирует базовую логику устройства.
3. **Вентильный уровень**: представляет схему через логические элементы (вентиля), что позволяет более детально описать взаимодействие компонентов и оптимизировать логику.
4. **Топологический уровень**: размещает компоненты на кристалле, прокладывает соединения и формирует тактовое дерево, создавая физическую модель микросхемы.

5. **Физический уровень**: Завершающий этап, адаптирует топологическую модель к реальному производству с учетом ограничений кристалла и технологического процесса.

ЭТАПЫ ТОПОЛОГИЧЕСКОГО МОДЕЛИРОВАНИЯ

- Синтез RTL-модели: первичная генерация схемы на основе логики описания.
- Имплементация:
    - Для ПЛИС (FPGA):
        - Оптимизация синтезированного представления: улучшение логической структуры для уменьшения задержек и снижения энергопотребления.
        - Размещение компонентов: определение положения отдельных блоков на кристалле.
        - Трассировка соединительных линий: обеспечение необходимых связей между компонентами на кристалле, что критично для повышения производительности и сокращения задержек.
    - Для СБИС (ASIC): возможна дополнительная укладка соединений, создание тактового дерева, адаптация для производства и т. д.

      * СБИС – СверхБольшая Интегральная Схема

ФАКТОРЫ, ОКАЗЫВАЮЩИЕ ВЛИЯНИЕ НА ТОПОЛОГИЧЕСКОЕ ПРЕДСТАВЛЕНИЕ

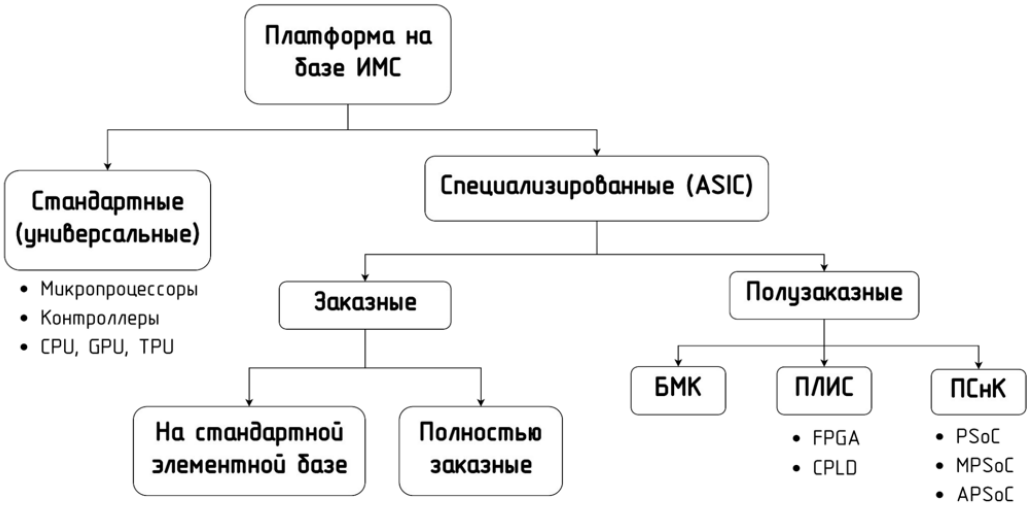
Основными факторами, которые влияют на топологию, являются:

- Архитектура RTL-модели;
- Алгоритмы синтеза и оптимизации синтезированного решения;
- Алгоритмы размещения и трассировки;
- Аппаратная платформа.

КЛАССИФИКАЦИЯ АППАРАТНЫХ ПЛАТФОРМ

Аппаратные платформы можно классифицировать следующим образом:

- Заказные микросхемы (ASIC): разрабатываются с нуля, обладают высокой производительностью и энергоэффективностью, но требуют значительных временных и финансовых затрат.
- Стандартные (универсальные) микросхемы: менее гибкие, но имеют более короткий цикл разработки и лучше подходят для серийного производства.
- Полузаказные микросхемы: представляют собой компромисс, сочетающий элементы заказных и стандартных микросхем.



** БМК – Базовый матричный кристалл*

** ПЛИС – Программи́руемая логи́ческая интегра́льная схе́ма*

** ПСнК – Программируемая Система на Кристалле*

СРАВНИТЕЛЬНАЯ ХАРАКТЕРИСТИКА FPGA И ASIC

Основные различия между FPGA и ASIC включают следующие аспекты:

- **Регистры**: в FPGA фиксированы позиции регистров, при этом требуются логические блоки (LUT), а ASIC позволяет более свободно размещать регистры.
- **Статическая память**: в FPGA количество и размер памяти фиксированы, тогда как в ASIC возможно гибкое конфигурирование под техпроцесс.
- **Блоки умножения**: в FPGA оптимизированы для готовых задач, а для ASIC часто требуют разработки с нуля.
- **Тактовое дерево и линии сброса**: в FPGA реализованы и разведены по кристаллу, в ASIC требуется специальная трассировка.

ОСНОВНЫЕ РИСКИ ПРИ ВЫБОРЕ ПЛАТФОРМЫ:

При выборе аппаратной платформы следует учитывать экономические и технические риски:

- **Экономические риски:**
    - Время и стоимость разработки (для заказных ASIC эти параметры существенно выше).
    - Доступность производственных ресурсов.

- **Технические риски:**
    - Сложность отладки и отсутствие гарантии на безошибочность (особенно в случае заказных микросхем).
    - Ограничения стандартных микросхем, что затрудняет решение некоторых специализированных задач.
    - Полузаказные микросхемы часто предлагают гибкость, однако требуют большего внимания к отладке и тестированию.

ОБЗОР ПЛИС XILINX/AMD

Xilinx, приобретенная AMD, предлагает широкий спектр FPGA с различными уровнями производительности и применений:

- **Spartan** — бюджетная линейка, предназначена для малобюджетных проектов и встраиваемых систем. Подходит для проектов в Интернете вещей (IoT) и сетевой инфраструктуры.
- **Artix** — средний ценовой сегмент, применяется в аппаратных ускорителях для видеообработки, криптографии, анализа сетевого трафика.
- **Kintex** — высокая производительность, подходит для беспроводных сетей и оптоволоконных систем, обладает увеличенным объёмом памяти.
- **Virtex** — передовая линейка, используется для телекоммуникационных систем 5G, радиолокации, высокопроизводительных вычислительных ускорителей.

ЛЕКЦИЯ №3 - АРХИТЕКТУРНЫЕ АСПЕКТЫ ПРОЕКТИРОВАНИЯ. ЧАСТЬ 1.

ПРОБЛЕМЫ РАССТАНОВКИ ЭЛЕМЕНТОВ НА КРИСТАЛЛЕ.

При расстановке элементов на кристалле, возникает ряд архитектурных и технических проблем, которые напрямую влияют на производительность, энергопотребление и надежность конечного устройства. Рассмотрим основные из них:

1. Оптимизация плотности размещения элементов

- Чем больше элементов можно уместить на кристалле, тем выше производительность устройства и экономичнее использование площади. Однако увеличение плотности создает множество проблем: ухудшается тепловыделение, усложняется разводка и увеличивается риск паразитных эффектов, таких как перекрестные наводки и утечки.

2. Минимизация задержек и длины соединений

- Расположение элементов на кристалле сильно влияет на длину соединений между ними. Длинные проводники создают дополнительные задержки и увеличивают энергопотребление, а также подвержены индуктивным и емкостным эффектам. Поэтому при размещении важно минимизировать расстояние между часто взаимодействующими элементами для уменьшения задержек.

3. Тепловые проблемы

- Выделение тепла на кристалле становится серьезной проблемой при увеличении плотности размещения. Неравномерное тепловыделение может привести к появлению "горячих точек", что снижает долговечность устройства. Важно учитывать тепловую карту кристалла и располагать элементы таким образом, чтобы распределить тепловую нагрузку равномерно.

4. Энергопотребление и баланс мощности

- При расстановке элементов необходимо учитывать энергопотребление каждого блока. Высокое потребление в определенной области может привести к увеличению тепловыделения и нестабильной работе из-за локальных скачков напряжения. Также стоит учитывать емкостные и резистивные свойства соединений, влияющих на потери энергии.

КОНЦЕПЦИЯ «NO SILVER BULLET»

Концепция *«No Silver Bullet»* была введена в 1986 году Фредериком Бруксом. Она утверждает, что не существует единственного решения, способного радикально улучшить процесс разработки программного обеспечения. В этой метафоре «серебряная пуля» обозначает универсальный инструмент или метод, который мог бы решить все основные проблемы в разработке ПО раз и навсегда.

Брукс утверждает, что хотя новые методы, технологии и инструменты могут приносить улучшения в некоторых аспектах разработки, они не смогут устранить фундаментальные сложности, присущие самому процессу создания ПО. Он выделил два типа сложности в разработке:

1. **Сущностная сложность** — это неизбежные сложности, связанные с природой самих программных задач.
2. **Случайная сложность** — это дополнительные проблемы, возникающие из-за недостатков используемых технологий и инструментов.

Ключевые идеи

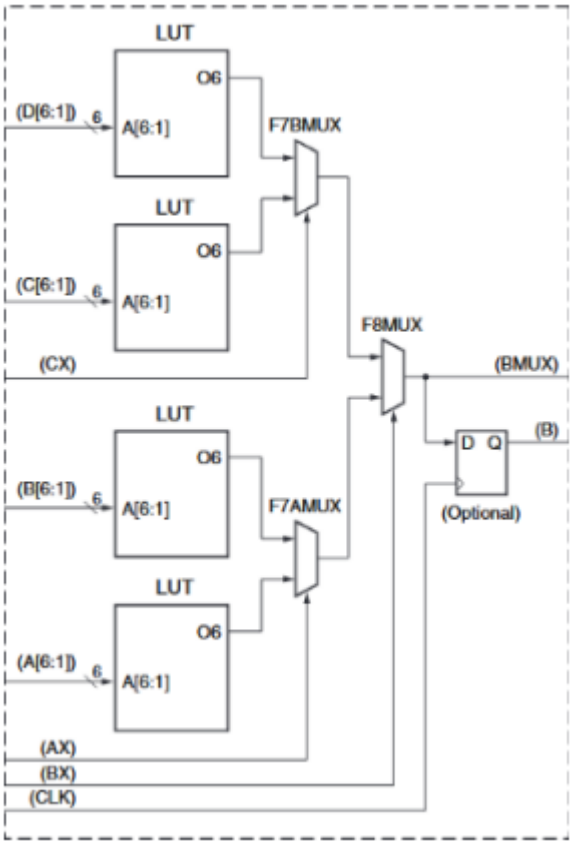
Брукс утверждает, что большинство разработок в сфере программирования (новые языки, методы проектирования и инструменты) в основном помогают снизить *случайные сложности*, но не затрагивают *сущностные*. Поэтому:

- **Нет и не может быть технологии или подхода, который решит все проблемы разработки ПО**, потому что сущностные сложности — это не технические барьеры, которые можно преодолеть, а особенности задач, которые просто нужно решать.
- **Ожидание «прорыва» в виде универсального решения в программной инженерии является ошибочным**, так как развитие в этой области происходит эволюционно, а не революционно.
- Улучшения возможны, но они будут происходить постепенно. И вместо поиска волшебного решения важно работать над постепенным совершенствованием процессов, инструментов и методов.

ПАТТЕРН ПРОЕКТИРОВАНИЯ «КОМБИНАЦИОННАЯ СХЕМА»

Паттерн проектирования *«Комбинационная схема»* (или *«Комбинаторная логика»*) — это подход к проектированию цифровых схем, который подразумевает построение системы на основе логических операций, не зависящих от внутренних состояний и временных характеристик. В отличие от последовательных схем,

комбинационные схемы не имеют памяти и выполняют свои функции исключительно на основе текущих входных данных.



Основные свойства комбинационной схемы

1. **Отсутствие состояния**: комбинационные схемы не сохраняют предыдущее состояние. Выход зависит только от входных данных в текущий момент времени.
2. **Детерминированность**: для каждой уникальной комбинации входов комбинационная схема всегда дает один и тот же результат на выходе.
3. **Независимость от времени**: комбинационная логика мгновенно преобразует входные сигналы в выходные, то есть не имеет задержек, обусловленных состоянием или внутренним состоянием устройства. В реальности, однако, есть физические задержки, связанные с прохождением сигнала по логическим элементам.

Примеры комбинационных схем

Комбинационные схемы используются для создания таких базовых устройств, как: Сумматоры и вычитатели, Декодеры и кодировщики, Мультиплексоры и демультиплексоры, Логические операторы.

Основные компоненты комбинационных схем

Комбинационные схемы состоят из базовых логических элементов, таких как: И (AND), ИЛИ (OR), НЕ (NOT), И-НЕ (NAND), ИЛИ-НЕ (NOR)

Преимущества и недостатки комбинационных схем

Преимущества:

- **Высокая скорость работы**: комбинационные схемы обрабатывают сигналы практически мгновенно (задержка связана только с прохождением сигнала по элементам).
- **Простота в анализе и моделировании**: отсутствуют состояния, что упрощает проектирование и тестирование.
- **Детерминированное поведение**: фиксированное соответствие входов и выходов делает комбинационные схемы предсказуемыми.

Недостатки:

- **Невозможность хранения данных**: отсутствие памяти означает, что комбинационные схемы не подходят для задач, требующих сохранения состояний.

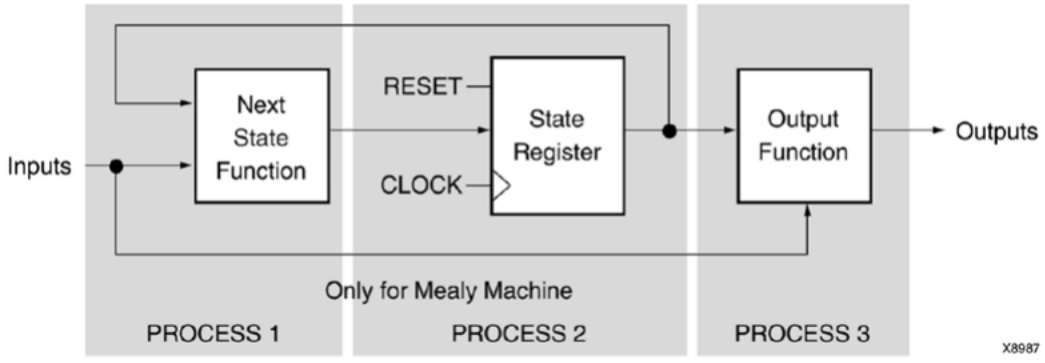
- **Потребность в большом количестве логических элементов** для сложных задач: сложные функции могут требовать большое количество элементов, что увеличивает площадь на кристалле и энергопотребление.

Применение комбинационных схем в разработке

Комбинационные схемы часто служат базовыми блоками для более сложных цифровых устройств, таких как процессоры, микроконтроллеры и ASIC. Они особенно полезны в таких задачах, как обработка сигналов, реализация математических операций и управление процессами. В современных цифровых устройствах комбинационные схемы сочетаются с последовательными схемами, создавая гибридные системы, которые сочетают преимущества обеих архитектур.

ПАТТЕРН ПРОЕКТИРОВАНИЯ «КОНЕЧНЫЙ АВТОМАТ»

Паттерн проектирования *«Конечный автомат»* используется для создания систем, поведение которых зависит от текущего состояния. Паттерн широко применяется в задачах, где объект должен изменять своё поведение в зависимости от перехода между различными состояниями. Конечный автомат имеет несколько ключевых элементов: состояния, действия и переходы.



Основные свойства конечного автомата

1. **Набор состояний**: у конечного автомата всегда есть конечное число состояний.
2. **Переходы между состояниями**: автомат меняет своё состояние на основе определённых условий.
3. **Действия**: при переходе между состояниями или при нахождении в состоянии могут выполняться определенные действия.

Основные компоненты конечного автомата

1. **Контекст**: объект, состояние которого мы отслеживаем и контролируем. Он управляет текущим состоянием и переключениями между ними.
2. **Состояния**: отдельные классы, представляющие возможные состояния контекста. Эти классы содержат логику, специфичную для определенного состояния.
3. **Переходы**: логика перехода определяет условия, при которых возможен переход между состояниями.
4. **События**: внешние или внутренние события, запускающие переходы. Например, нажатие кнопки или сигнал от системы.

Принципы работы конечного автомата

1. **Текущие состояние**: автомат в любой момент находится в одном конкретном состоянии.

2. **Событие или действие**: при получении события автомат может либо остаться в текущем состоянии, либо перейти в другое, выполняя при этом нужные действия.
3. **Таблица переходов**: конечный автомат можно описать таблицей переходов, в которой указано, как реагировать на события в каждом состоянии.

Преимущества и недостатки паттерна

Преимущества:

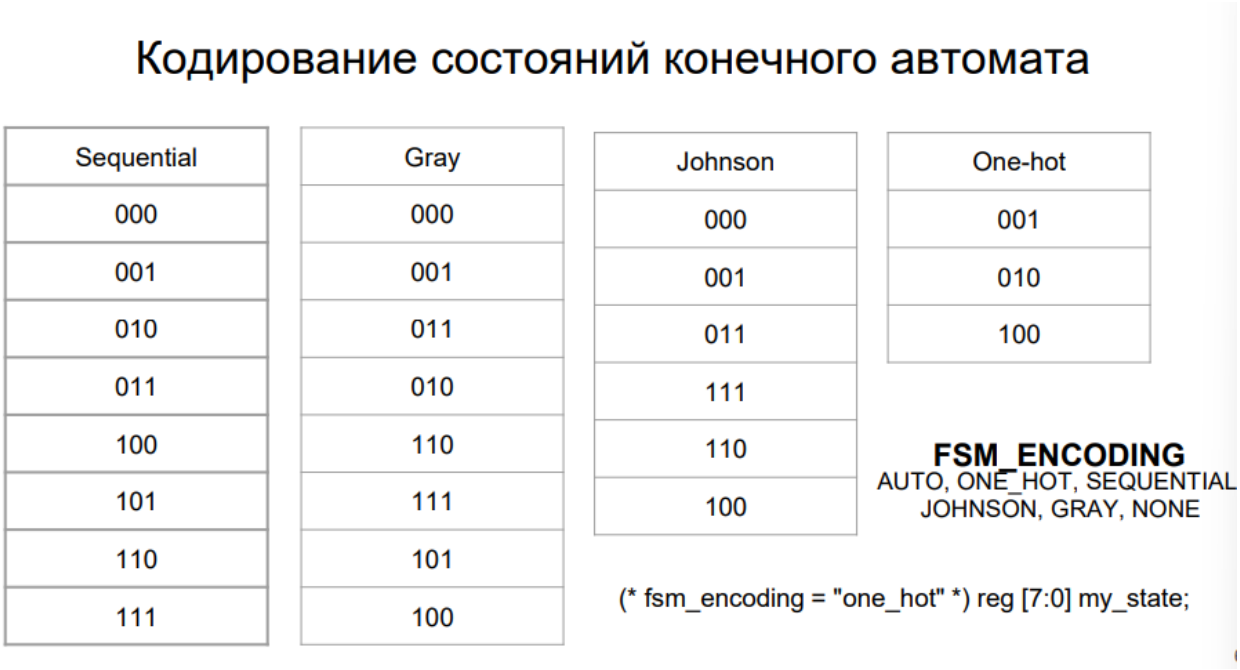
- **Изоляция логики состояний**: каждый класс-состояние отвечает только за своё состояние, что упрощает поддержку.
- **Легко расширяем**: для добавления нового состояния достаточно создать новый класс и определить его переходы.
- **Повышение читаемости кода**: поведение системы разбивается на логичные блоки, что упрощает понимание.

Недостатки:

- **Увеличение числа классов**: для каждого состояния требуется отдельный класс, что может увеличить сложность проекта.
- **Не всегда оправдано**: в простых случаях использование конечного автомата может добавить излишнюю сложность.

КОДИРОВАНИЕ СОСТОЯНИЙ КОНЕЧНОГО АВТОМАТА

Кодирование состояний конечного автомата — это процесс представления каждого состояния автомата уникальным набором сигналов, который позже можно использовать в логической схеме для реализации переходов и действий. Это один из ключевых этапов проектирования конечных автоматов (КА) на уровне цифровых схем, так как от выбора метода кодирования зависят размер схемы, скорость работы и устойчивость к ошибкам.



Основные подходы к кодированию состояний

Существуют различные методы кодирования состояний конечного автомата, включая: Прямое двоичное кодирование, Одноразрядное (однотактное) кодирование, Серийное (серийное двоичное) кодирование.

Прямое двоичное кодирование

В этом методе каждое состояние конечного автомата представляется уникальным двоичным числом. Это — эффективный способ в плане количества

бит, поскольку используется минимальное количество разрядов: если имеется n состояний, потребуется всего log2(n) бит.

- **Плюсы**: меньшее количество триггеров, минимальный объем памяти.
- **Минусы**: сложная логика переходов, что может замедлить схему при большом количестве состояний.

Одноразрядное (однотактное) кодирование

В этом подходе каждое состояние кодируется одним отдельным битом, что означает, что если у конечного автомата n состояний, то потребуется n разрядов. Только один разряд установлен в «1» для каждого состояния, остальные — «0».

- **Плюсы**: упрощенная логика переходов, высокая скорость работы, так как каждое состояние кодируется одним битом, и легко строить переходы.
- **Минусы**: требует большого количества триггеров и ресурсов, особенно если много состояний.

Серийное (серийное двоичное) кодирование

При этом методе используется определённая последовательность для переходов. Состояния кодируются так, чтобы последовательность кодов была максимально упорядоченной, что упрощает построение логики переходов. Например, при серийном двоичном кодировании разница между кодами соседних состояний может быть минимальной, что упрощает переходы.

- **Плюсы**: относительно простой для проектирования переходов.
- **Минусы**: ограничение на структуру автомата, что делает его менее универсальным.

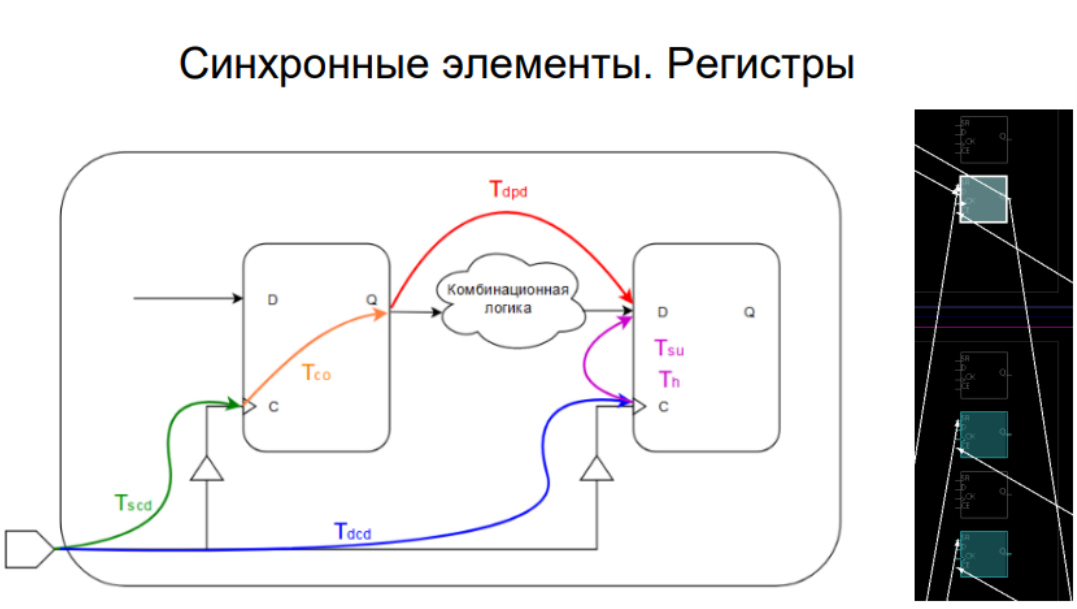
Выбор подхода к кодированию состояний

Выбор метода кодирования зависит от целого ряда факторов:

- **Количество состояний**: одноразрядное кодирование подходит для небольших автоматов, тогда как для больших лучше использовать двоичное.
- **Ограничения по скорости и площади**: одноразрядное кодирование проще и быстрее, но требует больше триггеров.
- **Надежность и устойчивость к ошибкам**: одноразрядное кодирование менее подвержено ошибкам, так как переходы легче контролировать.

ЭЛЕМЕНТЫ ПАМЯТИ

Синхронные регистры — это устройства, предназначенные для хранения и передачи данных в цифровых схемах, работающие в синхронизации с тактовым сигналом.



Основные типы синхронных регистров

Существует несколько типов синхронных регистров, выполняющих различные функции:

1. **Параллельные регистры**: обеспечивают параллельный ввод и вывод данных. Все биты данных записываются и считываются одновременно, что делает их удобными для одновременной обработки данных, команд или других многобитных значений.
2. **Сдвиговые регистры**: позволяют сдвигать данные влево или вправо на один бит с каждым тактовым импульсом. Используются для последовательной передачи данных, например, а также для преобразования данных из последовательного формата в параллельный и обратно.
3. **Счётные регистры**: специальные регистры, которые инкрементируют или декрементируют своё значение при каждом тактовом сигнале. Такие регистры часто применяются для подсчёта событий или отсчёта временных интервалов в процессорах и контроллерах.
4. **Регистр с выборочной загрузкой**: позволяет записывать данные в регистр при подаче специального сигнала загрузки. Это делает возможной выборочную запись, когда данные изменяются только при необходимости.
5. **Циклические сдвиговые регистры**: сдвигают данные по кругу, перенося крайний бит в противоположный конец регистра. Используются в генераторах случайных последовательностей и для хранения данных в специфических циклических процессах.

Принцип работы синхронного регистра

Работа синхронного регистра основана на синхронизации с тактовым сигналом, который контролирует запись и вывод данных. Основные элементы управления включают:

1. **Тактовый сигнал**: Синхронные регистры изменяют своё состояние при поступлении тактового импульса. В момент фронта тактового сигнала данные либо записываются в регистр, либо выполняются другие операции (например, сдвиг или сброс).
2. **Управляющие сигналы**: Некоторые регистры дополнительно оснащены управляющими входами:
    - **Сигнал записи (Load)**: инициирует запись данных в регистр.
    - **Сигнал сдвига (Shift)**: запускает сдвиг данных влево или вправо.
    - **Сигнал сброса (Reset)**: устанавливает регистр в начальное состояние, обычно обнуляя его.
3. **Входы и выходы**:
    - **Параллельные входы и выходы**: используются для одновременного ввода и вывода всех битов регистра.
    - **Последовательные входы и выходы**: применяются для последовательного ввода и вывода данных, особенно в сдвиговых регистрах.

Основные операции синхронного регистра

1. **Запись данных**: запись данных на входах в регистр при поступлении тактового импульса, если разрешён сигнал записи.
2. **Сдвиг данных**: сдвиг данных влево или вправо на один разряд при поступлении сигнала сдвига (в сдвиговых регистрах).
3. **Сброс регистра**: обнуление регистра при подаче сигнала сброса.

Преимущества и недостатки синхронных регистров

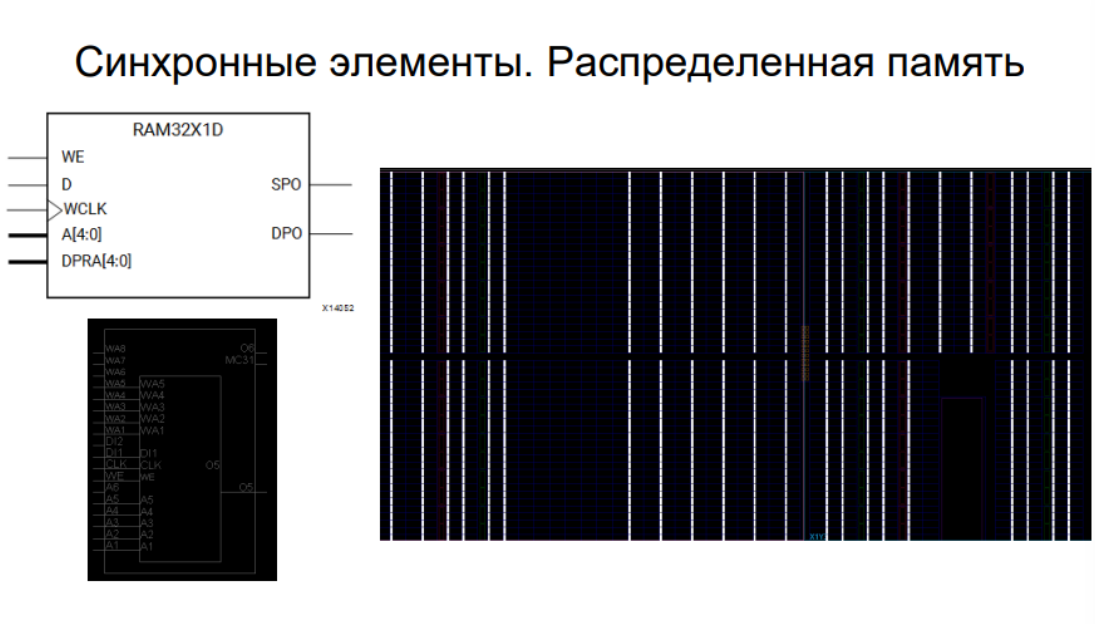
Преимущества:

- **Согласованность работы системы**: все изменения происходят строго в такт с системным сигналом, обеспечивая стабильную работу.
- **Упрощение проектирования**: синхронизация позволяет строить схемы с упрощённой логикой и управляемыми переходами состояний.
- **Высокая скорость работы**: синхронные регистры обеспечивают быстрый доступ и обработку данных в рамках тактового цикла.

Недостатки:

- **Зависимость от тактовой частоты**: быстродействие ограничено частотой тактового сигнала.
- **Энергопотребление**: при высокой частоте тактового сигнала увеличивается энергопотребление регистра.
- **Задержки на такт**: данные могут записываться только в момент тактового импульса, что может вызвать задержки при обмене данными.

Распределённая память — это способ организации памяти внутри FPGA (или других интегральных схем), при котором небольшие участки памяти распределены по кристаллу рядом с логическими элементами. В отличие от централизованных блоков памяти (например, блоков RAM или BRAM), распределённая память располагается децентрализованно, что позволяет каждому логическому элементу быстро обращаться к данным в своём окружении, минимизируя задержки. В FPGA распределённая память часто реализуется с помощью таблиц истинности (LUT), которые можно сконфигурировать для хранения небольшого объёма данных.



Основные особенности распределённой памяти в схемотехнике

1. **Гибкость в размещении**: распределённая память может быть сконфигурирована рядом с логическими элементами, которые к ней обращаются. Это позволяет сократить длину трасс и задержки на передачу данных, что особенно важно в высокоскоростных схемах.
2. **Малый объём памяти**: распределённая память обычно ограничена небольшими объёмами данных, так как каждый блок памяти занимает несколько логических элементов и их ресурсов. В основном она используется

для хранения таблиц значений, коэффициентов или буферов, где данные могут быть небольшими, но часто доступными.

3. **Синхронизация через тактовый сигнал**: распределённая память работает синхронно с остальной схемой, что упрощает управление её состояниями и синхронизацию данных с другими элементами.
4. **Высокая пропускная способность**: благодаря близкому размещению к логике и минимизации задержек, распределённая память обеспечивает высокую скорость доступа, что важно для выполнения критических по времени операций.

Реализация распределённой памяти в FPGA

Распределённая память может быть реализована в нескольких вариантах:

1. **Однопортовая память (Single-Port RAM):** обеспечивается доступ к памяти через один порт для записи или чтения данных.
2. **Дву- и многопортовая память (Dual-Port/Mult-Port RAM)**: используются два или более порта для параллельного доступа к памяти. Это позволяет читать и записывать данные одновременно, что повышает производительность системы.

Преимущества и недостатки распределённой памяти в схемотехнике

Преимущества:

- **Минимизация задержек**: память располагается вблизи логики, что уменьшает задержки доступа.

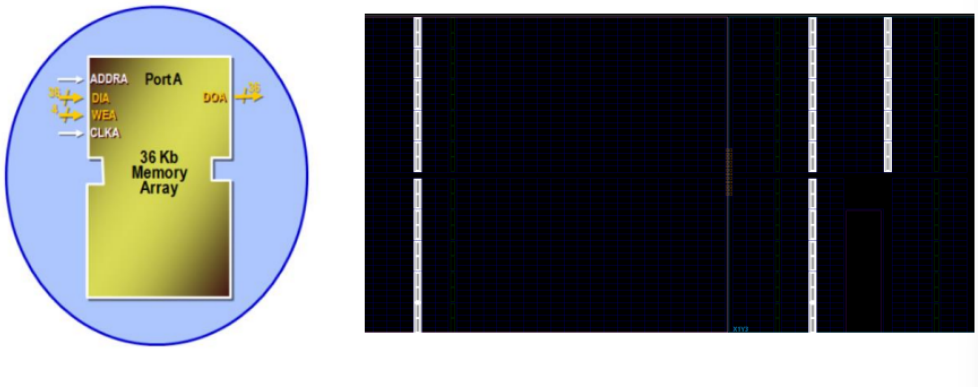
- **Высокая пропускная способность**: подходит для хранения данных, к которым нужен частый и быстрый доступ.
- **Гибкость использования**: можно динамически настраивать, используя LUT как память или как логические элементы.

Недостатки:

- **Ограниченный объём**: распределённая память имеет малую ёмкость и занимает ресурсы, которые могли бы использоваться для реализации логики.
- **Сложность управления**: конфигурация памяти и маршрутизация данных могут быть сложнее, особенно при проектировании схем с большим объёмом распределённой памяти.
- **Затраты на программирование и настройку**: требуют грамотного проектирования, чтобы правильно распределить ресурсы LUT для нужд памяти и логики.

Блочная память (Block RAM, BRAM) — это крупные массивы памяти, встроенные в FPGA и другие цифровые схемы, предназначенные для хранения значительных объёмов данных. В отличие от распределённой памяти, которая использует малые участки памяти, встроенные в логические элементы, блочная память представляет собой специализированные большие блоки памяти, расположенные в пределах кристалла FPGA и доступные через интерфейсы.

Синхронные элементы. Блочная память



Особенности блочной памяти в схемотехнике

1. **Большой объём памяти**: блочная память предназначена для хранения значительных объёмов данных (сотен или тысяч бит), что делает её удобной для хранения массивов данных, буферов и таблиц.

2. **Синхронный доступ**: доступ к блочной памяти контролируется тактовым сигналом, что позволяет согласовывать чтение и запись данных с другими синхронными элементами в цифровой схеме.

3. **Дву- и многопортовый доступ**: блочные памяти часто поддерживают многопортовый доступ, позволяя одновременно читать и записывать данные через разные порты. Это особенно полезно для параллельных вычислений и задач с высокими требованиями к пропускной способности.

4. **Гибкая конфигурация ширины и глубины**: многие FPGA позволяют настраивать блочную память для работы с разной шириной и глубиной данных. Например, можно использовать блочную память как несколько маленьких блоков или один большой.

5. **Специальные режимы и функции**: блочная память в FPGA часто имеет дополнительные функции, такие как инициализация данных при запуске, режим FIFO для организации очередей, и режимы автосброса.

Принцип работы блочной памяти

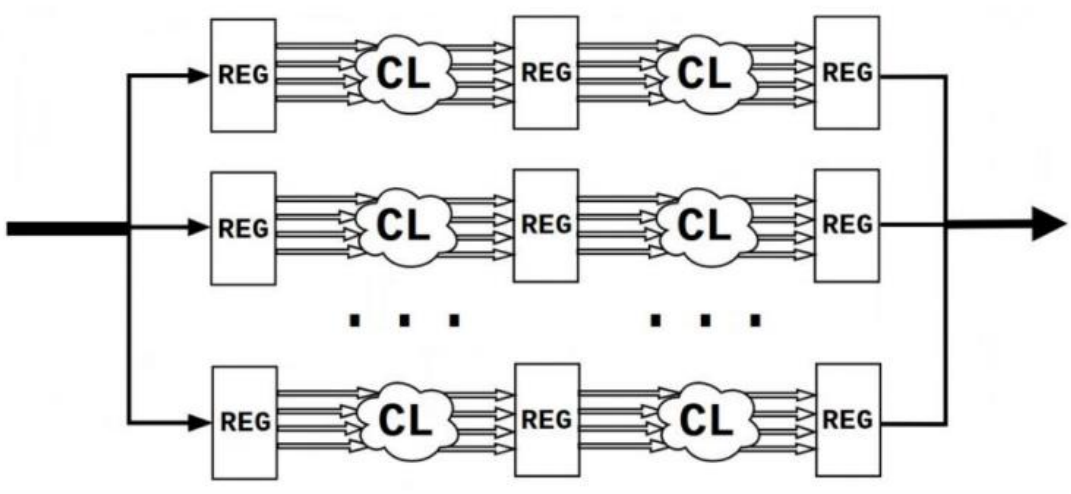
Блочная память управляется с помощью тактового сигнала и сигнала разрешения (enable) для операций чтения и записи, при этом каждый порт работает синхронно с остальными. Основные компоненты блочной памяти включают:

1. **Адресные входы**: определяют место хранения данных, позволяя обращаться к различным ячейкам памяти.
2. **Данные на входе и выходе**: для записи данные подаются на вход, а при чтении данные возвращаются с выхода памяти.
3. **Сигнал чтения и записи**: выбирает режим работы (чтение или запись) для каждого порта.
4. **Сигнал сброса**: используется для инициализации или очистки памяти в начальное состояние, если это необходимо.

Блочная память может быть настроена для параллельного чтения и записи, что делает её идеальной для задач, требующих одновременного доступа к данным.

ПАТТЕРН ПРОЕКТИРОВАНИЯ «КОНВЕЙЕР»

Паттерн проектирования «Конвейер» — это способ организации обработки данных, при котором последовательность вычислительных этапов разделена на отдельные блоки (стадии), работающие параллельно и последовательно передающие данные от одной стадии к другой. В каждом тактовом цикле конвейерная схема обрабатывает часть данных на одной из стадий, что позволяет достичь высокой производительности благодаря параллелизму. Конвейер особенно эффективен в задачах, требующих многократной обработки однотипных данных, таких как арифметические операции, цифровая обработка сигналов, и графические вычисления.



Основные особенности конвейера в схемотехнике

1. **Разделение задач на стадии**: задача разбивается на несколько логически последовательных стадий, каждая из которых выполняет определённую операцию над входными данными.
2. **Параллельная обработка**: данные, поступающие на вход, разделяются на пакеты, которые проходят обработку по стадиям одновременно, но на разных этапах. Например, пока одна порция данных находится на первой стадии, другая уже может обрабатываться на второй.
3. **Синхронный режим**: каждая стадия синхронизируется с тактовым сигналом, обеспечивая одновременное переключение между стадиями. Тактовая частота определяет время, в течение которого данные перемещаются на следующую стадию.
4. **Буферизация данных**: между стадиями обычно добавляются регистры или элементы памяти для хранения промежуточных данных, позволяя передавать результаты одной стадии на вход следующей в каждом тактовом цикле.

Преимущества и недостатки конвейера

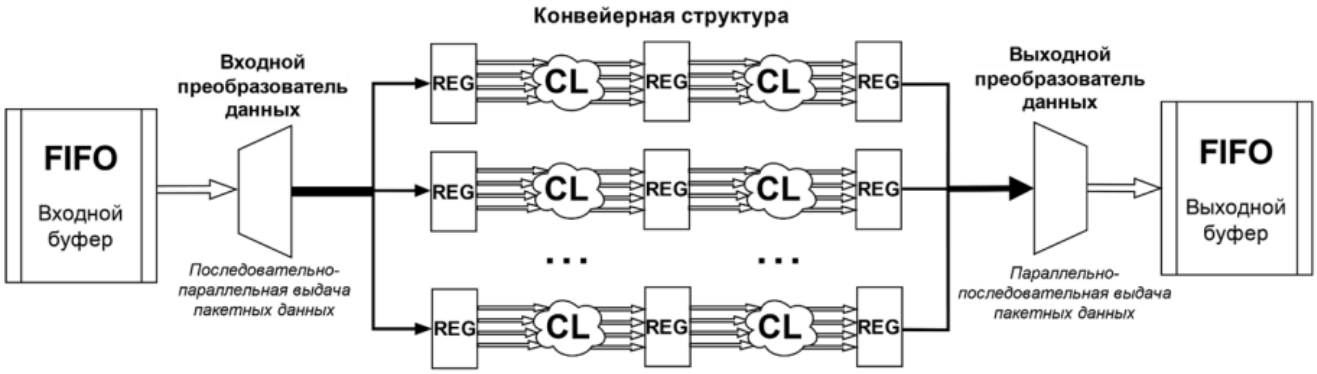
Преимущества:

- **Высокая производительность**: конвейер позволяет обрабатывать несколько данных одновременно, что повышает общую пропускную способность системы.
- **Масштабируемость**: можно добавлять или изменять количество стадий в конвейере для адаптации к конкретным задачам, требующим высокой производительности.
- **Снижение времени ожидания**: поскольку конвейер выполняет разные части задачи одновременно, он снижает общее время обработки больших объёмов данных.

Недостатки:

- **Сложность управления и синхронизации**: требуется точная синхронизация всех стадий и буферов, чтобы избежать пропусков или потерь данных.
- **Задержка заполнения (latency)**: конвейер нуждается в нескольких тактах на начальное заполнение перед тем, как начнёт выдавать результат.
- **Трудности при сбое**: если одна из стадий выходит из строя, работа всего конвейера может быть нарушена, что потребует добавления дополнительных механизмов контроля и восстановления.

Интеграция конвейера в вычислительную систему

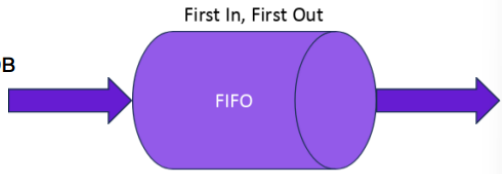


Интеграция. Входной/выходной буферы данных

- Собственная тактовая частота работы
- Последовательный (друг за другом) приём и выдача данных

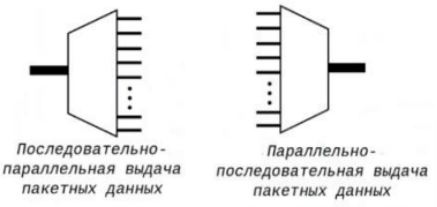
Параметры:

- глубина буферов
- размер шины данных буферов



Интеграция. Преобразователи данных

- Собственная внутренняя размерность данных
- Аккумуляция пакета данных



Параметры:

- размеры входной и выходной шин;
- тип применяемого сдвига битов данных

Дополнительные ограничения на примере конвейера

- Плотная компоновка компонентов в рамках одной стадии
- Близкое расположение стадий относительно друг друга в строгом порядке следования
- Проверка выхода за границы тактовых регионов, установка элементов для синхронизации на границе

Инструменты Vivado:

- floorplaning
- Bel
- Loc
- Pblock

СРЕДСТВА И СТРАТЕГИИ РАЗМЕЩЕНИЯ САПР

Средства автоматизированного проектирования (САПР) помогают инженерам автоматизировать процесс разработки цифровых схем и микросхем.

Средства для размещения в САПР:

1. **Cadence Innovus**: одно из популярных решений для размещения и трассировки в цифровом дизайне. Innovus поддерживает оптимизацию по производительности, площади и энергопотреблению с учетом актуальных ограничений, таких как электромиграция и тепловые ограничения.
2. **Synopsys IC Compiler**: инструмент для размещения и трассировки, широко используемый в индустрии. Он поддерживает различные методы оптимизации для высокопроизводительных и низкоэнергетических дизайнов.
3. **Xilinx Vivado**: специализированная среда для разработки FPGA, включающая мощные средства для оптимизации размещения на кристалле. Vivado может

анализировать и оптимизировать маршруты сигналов для повышения производительности и уменьшения задержек.

4. **Quartus Prime (Intel)**: используется для разработки схем на FPGA производства Intel. Поддерживает автоматическое размещение и маршрутизацию с возможностью оптимизации производительности и плотности.
5. **OpenROAD**: открытый проект, предоставляющий инструменты для автоматизации размещения и маршрутизации. OpenROAD предназначен для исследовательских и учебных целей и может быть полезен в разработке малых и средних проектов.

Стратегии размещения в САПР:

Стратегии размещения Vivado

Performance_Explore	Performance_ExplorePostRoutePhysOpt	Performance_LBlockPlacement	Performance_LBlockPlacementFanoutOpt
Performance_EarlyBlockPlacement	Performance_NetDelay_high	Performance_NetDelay_low	Performance_Retiming
Performance_ExtraTimingOpt	Performance_RefinePlacement	Performance_SpreadSLL	Performance_BalanceSLL
Congestion_SpreadLogic_high	Congestion_SpreadLogic_medium	Congestion_SpreadLogic_low	Congestion_SpreadLogic_Explore
Congestion_SSI_SpreadLogic_high	Congestion_SSI_SpreadLogic_low	Area_Explore	Area_ExploreSequential
Area_ExploreWithRemap	Power_DefaultOpt	Power_ExploreArea	Flow_RunPhysOpt
Flow_RunPostRoutePhysOpt	Flow_RuntimeOptimized	Flow_Quick	

1. Стратегия **Performance_Explore** в САПР используется для оптимизации размещения с акцентом на повышение производительности устройства. Она ориентирована на минимизацию задержек критических путей и улучшение временных характеристик. Данная стратегия особенно полезна для высокопроизводительных схем, где критично достижение максимальной скорости работы.
2. Стратегия **Performance_EarlyBlockPlacement** в САПР направлена на раннее размещение крупных и критически важных блоков (например, процессорных ядер, блоков памяти, DSP и других крупных логических элементов), чтобы обеспечить оптимальные условия для дальнейшего размещения и маршрутизации. Эта стратегия помогает улучшить временные характеристики, уменьшить задержки и упростить дальнейшую оптимизацию схемы.
3. Стратегия **Performance_ExtraTimingOpt** в САПР ориентирована на максимальное улучшение временных характеристик и устранение задержек в критических цепях. Она используется для схем, где особенно важно достижение высоких частот работы или минимизация временных задержек. Эта стратегия выполняет дополнительные этапы оптимизации таймингових характеристик, чтобы повысить производительность, даже если это требует больше времени на обработку и ресурсов.
4. Стратегия **Congestion_SpreadLogic_high** в САПР направлена на эффективное распределение логических элементов для снижения перегрузок в маршрутизации. Эта стратегия применяется для схем с высокой плотностью логики, где требуется максимальное распределение логических элементов для улучшения маршрутизации и предотвращения заторов. Уровень "high" (высокий) указывает на агрессивный подход к распределению логики для достижения наилучших результатов по снижению плотности в узких местах.

5. Стратегия **Congestion_SSI_SpreadLogic_high** в САПР предназначена для минимизации перегрузки маршрутизации в проектах с высокой плотностью логики и большим количеством соединений. Стратегия акцентирует внимание на агрессивном распределении логических элементов по всей площади кристалла, используя подход Spread Structured Interconnect (SSI), чтобы избежать концентрации маршрутов в ограниченных областях. Уровень "high" в названии стратегии указывает на высокий уровень агрессивности в распределении, что позволяет минимизировать заторы даже в сложных дизайнах.
6. Стратегия **Area_ExploreWithRemap** в системах автоматизированного проектирования (САПР) ориентирована на оптимизацию размещения элементов на кристалле с акцентом на использование доступной площади. Основная цель этой стратегии — улучшение плотности размещения логических блоков, что позволяет максимально эффективно использовать пространство на чипе и снизить вероятность перегрузок при маршрутизации.
7. Стратегия **Flow_RunPostRoutePhysOpt** в системах автоматизированного проектирования (САПР) предназначена для выполнения физической оптимизации схемы после завершения этапа маршрутизации. Этот этап важен для повышения производительности и улучшения параметров надежности созданной схемы, так как включает в себя анализ и корректировку размещения элементов и трассировки для устранения возможных проблем, возникающих на предыдущих этапах проектирования.
8. Стратегия **Performance_ExplorePostRoutePhysOpt** в системах автоматизированного проектирования (САПР) направлена на оптимизацию размещения и маршрутизации элементов после завершения основных этапов проектирования. Она фокусируется на улучшении временных характеристик схемы, минимизации задержек и повышении производительности, учитывая результаты маршрутизации и физического размещения.

9. Стратегия **Performance_NetDelay_high** в системе автоматизированного проектирования Vivado от Xilinx предназначена для оптимизации задержек сетей (net delays) в цифровых схемах. Эта стратегия акцентирует внимание на минимизации задержек сигналов, проходящих через критические цепи, что является важным аспектом для достижения высокой производительности проектируемых систем на FPGA (Field-Programmable Gate Array).
10. Стратегия **Performance_RefinePlacement** в системе автоматизированного проектирования Vivado от Xilinx фокусируется на уточнении размещения логических элементов на FPGA после первичного этапа размещения. Основная цель этой стратегии — улучшение временных характеристик схемы, что достигается путем оптимизации расположения элементов и минимизации задержек на критических путях.
11. Стратегия **Congestion_SpreadLogic_medium** в системе автоматизированного проектирования Vivado от Xilinx предназначена для управления перегрузкой (congestion) и оптимизации размещения логических элементов на FPGA. Эта стратегия сосредоточена на распределении логических блоков с целью уменьшения плотности и избежания перегрузок в узких местах кристалла.
12. Стратегия **Congestion_SSI_SpreadLogic_low** в системе автоматизированного проектирования Vivado от Xilinx направлена на управление перегрузкой и оптимизацию размещения логических элементов с использованием подхода Spread Structured Interconnect (SSI). Эта стратегия предназначена для создания более равномерного распределения логики по площади кристалла, что позволяет минимизировать риски перегрузки, особенно в сложных схемах с высокой плотностью соединений.
13. Стратегия **Power_DefaultOpt** в системе автоматизированного проектирования Vivado от Xilinx нацелена на оптимизацию потребления энергии в проектируемых схемах, особенно в FPGA. Эта стратегия

применяется для улучшения энергетической эффективности устройства при сохранении приемлемой производительности.

14. Стратегия **Flow_RuntimeOptimized** в системе автоматизированного проектирования Vivado от Xilinx фокусируется на оптимизации временных характеристик схемы в контексте времени выполнения (runtime). Эта стратегия применяется для достижения наилучших результатов по производительности в процессе функционирования схемы, что особенно важно для приложений, требующих высокой скорости обработки данных.
15. Стратегия **Performance_LBlockPlacement** в системе автоматизированного проектирования Vivado от Xilinx направлена на оптимизацию размещения логических блоков (L-blocks) с акцентом на повышение производительности схемы. Эта стратегия применяется для обеспечения наилучших временных характеристик и минимизации задержек на критических путях.
16. Стратегия **Performance_NetDelay_low** в системе автоматизированного проектирования Vivado от Xilinx предназначена для оптимизации задержек сетей (net delays) в цифровых схемах с акцентом на умеренное улучшение временных характеристик. Эта стратегия применяется, когда необходимо минимизировать задержки сигналов, проходящих через критические цепи, с менее агрессивным подходом, чем в высокопроизводительных стратегиях.
17. Стратегия **Performance_SpreadSLL** в системе автоматизированного проектирования Vivado от Xilinx предназначена для оптимизации размещения логических элементов с использованием подхода, основанного на распределении (spread) логических блоков. SLL (Small Logic Level) относится к малым логическим элементам, которые могут включать такие компоненты, как LUT (Look-Up Table), регистры и другие базовые логические единицы.

18. Стратегия **Congestion_SpreadLogic_low** в системе автоматизированного проектирования Vivado от Xilinx нацелена на управление перегрузкой в проектируемых схемах с высокой плотностью логических элементов. Она подразумевает умеренное распределение логики, что помогает уменьшить риски перегрузки и улучшить маршрутизацию.
19. Стратегия **Area_Explore** в системе автоматизированного проектирования Vivado от Xilinx направлена на оптимизацию размещения логических элементов на кристалле с акцентом на использование доступной площади. Эта стратегия применяется для исследования и улучшения размещения в контексте плотности и эффективности, что может помочь избежать перегрузок и улучшить производительность схемы.
20. Стратегия **Power_ExploreArea** в системе автоматизированного проектирования Vivado от Xilinx направлена на оптимизацию потребления энергии в проектируемых схемах, с акцентом на эффективное использование доступной площади. Эта стратегия помогает находить баланс между площадью, производительностью и энергопотреблением, что является критически важным в современных проектах на FPGA.
21. Стратегия **Flow_Quick** в системе автоматизированного проектирования Vivado от Xilinx предназначена для быстрого выполнения процессов проектирования, включая размещение и маршрутизацию логических элементов. Эта стратегия нацелена на сокращение времени, необходимого для завершения разработки, и часто используется в случаях, когда требуется быстрое прототипирование или первичное тестирование дизайна.

22. Стратегия **Performance_LBlockPlacementFanoutOpt** в системе автоматизированного проектирования Vivado от Xilinx нацелена на оптимизацию размещения логических блоков с учетом их распредения по выходам (fan-out). Эта стратегия направлена на улучшение временных характеристик схемы, особенно в контексте управления задержками и эффективной маршрутизации сигналов.
23. Стратегия **Performance_Retiming** в системе автоматизированного проектирования Vivado от Xilinx предназначена для улучшения временных характеристик схемы через перераспределение регистров и оптимизацию задержек в логических цепях. Этот процесс позволяет уменьшить задержки на критических путях, что, в свою очередь, помогает достичь более высокой производительности устройства.
24. Стратегия **Performance_BalanceSLL** в системе автоматизированного проектирования Vivado от Xilinx направлена на оптимизацию размещения логических элементов с целью достижения баланса в производительности для малых логических уровней (Small Logic Levels, SLL). Эта стратегия помогает улучшить временные характеристики схемы, обеспечивая эффективное распределение ресурсов на кристалле.
25. Стратегия **Congestion_SpreadLogic_Explore** в системе автоматизированного проектирования Vivado от Xilinx нацелена на исследование и оптимизацию размещения логических элементов с целью снижения рисков перегрузки (congestion) в проектируемых схемах. Эта стратегия помогает найти баланс между производительностью и эффективностью использования площади кристалла.
26. Стратегия **Area_ExploreSequential** в системе автоматизированного проектирования Vivado от Xilinx ориентирована на оптимизацию размещения последовательных логических элементов, таких как регистры и другие элементы, которые участвуют в обработке данных во времени. Эта стратегия

помогает эффективно использовать площадь кристалла, а также улучшать производительность и временные характеристики схемы.

27. Стратегия **Flow_RunPhysOpt** в системе автоматизированного проектирования Vivado от Xilinx направлена на оптимизацию физического размещения и маршрутизации схемы после выполнения начальных этапов проектирования. Эта стратегия фокусируется на улучшении временных характеристик и снижении энергопотребления, обеспечивая более эффективное использование ресурсов FPGA.

---

1. **Архитектурные аспекты проектирования. Часть 2**

Архитектурное проектирование электронных систем направлено на создание структуры, обеспечивающей высокую производительность, гибкость и масштабируемость. Оно учитывает функциональные и нефункциональные требования системы, включая энергоэффективность, надежность и поддержку расширяемости. Ключевые архитектурные подходы включают выбор типа процессора (или микропроцессорного ядра), способа взаимодействия между компонентами, а также оптимизацию взаимодействий между аппаратной и программной частями системы. Например, можно выбрать многоядерную архитектуру для повышения параллелизма или использовать специализированные ускорители для конкретных задач, таких как графика или машинное обучение.

---

2. **Паттерн проектирования «Процессорное ядро»**

Паттерн «Процессорное ядро» используется для построения иерархической, модульной структуры микропроцессора, где процессорное ядро выполняет основные вычисления и управление потоком данных. Оно является ядром вычислительной системы, объединяющим несколько компонентов, таких как регистры, арифметико-логическое устройство (АЛУ) и контроллер команд. Процессорное ядро обрабатывает инструкции, поддерживает параллельные вычисления и выполняет базовые арифметические, логические и побитовые операции. Этот паттерн облегчает проектирование и упрощает масштабирование процессоров.

Основные компоненты процессорного ядра

Паттерн «Процессорное ядро» предполагает, что его структура должна состоять из нескольких ключевых компонентов:

1. **Арифметико-логическое устройство (АЛУ)**:
   - Выполняет основные арифметические и логические операции (сложение, вычитание, логическое «и», «или» и др.).
   - Часто включает блоки для более сложных математических вычислений, таких как умножение и деление.
2. **Регистровый файл**:
   - Набор регистров, которые используются для хранения промежуточных данных, адресов и результатов операций.
   - Регистр состояния (или флагов), который фиксирует результаты операций (например, результат равен нулю, отрицательный результат, переполнение и др.).
3. **Блок управления**:
   - Управляет порядком выполнения команд, поступающих из памяти, и задает последовательность сигналов для других блоков ядра.
   - Включает декодер команд, который интерпретирует каждую инструкцию и направляет ее выполнение.
4. **Контроллер памяти**:
   - Управляет доступом к памяти и взаимодействием ядра с внешней оперативной памятью (ОЗУ) и постоянной памятью (ПЗУ).
   - В современных процессорах контроллер памяти может быть встроен в ядро, что улучшает производительность.

5. **Шины данных и адресов**:
    - Используются для передачи данных и адресов между компонентами ядра и внешними устройствами.
    - В зависимости от архитектуры (например, Гарвардской или фон Неймана) шины могут быть объединены или разделены для данных и команд.

Основные этапы проектирования процессорного ядра

При использовании паттерна «Процессорное ядро» обычно применяются следующие этапы проектирования:

1. **Определение набора команд (ISA)**:
    - Определяются инструкции, которые ядро может выполнять. Например, это может быть RISC (с минимальным набором простых команд) или CISC (с более обширным и сложным набором команд).
    - Определяется ширина команды, что может варьироваться от 8 бит до 64 бит и выше.
2. **Проектирование АЛУ**:
    - Разрабатывается АЛУ, способное выполнять все необходимые операции.
    - Выбирается подход к обработке чисел (например, целочисленные или с плавающей точкой).
3. **Создание регистрового файла и блоков памяти**:
    - Проектируются регистры, обеспечивающие временное хранение данных и инструкций.
    - Настраиваются параметры регистров, такие как размер, количество и доступность (например, общее или отдельное хранение данных и адресов).
4. **Разработка системы управления**:
    - Создается контроллер команд, управляющий порядком и условиями выполнения инструкций.
    - Система управления также отвечает за обработку прерываний и исключений, обеспечивая надежное выполнение кода.
5. **Сопряжение с внешними устройствами и интерфейсами**:
    - Разрабатываются шины и контроллеры для передачи данных между ядром и периферийными устройствами, такими как память, ввод-вывод и сеть.

---

3. **Переход от конечного автомата к процессорному ядру**

Проектирование на основе конечного автомата полезно на начальных этапах создания вычислительных систем, где простая логика и ограниченное количество состояний управляют системой. Однако, при увеличении сложности задач возникает необходимость перехода к процессорному ядру, которое способно выполнять сложные последовательности команд и обеспечивать более гибкое управление. Процессорное ядро предлагает мощную модель обработки, где набор команд позволяет запускать последовательности состояний и операций, выходящих за рамки фиксированных переходов конечного автомата.

Конечный автомат — это система с конечным количеством состояний, где каждое состояние определяет, какое действие выполняется системой. Конечные автоматы эффективны для выполнения задач с фиксированной логикой, такими как:

- Простое управление процессом или сигналами;
- Линейная последовательность действий, зависящая от условий и ограниченного набора переходов между состояниями;
- Управление простыми цифровыми системами (например, контроллерами).

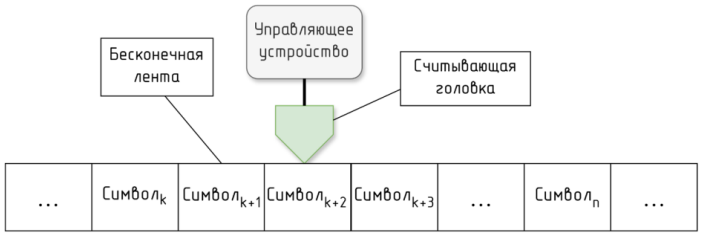
Конечные автоматы ограничены в своей гибкости, так как изменения логики требуют полного пересмотра модели автоматов и часто сложны в реализации для более обширных и абстрактных программ.

---

4. **Понятие абстракции машины Тьюринга**

Машина Тьюринга представляет собой абстрактную математическую модель вычислений. Она состоит из бесконечной ленты, разделенной на ячейки, головки для чтения и записи, и конечного набора правил. Эта модель помогает абстрактно понять и спроектировать вычислительные системы, так как машина Тьюринга способна выполнять любые вычисления, которые может выполнить цифровой компьютер, следуя заранее заданной последовательности команд. При проектировании процессорных ядер абстракция машины Тьюринга используется для создания общей логики выполнения команд и управления состоянием системы.

Абстракция машины Тьюринга



---

5. **Структура процессорного ядра**

Основные компоненты процессорного ядра включают:

- **Регистровый файл**: Совокупность регистров, используемых для хранения временных данных и операндов.

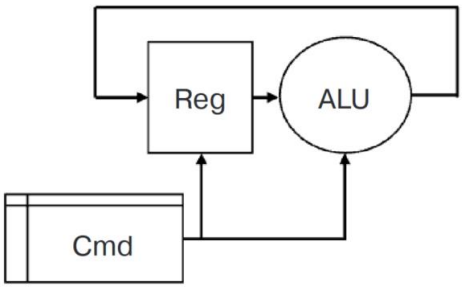
- **АЛУ (арифметико-логическое устройство)**: Выполняет арифметические и логические операции.

- **Контроллер команд**: Отвечает за выбор и выполнение команд из памяти.

- **Кэш и блок управления памятью**: Обеспечивает хранение данных и команд, управляет доступом к памяти.

- **Шины данных и адресов**: Используются для передачи данных между компонентами ядра и внешними устройствами.

Структура процессорного ядра



Эти элементы работают совместно, обрабатывая инструкции и выполняя операции, задающие основные функции ядра процессора.

---

6. **Классификация процессоров**

Процессоры можно классифицировать по нескольким параметрам:

- **По архитектуре набора команд**: CISC (Complex Instruction Set Computer) и RISC (Reduced Instruction Set Computer).

- **По числу команд, исполняемых за такт**: скалярные и векторные процессоры.

- **По типу организации памяти**: процессоры Гарвардской архитектуры и архитектуры фон Неймана.

- **По числу ядер**: однопроцессорные и многоядерные системы.

- **По назначению**: универсальные процессоры, специализированные (например, графические, цифровые сигнальные процессоры).

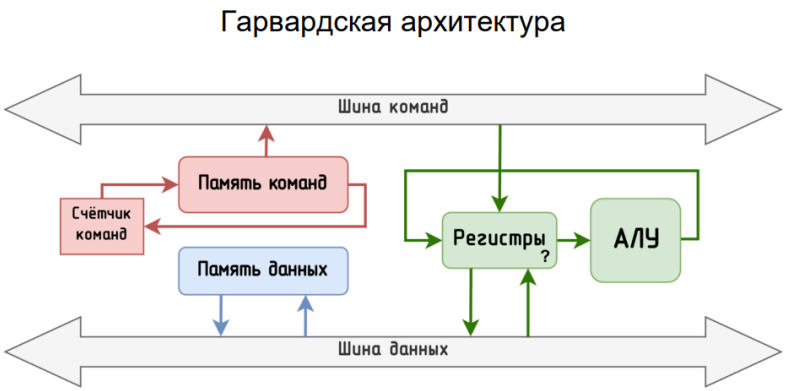
Классификация процессоров

Каждая из этих классификаций отражает различия в архитектурных подходах и назначении процессоров.

---

7. **Гарвардская архитектура**

Гарвардская архитектура предполагает раздельные шины и память для команд и данных, что позволяет одновременно выполнять операции чтения команд и данных, повышая скорость работы. Эта архитектура популярна в микроконтроллерах и DSP-процессорах, так как обеспечивает высокую производительность и минимизирует задержки при доступе к памяти. Примером использования Гарвардской архитектуры являются микроконтроллеры семейства PIC и DSP-процессоры, где параллельная работа команд и данных снижает временные издержки.

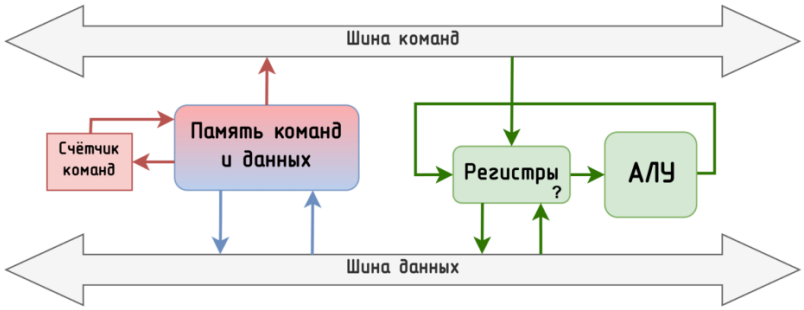


---

8. **Архитектура фон Неймана**

В архитектуре фон Неймана используется единая память и шина как для данных, так и для команд. Команды и данные передаются через общую шину, что создает узкое место, известное как «бутылочное горлышко фон Неймана», поскольку одновременно возможен только один доступ к памяти. Эта архитектура проще и дешевле, чем Гарвардская, и подходит для компьютеров общего назначения. Современные компьютеры, такие как x86-процессоры, используют модифицированную архитектуру фон Неймана с кэшированием, чтобы минимизировать проблемы узкого места.

Архитектура Фон-Неймана



---

9. **Пример построения простого процессорного ядра**

Пример простого процессорного ядра может включать следующие шаги:

- **Создание АЛУ**: Это устройство выполняет базовые арифметические и логические операции, такие как сложение, вычитание и побитовые операции.

- **Реализация регистрового файла**: Набор регистров используется для хранения данных, доступных на всех этапах выполнения команд.

- **Добавление блока управления**: Этот блок управляет порядком выполнения команд, извлекая команды из памяти и направляя их на выполнение.

- **Организация памяти**: В простом ядре можно использовать один блок памяти для хранения команд и данных.

- **Подключение шины данных и адресов**: Она связывает блоки памяти с АЛУ и регистровым файлом.

Пример в Vivado

Код операции	Адрес операнда 1	Адрес операнда 2	Адрес результата
1 бит	n бит	n бит	n бит

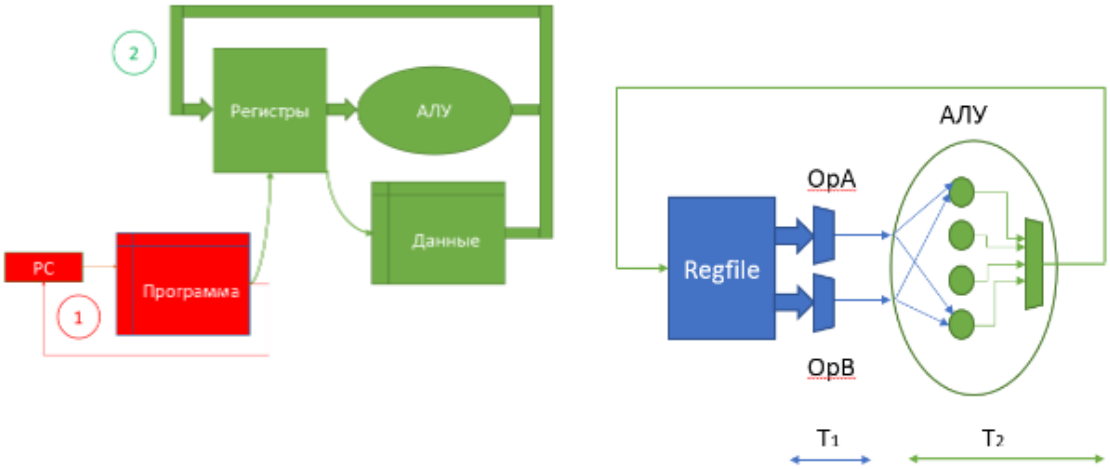
Такой минималистичный процессор, выполняющий простые инструкции (например, операции с фиксированным числом тактов на команду), можно построить на основе простого конечного автомата, где каждое состояние представляет одну инструкцию или ее часть.

ЛЕКЦИЯ №5 - АРХИТЕКТУРНЫЕ АСПЕКТЫ ПРОЕКТИРОВАНИЯ. ЧАСТЬ 3

ПРОБЛЕМЫ ДВУХТАКТНОЙ АРХИТЕКТУРЫ.

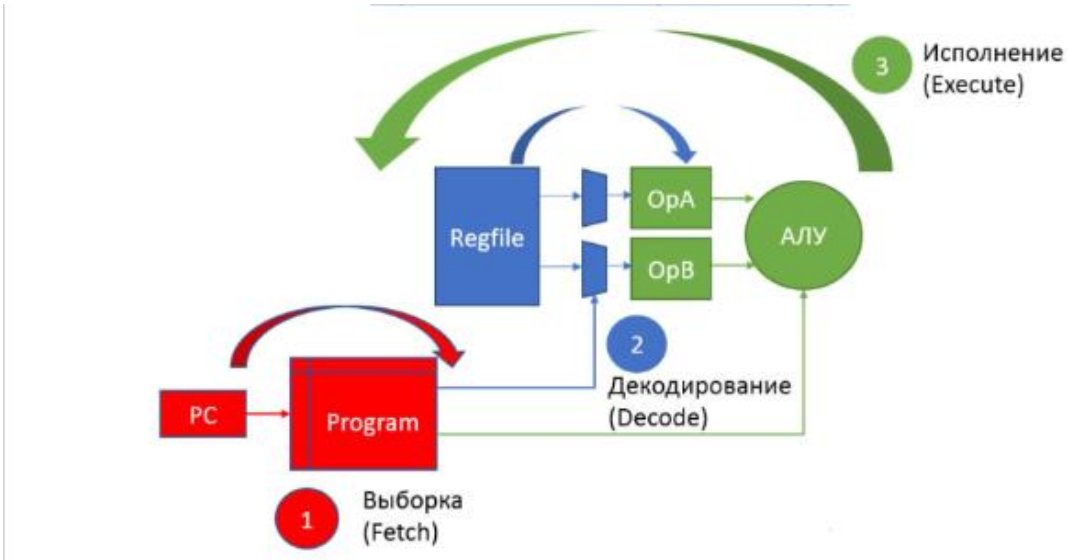
Простейший конвейер, работающий по схеме «выборка – исполнение», имеет явно видимый потенциал развития. Рассмотрим рис. 1, где схематично показаны задержки в тракте данных. Здесь видно, что для подготовки операндов OpA и OpB для арифметико-логического устройства нужно сначала выбрать их из регистрового файла с помощью мультиплексора. Чем больше размер регистрового файла, тем более сложный мультиплексор для этого понадобится. Полученные операнды будут поданы на входы АЛУ, в котором также имеется мультиплексор, выбирающий результат из нескольких вариантов. Повышение тактовой частоты возможно, если применить конвейеризующие регистры, очевидным расположением для которых являются точки OpA и OpB на входе АЛУ.

При увеличении длины конвейера возникают и другие подобные ситуации, часть из них ведет к появлению логических ошибок работы программы, а часть — и к электрическим конфликтам. Для рассмотрения основных проблем, подлежащих решению, необходимо сначала определить, каким целям служит увеличение числа стадий конвейера.



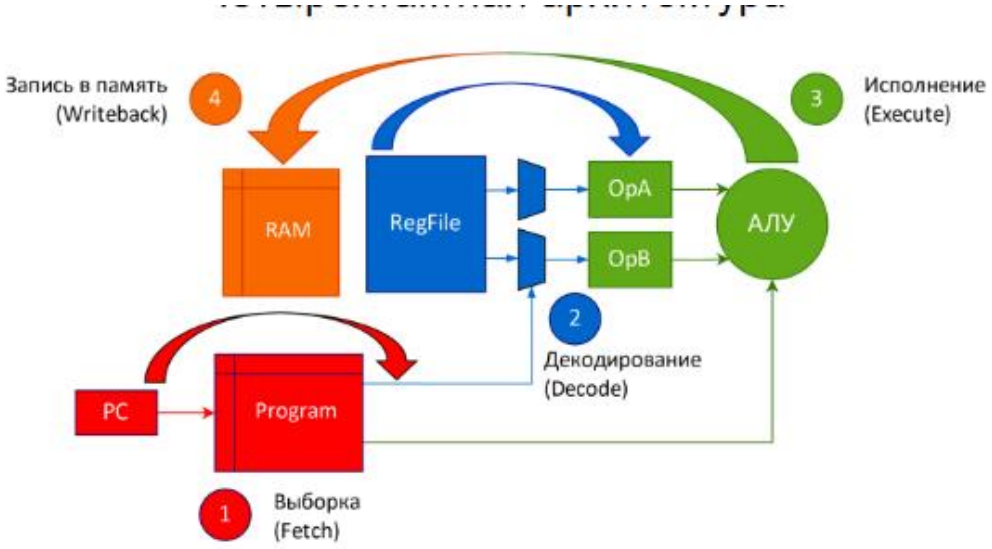
ТРЁХТАКТНАЯ АРХИТЕКТУРА

На рис показана архитектура, использующая трехтактный конвейер со стадиями «выборка» (Fetch), «декодирование» (Decode), «исполнение» (Execute). Такая схема потенциально достигает более высокой тактовой частоты, поскольку тракт данных конвейеризован и задержки чтения операндов из регистрового файла больше не прибавляются к задержке в АЛУ. На практике, тем не менее, необходимо оценить получаемый выигрыш, поскольку тактовая частота может быть ограничена и другими факторами.



ЧЕТЫРЁХТАКТНАЯ АРХИТЕКТУРА

Дополнительный такт для сложных операций или для реализации более сложных инструкций. Добавляется writeback (запись в память)



ПЯТИТАКТНАЯ АРХИТЕКТУРА

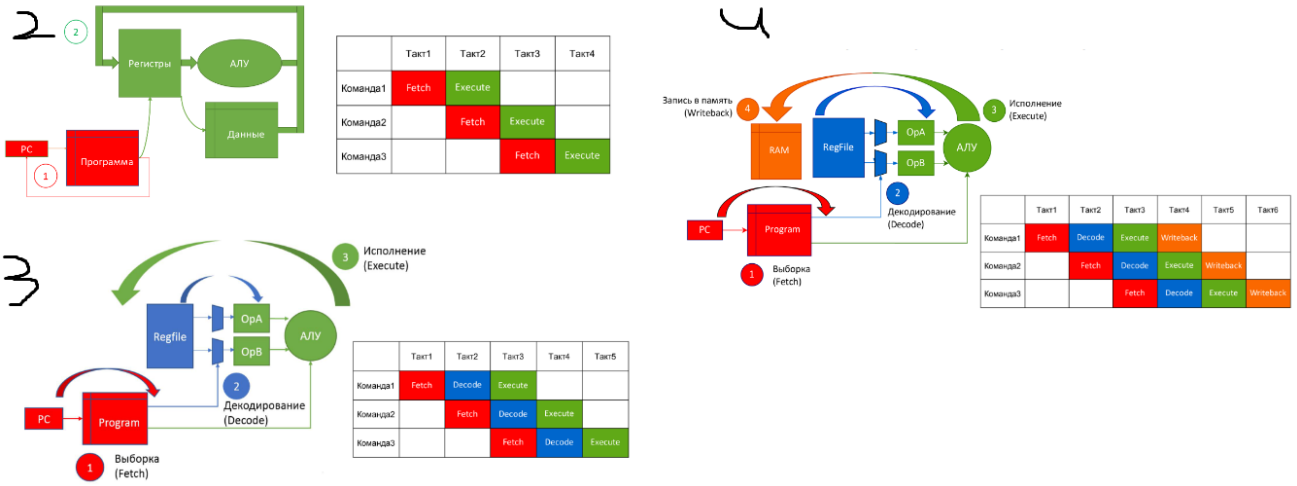
Используется для реализации очень сложных инструкций, например, с обработкой векторов или с многоступенчатыми операциями.

КОНВЕЙЕРНОЕ ПРОЦЕССОРНОЕ ЯДРО 2, 3, 4 ТАКТА:

* 2 такта: Инструкция разбивается на две стадии: получение и исполнение.

* 3 такта: Добавляется стадия записи результата.

* 4 такта: Может добавляться стадия для более сложных операций, например, для вычисления адреса памяти.



ОСНОВНЫЕ КОНФЛИКТЫ В КОНВЕЙЕРНОЙ АРХИТЕКТУРЕ.

«Чтение после чтения» (RAR, Read After Read) — из одного и того же физического ресурса (регистра или ячейки памяти) производятся два последовательных чтения. Эта ситуация не формирует конфликтов, поскольку первое чтение не влияет на содержимое регистра и не мешает прочитать его второй раз.

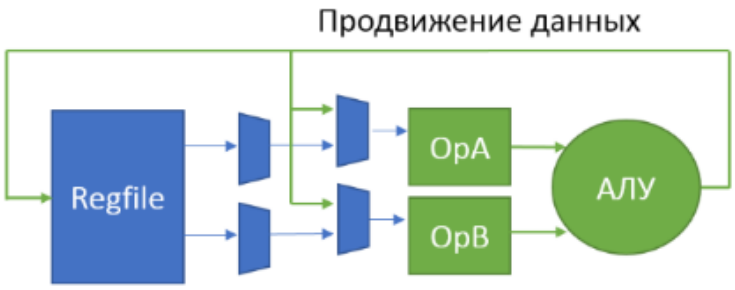
«Чтение после записи» (RAW, Read After Write) — наиболее проблемное положение, соответствующее показанному на рис. 4. При наличии конвейеризации может возникнуть ситуация, когда данные, предназначенные для записи, еще не попали в регистр (или память), однако следующая команда, находясь в конвейере, уже производит их чтение.

«Запись после чтения» (Write After Read).

«Запись после записи» (Write After Write).

АРХИТЕКТУРА MIPS

Процессор без блокировок в конвейере (Microprocessor without Interlocked Pipeline Stages) — это и способ организации конвейера, и конкретный продукт, разработанный компанией MIPS Technologies (ранее MIPS Computer Systems). В данной архитектуре используется полезный прием разрешения конфликтов по доступу к регистрам, называемый «продвижение данных» (data bypass). Его суть можно проиллюстрировать на рис.



На рис можно видеть способ разрешения конфликта при последовательном доступе к одному и тому же регистру на запись и на чтение. Как было показано выше, такая ситуация должна вызвать конфликт, поскольку в момент записи нового значения на вход АЛУ попадает старое содержимое регистра. При реализации подхода data bypass новое значение записывается не только в сам регистр, но и в регистр операнда, если номер регистра для записи результата совпадает с номером регистра операнда.

Несмотря на то, что на рис показан дополнительный мультиплексор, при должном подходе снижение тактовой частоты такой схемы окажется

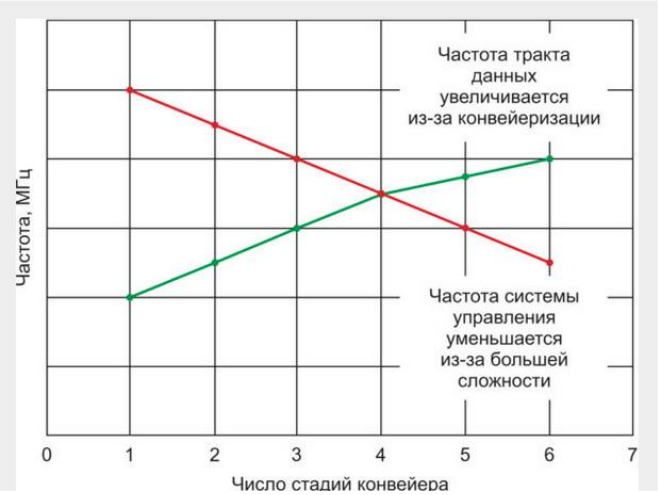
несущественным. Для этого необходимо убедиться, что данные из АЛУ подаются именно на последний мультиплексор в цепочке, то есть после мультиплексирования одного из регистров в регистры OpA, OpB записывается либо значение регистра, либо данные из АЛУ. В этом случае суммарная задержка распространения сигнала от OpA, OpB через АЛУ увеличится всего на один мультиплексор «2-в-1».

ПРОБЛЕМА ОТЛОЖЕННОГО ПЕРЕХОДА

Программный обход блокировок конвейера в чем-то подобен отложенным переходам. В процессорах с отложенным переходом команды, уже находящиеся в конвейере, будут выполнены даже при необходимости нарушить последовательное выполнение, однако разрешение этой проблемы переложено на компилятор.

Соотношение производительности тракта данных и управляющей схемы

При чрезмерно агрессивной оптимизации частота тракта данных будет достаточно высокой, но при этом общая частота процессора будет ограничена уже сложностью управляющих систем, проверяющих конфликты из-за зависимостей по данным на разных стадиях конвейера.



АРХИТЕКТУРЫ VLIW, EPIC

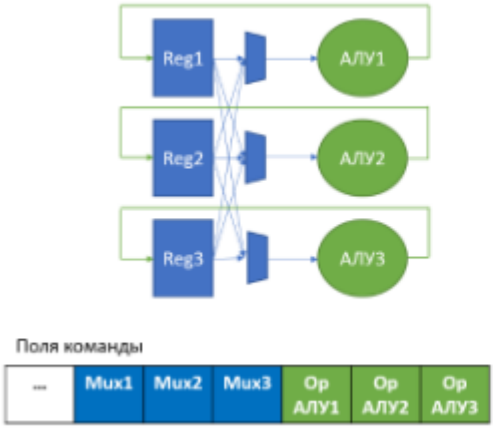
Сверхдлинное командное слово требуется, если в команде процессора оказывается много отдельных полей. Потребность в дополнительных полях может возникнуть, если за один такт будет параллельно выполняться несколько операций. Пример организации конвейера данных VLIW-процессора показан на рис. 6. На рисунке видно, что несколько регистровых файлов (Reg1–Reg3) через коммутаторы подают операнды на несколько АЛУ (АЛУ1–АЛУ3). Коммутаторы могут иметь разные варианты организации (в примере показано, что выходы любого регистрового файла могут быть поданы на входы любого АЛУ), поэтому в таком процессоре возможно параллельное выполнение трех разных команд в трех АЛУ. Подобный подход соответствует параллелизму уровня инструкций (ILP, Instruction Level Parallelism).

Размер команды на рис. 6 увеличивается, потому что каждому из АЛУ необходимо определить выполняемую на этом такте операцию, а каждому коммутатору — выбираемый им операнд. Однако это не означает, что размер команды пропорционален количеству АЛУ. Если предположить, что команда имеет 32 разряда, добавление АЛУ потребует 4–5 разрядов для выбора операции, а дополнительному мультиплексору понадобится 2–3 разряда (в зависимости от количества его входов).

Размер команды для VLIW-процессоров указывается в пределах 128–256 разрядов. Рекордсменом является процессор Avispa+ компании Silicon Hive, имеющий 768-разрядную команду, однако при такой разрядности команды этот процессор имеет 60 слотов операций.

Преимущество VLIW заключается в возможности выполнения нескольких операций за один такт. Однако это формальное преимущество не всегда можно применить на практике. Для того чтобы параллельно выполнить несколько операций, для них должны быть готовы данные. Если типичные целевые алгоритмы предусматривают в основном линейные последовательности команд, в которых

вычисленный результат используется как операнд следующей команды, дополнительные АЛУ, скорее всего, будут простаивать. Поэтому для VLIW большую роль играют предварительные исследования алгоритмов и возможности имеющегося компилятора.



- Very Logic Instruction Word (VLIW) - сверхдлинное командное слово
- Explicit Parallel Instruction Compiler (EPIC) - процессор с явным параллелизмом

Архитектура VLIW имеет разновидность, отмечающую тот факт, что в процессоре происходит прямое управление всеми вычислительными и коммутационными устройствами. Такая архитектура называется Explicit Parallel Instruction Computer (EPIC) — процессор с явным параллелизмом. Если для VLIW-процессоров часто использовались аппаратные планировщики, обеспечивавшие автоматический выбор операндов для АЛУ по мере их готовности, то в EPIC делается больший акцент на возможности компилятора по прямому управлению всеми устройствами процессора. Как и для VLIW, сам факт применения архитектуры EPIC не позволяет достичь увеличения производительности, если особенности выполняемой программы не предполагают параллельных вычислений.

РЕГИСТРОВАЯ МОДЕЛЬ ПРОЦЕССОРА (СЛИШКОМ МНОГО)

https://fpga-e.ru/design/proektirovanie-proczessornyh-yader-chast-2/\

РЕГИСТРОВЫЙ ФАЙЛ

Регистровый файл — это группа регистров, имеющих сходное назначение. Для регистрового файла обычно предусмотрен общий интерфейс, допускающий единообразный доступ к любому регистру. Поэтому удобнее описывать регистровый файл в виде линейного массива регистров.

АДРЕСНОСТЬ КОМАНД

Под адресностью понимается количество адресов (индексов регистров процессора), участвующих в выполнении команды. «Адрес» можно трактовать в широком смысле, как любое указание на ресурс процессора, участвующий в вычислениях или получающий результат.

Примером трехадресной команды служит команда вида:

R1 = R2+R3

В данном случае команда содержит указания на три регистра — получатель результата, первый операнд и второй операнд. Для того чтобы определить все три регистра, команда должна иметь три битовых поля, где будут помещены соответствующие номера. Размер полей зависит от количества регистров, которые можно адресовать таким способом. Так, 16 регистров потребуют 4-разрядных полей. Примером 3-адресных команд являются команды процессоров ARM.

Архитектуры RISC, CISC

Эти аббревиатуры означают Complex Instruction Set Computer и Reduced Instruction Set Computer, или компьютеры (процессоры) с полным набором команд и процессоры с сокращенным набором команд. Это достаточно старое разделение и соответствует выбору между сложным многотактным управляющим автоматом и простой системой работы, при которой каждый машинный цикл соответствует завершенной команде. Современные процессоры обычно выполняются по архитектуре RISC, однако это не означает,

будто их возможности в чем-то сокращены. Команды CISC обычно могут быть выражены программно с помощью нескольких команд RISC.

Характеристика	CISC	RISC
Длина команды	Произвольная в зависимости от характера инструкции	фиксированная
Набор команд	Большой, адаптирован для удобства программиста	Средний, адаптирован для удобства выполнения процессором
Доступ к памяти	Разрешён для инструкций разного рода	Разрешен только для инструкций загрузки и сохранения

Простые системные шины — это каналы передачи данных, которые соединяют различные компоненты компьютера. Основные моменты включают:

- Типы шин: могут быть параллельными (одновременный обмен данными) и последовательными (по очереди).

- Архитектура: системная шина состоит из трех основных шины — данных, адреса и управления.

  - Шина данных: передает саму информацию.

  - Адресная шина: указывает, куда отправляется информация.

  - Управляющая шина: контролирует процессы передачи данных.

- Пропускная способность: определяется количеством битов, передаваемых за единицу времени; чем выше, тем лучше производительность.

- Стандарты: AT, ISA, PCI, PCIe — разные системы с различными характеристиками и скоростями передачи.

Эти системы играют ключевую роль в эффективной работе компьютера.

Параллельные системные шины – это системы передачи данных, где несколько бит информации передаются одновременно по отдельным проводникам. Это обеспечивает высокую скорость передачи данных, что важно для современных вычислительных устройств.

Основные характеристики:

- Ширина шины: Определяется количеством передаваемых бит одновременно (например, 8, 16, 32 бита).

- Скорость: Позволяет передавать данные быстрее по сравнению с последовательными системами.

- Применение: Используются в компьютерах для связи между процессором, памятью, и периферийными устройствами.

Отличие от простой (последовательной) шины:

- Параллельная шина передает множество бит одновременно, тогда как простая шина передает биты по одному.

- Это приводит к более высокой пропускной способности у параллельных шин, но они требуют больше проводников и могут быть подвержены интерференции.

Системные шины ISA (Industry Standard Architecture) – это стандартизированный интерфейс для подключения периферийных устройств к ноутбукам и настольным компьютерам.

Ключевые характеристики:

- Ширина шины: 8 или 16 бит.

- Частота: Обычно до 8 МГц.

- Совместимость: Поддерживает множество устройств, таких как видеокарты, звуковые карты и сетевые адаптеры.

- Стандарт: Применяется в основном в старых ПК, начиная с IBM PC XT в 1981 году.

Преимущества:

- Простота подключения и настройки.

- Огромный выбор доступных устройств.

Недостатки:

- Ограниченная скорость передачи данных и пропускная способность по сравнению с современными шинами (например, PCI).

Системные шины PSI (Peripheral System Interface)

- Описание: PSI — это интерфейс для соединения между различными компонентами компьютерной системы, такими как процессоры, память и периферийные устройства.

- Типы шин: Существуют разные типы системных шин, включая адресные, данные и управляющие.

- Функция: Обеспечивает передачу данных и команд между устройствами, что обеспечивает эффективное взаимодействие.

- Применение: Используется в различных архитектурах компьютеров и встраиваемых системах для обеспечения совместимости и быстродействия.

- Переход на новые стандарты: С развитием технологий, такие как PCIe, становятся более популярными, так как предлагают большую скорость передачи данных.

Если нужно больше деталей, дай знать!

Wishbone — это открытый стандарт интерфейса для соединения цифровых компонентов, особенно в системах на кристалле (SoC). Основные характеристики:

- Модульная архитектура: Wishbone позволяет легко интегрировать различные компоненты, такие как процессоры, память и устройства ввода-вывода, создавая гибкие системы.

- Типы интерфейсов: включает 4 основных типа интерфейсов: Wishbone B (параллельная шина), Wishbone C (синхронный интерфейс), Wishbone D (для асинхронных соединений) и Wishbone R (для повышения производительности).

- Универсальность: поддерживает различные протоколы передачи данных, что делает его совместимым с разными архитектурами.

- Производительность: оптимизирован для минимизации задержек и повышения пропускной способности.

- Сообщество: активно используется в проектах с открытым исходным кодом, что способствует его популярности и развитию.

Сочетание процессорного управления и автономной работы

Процессорное управление основано на использовании микропроцессоров для выполнения задач управления в различных системах. Такое управление позволяет автоматизировать процессы, обеспечивая высокую точность и скорость реакции.

Автономная работа предполагает, что устройства функционируют без постоянного подключения к внешним источникам питания или сетям. Это критично для мобильных и удалённых систем, таких как дронов, роботов и сенсоров.

Ключевые преимущества:

1. Эффективность: Процессоры обрабатывают данные быстрее, чем механические системы.

2. Независимость: Устройства могут работать в любых условиях, не завися от внешних источников.

3. Гибкость: Легкость в настройке и адаптации под различные задачи.

Применения: используются в IoT, мобильной технике, экологическом мониторинге и робототехнике.

AXI (Advanced eXtensible Interface) — это интерфейс, разработанный для связи между компонентами систем на кристалле (SoC). Он предлагает высокую пропускную способность и низкую латентность.

AXI4-Lite — это версия AXI, предназначенная для менее требовательных приложений, которая поддерживает:

- Однонаправленную передачу данных на короткие расстояния.

- Простую архитектуру передачи данных, подходящую для управления устройствами.

- Ограниченная поддержка функций, что делает его легким в реализации.

- Подходит для конфигурации и управления периферийными устройствами в цифровых системах.

В целом, AXI4-Lite упрощает взаимодействие с устройствами, обеспечивая быструю и надежную связь.

Примером процессорной системы класса СНК (системы навигации и контроля) является GPS-навигатор. Эти устройства используют микропроцессоры для обработки сигналов от спутников GPS, позволяя пользователю точно определить свое местоположение.

Основные компоненты:

1. Микропроцессор: Обрабатывает данные о местоположении и выполняет вычисления.

2. Приемник GPS: Улавливает спутниковые сигналы.

3. Датчики: Могут включать акселерометры и гироскопы для улучшения точности.

4. Интерфейс пользователя: Экран и кнопки для ввода данных и отображения маршрутов.

Функции:

- Определение текущего местоположения.

- Построение маршрутов.

- Предоставление информации о пробках и дорожных условиях.

Такой GPS-навигатор работает автономно, что делает его полезным в поездках и в условиях ограниченного доступа к интернету.

Процессорные системы класса СНК (системы навигации и контроля) характеризуются следующими основными особенностями:

1. Высокая скорость обработки данных: Используют мощности современных процессоров для быстрого анализа информации.

2. Автономность: Многие устройства могут функционировать без постоянного подключения к сети.

3. Ключевые компоненты: Включают GPS-приемники, датчики и средства связи, что позволяет осуществлять комплексное отслеживание информации.

4. Многофункциональность: Способны выполнять различные задачи — от навигации до мониторинга состояния.

5. Интерактивность: Предоставляют пользователю возможность ввода данных и получения информации в реальном времени.

6. Интеграция с другими системами: Часто соединяются с мобильными и веб-приложениями для повышения функциональности.

Эти характеристики делают такие системы незаменимыми в области транспорта, логистики и управления.

ЛЕКЦИЯ №7 - СОПРЯЖЕНИЕ ИЗМЕРИТЕЛЬНЫХ И СИЛОВЫХ УСТРОЙСТВ С ЦИФРОВЫМИ СИСТЕМАМИ.

Для соединения измерительных и силовых устройств с цифровыми системами используют различные интерфейсы и преобразователи (например, АЦП и ЦАП), чтобы передать данные с аналоговых датчиков на микроконтроллеры и микропроцессоры. Также применяется гальваническая развязка для защиты от помех и электрической изоляции цепей.

Для преобразования сигналов между аналоговым и цифровым представлением используются специальные микросхемы – аналогоцифровые (АЦП) и цифро-аналоговые (ЦАП) преобразователи. Они широко используются во встраиваемых устройствах, управляющих бытовыми приборами, устройствами связи, промышленными установками и т.д.

ВВОД АНАЛОГОВЫХ СИГНАЛОВ В КОМПЬЮТЕРНЫХ СИСТЕМАХ. АЦП И ЕГО ХАРАКТЕРИСТИКИ:

Аналого-цифровой преобразователь (АЦП) позволяет преобразовать аналоговые сигналы в цифровые для дальнейшей обработки. Основные характеристики АЦП — это разрядность (количество бит), скорость выборки, точность, линейность и разрешение. Эти параметры определяют, насколько точно и быстро устройство может измерять и преобразовывать входные сигналы.

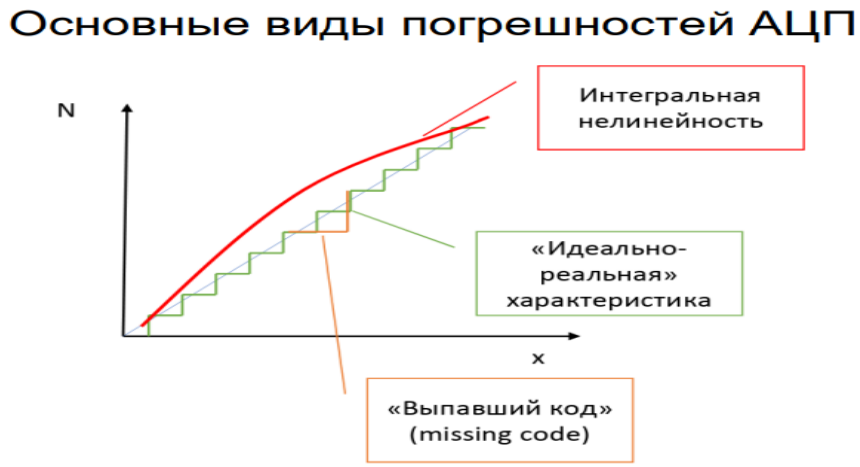
Характеристики АЦП:

- Разрядность (8-16, 24)
- Частота преобразования (от кГц до ГГц)
- Дополнительные характеристики:
    - Интерфейс
    - Диапазон входных данных
    - Архитектура (влияет на возможные ошибки измерения)

Метрологические характеристики АЦП:

«Точность» и «разрешающая способность» - термины, которые часто путают при практическом использовании

- Точность – на сколько измеренная величина отличается от значения, измеренного эталонным прибором в системе СИ
- Разрешающая способность – на сколько одно значение отличается от другого.
- 10 бит АЦП – 1024 возможных значения. Например, от 0 до 1,023 В (с шагом 1 мВ)
- У АЦП должен быть точный источник опорного напряжения. На практике – 12-14 бит могут использовать внутренний источник.



АРХИТЕКТУРЫ И ИНТЕРФЕЙСЫ АЦП.

Существуют разные архитектуры АЦП: интегрирующие, сигма-дельта, последовательного приближения. Выбор архитектуры зависит от требований к точности, скорости и стоимости. Интерфейсы могут быть последовательными (SPI, I2C) или параллельными, что влияет на скорость передачи данных и сложность схемы сопряжения.

Существуют различные архитектуры АЦП, каждая из которых имеет свои особенности и подходит для разных приложений:

- **АЦП прямого преобразования (flash ADC):** обеспечивает высокую скорость преобразования, но требует большого количества компонентов, что ограничивает его использование в высокоразрядных системах.
- **АЦП последовательного приближения (SAR ADC):** характеризуется хорошим соотношением между точностью и скоростью. Эта архитектура широко применяется благодаря сбалансированным характеристикам.
- **Сигма-дельта АЦП:** отличается высокой точностью и хорошей фильтрацией шумов, используется для высококачественных аудиосистем и прецизионных измерений.

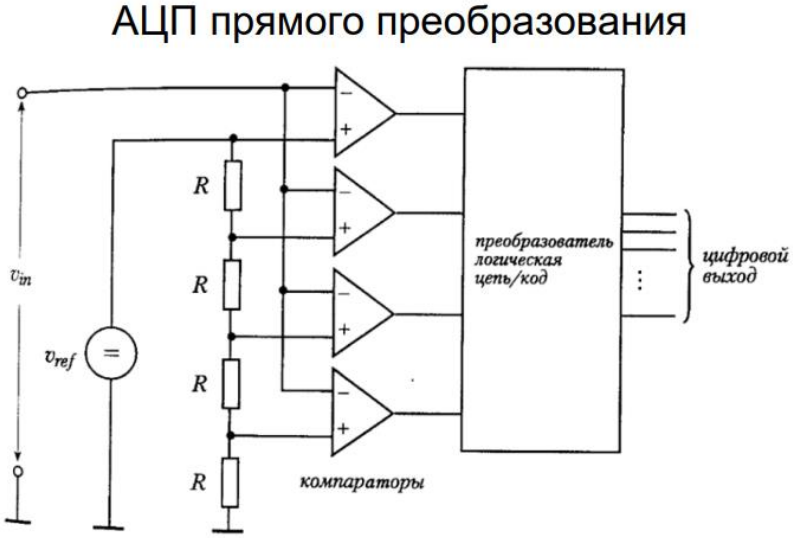
Flash-АЦП использует параллельную структуру, которая позволяет преобразовывать аналоговый сигнал в цифровой за один такт. Сигнал одновременно подается на большое количество (2^N - 1) компараторов, где N — разрядность АЦП. Каждый компаратор сравнивает входное напряжение с фиксированным порогом, генерируя двоичный код, представляющий входное значение.

Преимущества

- **Высокая скорость**: Flash-АЦП могут работать на частотах в несколько гигагерц и выполнять преобразование практически за один тактовый цикл, что делает их отличными для приложений, где важна высокая скорость (например, в осциллографах, системах связи и радарах).
- **Простота использования**: Мгновенное преобразование, отсутствие задержек.

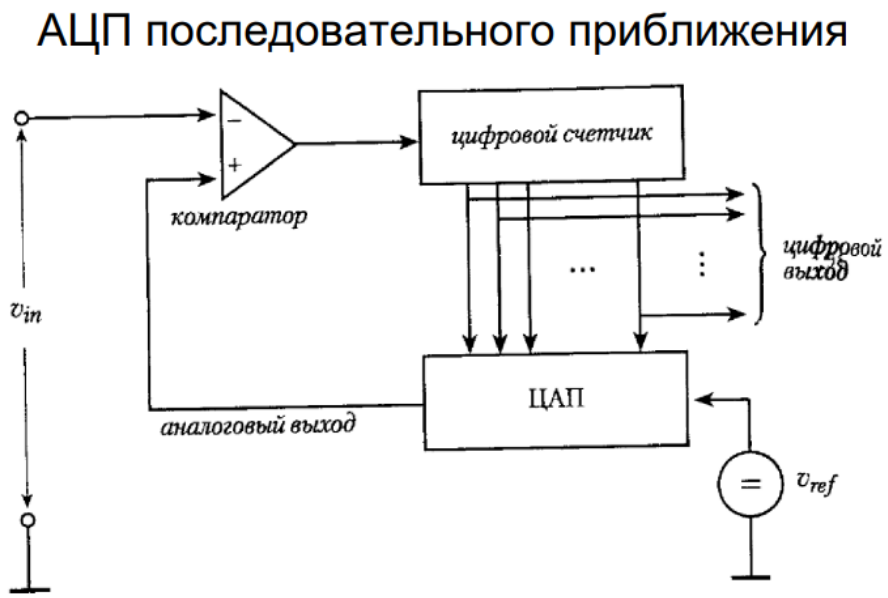
Недостатки

- **Высокое потребление энергии**: Параллельная структура и большое количество компараторов требуют много энергии, что ограничивает использование Flash-АЦП в энергоэффективных системах.
- **Большая сложность и стоимость**: С ростом разрядности число компараторов увеличивается экспоненциально (для 8-битного АЦП нужно 255 компараторов), что делает такие преобразователи дорогими и объемными.
- **Ограниченное разрешение**: Flash-АЦП обычно используются для низкой разрядности (4–8 бит) из-за сложности реализации на высоких разрядностях.



АЦП последовательного приближения (SAR, successive-approximation ADC)

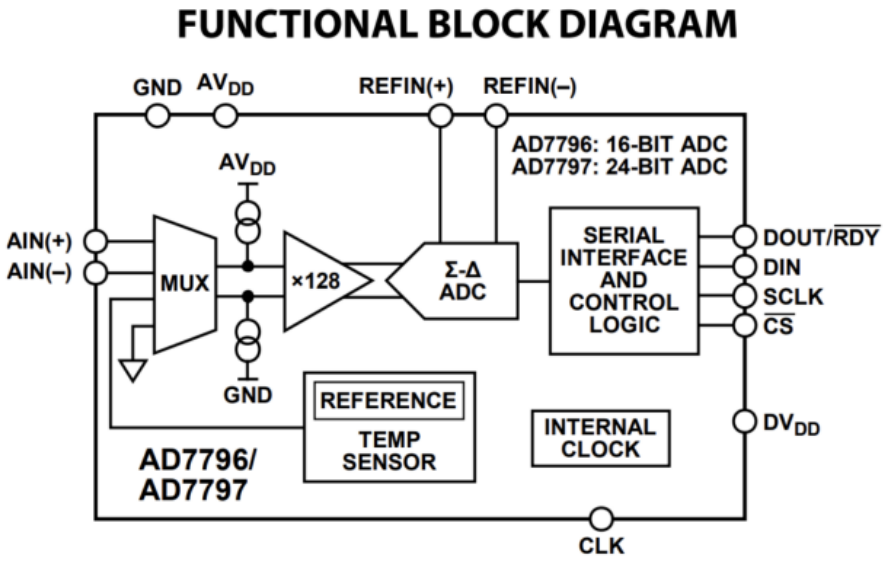
- **Принцип работы:** SAR-АЦП выполняет преобразование, приближаясь к значению входного сигнала пошагово. Внутри SAR-АЦП находится ЦАП и компаратор, которые делят входное напряжение пополам и итеративно проверяют, превышает ли входной сигнал это значение. Этот процесс продолжается до достижения заданного уровня точности.
- **Особенности:** SAR-АЦП очень быстрые и обеспечивают относительно высокое разрешение, поэтому они используются в приложениях, где требуется высокая скорость преобразования (например, в измерительных приборах, медицинской технике).
- **Плюсы:** Высокая скорость и хорошее разрешение, небольшая латентность, подходит для различных применений.
- **Минусы:** SAR-АЦП могут иметь более высокую стоимость и сложность, особенно при требовании высокой точности и разрядности.



АЦП с архитектурой “сигма-дельта” (sigma-delta)

- **Принцип работы**: Сигма-дельта АЦП (ΣΔ) используют принцип модуляции с увеличением частоты выборки и фильтрацией. Входной сигнал сначала поступает на схему, которая интегрирует его, и получается результат с высокой частотой выборки, состоящий из отдельных битов. Затем сигнал проходит через цифровой фильтр и понижается до нужной частоты.
- **Особенности**: Благодаря внутреннему фильтрованию, сигма-дельта АЦП достигают высокого разрешения и точности даже при наличии шума. Идеальны для звуковых и других низкочастотных сигналов.
- **Плюсы**: Высокая точность и разрешение, хорошая защита от шумов, особенно низкочастотных.
- **Минусы**: Достаточно высокая латентность и сравнительно низкая скорость, поэтому они не всегда подходят для быстрой обработки сигналов.

АЦП с архитектурой «сигма-дельта»



СОПРЯЖЕНИЕ АЦП С ЦИФРОВЫМИ СИСТЕМАМИ

Для подключения АЦП к цифровой системе (например, микроконтроллеру) используются интерфейсы, такие как SPI или I2C. Это позволяет передавать данные с минимальными задержками и защищает данные от внешних помех. Важно также учитывать напряжение питания и сигналы синхронизации.

Интерфейс между АЦП и микроконтроллером (или микропроцессором) определяет, насколько быстро и эффективно будут передаваться данные. Наиболее распространенные интерфейсы для сопряжения АЦП:

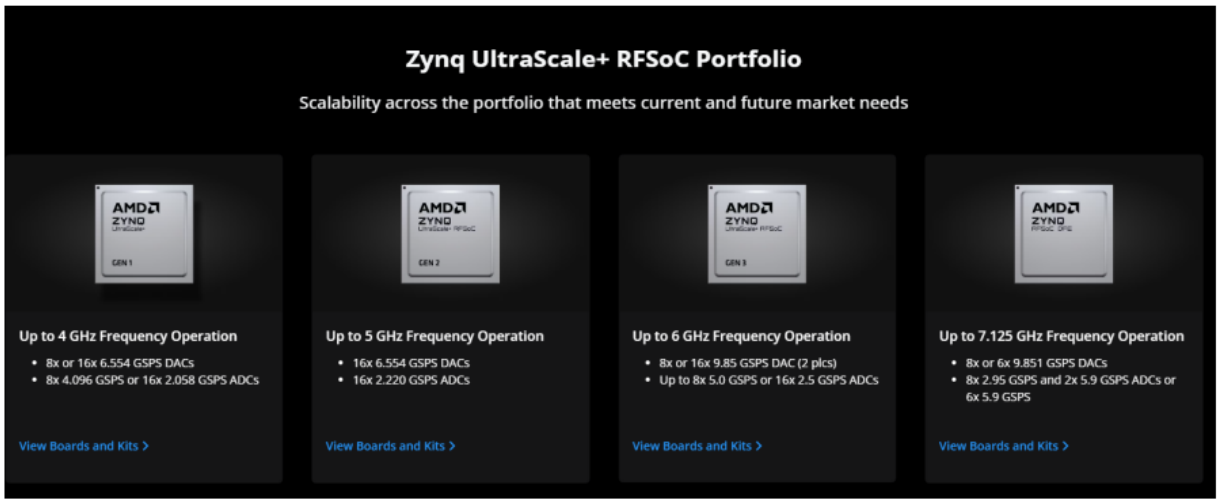
- **SPI (Serial Peripheral Interface)**: Последовательный интерфейс, обеспечивающий высокую скорость передачи данных. Он требует минимального количества соединений (четыре провода: MISO, MOSI, SCK, CS) и часто используется в высокоскоростных приложениях, поскольку позволяет передавать данные на тактовой частоте до нескольких десятков мегагерц.
- **I²C (Inter-Integrated Circuit):** Последовательный интерфейс с двумя проводами (SDA и SCL). Обладает меньшей скоростью по сравнению с SPI, но позволяет подключить несколько устройств к одной шине. Он предпочтителен, когда важно сэкономить выводы на микроконтроллере и использовать шину с несколькими устройствами.

ЦАП, ИНТЕРФЕЙСЫ ЦАП

Интерфейсы АЦП и ЦАП не имеют общепринятого стандарта и реализуются производителями микросхем в зависимости от частоты преобразования и назначения микросхемы.

Цифро-аналоговый преобразователь (ЦАП) преобразует цифровые данные в аналоговый сигнал, который может использоваться для управления аналоговыми устройствами. Основные интерфейсы для подключения ЦАП — это также SPI и I2C, которые обеспечивают совместимость с микроконтроллерами и микропроцессорами.

AMD RFSoC с встроенными блоками АЦП и ЦАП

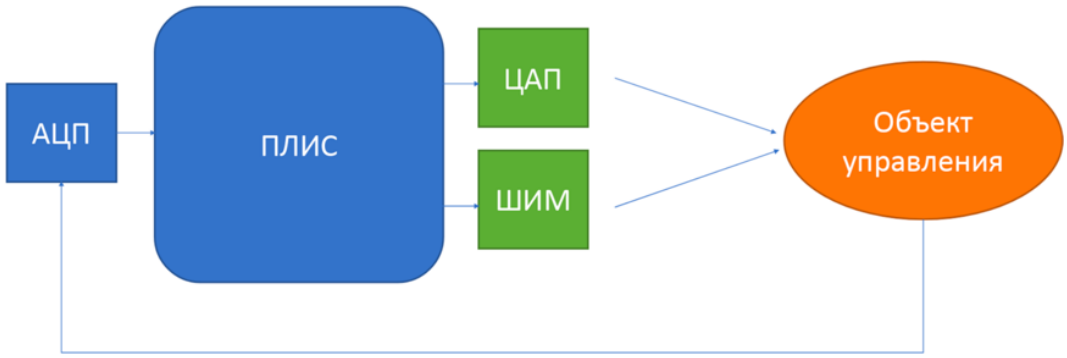


СОПРЯЖЕНИЕ ЦАП С ЦИФРОВЫМИ СИСТЕМАМИ

Для подключения ЦАП к цифровой системе также часто используются стандартные интерфейсы (SPI или I2C), что позволяет уменьшить количество выводов и упростить схему. При сопряжении важно учитывать временные характеристики для достижения необходимого качества сигнала.

УПРАВЛЕНИЕ СИЛОВЫМИ УСТРОЙСТВАМИ С ПОМОЩЬЮ ШИМ

Широтно-импульсная модуляция (ШИМ) используется для управления мощными устройствами, такими как электродвигатели и светодиодные ленты, за счёт изменения среднего значения выходного сигнала. Этот метод работает путем изменения соотношения времени включения и выключения сигнала (ширины импульса) при фиксированной частоте, что позволяет управлять мощностью, подаваемой на нагрузку, не изменяя уровня напряжения. ШИМ активно применяется в цифровых системах, где необходимо точное управление энергией, подаваемой на исполнительные устройства.



Эти темы охватывают основные аспекты сопряжения аналоговых и цифровых устройств, включая преобразование сигналов, характеристики и архитектуры преобразователей, а также управление мощными устройствами через ШИМ, что позволяет эффективно интегрировать аналоговые элементы в цифровые системы управления.

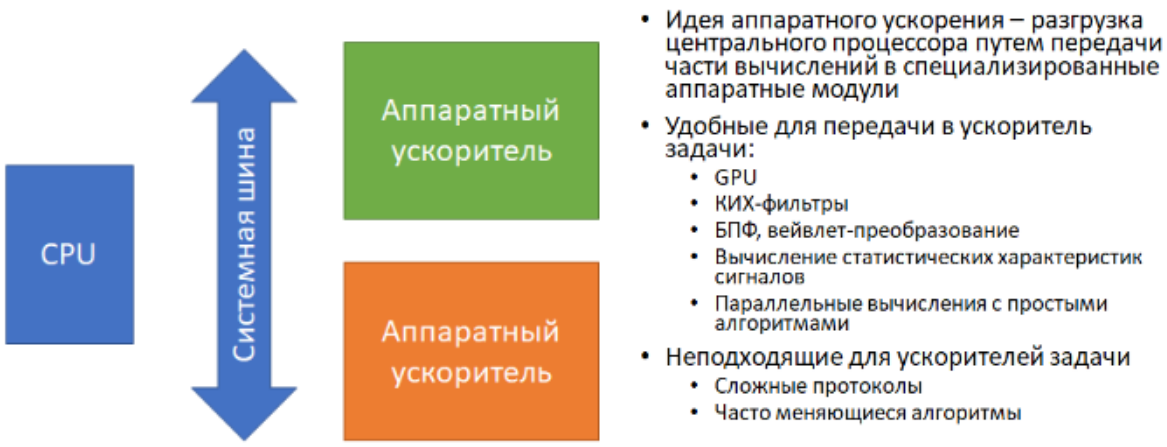
Датчики и исполнительные устройства

- В системе управления присутствуют датчики («чувствительные элементы») и «исполнительные устройства»
- Многие датчики представляют собой первичные преобразователи, размещенные в корпусе или на плате

- Для того, чтобы ввести данные о состоянии объекта управления, требуется комбинация АЦП и цифровых интерфейсов различных типов
- Для управления исполнительными устройствами используются цифровые интерфейсы, ЦАП, ШИМ (это цифровой интерфейс, но на него обращается внимание как на альтернативу ЦАП)

ЛЕКЦИЯ №8 - ПРОЕКТИРОВАНИЯ АППАРАТНОГО ОБЕСПЕЧЕНИЯ С ПРИМЕНЕНИЕМ ЯЗЫКОВ ВЫСОКОГО УРОВНЯ

АППАРАТНОЕ УСКОРЕНИЕ ВЫЧИСЛЕНИЙ.



Аппаратное ускорение вычислений — это процесс, в котором специализированные аппаратные компоненты используются для повышения производительности вычислений по сравнению с традиционными программными решениями, выполняемыми на общих процессорах. Это может включать в себя использование таких технологий, как FPGA (Field-Programmable Gate Arrays), ASIC (Application-Specific Integrated Circuits), GPU (Graphics Processing Units) и других специализированных устройств. Рассмотрим основные аспекты аппаратного ускорения вычислений.

1. Принципы работы аппаратного ускорения

- **Специализация**: Аппаратные ускорители проектируются для выполнения конкретных задач или классов задач, что позволяет им обрабатывать данные гораздо быстрее и эффективнее, чем универсальные процессоры.
- **Параллелизм**: Большинство аппаратных ускорителей могут обрабатывать множество операций одновременно, что особенно полезно в задачах, требующих больших объемов данных, таких как обработка изображений, видео, машинное обучение и научные вычисления.

2. Типы аппаратных ускорителей

- **FPGA**: Эти устройства могут быть программируемыми для выполнения конкретных алгоритмов или функций. Они обеспечивают высокий уровень гибкости и позволяют пользователям настраивать аппаратное обеспечение под свои нужды.
- **ASIC**: Эти чипы проектируются для выполнения конкретных задач и не могут быть перепрограммированы. Они предлагают максимальную производительность и эффективность для определенных приложений, но их разработка требует значительных затрат и времени.
- **GPU**: Графические процессоры изначально разрабатывались для обработки графики, но их архитектура, позволяющая выполнять множество параллельных вычислений, делает их отличными для задач машинного обучения и научных расчетов.

3. Преимущества аппаратного ускорения

- **Высокая производительность**: Специализированные устройства могут выполнять вычисления намного быстрее, чем универсальные процессоры, что критично для задач с высоким уровнем вычислительной нагрузки.

- **Энергоэффективность**: Часто специализированные чипы могут выполнять те же задачи с меньшим потреблением энергии по сравнению с общими процессорами, что особенно важно для мобильных устройств и крупных вычислительных центров.
- **Скорость обработки данных**: Аппаратное ускорение позволяет обрабатывать большие объемы данных в реальном времени, что критически важно для приложений, таких как видеоаналитика, финансовые расчеты и научные исследования.

АППАРАТНЫЕ ПЛАТФОРМЫ

Аппаратные платформы в схемотехнике представляют собой комбинацию аппаратных компонентов и технологий, используемых для проектирования, реализации и тестирования электронных систем и устройств. Эти платформы могут включать в себя как стандартные, так и специализированные компоненты, обеспечивая необходимую гибкость и производительность для различных приложений.

Основные аспекты аппаратных платформ в схемотехнике

- **FPGA (Field-Programmable Gate Array)**:
    - **Описание**: FPGA представляют собой программируемые логические устройства, которые могут быть настроены для выполнения различных логических операций.
    - **Применение**: Используются для проектирования цифровых схем, прототипирования, обработки сигналов и реализации алгоритмов машинного обучения.

    - **Преимущества**: Высокая степень гибкости и возможность изменения функциональности после производства.
- **ASIC (Application-Specific Integrated Circuit)**:
    - **Описание**: Специализированные интегральные схемы, разработанные для выполнения конкретных задач.
    - **Применение**: Широко используются в высокопроизводительных приложениях, таких как мобильные телефоны, серверы и системы обработки данных.
    - **Преимущества**: Высокая производительность и энергоэффективность, но высокая стоимость и время разработки.
- **SoC (System-on-Chip)**:
    - **Описание**: Интегрированные решения, которые объединяют процессоры, память и интерфейсы ввода-вывода на одном чипе.
    - **Применение**: Применяются в мобильных устройствах, встраиваемых системах и IoT-устройствах.
    - **Преимущества**: Компактность, снижение стоимости и потребления энергии.

Архитектуры аппаратных платформ

- **Архитектура von Neumann**:
    - Одна из самых распространенных архитектур, использующая единую память для хранения как данных, так и инструкций.

- **Архитектура Гарварда**:
    - Разделяет память для данных и инструкций, позволяя параллельное выполнение операций и улучшая производительность.
- **Многопроцессорные системы**:
    - Используют несколько процессоров для одновременной обработки данных, что позволяет значительно увеличить производительность в вычислительно интенсивных задачах.

Критерии выбора аппаратной платформы в схемотехнике

- **Производительность**: Определяет, насколько быстро и эффективно платформа может выполнять необходимые задачи.
- **Энергоэффективность**: Важно учитывать потребление энергии, особенно в встраиваемых системах и мобильных устройствах.
- **Гибкость**: Некоторые платформы, такие как FPGA, обеспечивают возможность настройки и адаптации для различных приложений.
- **Совместимость**: Платформа должна поддерживать необходимые программные и аппаратные компоненты для успешной реализации решения.

СОВРЕМЕННЫЕ ТЕНДЕНЦИИ И ПРОБЛЕМЫ

- Основной инструмент проектирования аппаратного обеспечения – языки описания аппаратуры
- Высокий порог вхождения, необходимость добавления в рабочую группу специалистов по схемотехнике
- Сложность обеспечения взаимодействия, различные области компетенций по сравнению с программистами
- Длительный цикл проектирования по сравнению с программированием, разработка тяготеет с каскадной модели с предварительным формулированием детализированного технического задания
- Требуется:
  - Снижение порога вхождения в разработку и привлечение программистов и специалистов в предметной области

СИСТЕМЫ КЛАССА HLS.

- Следующее поколение языков проектирования аппаратуры, дополняющее HDL
- Языки класса «C-RTL», выполняющие преобразование Си-подобного описания аппаратуры в RTL-представление на одном из языков описания аппаратуры
- Первое поколение (2000) – Catapult-C, Handel-C
- 2012 год – Berkeley Design Technology Inc (BDTI) – HLS для САПР ПЛИС

Системы класса High-Level Synthesis (HLS) представляют собой инструменты и методологии, позволяющие разработчикам проектировать аппаратные системы на более высоком уровне абстракции, чем традиционное проектирование с использованием языков описания аппаратуры (HDL), таких как VHDL или Verilog. Основная цель HLS — упростить и ускорить процесс разработки, позволяя проектировщикам работать с алгоритмами и структурами данных на уровне, близком к языкам программирования высокого уровня (например, C или C++).

Процесс HLS

1. **Разработка алгоритма**: Проектировщик разрабатывает алгоритм на высокоуровневом языке.
2. **Оптимизация**: HLS инструменты автоматически анализируют код для выявления возможностей параллелизма, оптимизации ресурсов и минимизации временных задержек.
3. **Генерация RTL**: На основе анализа генерируется регистровое уровень описание аппаратуры (RTL), которое можно синтезировать для создания физического устройства.
4. **Синтез**: RTL код синтезируется в конкретные элементы аппаратуры (например, FPGA или ASIC).
5. **Верификация и тестирование**: Созданная аппаратная система проходит верификацию для проверки ее функциональности и производительности.

HLS В МАРШРУТЕ ПРОЕКТИРОВАНИЯ

Роль HLS в маршруте проектирования

HLS позволяет проектировщикам переходить от высокоуровневого описания алгоритмов к аппаратным решениям с помощью инструментов, которые автоматически генерируют регистровое уровень описание аппаратуры (RTL) из кода на языках высокого уровня (например, C, C++ или SystemC). Это снижает необходимость в ручном кодировании на HDL (VHDL или Verilog) и упрощает процесс проектирования.

Этапы маршрута проектирования с использованием HLS

a. Определение спецификаций

На этом начальном этапе проектировщики определяют функциональные и производительные требования к системе. HLS помогает формулировать эти требования более наглядно, используя алгоритмические описания.

b. Разработка высокоуровневого кода

- **Алгоритмическое проектирование**: Проектировщик пишет алгоритмы на высокоуровневых языках программирования (например, C/C++), описывающих функциональность системы.
- **Оптимизация**: Используются HLS инструменты для оптимизации алгоритмов, выявления параллелизма и распределения ресурсов.

c. Синтез в RTL

HLS инструмент анализирует написанный код и генерирует RTL описание, которое может быть синтезировано для создания аппаратных компонентов. На этом этапе HLS может выполнять различные оптимизации, такие как:

- **Pipeline**: Разделение операций на несколько этапов для повышения производительности.
- **Loop unrolling**: Раскрытие циклов для повышения параллелизма.
- **Resource sharing**: Оптимизация использования аппаратных ресурсов.

d. Синтез и реализация

RTL код затем проходит процесс синтеза, который преобразует его в сетевую таблицу, используемую для реализации на целевой аппаратной платформе (FPGA или ASIC).

e. Верификация и тестирование

После генерации аппаратного описания необходимо провести верификацию, чтобы убедиться, что разработанная система соответствует заданным требованиям. HLS инструменты часто предоставляют средства для автоматической верификации функциональности и производительности на высоком уровне.

Преимущества HLS в маршруте проектирования

- **Скорость разработки**: HLS значительно ускоряет процесс проектирования, позволяя инженерам быстро переходить от алгоритма к аппаратному решению.
- **Упрощение сложности**: Проектировщики могут сосредоточиться на алгоритмах, не беспокоясь о низкоуровневых деталях реализации.
- **Улучшение производительности**: HLS инструменты могут автоматически оптимизировать код, позволяя лучше использовать ресурсы аппаратуры и повышать производительность.
- **Гибкость**: Возможность легко изменять и адаптировать алгоритмы на высоком уровне для различных приложений и платформ.

Недостатки и ограничения HLS

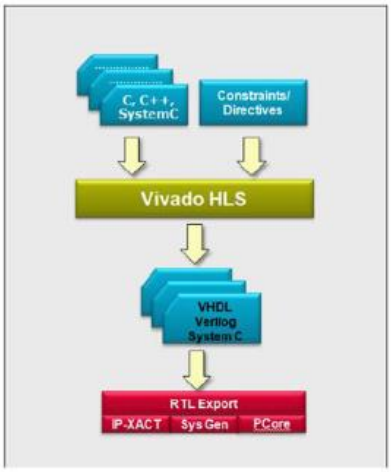
Несмотря на свои преимущества, HLS также имеет свои недостатки:

- **Ограничения производительности**: В некоторых случаях производительность, достигаемая с помощью HLS, может быть ниже, чем при ручном проектировании на HDL, особенно для сложных систем.

- **Сложность оптимизации**: Оптимизация алгоритмов на высоком уровне может быть сложной задачей, требующей глубокого понимания как алгоритмов, так и аппаратных архитектур.
- **Долгое время компиляции**: Процесс генерации RTL и синтеза может занимать значительное время, особенно для больших и сложных систем.

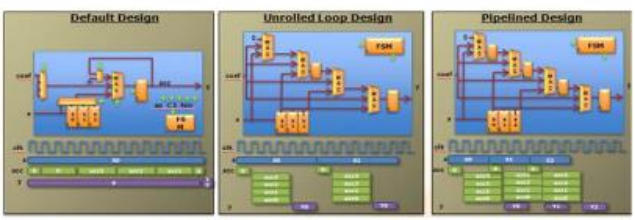
ВЗАИМОДЕЙСТВИЕ HLS и VIVADO

- Vivado HLS создает RTL-представление из исходного высокоуровневого кода на языке C/C++/SystemC
- Генерация RTL производится с учетом директив компилятора и возможностей ПЛИС
- Обработка RTL-представления производится в САПР Vivado



ДИРЕКТИВЫ

- Один и тот же исходный текст может быть реализован различными схемами, различающимися числом тактов на реализацию и объемом аппаратных ресурсов
- Управление реализацией не является частью С-подобного языка и достигается добавлением директив компилятора



ПЛАНИРОВАНИЕ И СВЯЗЫВАНИЕ (SCHEDULING AND BINDING)

- Планирование определяет, на каком именно такте производится каждая из операций
- Учитываются построенные datapath и control flow, а также директивы и возможности технологической библиотеки
- Связывание определяет ресурсы ПЛИС, которые будут использованы для выполнения операций

Связывание

- Распределение ресурсов производится с учетом директив компилятора
- Пример 1: требуется два умножителя, т.к. обе операции умножения привязаны к одному и тому же такту

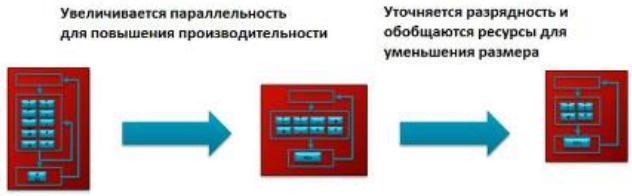
- Пример 2: операции умножения выполняются на разных тактах

ПРОЦЕСС СИНТЕЗА В HLS

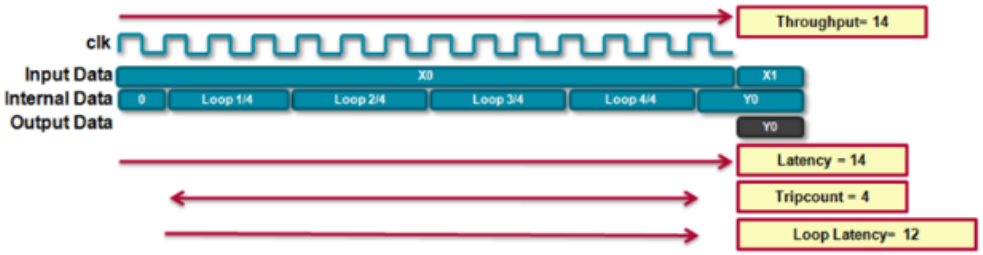
- Синтезатор HLS определяет, на каком такте выполняется каждое из действий
- Определяются устройства, выполняющие каждую из операций
- Производится оптимизация:
  - Производительность (частота и пропускная способность
  - Минимизируется латентность
  - Минимизируется размер

ПОРЯДОК ОПТИМИЗАЦИИ В HLS

- Синтезируется начальный вариант
- Проверяется достижение заданной производительности
- Уменьшается размер



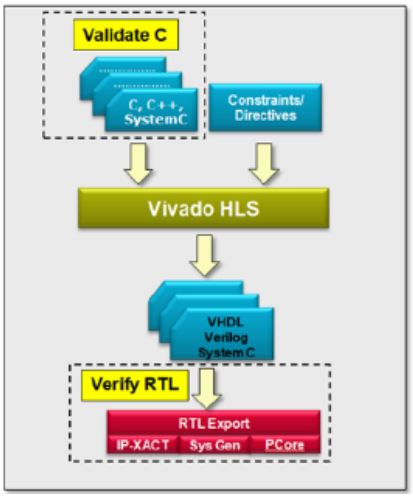
ТЕРМИНЫ ДЛЯ ОПИСАНИЯ КОНВЕЙЕРИЗОВАННОЙ СХЕМЫ В HLS



Латентность - число тактов от входа до выхода
Пропускная способность (throughput) - число тактов между соседними входными отсчетами
Trip count - число тактов в одной итерации
Loop latency - латентность одного цикла

МОДЕЛИРОВАНИЕ И ТЕСТОВЫЕ ВОЗДЕЙСТВИЯ

- Два шага в процессе моделирования
  - RTL validation – выполняется над исходным текстом на C
  - RTL verification – выполняется симулятором RTL-представления после синтеза схемы



- Тест является модулем на C, находящимся выше верхнего уровня синтезируемой части проекта
- Тест содержит функцию main()
- Рекомендации:
  - Самопроверяющийся проект (сравнение результатов с эталоном встроено в тест)
  - При успехе возвращать 0 (ненулевое значение трактуется в HLS как ошибка моделирования с выводом сообщения в консоль)

```
int main () {
  int ret=0;
  ...
  ret = system("diff --brief -w output.dat output.golden.dat");

  if (ret != 0) {
      printf("Test failed !!!\n");
      ret=1;
  } else {
      printf("Test passed !\n");
  }
  ...
  return ret;
}
```