**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ**

**РОССИЙСКОЙ ФЕДЕРАЦИИ**

## Федеральное государственное автономное образовательное учреждение высшего образования «ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт компьютерных технологий и информационной безопасности Инженерно-технологической академии

Южного федерального университета

**Кафедра интеллектуальных и многопроцессорных систем** (ИМС) — базовая кафедра Научно-исследовательского центра

«Супер-ЭВМ и нейрокомпьютеров»

**ПЛИС-ТЕХНОЛОГИИ И МЕТОДЫ СОЗДАНИЯ ЭФФЕКТИВНЫХ ПРИКЛАДНЫХ ПРОГРАММ ДЛЯ ПЛИС**

Конспект лекций (1 часть 1 семестра)

Магистрант \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Таганрог 2020

Содержание

Предыстория и история вычислений на ПЛИС 3

Комбинационные асинхронные устройства 24

Дискретные устройства. Синхронизаторы,

триггеры, регистры, счетчики 38

Структура ПЛИС 52

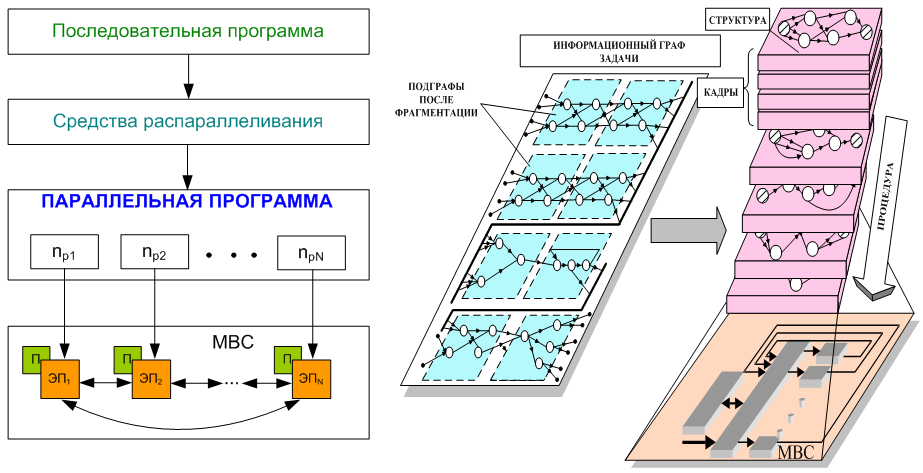
Конечные автоматы 65

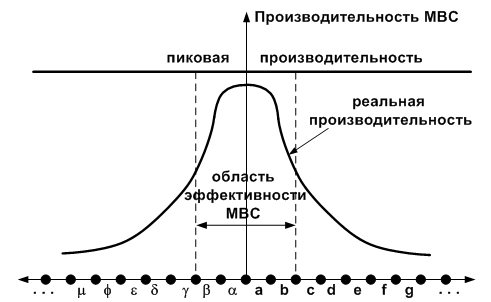
Арифметические операции 74

# Предыстория и история вычислений на ПЛИС

## ОСНОВНЫЕ ПРОБЛЕМЫ СОВРЕМЕННЫХ КЛАСТЕРНЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

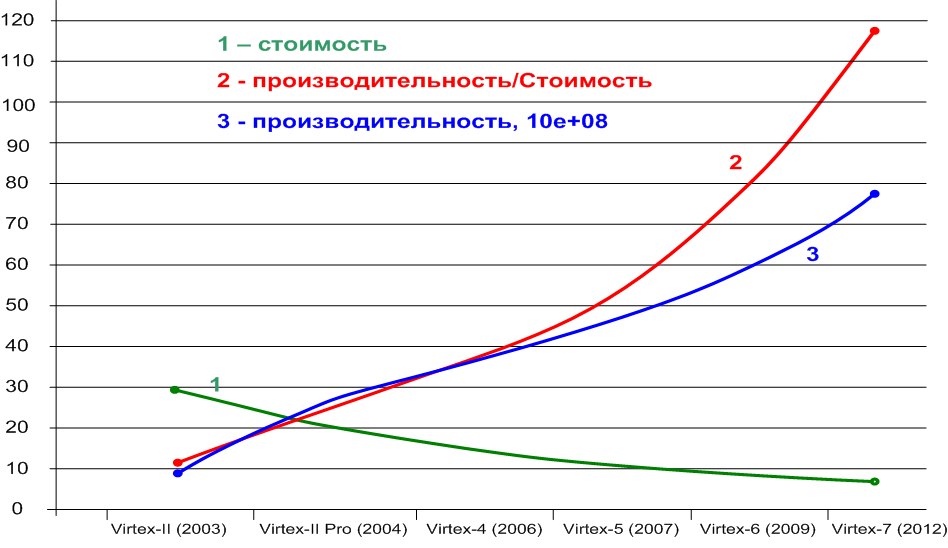
* **Низкая реальная производительность** – не более 5-10% от пиковой производительности системы, поэтому часто система всего лишь в несколько раз быстрее ПК;
* **Низкая удельная производительность** (производительность в единице объема) – необходимы дополнительные стойки для охлаждения и коммутации;
* **Низкая энергоэффективность** – большое количество дополнительных неэффективных расходов энергии на охлаждение, коммутацию и организацию вычислений.
* **Ограниченность роста производительности**, зачастую при увеличении вычислительного ресурса системы производительность начинает снижаться.





# Классы задач

Эволюция технических характеристик ПЛИС



**Реконфигурируемая вычислительная система** – вычислительная система (ВС), аппаратно-программные средства которой позволяют **адаптировать архитектуру ВС** под структуру решаемой задачи для каждой новой решаемой задачи.

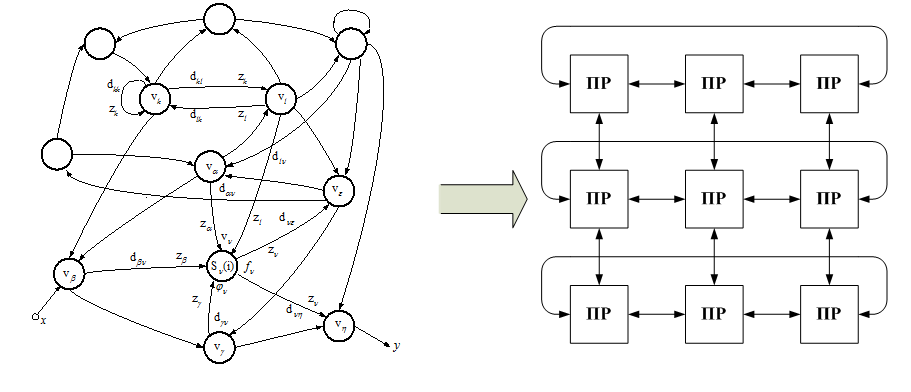
В полной мере преимущества от использования РВС достигаются при использовании в качестве основного вычислительного элемента аппаратного ресурса ПЛИС, объединенных в единое вычислительное поле высокоскоростными каналами передачи данных.

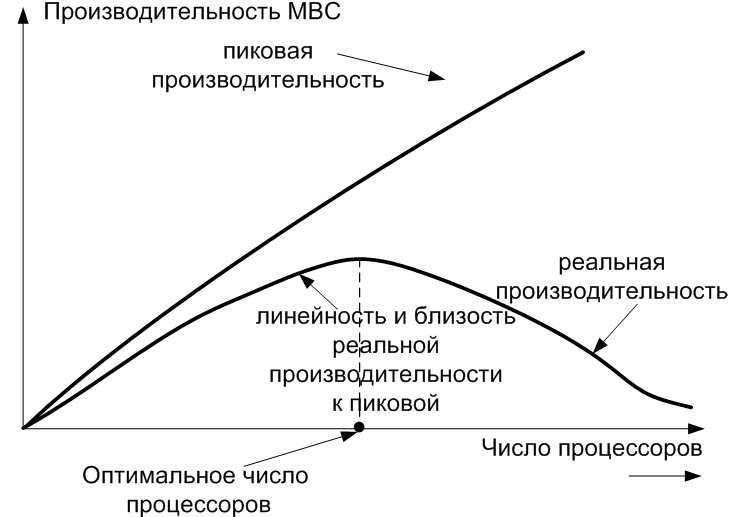
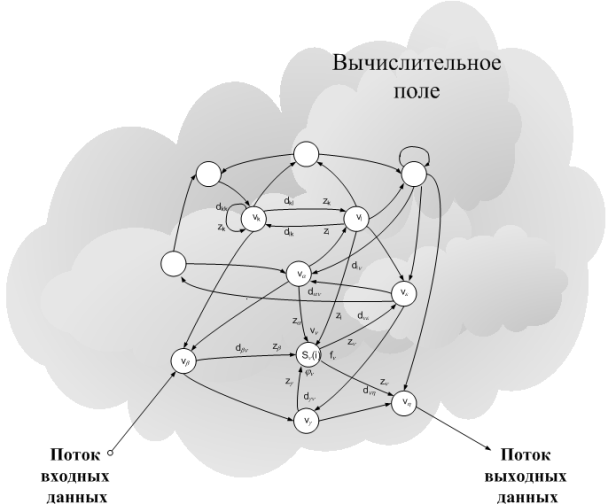
**Существующий подход**

**Адаптация задачи под архитектуру вычислительной системы.**

**Альтернативный подход**

**Адаптация архитектуры вычислительной системы под задачу.**

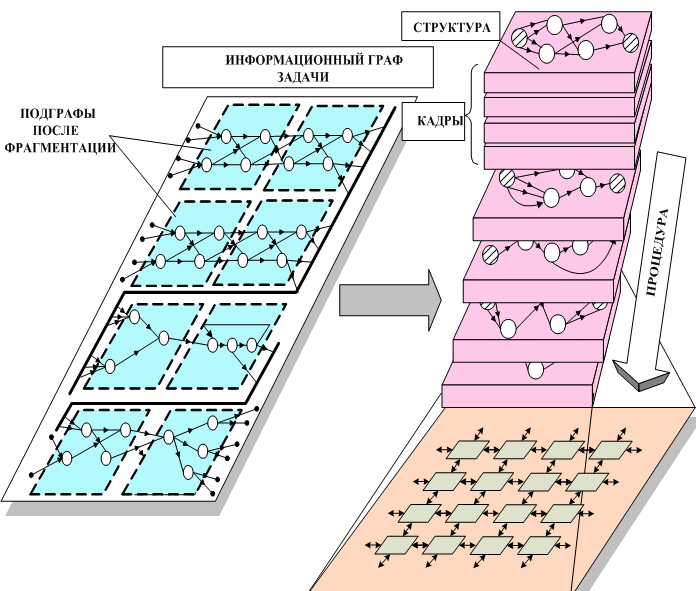




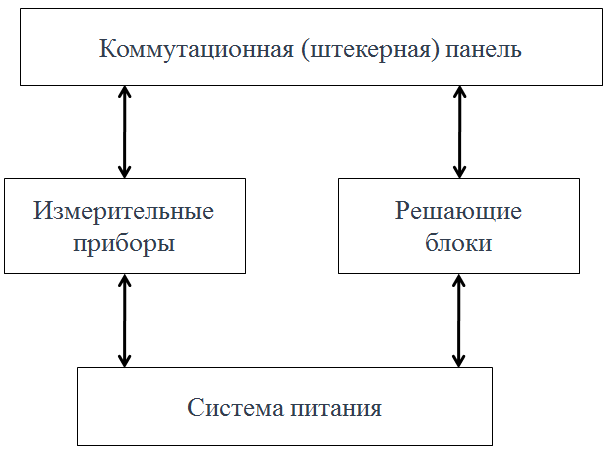
**Эволюция развития реконфигурируемых вычислительных систем**

1. **Аналоговые вычислительные машины – 50-е–60-е годы XX века.**
2. **Цифровые дифференциальные анализаторы – 60-е–70-е годы XX века.**
3. **Однородные вычислительные среды – 70-е–80-е годы XX века.**
4. **Многопроцессорные вычислительные системы с программируемой архитектурой – 80-е–90-е годы XX века.**

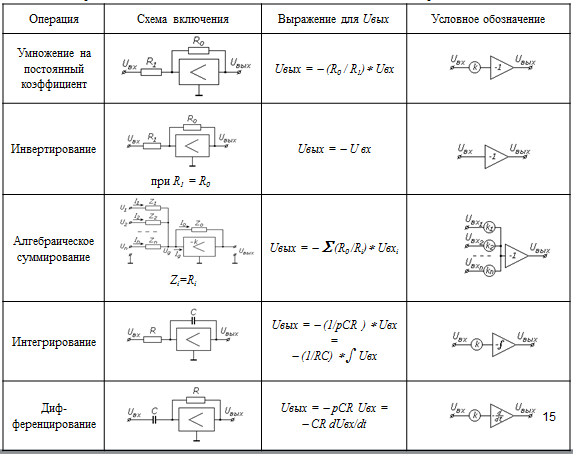
## АДАПТАЦИЯ АРХИТЕКТУРЫ ВЫЧИСЛИТЕЛЬНОЙ СИСТЕМЫ ПОД ЗАДАЧУ



**Обобщенная структура АВМ**



## Набор основных математических операций АВМ



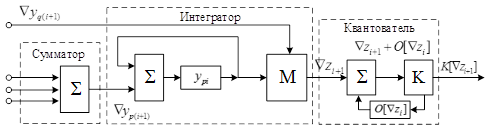
**Аналоговая ЭBM MH-7 (1955 г.)**

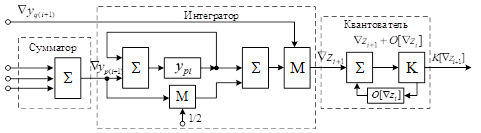
# 

**Обобщенная структура ЦИМ**

# 

## Структурная схема интегратора на основе формул: прямоугольников и трапеций

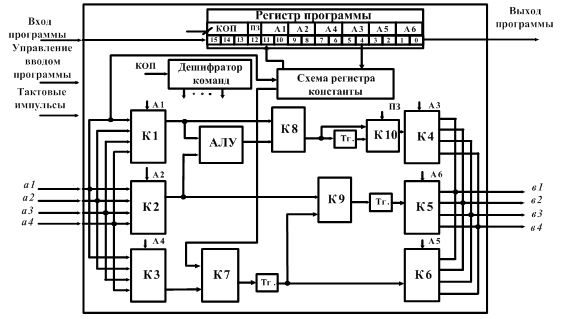
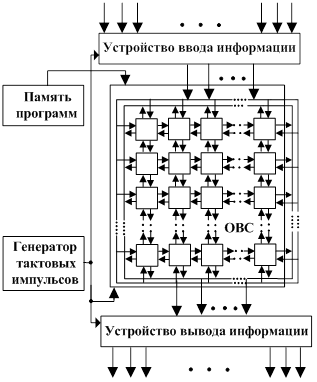




## Структура ЦИМ интерполяционного типа

# 

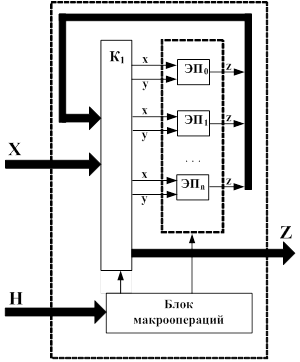
## Структурная схема и вычислительная ячейка ОВС



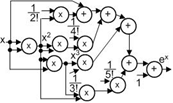
## Реализация задачи в поле ОВС и модифицированный спецпроцессор операции сложения

# 

## Структурная схема макропроцессора Обобщенная структура МВС ПА

Макрооперация экспоненты



## Архитектура ПЛИС



**- блоки ввода/вывода IOB;**

**- конфигурируемые логические блоки CLB;**

**- секции блочной памяти Block SelectRAM;**

**- блоки аппаратных умножителей;**

**- цифровые модули управления синхронизацией DCM;**

**- трассировочные ресурсы.**

1970 Разработан первый тип SPLD - микросхема ППЗУ.

1971 Изобретение нового типа ROM(Read-Only Memory)-памяти – EPROM (ПЗУ с УФ стиранием).

1975 Появление нового типа SPLD - PLA (Programmable Logic Arrays). В основе PLA лежит развитие идеи микросхем ППЗУ.

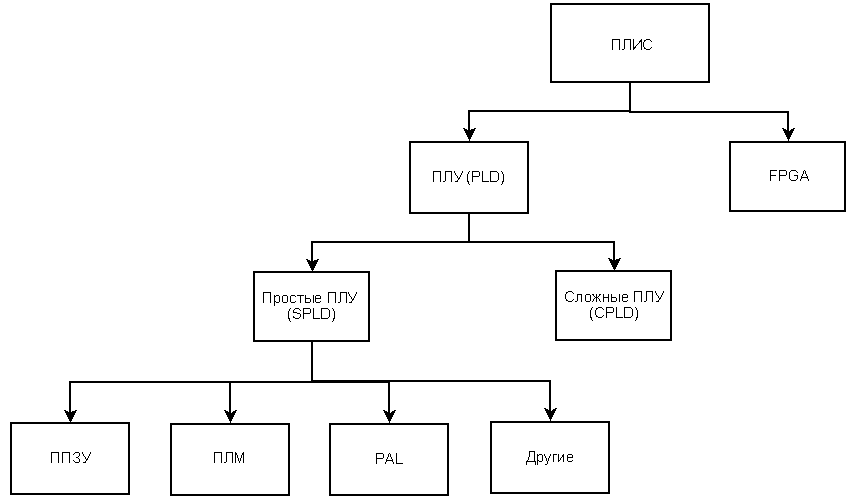
Конец 70-х - Разработан ещё один класс SPLD - программируемый массив логики   
(PAL - Programmable Array Logic).

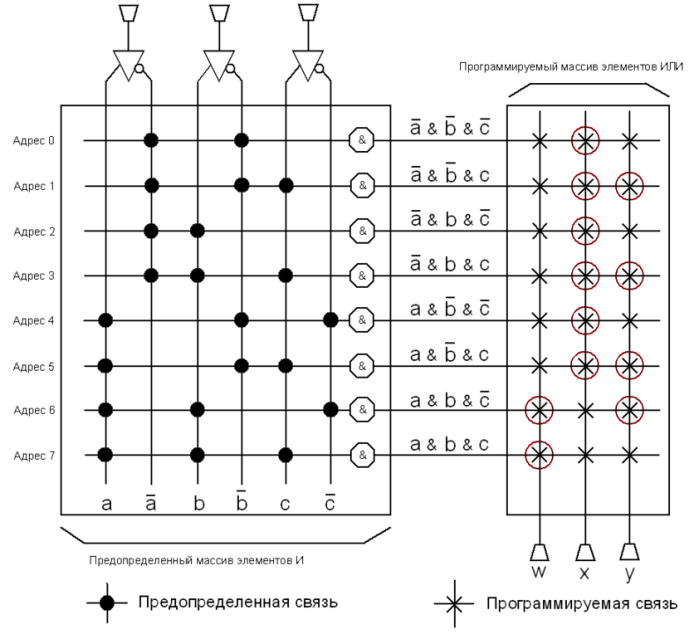
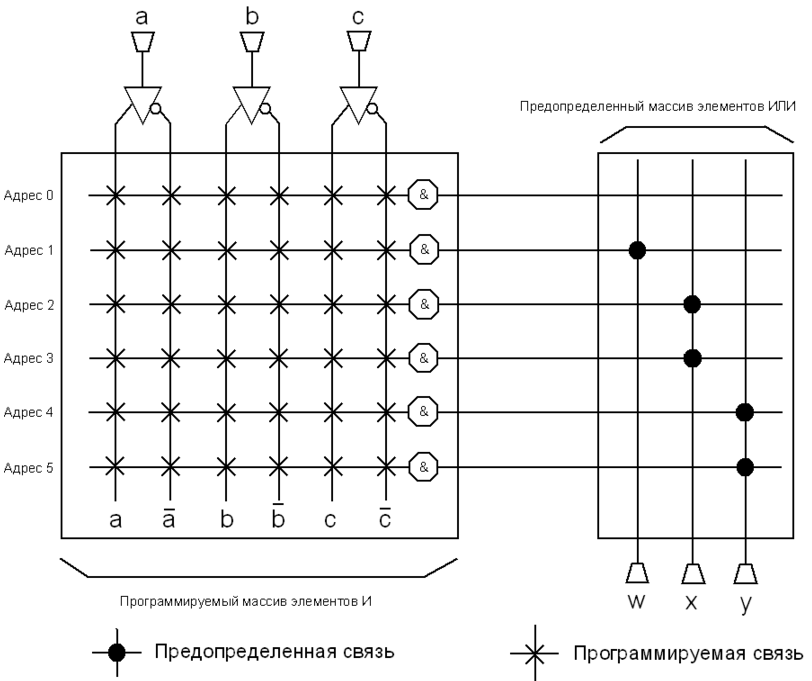
1983 Появление памяти EEPROM, ПЗУ с электрическим стиранием

1984 Появились первые сложные программируемые логические устройства (CPLD) от созданной в этом же году компании Altera.

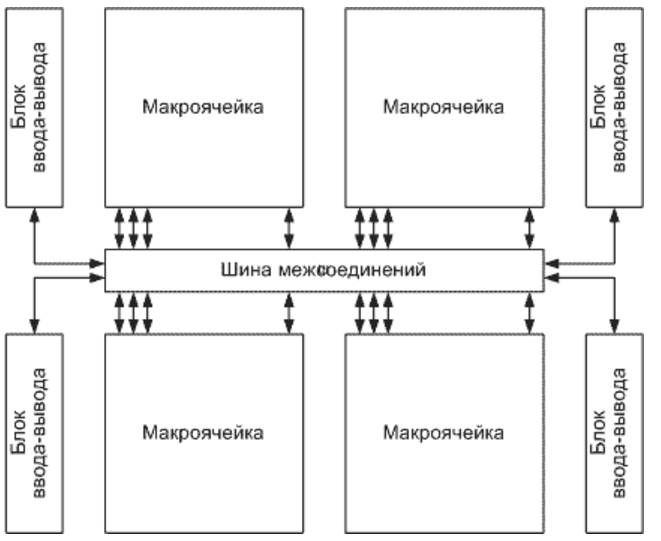
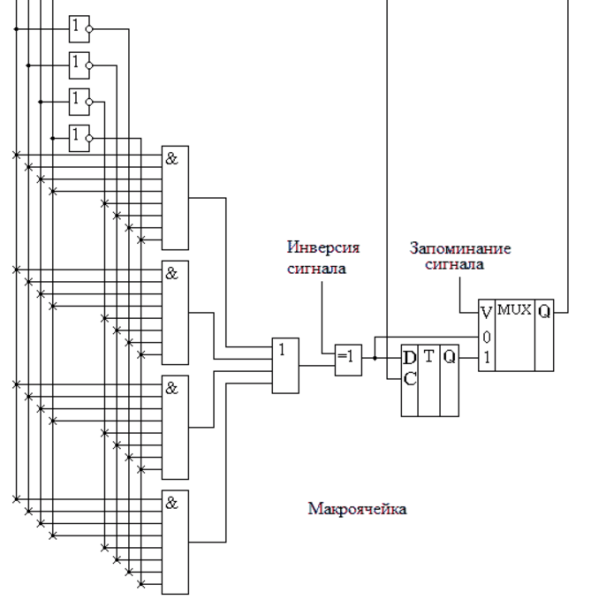
1984 Основание компании Xilinx. Росс Фриман разработал FPGA-микросхемы.

field-programmable gate array

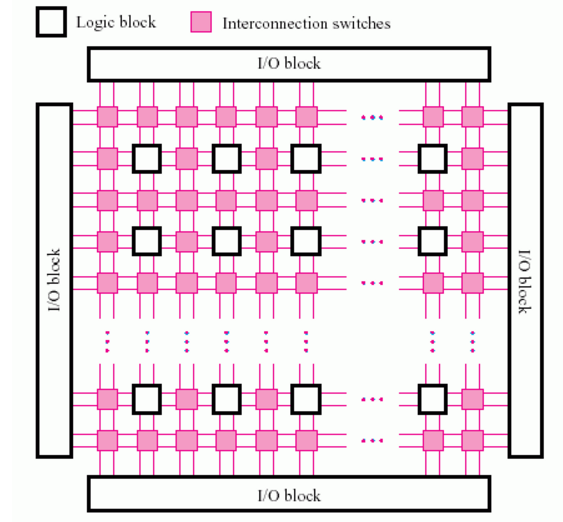


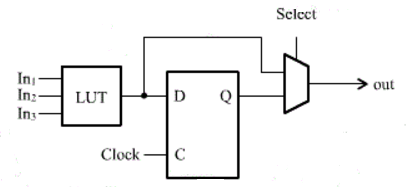
CPLD- Complex PLD CPLD- Complex PLD

функции в FPGA реализуются не с помощью ДНФ, а помощью программируемых таблиц соответствия (LUT-таблиц)



LUT с триггером



1985 Выпуск первой FPGA-микросхемы, Xilinx XC2064 с 1000 вентилей, 3 мкм технология

1988 Основана компания Actel.

1992 NASA впервые применила FPGA в космических технологиях.

1995 Altera и Xilinx начали выпуск FPGA-микросхем для цифровой обработки сигналов.

С этого момента FPGA начали активно использоваться в данной области.

1996 Actel, выпуск микросхем RH1280, радиационно-стойкие FPGA-микросхем.

1998 Появление семейства Virtex от компании Xilinx.

1999 Компания Altera выпустила FPGA с 1.5 миллионами вентилей - APEX EP20K.

2000 Actel запустила семейство ProASIC на Flash-технологии (первая серия - ProASIC 500K).

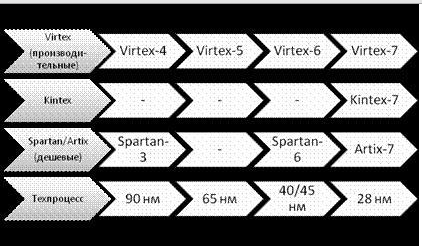
2002 Компания Lattice Semiconductor начала разработку и производство FPGA- и FPSC-микросхем.

2003 В производстве FPGA произошел переход на 90-нм технологию (серия Spartan-3 от Xilinx). Число вентилей превысило 5 миллионов.

2004 Компания Cray выпустила ПЛИС-сопроцессоры XD1.

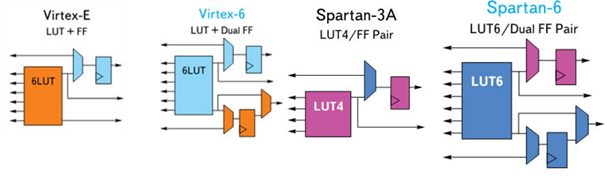
2005 Компания Actel представляет технологию Fusion, которая позволяет объединить логические блоки FPGA, Flash-память и аналоговые устройства на одной микросхеме.

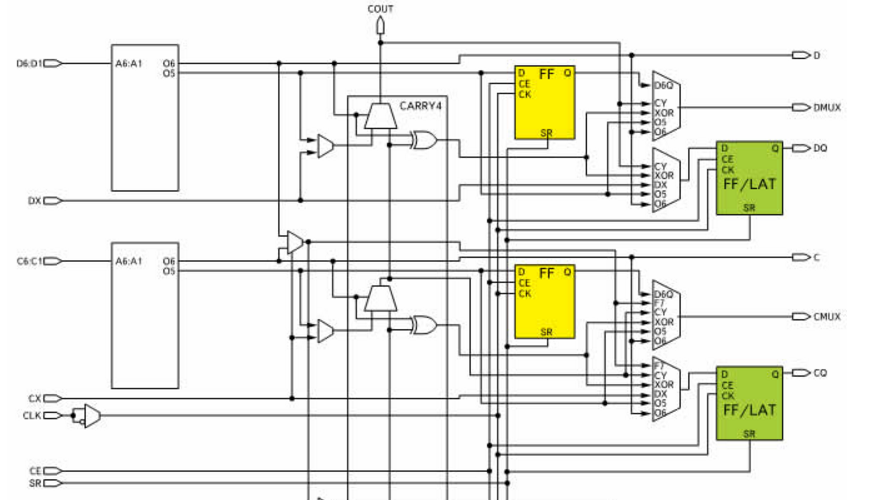
2006 - Появление серии Virtex-5 от Xilinx на основе 65-нм технологии. Число вентилей достигло 7 миллионов (Virtex-5 V5LX33OT).

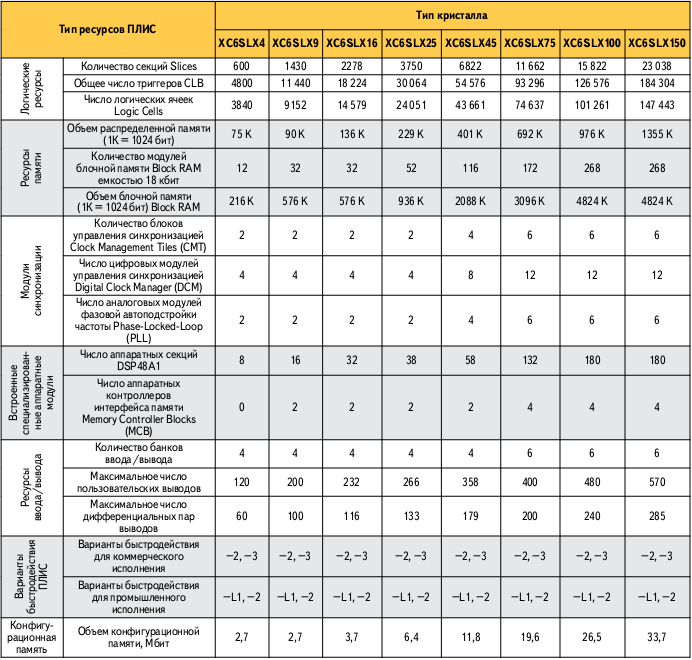


|  |  |
| --- | --- |
|  | блоки ввода/вывода IOB;  конфигурируемые логические блоки CLB;  секции блочной памяти Block SelectRAM;  блоки аппаратных умножителей;  цифровые модули управления синхронизацией DCM;  трассировочные ресурсы. |

Развитие логического блока ПЛИС

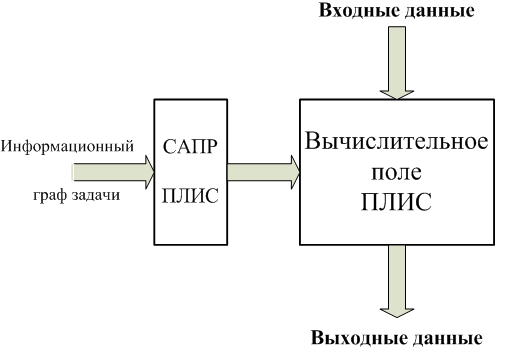






Встраиваемые процессоры





## ФОРМИРОВАНИЕ КАДРОВЫХ СТРУКТУР

**Условия представления информационного графа в функционально-фрагментированной форме:**

**Граф G ={ P1, P2, … Pk }: Pi Pj = ∅, если i ≠ j**

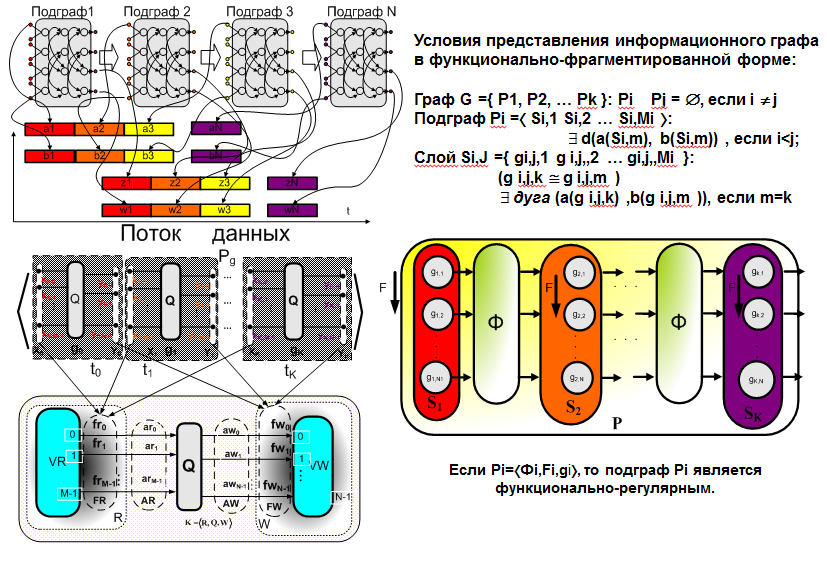
**Подграф Pi =〈 Si,1 Si,2 … Si,Mi 〉:**

**∃ d(a(Si,m), b(Si,m)) , если i<j;**

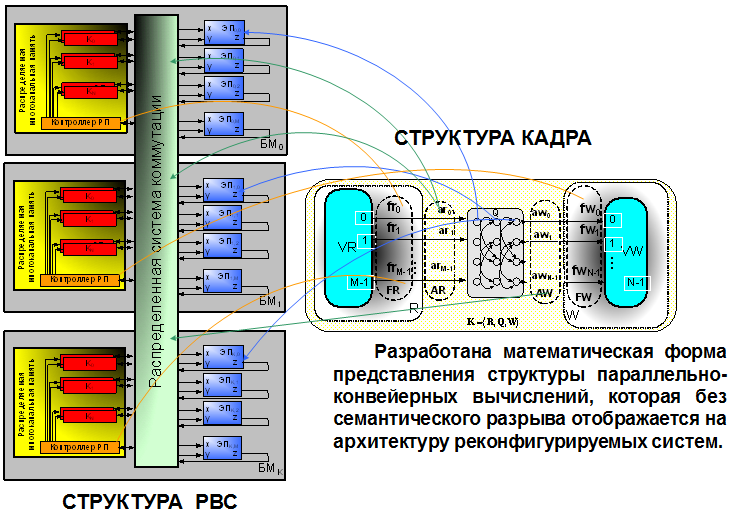
**Cлой Si,J ={ gi,j,1 g i,j,,2 … gi,j,,Mi }:**

**(g i,j,k ≅ g i,j,m )**

**∃ *дуга* (a(g i,j,k)** **,b(g i,j,m** **)), если m=k**



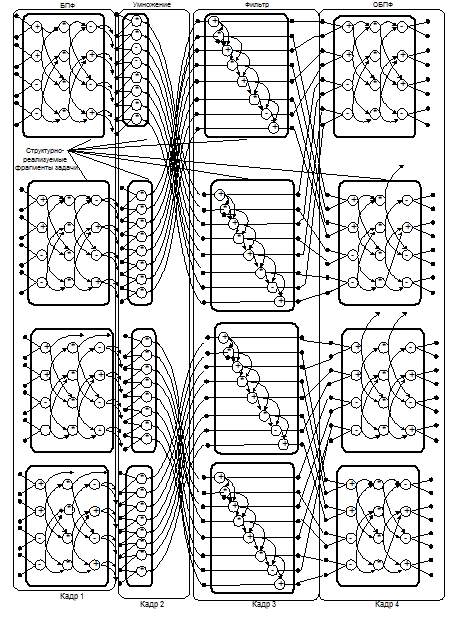
## ОТОБРАЖЕНИЕ КАДРОВОЙ ФОРМЫ В СТРУКТУРУ МВС

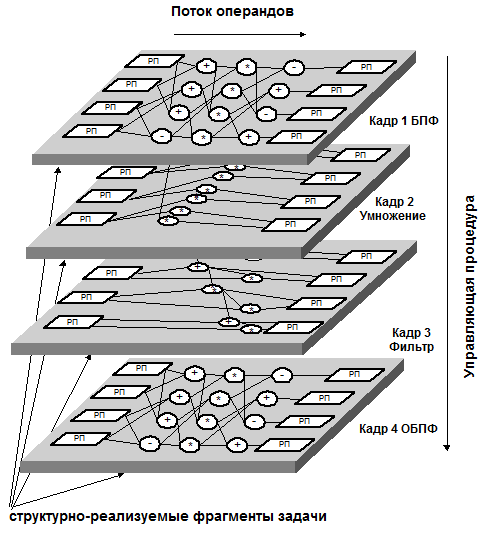


## Проблема 1

Физические ограничения поля ПЛИС (невозможность отображения всего информационного графа задачи в поле ПЛИС).

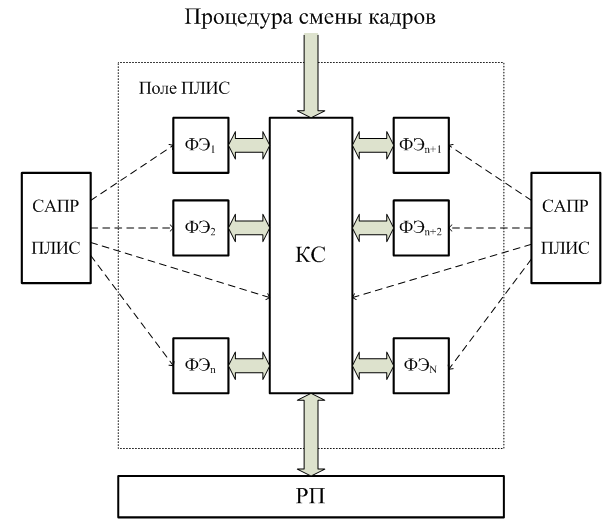
Информационный граф базовой процедуры ЦОС





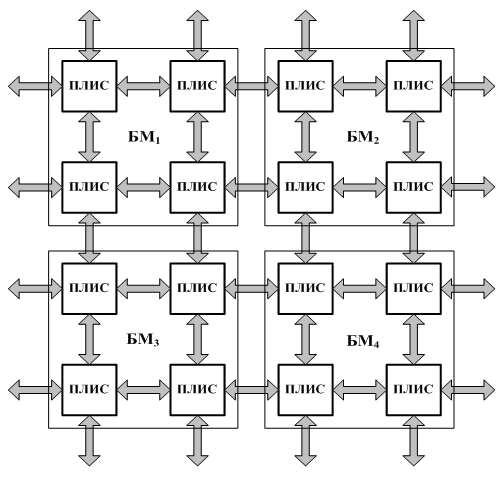
## Проблема 2

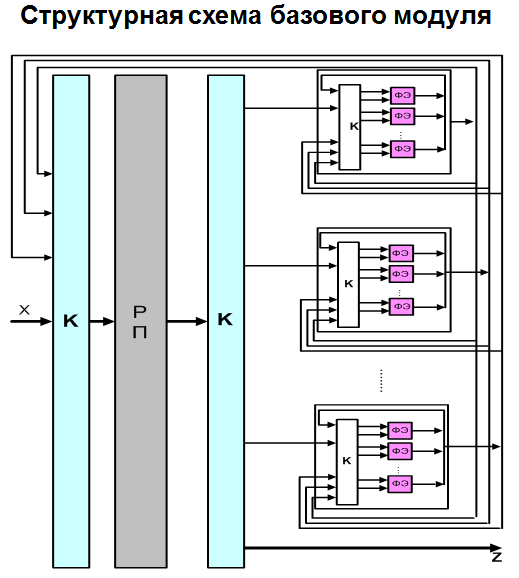
Длительное время загрузки файла конфигурации ПЛИС.



## Проблема 3 Ограничения по количеству ПЛИС, физически располагаемой на одной плате.

Разбиение вычислительного поля ПЛИС на базовые модули





## БАЗОВЫЙ МОДУЛЬ НИИ МВС ТРТУ, 2001 ГОД

Число ПЛИС Vertex E-300, шт. 33

Объем памяти, Мбайт 128

Число макропроцессоров 16

Число элементарных процессоров 64

Число каналов распределенной памяти 32

Производительность, Гфлопс 8

Частота ПЛИС, МГц 50

Частота платы, МГц 50

## ХАРАКТЕРИСТИКИ БАЗОВОГО МОДУЛЯ 4К32

БАЗОВЫЙ МОДУЛЬ РМВС «4К32» В КОНСТРУКТИВЕ ЕВРОМЕХАНИКА, 2004 ГОД

Число ПЛИС Virtex II Pro, шт. 4

Объем памяти, МБайт 256

Производительность, Гфлопc 32

Рабочая частота, МГц 200

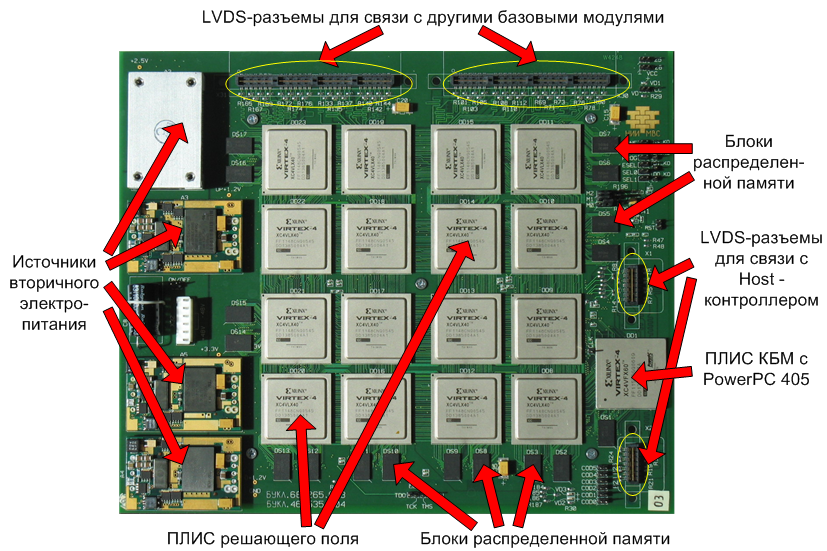
Потребляемая мощность, Вт 86

Количество каналов LVDS, шт 100

Скорость обмена, Гбит/сек 20

Максимальный размер БПФ 132072

Число эквивалентных вентилей, млн. 40



Число ПЛИС Virtex IV, шт. 16

Число эквивалентных модулей

в каждой ПЛИС, млн. 4

Число элементарных процессоров, шт. 256

Объем памяти, ГБайт 1

Производительность, Гфлопc 50

Рабочая частота, МГц 160

Количество каналов LVDS, шт. 112

Суммарная пропускная способность

LVDS каналов, Гбит/сек 2,24

Потребляемая мощность, КВт 0,16

Габариты, мм 233,35х280

Общее число эквивалентных вентилей, млн. 64

## РЕКОНФИГУРИРУЕМАЯ МВС «МЕДВЕДЬ», 2006 ГОД

Количество унифицированных

базовых модулей 4

Число ПЛИС Virtex IV, шт. 64

Объем памяти, Гбайт 4

Производительность пиковая, ГФлопс 200

Потребляемая мощность, КВт 0,9

Объем, м.куб. 0,39

Общее число эквивалентных вентилей, млн. 256

## БАЗОВЫЙ МОДУЛЬ РМВС «16М50-М», 2006 ГОД

Число ПЛИС Virtex IV, шт. 16

Число эквивалентных вентилей

в каждой ПЛИС, млн. 4

Объем памяти, ГБайт 1,3

Производительность, Гфлопc 50

Рабочая частота, МГц 160

Скоростной информационный канал гипертранспорт

Потребляемая мощность, КВт 0,4

Габариты, мм 233,35х280

Общее число эквивалентных

вентилей, млн. 64

## БАЗОВЫЙ МОДУЛЬ РМВС «16TР75», 2007 ГОД

Число ПЛИС Vertex IV, 16

Число элементарных процессоров 256(512)

Объем памяти, Гбайт 2

Производительность, Гфлопс 64(128)

Частота ПЛИС, МГц 250

Частота платы, МГц 250

Габариты, мм 259х379х98

Общее число эквивалентных вентилей, млн. 128

## БАЗОВЫЙ МОДУЛЬ РМВС «8ТО50», 2007 ГОД

Число ПЛИС, шт. 8

Число элементарных процессоров, шт. 256

Объем статической памяти, Мбайт 80

Частота, МГц 160

Потребляемая мощность, КВт 0,16

Габариты ВМ, мм 74х258х245

Производительность, Гфлопс 50

Общее число эквивалентных

вентилей, млн.шт. 64

## РЕКОНФИГУРИРУЕМАЯ МВС НА ОСНОВЕ 4-Х БМ 8ТО50, 2007 ГОД

Количество базовых модулей 4

Общее число ПЛИС, шт. 32

Производительность, оп/с, не менее 4\*1012

Объем статической памяти, Мбайт 320

Тактовая частота, МГц 160

Потребляемая мощность, КВт 0,9

Габариты,мм 482х266х675

Общее число эквивалентных

вентилей, млн. 256

ТЕХНИЧЕСКИЕ ХАРАКТЕРИСТИКИ СТОЙКИ РМВС НА ОСНОВЕ   
24-Х БМ 8ТО50

|  |  |
| --- | --- |
| Стойка_без фона | Количество базовых  модулей 24  Общее число ПЛИС, шт. 192  Производительность, оп/с,  не менее 2,4\*1013  Объем статической памяти,  Мбайт 1920  Тактовая частота, МГц 160  Потребляемая мощность,  КВт 5,4  Габариты,мм 610х2300х1000  Общее число эквивалентных  вентилей, млн. 1536 |

## БАЗОВЫЙ МОДУЛЬ РМВС «4V4-25», 2008

Количество ПЛИС Virtex 4,шт. 4

Объем динамической памяти, Мбайт 576

Производительность, Гфлопс 25

Частота платы модуля, МГц 160

Скорость обмена данными

по каналу Ethernet, Гбит/с 1

Потребляемая мощность, Вт 145

Размеры базового модуля, мм. 150×190

Общее число эквивалентных

вентилей, млн. шт. 32

РЕКОНФИГУРИРУЕМЫЙ УСКОРИТЕЛЬ К ПЕРСОНАЛЬНОМУ КОМПЬЮТЕРУ НА ОСНОВЕ БМ «4V4-25» «Мангуст»



Система РВС-5, 2009

|  |  |
| --- | --- |
| РВС-5_зад_1 | Число вычислительных  блоков (ПЛИС), шт. 1280  Объем динамической  памяти, Гбайт 100  Производительность,  Гфлопс 15000  Суммарная скорость  обмена данными по  каналам Ethernet, Гбит/с 500  Потребляемая мощность,  КВт 24  Габариты пяти  вычислительных  стоек, мм 3000х1000х1800 |

## УСКОРИТЕЛИ ПЕРСОНАЛЬНОГО КОМПЬЮТЕРА «ФЕКДА» И «МЕРАК»

|  |  |
| --- | --- |
|  | РУПК-50 РУПК-25  1 плата  вычислительного  модуля 16V5-50 16S3-25;  процессоров  (IEEE-754) 256 128  тактовая частота МГц 250 200  - производительность  Гфлопс 50 25  обработка 64-разрядных данных IEEE-754  внешние интерфейсы – LVDS, Gigabit Ethernet |

## Вычислительный блок «ОРИОН-5» 1U

|  |  |
| --- | --- |
| 7smoll | Число базовых модулей 4  Число ПЛИС (11 млн.) 64  Производительность 800 Гфлопс  Частота платы 330 МГц  Частота обменов 1, ГГц  Высота 1U |

**РЕКОНФИГУРИРУЕМЫЕ ВЫЧИСЛИТЕЛЬНЫЕ СИСТЕМЫ НОВОГО ПОКОЛЕНИЯ НА ОСНОВЕ ПЛИС XILINX VIRTEX 7**

Основные отличия:

а) повышение плотности компоновки и

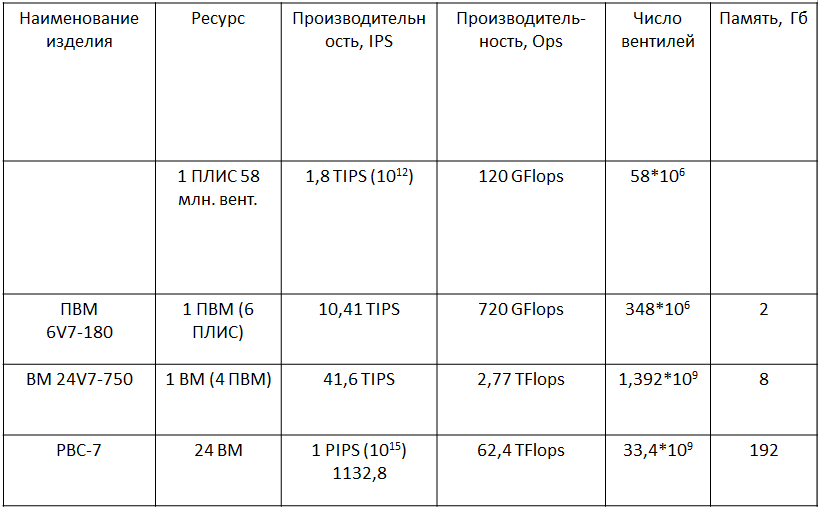
высокоскоростные каналы;

б) новые методы конструирования

в) повышение быстродействия по сравнению Virtex 6.

|  |  |
| --- | --- |
| Базовый модуль ПВМ 6V7-180  исп.1  TOP | Блок «Плеяда» |

ВЫЧИСЛИТЕЛЬНАЯ СИСТЕМА РВС-7, 24 ВМ 24V7-750, 47U, 2013, Производительность 1015 оп/c или 62 ТФлопс



# Комбинационные асинхронные устройства

Комбинационные схемы

Выходы комбинационных схем зависят только от текущих значений на входах, другими словами, такие схемы комбинируют текущие значения входных сигналов для получения значения на выходе.

Схема является комбинационной, если она состоит из соединенных между собой элементов и выполнены следующие условия:

- каждый элемент схемы сам является комбинационным;

- каждое соединение схемы является или входом, или подсоединено к выходу другого элемента схемы;

- схема не содержит циклических путей: каждый путь в схеме проходит через любое соединение не более одного раза.

## Булевы уравнения и логические устройства

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Не  Y <= NOT (A); | Not  Инверсия | \_\_\_\_ |  |  |
| И  Y <= A AND B; | And Конъюнкция | /\ |  |  |

## Комбинационные асинхронные устройства

## Булевы уравнения и логические устройства

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Или  Y <= A OR B; | Or  Дизъюнкция | \/ |  |  |
| Эквивалентность  Y <= not(A xor B); | Исключающее  И,  Равнозначность | ­  =1 |  |  |
| Полусуммирование  Y <= (A xor B); | Исключающее  или,  Неравнознач-ность,  Полусумма | ­  XOR |  |  |

Порядок операций при анализе булевых уравнений:

* Инверсия
* И
* Или



## Комбинационные асинхронные устройства

Минтерм и Макстерм

*Минтерм* (minterm, элементарная конъюнктивная форма) – это произведение, включающее все входы функции.

*Макстерм* (maxterm, элементарная дизъюнктивная форма) – это сумма всех

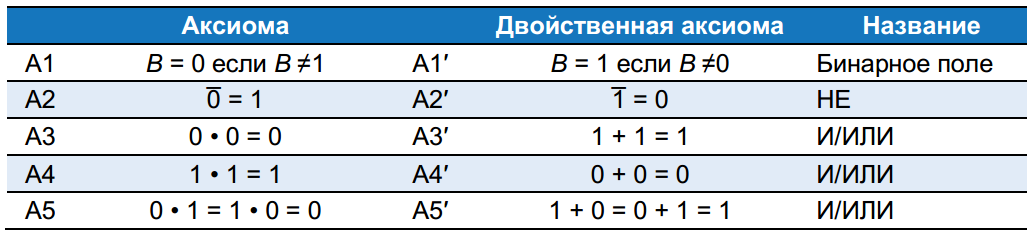
входов функции.

|  |  |
| --- | --- |
| Дизъюнктивная форма  Таблица истинности для функции *N* переменных содержит 2*N* строк, по одной для каждой возможной комбинации значений входов. Каждой строке в таблице истинности соответствует минтерм, который имеет значение ИСТИНА.  Просуммировав нужные нам минтермы, получим логическое выражение в совершенной дизъюнктивной форме. |  |
| Конъюктивная форма  Таблица истинности для функции *N* переменных содержит 2*N* строк, по одной для каждой возможной комбинации значений входов. Каждой строке в таблице истинности соответствует макстерм, который имеет значение ЛОЖЬ.  Перемножив нужные нам макстермы, получим логическое выражение в совершенной конъюктивной форме. |  |

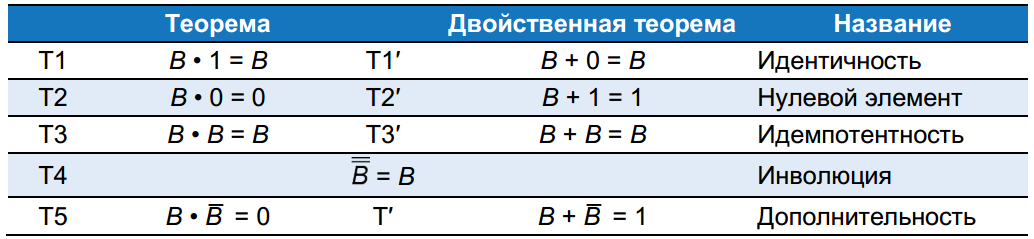
Задача: написать совершенную дизъюнктивную и совершенную конъюктивную формы для функции эквивалентности.

## Булевы уравнения и логические устройства

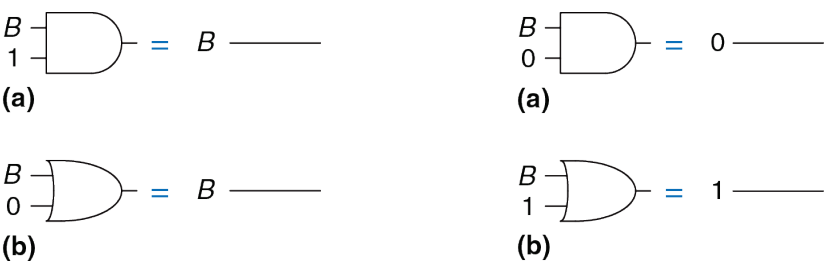
Аксиомы булевой алгебры

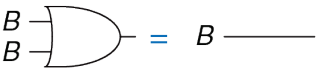
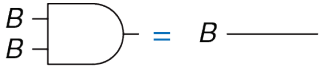


Теоремы булевой алгебры для одной переменной

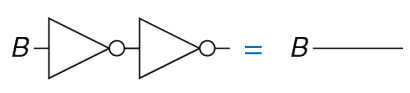


Теоремы булевой алгебры для одной переменной: схемотехнический эквивалент





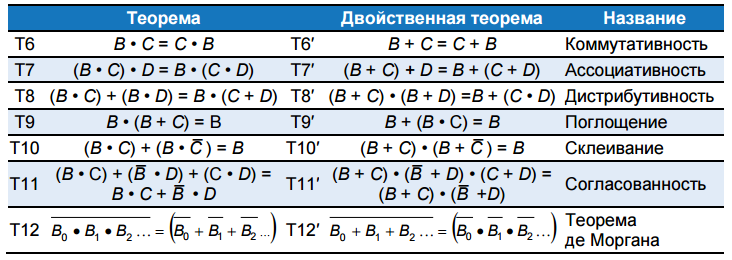




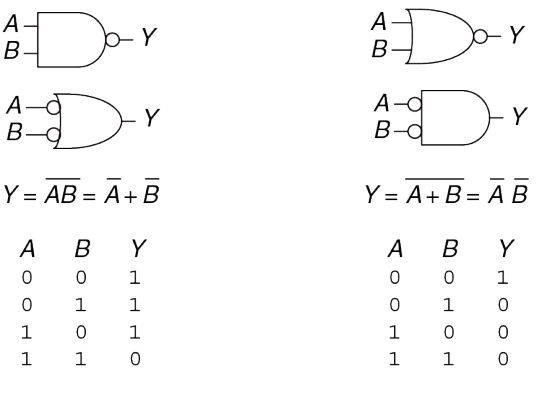
Последовательные цепочки инверторов используются в шинных формирователях

## Булевы уравнения и логические устройства

## Теоремы булевой алгебры для нескольких переменных



Теорема де Моргана: схемотехнические эквиваленты



Теорема де Моргана: словами

- перемещение инверсии назад (от выхода) или вперед (от входов) меняет тип элемента с И на ИЛИ и наоборот;

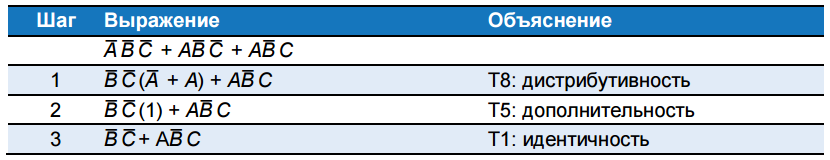
- перемещение инверсии с выхода назад ко входам приводит к тому, что на всех входах появляется инверсия;

- перемещение инверсии со всех входов элемента к выходу приводит к появлению инверсии на выходе.

## Комбинационные асинхронные устройства

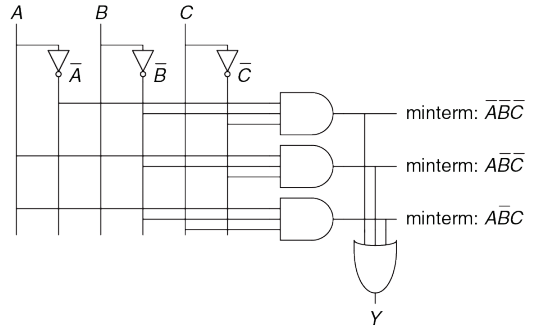
Пример оптимизации логического выражения





От логического выражения к логической схеме





NOTA <= NOT(A);

MINTERM1 <= NOTA AND NOTB AND NOTC;

NOTB <= NOT(В);

MINTERM2 <= A AND NOTB AND NOTC;

NOTC <= NOT(C);

MINTERM3 <= A AND NOTB AND C;

Y <= MINTERM1 OR MINTERM2 OR MINTERM3;

## Правила оформления логических схем

Входы изображаются на левой (или верхней) части схемы.

Выходы изображаются на правой (или нижней) части схемы.

Всегда, когда это возможно, элементы необходимо изображать слева направо.

Проводники лучше изображать прямыми линиями, чем линиями с множеством углов (неровные рваные линии отвлекают внимание: приходится следить за тем, куда ведут провода, а не думать о том, что делает схема).

Проводники всегда должны соединяться в виде буквы «Т».

Проводники, пересекающиеся без точки, не имеют соединения друг с другом.

## Карты Карно

Карты Карно представляют собой наглядный метод для упрощения булевых уравнений. Карты Карно очень удобны в случаях, когда уравнение содержит до четырёх переменных. Но, что более важно, они дают понимание сути при работе с логическими выражениями.

Логическая минимизация осуществляется путем склейки термов. Два терма, включающие в себя импликанту P и два логических значения некоторой переменной A, объединяются, при этом переменная A исключается. Карты Карно позволяют легко находить термы, которые можно склеить, располагая их в виде таблицы.

## Таблица истинности и карта Карно для функции трех переменных

Верхняя строка дает 4 возможных значения для переменных A и B. Левая колонка дает 2 возможных значения переменной C.

|  |  |  |
| --- | --- | --- |
|  |  |  |

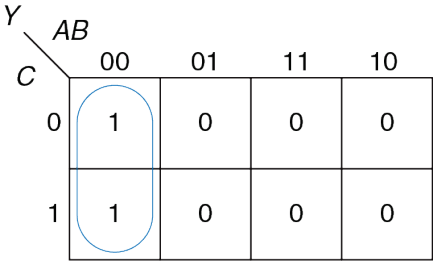
Каждая клетка карты Карно соответствует строке таблицы истинности и содержит значение функции Y из этой строки. Например, верхняя левая клетка соответствует первой строке таблицы истинности и показывает, что значение функции Y будет равно 1, когда ABC=000.

Как и каждая строка в таблице истинности, каждая клетка карты Карно представляет собой отдельный минтерм.

Как и раньше, мы могли бы использовать булеву алгебру для минимизации:

.

Карты Карно помогают нам делать это упрощение графически, обводя единицы в соседних клетках овалами. Для каждого овала мы пишем соответствующую ему импликанту.



Переменные, для которых прямая и комплементарная формы попадают в один овал, исключаются из импликанты.

Карты Карно обеспечивают простой визуальный способ минимизации логических выражений. Просто обведите все прямоугольные блоки с единицами на карте, используя наименьшее возможное число овалов. Каждый овал должен быть максимально большим. Затем прочитайте все импликанты, которые обведены.

Максимально возможный овал является первичной импликантой.

Правила для нахождения минимального уравнения из карт Карно: использовать меньше всего овалов, необходимых для покрытия всех 1.

Все клетки в каждом овале обязаны содержать 1.

Каждый овал должен охватывать блок, число клеток которого в каждом направлении равно степени двойки (то есть 1, 2 или 4).

Каждый овал должен настолько большим, насколько это возможно.

Овал может связывать края карты Карно.

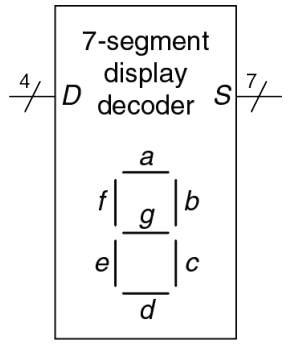
Единица на карте Карно может быть обведена сколько угодно раз, если это позволяет уменьшить число овалов, которые будут использоваться.

## Пример разработки

Задача: разработать декодер семисегментного индикатора.

Декодер получает на вход четырехбитные данные D[3:0] и формирует семь выходов для управления светодиодами для показа цифр от 0 до 9.

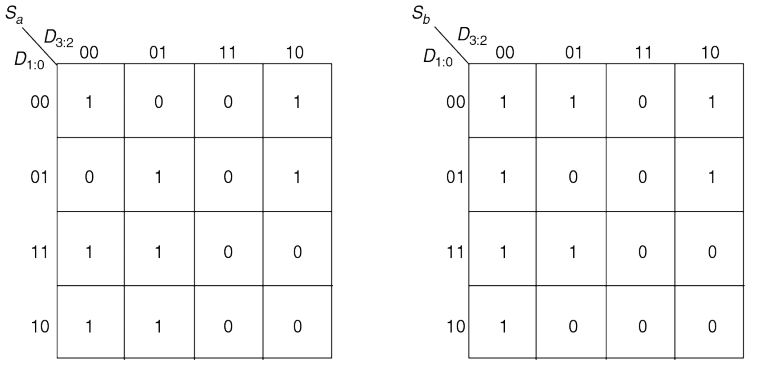
Семь выходов часто называют сегментами от a до g, или Sa–Sg.



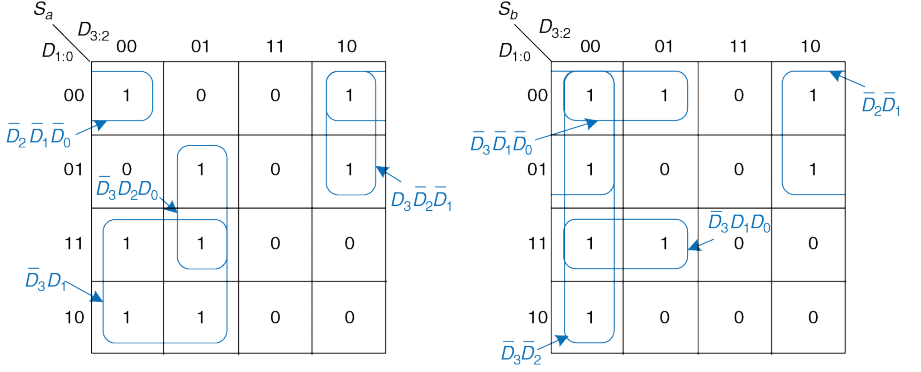
Строим таблицу истинности



Каждый из семи выходов является независимой функцией от четырех переменных. Карты Карно для выходов Sa и Sb:



Обводим первичные импликанты и записываем оптимизированные уравнения.





Остается зарисовать схемы, соответствующие разработанным уравнениям или VHDL код:



SA <= (NOT (D3) AND D1) OR (NOT(D3) AND D2 AND D0) OR

(D3 AND NOT(D2) AND NOT(D1)) OR

(NOT (D2) AND NOT (D1) AND NOT (D0));



SB <= (NOT(D3) AND NOT(D2)) OR(NOT(D2) AND NOT (D1)) OR

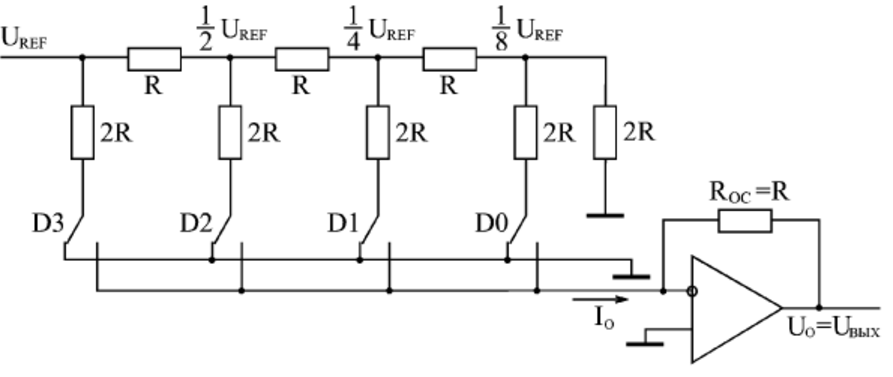
(NOT (D3) AND NOT(D1) AND NOT (D0));

и так далее для всех сегментов.

## Цифро- аналоговое преобразование

Суть преобразования входного цифрового кода в выходной аналоговый сигнал довольно проста. Она состоит в суммировании нескольких токов (по числу разрядов входного кода), каждый последующий из которых вдвое больше предыдущего. Для получения этих токов используются или транзисторные источники тока, или резистивные матрицы, коммутируемые транзисторными ключами.

Пример цифро- аналогового преобразователя



Первым (левым по рисунку) ключом коммутируется ток величиной UREF/2R, вторым ключом — ток UREF/4R, третьим — ток UREF/8R, четвертым — ток UREF/16R. То есть токи, коммутируемые соседними ключами, различаются вдвое, как и веса разрядов двоичного кода. Токи, коммутируемые всеми ключами, суммируются и преобразуются в выходное напряжение с помощью операционного усилителя с сопротивлением RОС=R в цепи отрицательной обратной связи.

При правом положении каждого ключа (единица в соответствующем разряде входного кода ЦАП ) ток, коммутируемый этим ключом, поступает на суммирование. При левом положении ключа (нуль в соответствующем разряде входного кода ЦАП ) ток, коммутируемый этим ключом, на суммирование не поступает.

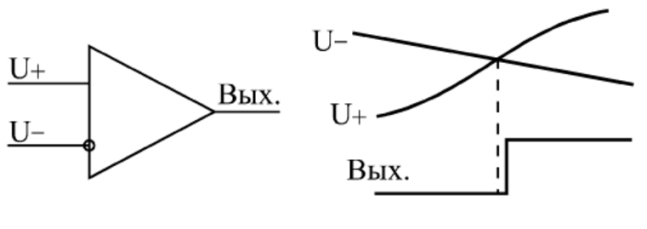
Суммарный ток IO от всех ключей создает на выходе операционного усилителя напряжение UO=IO RОС=IOR. То есть вклад первого ключа (старшего разряда кода) в выходное напряжение составляет UREF/2, второго — UREF/4, третьего — UREF/8, четвертого — UREF/16. Таким образом, при входном коде N = 0000 выходное напряжение схемы будет нулевым, а при входном коде N = 1111 оно будет равно –15UREF/16.

В общем случае выходное напряжение ЦАП при RОС = R будет связано со входным кодом N и опорным напряжением UREF простой формулой

UВЫХ = –N • UREF 2-n где n — количество разрядов входного кода. Знак минус получается из-за инверсии сигнала операционным усилителем.

## Аналого- цифровое преобразование

Аналого- цифровые преобразователи основаны на использовании компараторов напряжения:



Параллельный АЦП использует набор таких компараторов

## Параллельный АЦП

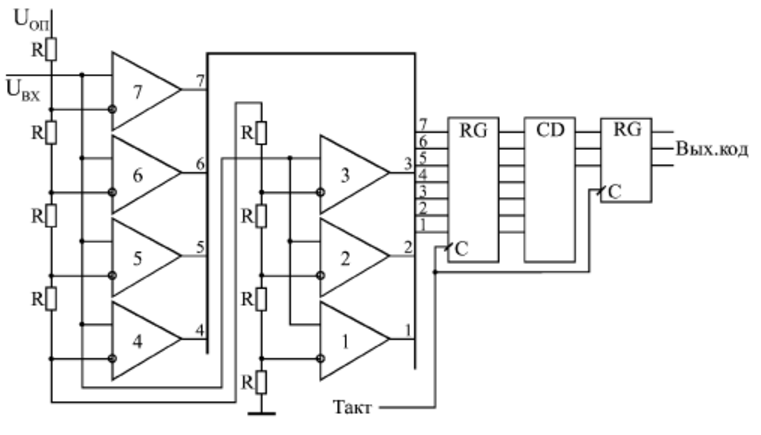


Схема такого АЦП включает в себя резистивный делитель из 2n одинаковых резисторов, который делит опорное напряжение на (2n–1) уровней.

## Комбинационные асинхронные устройства

## Мультиплексор

Мультиплексор передает на выход один из нескольких входных сигналов данных, основываясь на сигнале выбора:

- для двухвходового мультиплексора:

если S = 0, выход Y = D0,

если S = 1, то выход Y = D1.

S также называют управляющим сигналом, так как он управляет поведением мультиплексора.

|  |  |
| --- | --- |
| Обозначение | Таблица истинности |
| Вариант реализации | -- Поведенческое описание на VHDL  process(S, D0, D1)  begin  case S is  when "0" => Y <= D0;  when "1" => Y <= D1;  when others => Y <= "U";  end case;  end process;  -- "U" означает неопределенность |

## Мультиплексор на четыре входа: варианты

|  |  |
| --- | --- |
|  |  |

## Мультиплексор на четыре входа: описание на vhdl

**library** ieee;

**use** ieee.std\_logic\_1164.**all**;

**entity** mux4 **is**

**port**(a, b, c, d: **in** std\_logic\_vector(7 **downto** 0);

s: **in**  std\_logic\_vector(1 **downto** 0);

x: **out** std\_logic\_vector(7 **downto** 0));

**end** mux4;

**architecture** cond\_arch **of** mux4 **is**

**begin**

x <= a **when** (s = "00") **else**

b **when** (s = "01") **else**

c **when** (s = "10") **else**

d;

**end** cond\_arch;

-- Конструкция **when** **else** должна быть исчерпывающей**!**

## Дешифратор

В общем случае у дешифратора имеется N входов и 2N выходов. Он выдает единицу строго на один из выходов в зависимости от набора входных значений.

|  |  |
| --- | --- |
| Вариант обозначения | Таблица истинности |
| Схемотехническая реализация | -- описание на VHDL  Y0 <= **NOT** (A0) **AND NOT** (A1);  Y1 <= **NOT** (A0) **AND** A1 ;  Y2 <= A0 **AND NOT** (A1);  Y3 <= A0 **AND** A1 ; |

## Компаратор равенства

Выдает один выходной сигнал, показывая, равны ли А и В (A=B).

|  |  |
| --- | --- |
| Вариант обозначения | -- описание на VHDL  EQUAL <= **XOR** (A0,B0) **AND**   **XOR** (A1,B1) **AND  XOR** (A2,B2) **AND  XOR** (A3,B3); |
| Схемотехническая реализация | -- описание на VHDL  EQUAL <= '1' **when** 'A '= 'B' **else** '0';  -- описание на VHDL  **if** ('A' = 'B') **then**  EQUAL <= '1';  **elsif** EQUAL <= '0'; **end if**; |

## Однобитный полусумматор

Выдает арифметическую сумму двух однобитных данных:

|  |  |  |
| --- | --- | --- |
| Вариант обозначения | Таблица истинности | Описание |

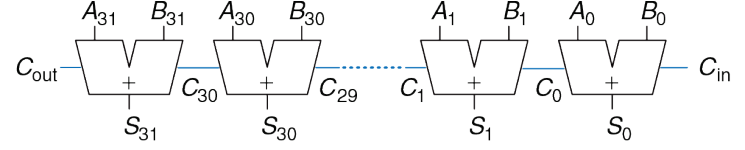
## Однобитный полный сумматор

Выдает арифметическую сумму двух однобитных данных:

|  |  |  |
| --- | --- | --- |
| Вариант обозначения | Таблица истинности | Описание |

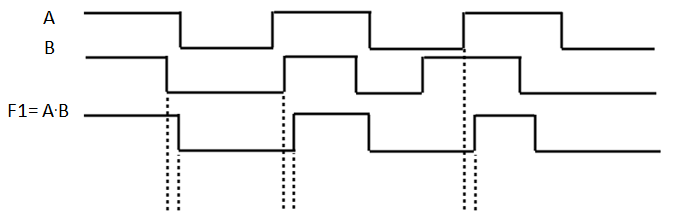
## Сумматор с последовательным переносом

Строится из однобитных полных сумматоров (в первом разряде - однобитный полусумматор).

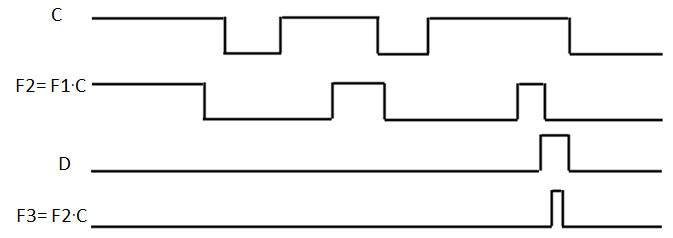


# Дискретные устройства. Синхронизаторы, триггеры, регистры, счетчики

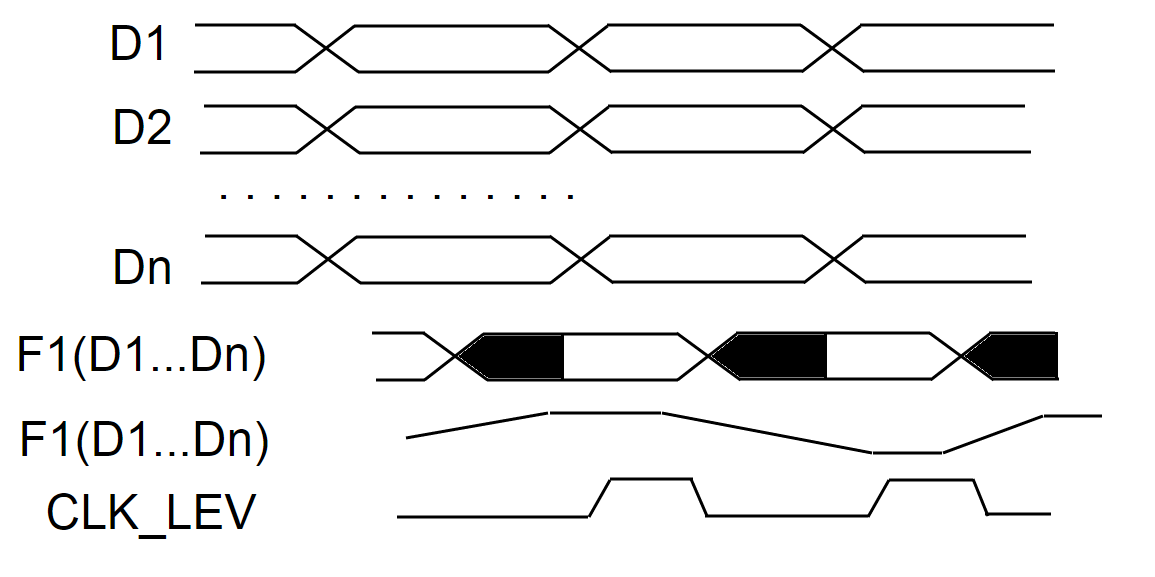
## Распространение сигналов. Быстродействие. Понятие о задержке



Задержкa 1->0 Задержкa 0->1 Задержкa 1->0



Сигналы A, B, C, D заданы, сигналы F1, F2, F3 получаются из них



Короткий по времени фронт позволяет зафиксировать короткий сигнал

Фиксация сигнала по уровню  может оказаться или ресурсоемкой, или ненадежной

|  |  |
| --- | --- |
| Переход от НИЗКОГО уровня к ВЫСОКОМУ называется положительным перепадом или фронтом. Аналогично переход от ВЫСОКОГО уровня к НИЗКОМУ называется соответственно отрицательным перепадом или срезом. |  |

|  |  |
| --- | --- |
| Комбинационная логика характеризуется задержкой распространения (propagation delay) и задержкой реакции или отклика (contamination delay).  Задержка распространения tpd –максимальное время от начала изменения входа до момента, когда все выходы достигнут установившихся значений. Задержка реакции tcd –минимальное время от момента, когда вход изменился, до момента, когда любой из выходов начнет изменять свое значение. |  |

Различные задержки в логическом устройстве

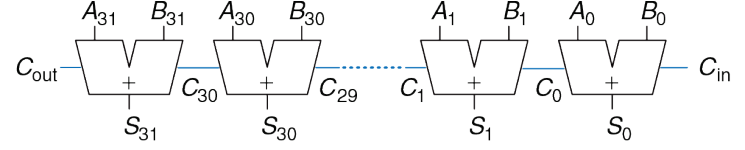
|  |  |
| --- | --- |
|  |  |

Варианты реализации мультиплексора: где больше задержка?

|  |  |  |
| --- | --- | --- |
|  |  |  |

Критические цепи. Критической является цепь с самой   
большой задержкой, например:

|  |  |  |
| --- | --- | --- |
|  |  |  |

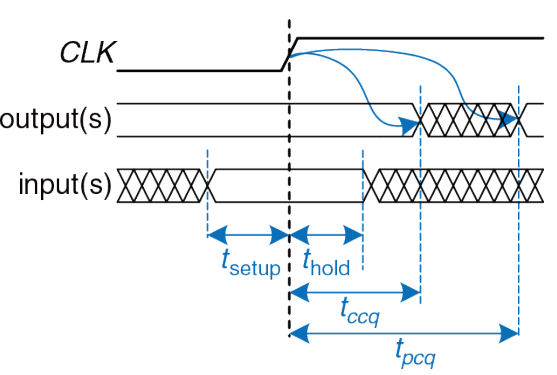
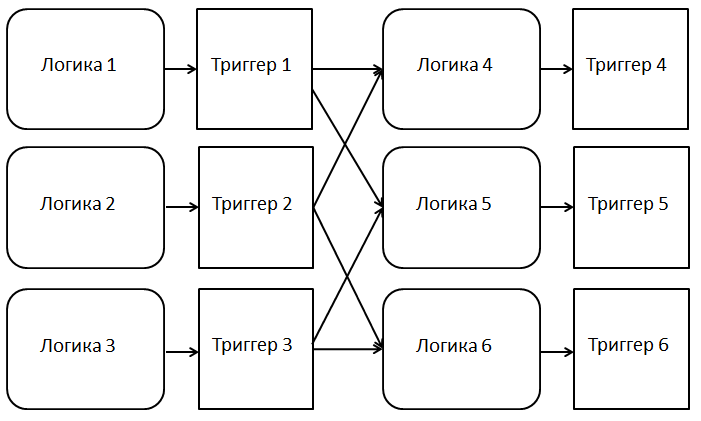


Цепь переноса в сумматоре является критической: в ней последовательно включено 2\*n элементов, где n- разрядность.

Примеры проблем, требующих синхронизации

|  |  |
| --- | --- |
|  |  |
|  | |

Базовый принцип синхронизации



Динамическая дисциплина

Входы синхронной последовательной схемы должны быть стабильны в течение времени предустановки до и времени удержания после фронта тактового импульса.

Выполнение этих требований гарантирует, что в процессе фиксации значения информационного входа триггером он не будет изменяться.

## Метастабильность

Сам фронт сигнала тактовой частоты немного растянут во времени, да и триггер переключается не мгновенно, на его переключение требуется время.

Чтобы успешно зафиксировать входное значение в D-триггере, входной сигнал должен быть стабилен некоторое время до фронта и после фронта тактовой частоты.

Если этого не произошло, триггер может оказаться в «промежуточном» состоянии, называемом метастабильным.

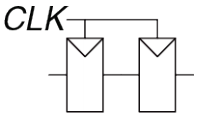
Метастабильное состояние триггера не является устойчивым. После выхода из метастабильного состояния триггер оказывается в произвольном состоянии.



Попадание триггера в метастабильное состояние - вероятностный процесс, причем вероятность связана как с параметрами микросхемы, так и с тактовой частотой.

Вероятность этого события можно уменьшить, используя два последовательно включенных триггера, при этом вероятность того, что оба триггера попадут в метастабильное состояние, равна квадрату исходной вероятности.

Если первый триггер и «поймает» метастабильное состояние, то, вероятно, оно должно пройти к моменту фиксации сигнала во втором триггере.

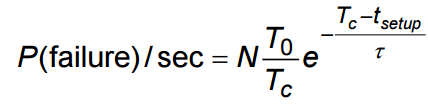


Простой синхронизатор и временная диаграмма его функционирования

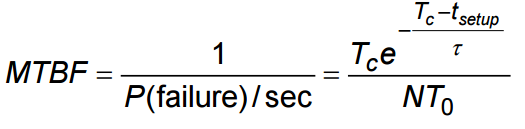
|  |  |
| --- | --- |
| Триггер F1 фиксирует значение входного сигнала D по переднему фронту тактового сигнала CLK. Если D изменяется в апертурное время, его выход D2 на некоторое время может стать метастабильным. Если период тактового сигнала достаточно велик, то с высокой вероятностью до конца периода D2 придет к корректному логическому уровню. Триггер F2 затем фиксирует D2, который теперь стабилен, и формирует корректный выходной сигнал. |  |

Вероятность сбоя синхронизатора

Если D изменяется N раз за секунду, то вероятность ошибки за секунду составит

.

A среднее время наработки на отказ (mean time between failures, MTBF) составит

.

MTBF растет экспоненциально с ростом времени ожидания синхронизатора *Тс*. Для большинства систем синхронизатор, который ожидает один период тактового сигнала, обеспечивает достаточную величину MTBF. В высокоскоростных системах может понадобиться ожидание на большее количество периодов тактового сигнала.

## Дискретные устройства - триггеры

Дискретное (последовательное) цифровое устройство

Выходное состояние устройства зависит от текущего состояния и предыдущих входных воздействий.

|  |  |
| --- | --- |
| Бистабильная защелка: |  |

Случай 1: Q=0

На вход поступает сигнал Q = 1. I2 инвертирует сигнал и подает на вход I1 сигнал Q¯ = 1. Соответственно, на выходе I1 – логический «0». В рассмотренном случае схема находится в стабильном состоянии.

Случай II: Q=1

На вход I2 поступает 1 (Q). I2 инвертирует сигнал и подает на вход I1 «0». Соответственно, на выходе I1 – логическая «1». В этом случае схема также находится в стабильном состоянии.

Так как инверторы, включенные перекрестно, имеют два стабильных состояния Q = 0 и Q = 1, то говорят, что схема бистабильна.

Попытка одновременно подать на входы одинаковые сигналы приведет к разрушению устройства.

|  |  |
| --- | --- |
| R-S триггер: |  |

Случай I: R=1, S=0

На входе N1 как минимум одна единица – вход R, следовательно, выход Q=0. Оба входа N2 – в состоянии логического нуля (Q=0 и S=0), поэтому выход Q=1.

Случай II: R=0, S=1

На вход N1 поступает 0 и Q¯ . Так как мы еще не знаем значения Q¯, мы не можем определить значение Q.

На вход N2 поступает как минимум одна единица S, поэтому на выходе Q¯ 0. Теперь можно вернуться к определению состояния выхода элемента N1. Мы знаем, что на обоих его входах 0, следовательно, Q=1.

Случай III: R=1, S=1

Запрещенная комбинация входных воздействий. Как на входе N1, так и на входе N2 как минимум по одной единице (R и S), поэтому на выходе каждой защелки – логический 0.

Случай IV: R=0, S=0

На вход N1 поступает 0 и Q¯. Так как мы еще не знаем значения Q¯, то не можем определить значение на выходе элемента N1.

На вход N2 поступают 0 и Q. Так как мы еще не знаем значения Q, то не можем определить значение на выходе элемента N2. Этот случай аналогичен случаю с двумя перекрестно включенными инверторами. Мы знаем, что Q должен быть равен либо 0, либо 1. Рассмотрим каждый из этих двух случаев.

Случай IVa: Q=0

Так как S и Q равны 0, то на выходе N2 будет логическая 1, Q¯=1. Теперь на входе N1 есть одна единица – Q¯, поэтому на его выходе Q=0.

Случай IVb: Q=1

Так как Q = 1, то на выходе N2 будет 0, Q¯ =0. Теперь на обоих входах N1 нули (R и Q¯), поэтому на его выходе логическая 1, Q=1. Устройство остается в предыдущем состоянии.

Итак, таблица истинности для R-S триггера выглядит следующим образом:

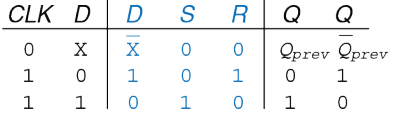
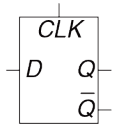
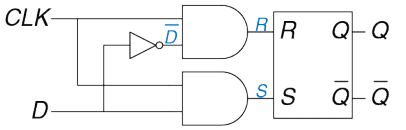
|  |  |
| --- | --- |
|  |  |

RS-триггер - бистабильный элемент с одним битом состояния, хранящимся в Q.

Состоянием можно управлять при помощи входов R и S. Когда на R поступает высокий уровень, выход сбрасывается в 0. Когда высокий уровень приходит на S, выход устанавливается в 1. Если ни на один вход не пришла логическая единица, триггер сохраняет свое предыдущее состояние, значение выходов не изменяется.

## D триггер с синхронизацией по уровню

Сигнал D записывается в R-S триггер по разрешению от сигнала CLK

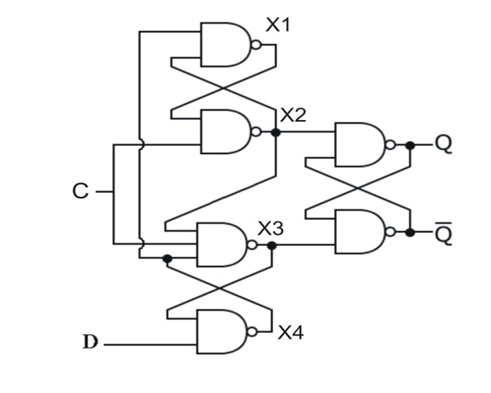


## D триггер, работающий по фронту

Из двух D триггеров, работающих по уровню, можно построить D триггер, работающий по фронту тактового сигнала. Полученное устройство будет переключаться по фронту нарастания сигнала.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  | Предыдущее Новое |

Другой вариант реализации D триггера, работающего по фронту:



|  |  |
| --- | --- |
| -- описание D триггера на vhdl  Process (clk)  Begin  if (clk='0' and c'event) then  q<=d;  End if;  End process; |  |

D триггер со сбросом

В триггере с функцией сброса добавляется вход RESET (сброс). Когда на RESET подан 0, триггер ведет себя как обычный D-триггер. Когда на RESET подана 1, такой триггер игнорирует вход D и сбрасывает выход в 0. В данном случае сброс возможен только по переднему фронту синхросигнала.

|  |  |
| --- | --- |
| -- описание D триггера со сбросом на vhdl  process (clk) begin  if (clk'event and clk = '1') then  if (reset = '1') then  q <= '0';  else q <= data;  end if;  end if;  end process; |  |

D триггер со сбросом и установкой

В триггере с функцией установки добавляется вход SET. Когда на SET подан 0, триггер ведет себя как обычный D-триггер. Когда на SET подан 1, такой триггер игнорирует вход D и устанавливает выход в 1.

|  |
| --- |
| process (clk) begin  if (clk'event and clk = '1') then  if (reset = '1') then q <= '0';  if (set = '1') then q <= ‘1';  else q <= data;  -- r=q=0 – хранение данных  end if; end if; end if;  end process; |

D триггер со сбросом, установкой и разрешением (вход CE-разрешение на запись)

|  |  |
| --- | --- |
|  |  |

## Счетный триггер (Т-триггер)

D-триггер, у которого инверсный выход подключен к входу, называется Т- триггером. После поступления на этот вход импульса состояние Т-триггера меняется на прямо противоположное. Счётным он называется потому, что подсчитывает количество импульсов, поступивших на его вход. Считать этот триггер умеет только до одного. При поступлении второго импульса T-триггер снова сбрасывается в исходное состояние.

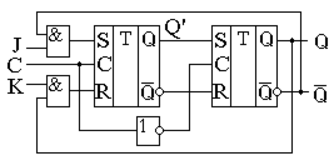
|  |  |
| --- | --- |
|  |  |

## 

## JK триггер

|  |  |
| --- | --- |
|  |  |

|  |  |
| --- | --- |
|  |  |



## Сдвиговый регистр



Cдвиговые регистры представляют собой последовательно соединенную цепочку триггеров. Основной режим их работы - это сдвиг разрядов кода, записанного в эти триггеры, то есть по тактовому сигналу содержимое каждого предыдущего триггера переписывается в следующий по порядку в цепочке триггер. Сдвиг бывает двух видов: вправо - это основной режим, который есть у всех сдвиговых регистров, и влево - этот режим есть только у некоторых реверсивных сдвиговых регистров.

Поведенческое описание сдвигового регистра:

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity SReg8 is

Port ( clk : in STD\_LOGIC;

D : in STD\_LOGIC;

Q : out STD\_LOGIC);

end SReg8;

architecture Behavioral of SReg8 is

Signal S : std\_logic\_vector(7 downto 0) := (others => '0');

begin

Process (clk)

Begin

If (rising\_edge(clk)) then

S(7 downto 1) <= S(6 downto 0);

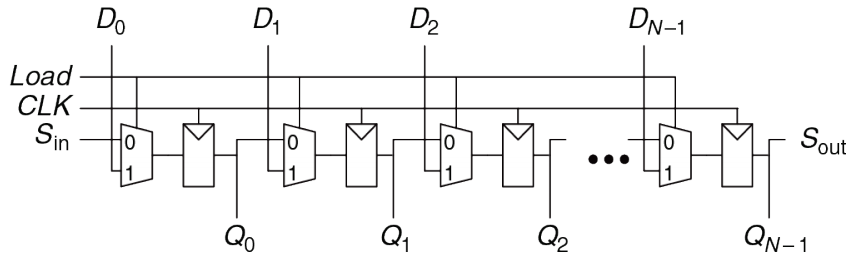
S(0) <= D;

End if;

End process;

Q <= S(7);

## end Behavioral; Сдвиговый регистр с параллельной загрузкой



library IEEE; use IEEE.STD\_LOGIC\_1164.ALL;

entity shiftreg is

generic(N: integer := 8);

port(clk, reset: in STD\_LOGIC; load, sin: in STD\_LOGIC;

d: in STD\_LOGIC\_VECTOR(N–1 downto 0);

q: out STD\_LOGIC\_VECTOR(N–1 downto 0);

sout: out STD\_LOGIC); end;

architecture synth of shiftreg is

process(clk, reset) begin

if reset = ‘1’ then q <= (OTHERS => ‘0’);

elsif rising\_edge(clk) then

if load then q <= d;

else q <= q(N–2 downto 0) & sin;

end if;

end if;

end process;

sout <= q(N–1);

end;

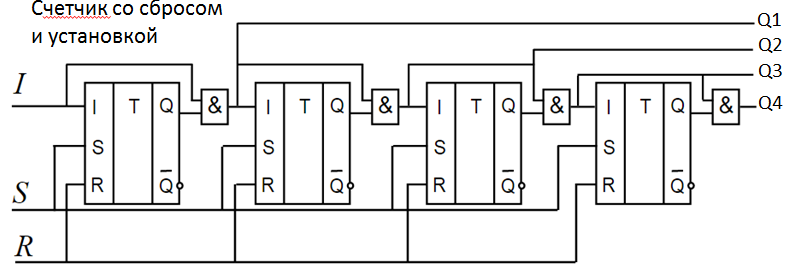
## Счетчики

Счетчик на основе Т-триггеров



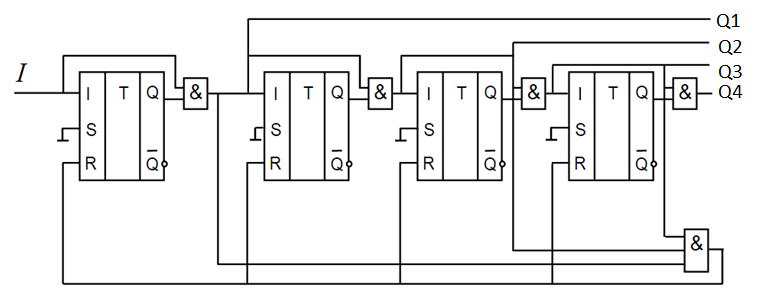
Последовательное соединение Т- триггеров формирует на выходах сигналы

0000, 0001, 0010 … 1111, 0000…



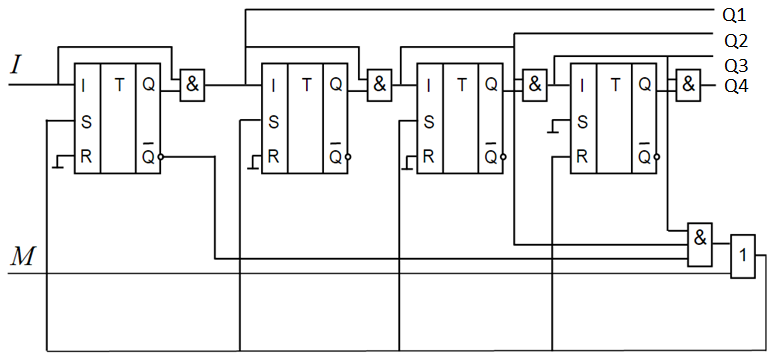
Используя входы сброса и установки триггеров, можно перевести схему в состояние 0000 (подав «1» на вход *R*) или 1111 (подав «1» на вход *S*).

## Счетчик с промежуточным сбросом



Сигнал сброса можно сформировать из сигналов (в данном случае из Q1, Q2 и Q3).

## Cчетчик с пропуском состояний и установкой в «0111»



В данном случае сигнал, сформированный из Qi устанавливает часть триггеров в «0», часть в «1»

## Поведенческое описание работы счетчика

-- Простой счетчик со сбросом

process(CLK)

begin

if CLK 'event and CLK = '1' then

if (rst) = '1' then

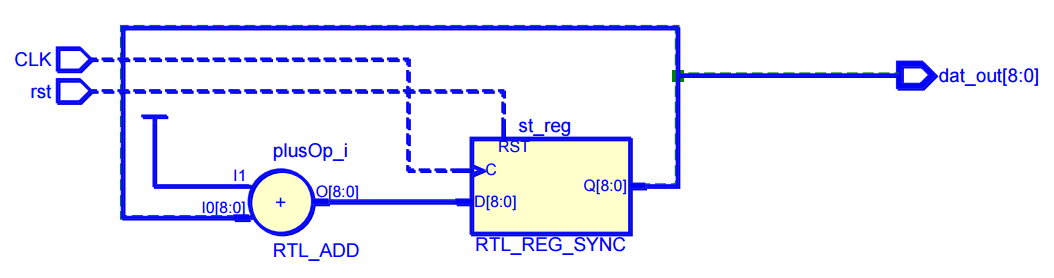
st <= "000000000";

else

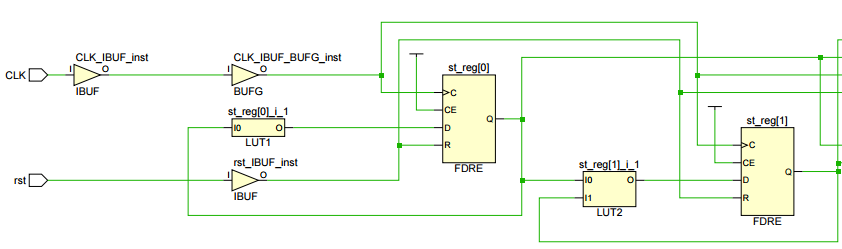
st <= st + 1;

end if;

Схема, синтезированная по поведенческому описанию

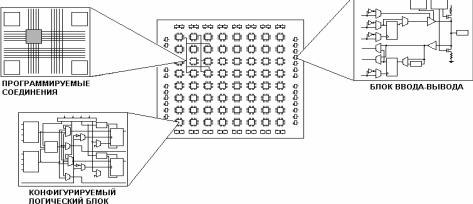


## Счетчик: поведенческое описание, переведенное САПР на элементы ПЛИС (фрагмент)



# Структура ПЛИС

В наиболее типичном варианте ПЛИС, выполненная по технологии FPGA, состоит из прямоугольной матрицы конфигурируемых логических блоков (Configurable Logic Blocks, CLB), блоков ввода-вывода (Input/Output Block, IOB) и дополнительных блоков. Между CLB располагаются программируемые трассировочные линии. Между матрицей CLB и блоками ввода-вывода имеются отдельные межсоединения, которые и обеспечивают подключение внешних сигналов.



**FPGA** – **это** сокращение от английского словосочетания Field Programmable Gate Array. Также используется термин CPLD (complex programmable logic device).

|  |  |
| --- | --- |
|  | Современные ПЛИС также содержат блоки памяти (BRAM), секции цифровой обработки сигналов (DSP) и в отдельных случаях аналого-цифровые и цифроаналоговые преобразователи.  Каждый модуль двухпортовой блочной памяти емкостью 36 кбит, который может использоваться в виде двух независимых блоков объемом 18 кбит, оснащен встроенной системой обнаружения и коррекции ошибок (ECC) и специальной схемой управления, необходимой для организации запоминающих устройств, функционирующих по принципу «первым вошел – первым вышел» (FIFO). |

|  |  |
| --- | --- |
|  | В состав секций SLICEL входит логика ускоренного переноса и мультиплексоры, предназначенные для расширения функциональных возможностей конфигурируемых логических блоков CLB.    В секциях типа SLICEM, кроме логики ускоренного переноса и мультиплексоров расширения количества аргументов выполняемых функций, применяются таблицы преобразования, которые могут конфигурироваться как элементы 64-разрядной распределенной памяти или как сдвиговые регистры различной разрядности (до 32 разрядов). |

|  |  |
| --- | --- |
|  | Основной элемент для реализации логических функций - ЛОГИЧЕСКИЙ ГЕНЕРАТОР 6LUT. Блоки 5LUT- статическая память.    Логический генератор (LUT) реализует любую функцию с не более чем 6 операндами.    Кроме того, возможно использование LUT блоков SLICEM в режиме с частично совмещенными входами, когда память разбивается на два фрагмента по 32 бита (формируются две логические функции с не более чем 5 операндами каждая).    Поскольку в самой LUT всего 6 входов, часть входов должна быть обобщена. Такой режим автоматически используется САПР. |

|  |  |
| --- | --- |
|  | Логические генераторы (LUT) могут также использоваться в режимах распределенной памяти (distributed memory) и сдвиговых регистров (SHR). Поскольку LUT представляет собой элемент статической памяти, который хранит таблицу истинности в явном виде, в SLICEM его можно использовать по прямому назначению – в качестве обычного компонента памяти.    Показанный элемент распределенной памяти имеет следующие сигналы:  – we – сигнал разрешения записи;  – d – данные для записи;  – wclk – вход тактового сигнала;  – a – адрес для записи;  – dpra – адрес для чтения, второй порт  (dualportreadaddress);  – spo – выход первого порта (singleportoutput);  – dpo – выход второго порта (dualportoutput). |

|  |  |
| --- | --- |
|  | LUT может функционировать и как сдвиговый регистр, использующий следующие сигналы:  – d – данные для записи в сдвиговый регистр;  – clk – тактовый сигнал;  – a – адресный вход;  – q – выход данных.  LUT в режиме сдвигового регистра может использоваться совместно с триггером той же логической ячейки, увеличивая глубину сдвигового регистра на 1. |

Временные диаграммы работы триггера

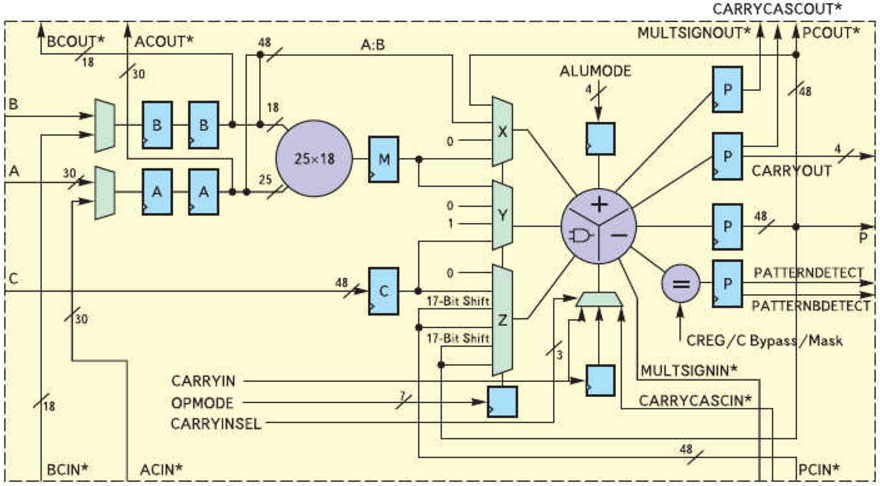
|  |  |
| --- | --- |
|  |  |

Сигналы PRE, CLR действуют асинхронно, то есть установка триггера может осуществляться без привязки к фронту тактового сигнала.

Для логической секции FPGA Xilinx существует понятие controlset. Оно соответствует набору управляющих сигналов (clk, ce, sr), которые должны быть идентичны для всех триггеров секции. Это означает, например, что невозможно использовать сигнал CE только для одного триггера в секции. Следует обратить внимание, что тактовый сигнал также включен в controlset, т.е. все триггеры должны тактироваться одним и тем же тактовым сигналом. На практике такое ограничение означает, что при необходимости использовать разные управляющие сигналы и соответствующие триггеры будут помещены в разные секции.

|  |  |
| --- | --- |
|  | Триггеры FPGA имеют гарантированное начальное значение после включения питания. Значения всех триггеров (и других компонентов системы) записаны в конфигурационном файле, и инициализация выполняется с помощью специальных трассировочных ресурсов. Поэтому отдельные цепи сброса для обеспечения начального состояния триггера являются избыточными и ухудшают проект. Самый простой способ начальной установки триггера – объявление сигнала с начальным значением:  signal reg1 : std\_logic := ‘1’; |

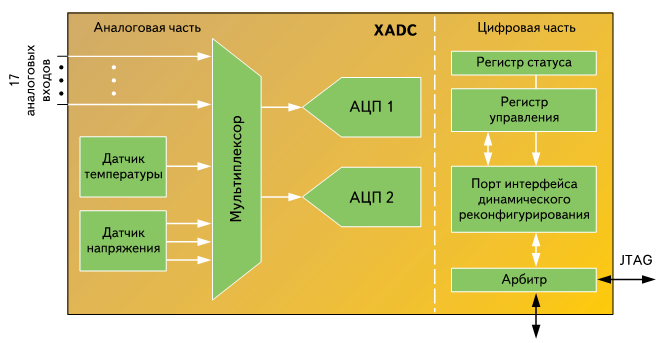
В этом примере на VHDL объявляется триггер с именем reg1, которому присваивается начальное значение 1. Это значение будет корректно учитываться средствами моделирования, и будет записано в конфигурационный файл FPGA.



В ПЛИС семейства VIRTEX (ARTIX-7 и Kintex-7) содержатся многофункциональные блоки, предназначенные для стандартных операций по цифровой обработке сигналов (DSP48E1).

Вычислительные возможности блока ЦОС видны из схемы АЛУ и блок умножения.

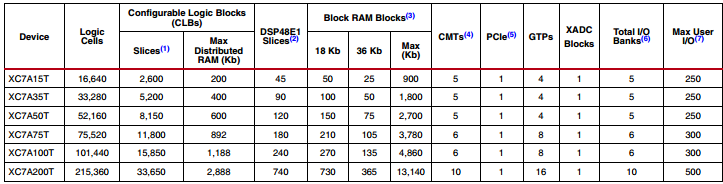
В ПЛИС Xilinx 7-й серии содержатся аналого-цифровые блоки XADC.



Каждый блок содержит два 12-разрядных аналого-цифровых преобразователя (АЦП) и мультиплексор. Кроме того, к аналого-цифровому блоку относятся внутрикристальные датчики напряжения питания и температуры.

Аналого-цифровой блок XADC поддерживает 17 внешних входных аналоговых каналов. Коммутация аналоговых сигналов на входы АЦП осуществляется с помощью мультиплексора. Каждый АЦП позволяет выполнять преобразование входного аналогового сигнала с частотой дискретизации 1 млн. выборок/с и точностью 0,1%. Наличие блока XADC в составе ПЛИС позволяет реализовать устройства цифровой обработки низкочастотных сигналов на базе одного кристалла без внешних АЦП.

Семейство Artix-7



Наименьшие значения потребляемой мощности и стоимости ориентированы на применение в составе серийно выпускаемой аппаратуры.

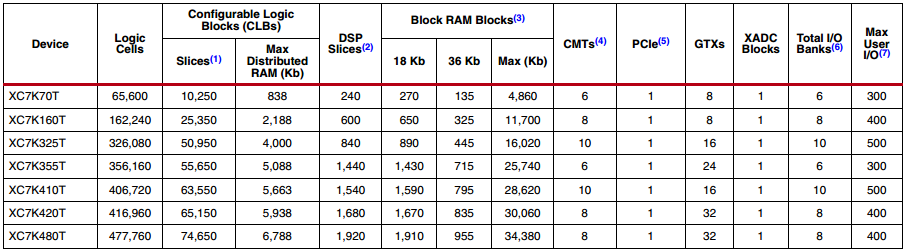
CLB - общее количество триггеров.

CMT - блоки управления синхронизацией.

PCIe - интерфейсы к PCI express.

XADC - аналого-цифровые преобразователи.

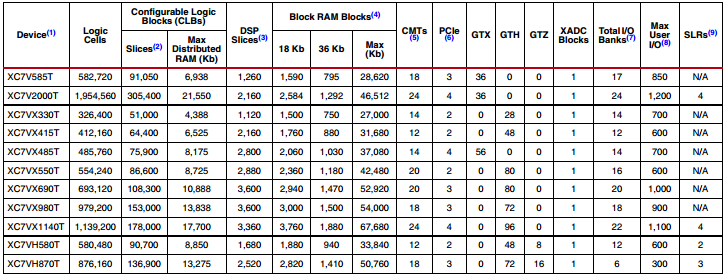
Семейство Кintex-7



Увеличено число ЦОС DSP48E1 и модулей двухпортовой блочной памяти Block RAM. Используются высокоскоростные приемопередатчики RocketIO типа GTX, поддерживающие скорость приема и передачи данных до 10,3125 Гбит/с.

Содержат аппаратный модуль интерфейса PCI Express, который поддерживает скорости передачи данных 2,5 и 5 Гбит/с.

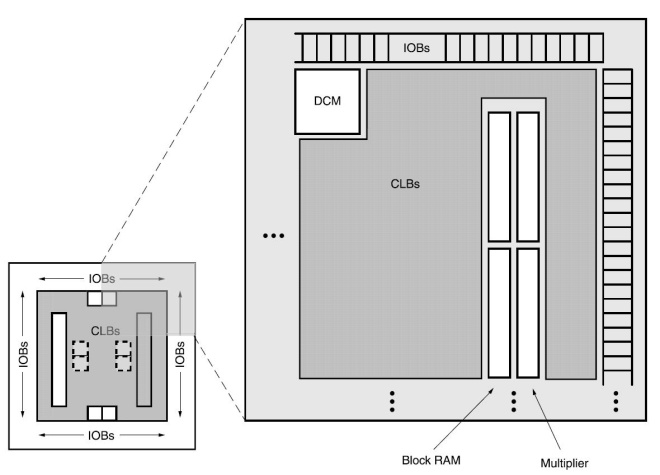
Семейство Virtex-7



Используются высокоскоростные приемопередатчики RocketIO (MGT) типа GTX+GTH+GTZ, поддерживающие скорость приема и передачи данных до 10,3125 (очень много) Гбит/с.

Содержат аппаратный модуль (блок) интерфейса PCI Express, который поддерживает (PCI Express Gen 1, 2 и 3) скорости передачи данных 2,5 и 5 Гбит/с.

Организация работы со статической памятью в ПЛИС



Внутри ПЛИС Xilinx есть специальный аппаратный ресурс – блочная память или BRAM (Block RAM).

Эти блоки выполнены аппаратно, т.е. часть логики кристалла зафиксирована под память и не может быть использована по другому назначению. Количество таких блоков зависит от конкретного типа кристалла ПЛИС и может быть составлять несколько сотен. Каждый блок имеет два аппаратных порта, которые могут работать на разных тактовых частотах.

Другой ресурс, который можно использовать как память, – это сами конфигурационные блоки ПЛИС. Из конфигурационных блоков можно собрать логическую схему статической памяти. Тогда биты информации, хранящиеся в такой памяти, будут «размазаны» или «распределены» по большей (по сравнению с BRAM, организованными локально) площади кристалла, и такая память будет называться распределенной памятью (Distributed RAM).

Сравнивая блочную и распределенную память, надо сказать следующее.

Блочная память – это аппаратные блоки.

Блочная память – дорогостоящий ресурс. Количество ее ограничено. Если требуется объем памяти меньший, чем объем одного блока (18 или 36 Кбит), то все равно будет использован весь блок, а неиспользованная его часть просто пропадет. При этом распределенная память может быть создана конкретного объема: ни один лишний конфигурационный блок не будет задействован.

При использовании блочной памяти ее разные порты могут работать на разных частотах. Это удобно использовать для синхронизации между разными блоками, работающими на разной тактовой частоте.

Схемы как блочной, так и распределенной памяти, создаются с использованием программы генератора процессорного ядра. В ней можно выбрать необходимый тип памяти, объем, ширины шин данных, особенности интерфейса. После создания IP блока работать с памятью надо через интерфейс, подобный рассмотренному на следующем слайде.

В VHDL есть возможность описать массив ячеек статической памяти с помощью сигналов типа «массив» и стандартных операторов.

entity static\_ram

PORT (

clk : IN STD\_LOGIC; -- тактовый сигнал

rst : IN STD\_LOGIC; -- сигнал сброса

en : IN STD\_LOGIC; -- сигнал разрешения работы (опционально)

we : IN STD\_LOGIC; -- write enable, сигнал определяющий операцию

записи или чтения

addr : IN STD\_LOGIC\_VECTOR(3 DOWNTO 0); -- шина адреса ячейки

din : IN STD\_LOGIC\_VECTOR(15 DOWNTO 0); -- входная шина данных

dout : OUT STD\_LOGIC\_VECTOR(15 DOWNTO 0) -- выходная шина данных

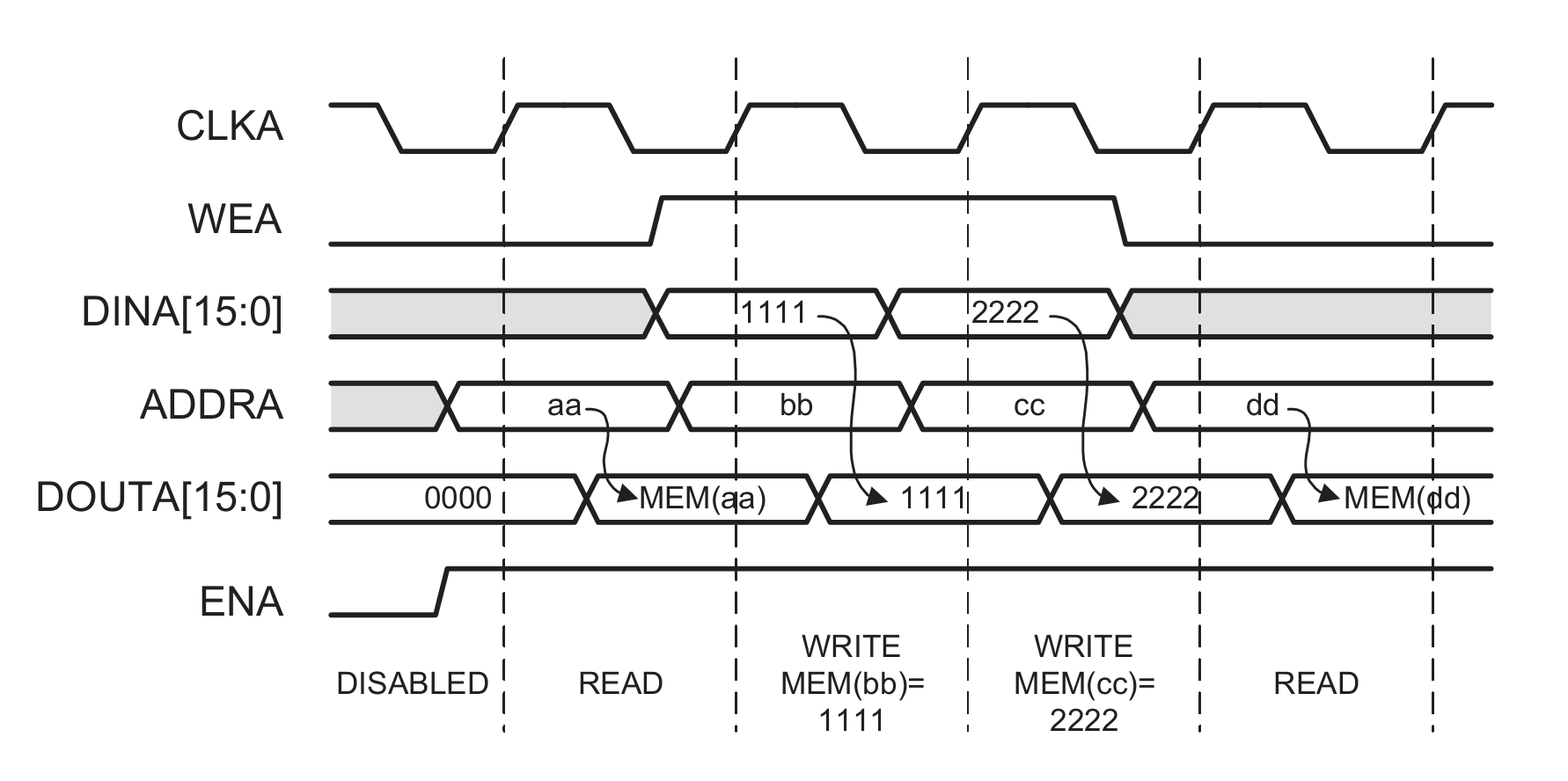
);

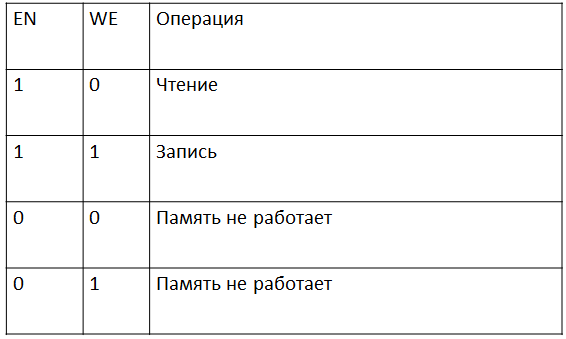
END static\_ram;

|  |  |
| --- | --- |
|  | Блок памяти имеет следующие порты:  – addra, addrb – адреса портов A и B соответственно;  – dia, dib – данные для записи для портов A и B (32 бита);  – dipa, dipb – дополнительные данные для записи (4 бита);  – wea, web – входы разрешения записи (побайтно);  – clka, clkb – тактовые сигналы для портов;  – ena, enb – входы разрешения работы блока памяти (при чтении состояние выходов не обновляется, если нет разрешающего сигнала);  – regcea, regceb – разрешение работы выходных регистров;  – rsta, rstb – сброс выходных регистров (не влияет на содержимое массивов памяти);  – doa, dob – выходы данных для портов A и B (32 бита);  – dopa, dopb – дополнительные выходы данных для портов A и B (4 бита). |

Физически размещенные в FPGA блоки памяти являются 18-битными. Такая разрядность позволяет реализовывать схемы контроля четности, когда каждые 8 бит имеют дополнительный 9-й бит для хранения бита четности. Соответственно, каждые 16 бит имеют 2 дополнительных бита четности, а 32 – 4 бита. Для удобства работы с дополнительными битами в графическом представлении модуля они выделены в отдельные шины dipa, dipb, dopa, dopb.

Дополнительные биты не являются автоматически заполняемыми и представляют собой разряды, доступные для записи в них произвольных значений. Разработчик может выбирать требуемую ему разрядность, включая 1, 2, 4, 9, 18 или 36 бит, в особых случаях - до 72 бит.





Организация работы со статической памятью в ПЛИС: двухпортовая память

library IEEE;

use IEEE.std\_logic\_1164.all;

use IEEE.std\_logic\_unsigned.all;

entity rams\_16 is

port(clka : in std\_logic;

clkb : in std\_logic;

ena : in std\_logic;

enb : in std\_logic;

wea : in std\_logic;

web : in std\_logic;

addra : in std\_logic\_vector(5 downto 0);

addrb : in std\_logic\_vector(5 downto 0);

dia : in std\_logic\_vector(15 downto 0);

dib : in std\_logic\_vector(15 downto 0);

doa : out std\_logic\_vector(15 downto 0);

dob : out std\_logic\_vector(15 downto 0));

end rams\_16;

architecture syn of rams\_16 is

type ram\_type is array (63 downto 0) of std\_logic\_vector(15 downto 0);

shared variable RAM : ram\_type;

begin

process (CLKA)

begin

if CLK 'event and CLK = ‘1‘ then

if ENA = ’1’ then

if WEA = ’1’ then

RAM(conv\_integer(ADDRA)) := DIA;

end if;

DOA <= RAM(conv\_integer(ADDRA));

end if;

end if;

end process;

process (CLKB)

begin

if CLKB 'event and CLKB = ‘1‘ then

if ENB = ’1’ then

if WEB = ’1’ then

RAM(conv\_integer(ADDRB)) := DIB;

end if;

DOB <= RAM(conv\_integer(ADDRB));

end if;

end if;

end process;

end syn;

# Память и IP ядра

IP cores (IP-ядра, от англ. Intellectual Property — интеллектуальный продукт.

В FPGA (ПЛИС) под Hard IP-Core понимают специализированные области кристалла, выделенные для определенных функций. В этих областях реализованы блоки неизменной структуры, спроектированные по методологии ASIC (как области типа БМК или схем со стандартными ячейками), оптимизированные для заданной функции и не имеющие средств её программирования.

В случае использования данного вида ядер размер площади, используемой на кристалле, сокращается, улучшаются характеристики быстродействия, но в данном случае происходит потеря универсальности.

Приложение CoreGenerator входит в состав САПР фирмы Xilinx и предлагает разработчику графический интерфейс быстрого конфигурирования стандартных цифровых блоков для реализации их в ПЛИС (IP-ядер). С помощью CoreGenerator имеется возможность создания шаблонов на языке Verilog, которые потом будут использованы как instance (см. componentinstantiation) в проектах пользователя. Генерация компонентов с помощью CoreGenerator обладает рядом преимуществ:

- IP-ядра, как правило, представляют собой верифицированные компоненты, имеющие качественное размещение на кристалле и вследствие этого способные работать на высокой тактовой частоте;

– ряд специфичных ресурсов ПЛИС, например модули формирования тактовой частоты, достаточно сложны для освоения, однако вносят в проект полезные функциональные возможности, поэтому рекомендуются к использованию; соответственно, CoreGenerator предоставляет возможности для их быстрого конфигурирования с последующим добавлением к проекту.

Запускаем генератор и выбираем тип IP ядра

|  |  |
| --- | --- |
|  |  |

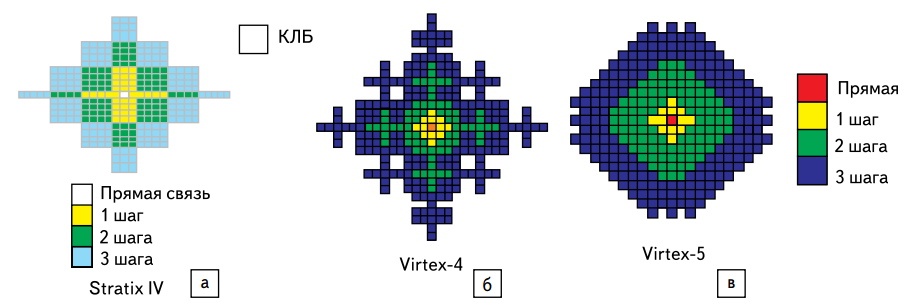
Выбираем параметры памяти, в частности, ширину (разрядность) данных (**Data Width**), количество данных (**Depth – глубина**), тип создаваемой памяти (**Memory Type**)

|  |  |
| --- | --- |
|  |  |

Создаем и подключаем файл coe-файл коэффициентов, после этого формируется память по заданным параметрам

|  |  |
| --- | --- |
|  |  |

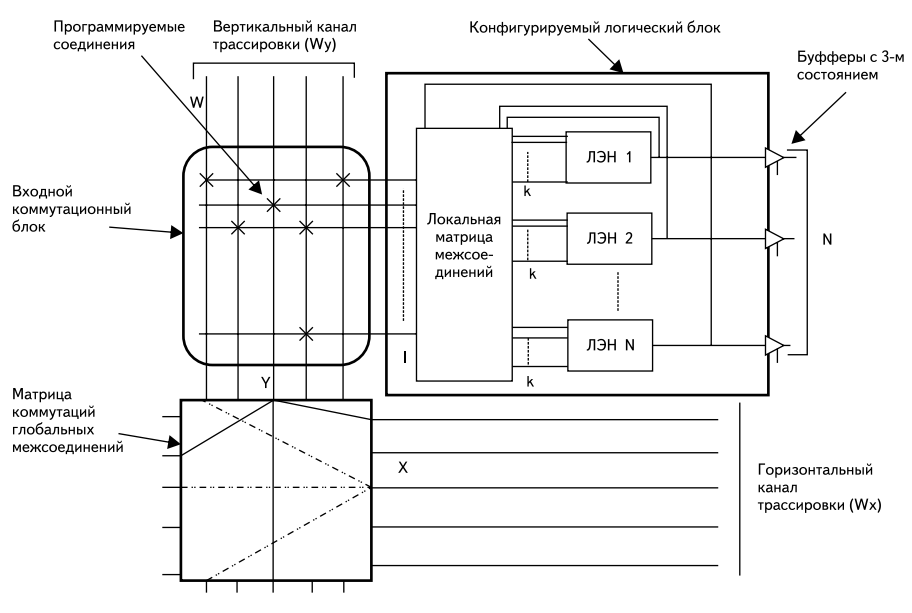
Коммутационная система ПЛИС



Трассировочные ресурсы ПЛИС:

а) Stratix IV фирмы Altera с использованием технологии DirectDrive;

б), в) фирмы Xilinx с использованием технологии ExpressFabric (б - Virtex-4; в - Virtex-5).



Пример организации взаимодействия между логическими блоками

Блоки ввода-вывода

Каждый доступный разработчику ввод-вывод (pin) может быть сконфигурирован как вход, выход или универсальный вход-выход.

Зачастую возможен выбор рабочих напряжений, для разработчика - чем меньше напряжение, тем на большей частоте можно работать, но тем меньше помехозащищенность.

|  |  |
| --- | --- |
|  |  |

# Конечные автоматы

Схема с k-регистрами может находиться в одном из 2k состояний, то есть в конечном числе состояний. У конечных автоматов (КА) М входов, N выходов и k бит состояний.

На вход КА также подаются тактовый сигнал и, возможно, сигнал сброса.

КА состоит из двух блоков комбинационной логики: логики перехода в следующее состояние и выходной логики, и из регистра, в котором хранится текущее состояние.

По фронту каждого тактового импульса автомат переходит в следующее состояние, которое определяется текущим состоянием и значениями на входах.

На основе цифровых автоматов, как правило, реализуются устройства управлением дискретными процессами.

Существует два основных класса конечных автоматов, которые отличаются своими функциональными описаниями.

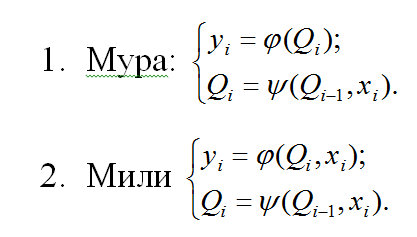
В автомате Мура (Moore) выходные значения зависят лишь от текущего состояния, в то время как в автомате Мили (Mealy) выход зависит как от текущего состояния, так и от входных данных.

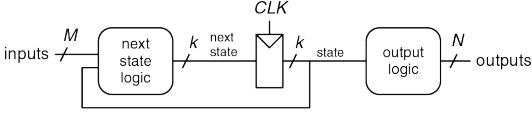
Конечные автоматы предоставляют систематический способ проектирования синхронных последовательностных схем по заданному функциональному описанию. Логические выражения для цифрового автомата разделяются на 2 составляющих.

I часть - выходная функция, зависящая от текущего входного сигнала и внутреннего состояния цифрового автомата в данный момент времени.

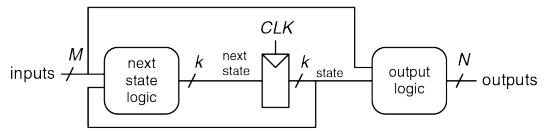
II часть - функция изменения внутреннего состояния, которая определяется предыдущим внутренним состоянием и текущим входным сигналом.

Для разных классов автоматов они записываются так:





КА Мура (Moore)

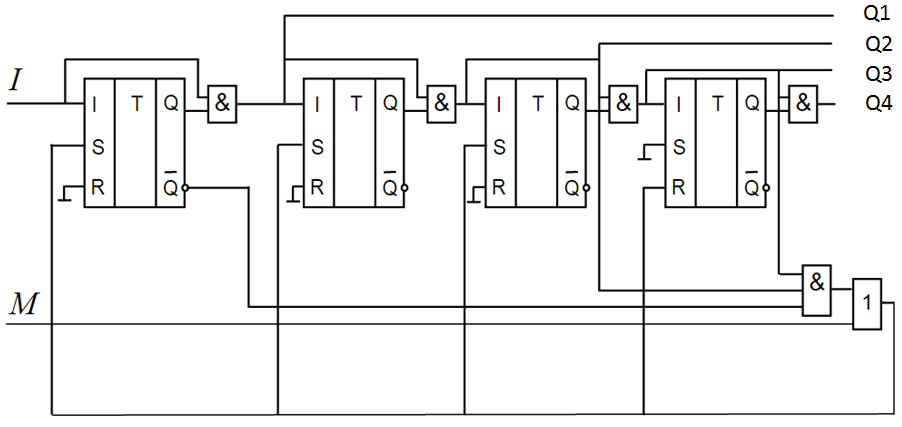


КА Мили (Mealy)

Cчетчик с пропуском состояний и внешней установкой в «0111»

Представляет собой конечный автомат, последующее состояние которого зависит от текущего состояния и входных воздействий (по сигналу *M*=1 схема переходит в состояние 1110).

Выходные значения полностью определяются текущим состоянием.



Синтез конечного автомата с помощью граф-схемы алгоритма (ГСА)

Граф-схема алгоритма является наглядным описанием работы конечного автомата. Для автомата Мура можно изобразить схему и граф.

|  |  |
| --- | --- |
|  |  |

Граф-схема алгоритма является наглядным описанием работы конечного автомата. Для автомата Мили можно изобразить ГСА и граф.

|  |  |
| --- | --- |
|  |  |

## Последовательность проектирования: формализация описания, построение графа состояний

|  |  |
| --- | --- |
|  |  |

При составлении ГСА следует придерживаться следующих двух **правил**:

1. Каждой входной комбинации должен соответствовать единственный выходной путь *из* данного узла ГСА. Допускается использование одного выходного пути для нескольких входных комбинаций, но не наоборот.
2. Выходной путь из узла ГСА обязательно должен вести к блоку состояния (либо другого, либо этого же узла ГСА).

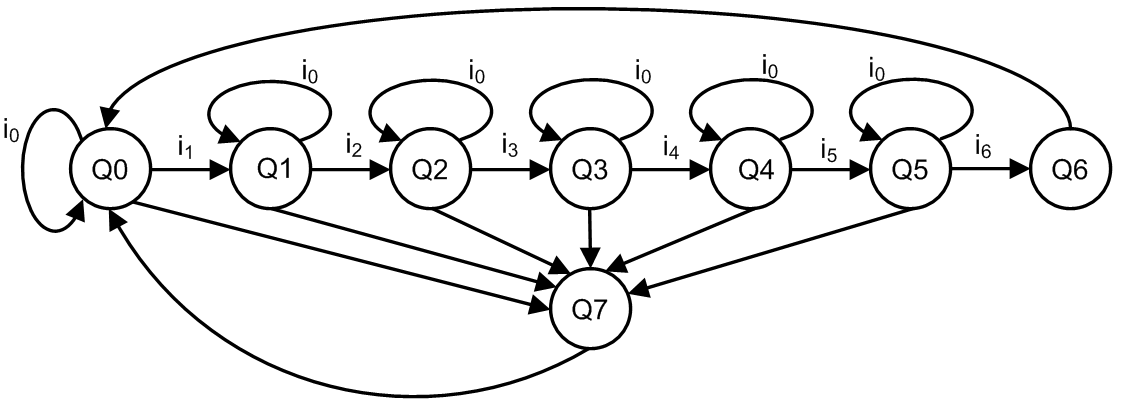
Рассмотрим синтез конечного автомата, отпирающего дверь при последовательном наборе «1234567» с произвольным числом «0» в коде и сигнализирующего, что при наборе произошла ошибка.

Выходные сигналы автомата «Открывание», «Ошибка». Определяем тип автомата. В данном случае выходные сигналы могут полностью определяться состоянием автомата, следовательно, речь идет об автомате Мура.

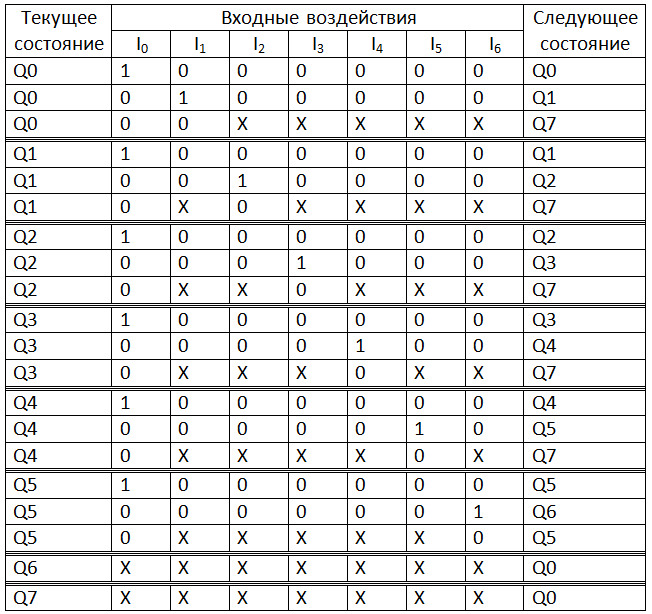
Состояния автомата:

1. Q0- ожидание нажатия
2. Q1- правильно введено «1»
3. Q2- правильно введено «2»
4. Q3- правильно введено «3»
5. Q4- правильно введено «4»
6. Q5- правильно введено «5»
7. Q6- правильно введено «6», сигнал «Открывание», переход в Q0
8. Q7- неправильный ввод, «Ошибка», переход в Q0

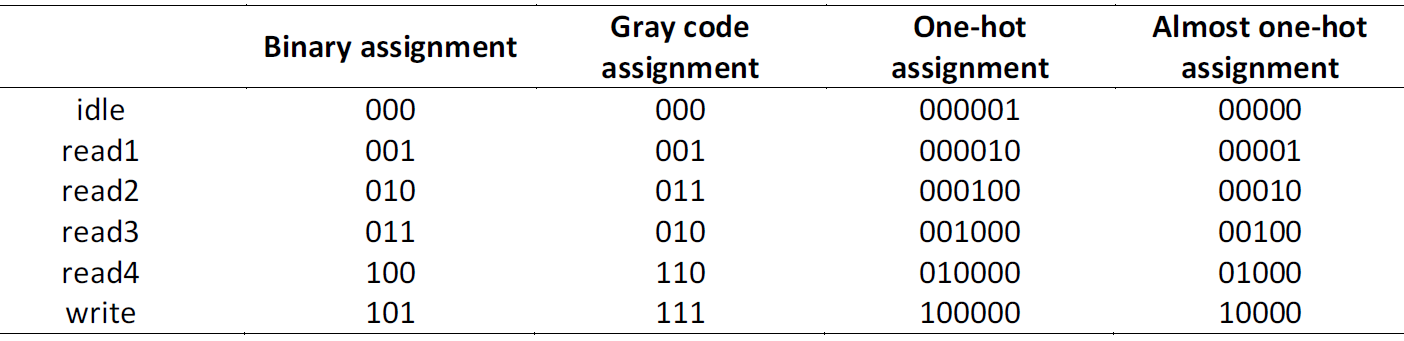
Текстовое описание, переведенное в граф



От графа к таблице



Способы кодирования состояний устройства



Кодирование «в одну единицу» приведет к тому, что КА станет сдвиговым регистром с дополнительными логическими цепями - просто и очень затратно по ресурсам.

Бинарное кодирование приведет к усложнению (в общем случае) конечного автомата. В рассмотренном примере применялось бинарное кодирование: перейти к более простому несложно.

Конечный автомат можно реализовать на D или RS триггерах.

При кодировании необходимо выполнить следующее:

1.   Каждому состоянию автомата аm (m = 1,2,...,M) ставится в соответствие целое число Nm, равное числу переходов в состояние аm (Nm равно числу появлений аm в поле таблицы переходов или числу дуг, входящих в аm при графическом способе задания автомата).

2.   Числа N1, N2, ..., Nm упорядочиваются по убыванию.

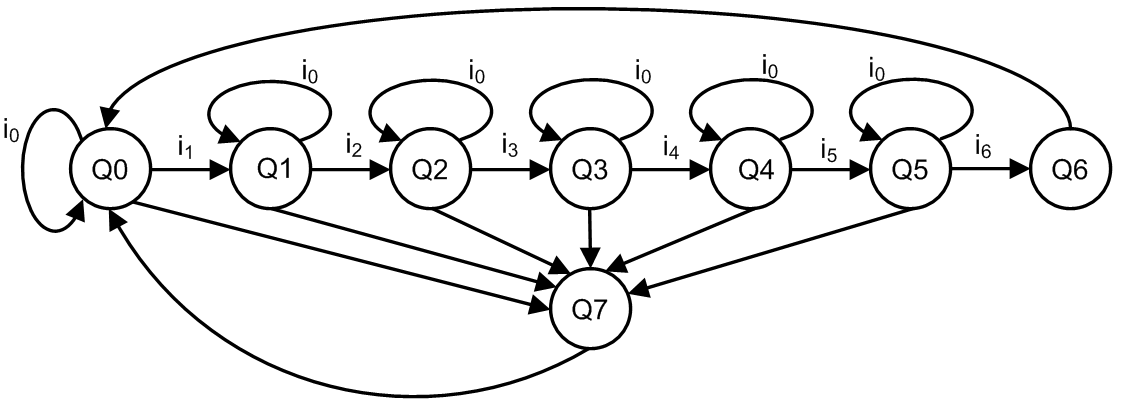
3.   Состояние аs с наибольшим Ns кодируется кодом, состоящим из R нулей, где R - количество элементов памяти.

4.   Следующие R состояний согласно списку пункта 2 кодируются кодами, содержащими только одну 1: 00 ... 01, 00 ... 10, ... , 01 ... 00, 10 ... 00.

5.   Для оставшихся состояний опять в порядке списка п.2 используют коды с двумя единицами, затем с тремя и так далее, пока не будут закодированы все состояния.

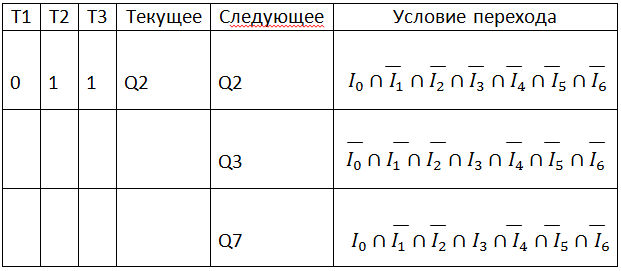
В результате получается такое кодирование, при котором чем больше имеется переходов в некоторое состояние, тем меньше единиц в его коде. Т.к. для D-триггеров функции возбуждения однозначно определяются кодом состояния перехода, то очевидно, что выражения для функций возбуждения будут проще. Этот метод особенно эффективен при отсутствии минимизации функций возбуждения, что имеет место в реальных автоматах с большим количеством внутренних состояний и входных переменных.

Пример:





Теперь можно записать таблицу переходов для различных состояний. Например, из состояния Q2 автомат может перейти только в состояния Q3, Q7 или остаться в состоянии Q2. В результате формируются логические функции, описывающие условия переходов между состояниями.



A = {a0, a1}, где a0 – логическая 1 на входе R, a1 – логическая единица на входе S.

B = {b0, b1}, где b0 – логический 0 на выходе Q, b1 – логическая единица на выходе Q.

Граф автомата Мили

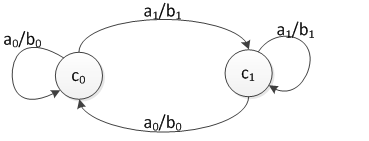
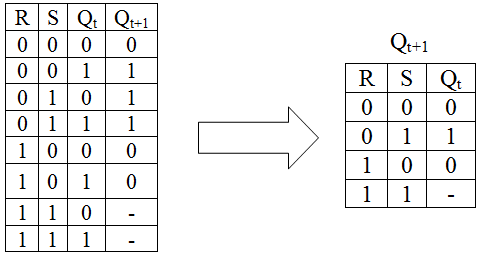


Таблица состояний и переходов

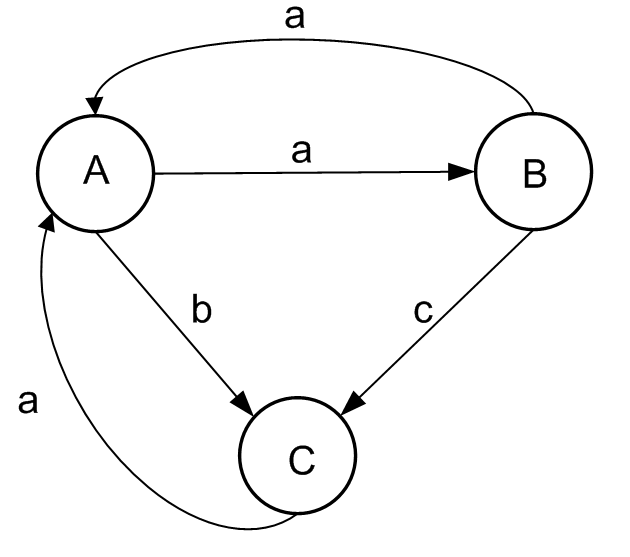


После минимизации остается .

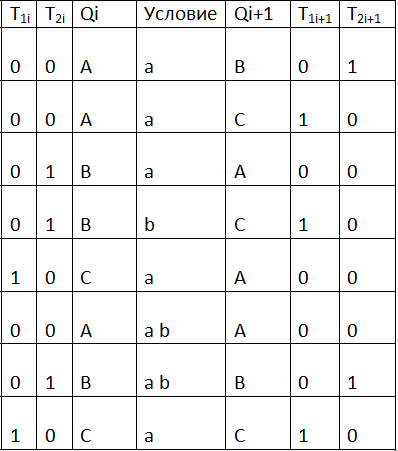
В базисе И-Не: .

## Полный цикл синтеза автомата на D- триггерах

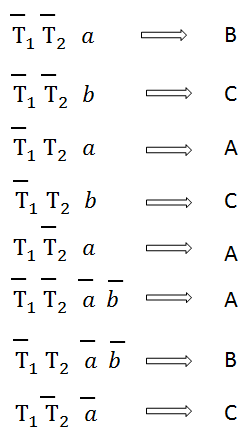
Формулировка задачи. Конечный автомат задан графом, разработать кодировку состояний и логику переходов



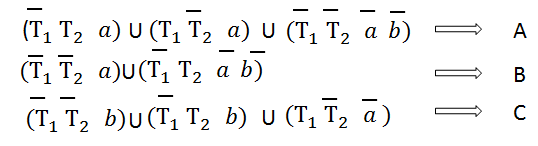
Автомат имеет три состояния, следовательно, может кодироваться двумя триггерами Т1 и Т2.



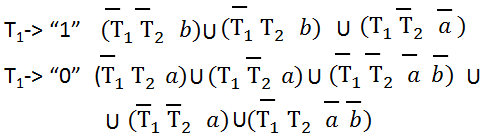
Из таблицы кодировки построим условия переходов



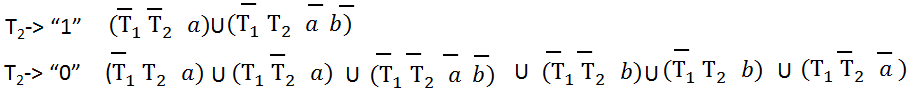
Условия можно записать в виде



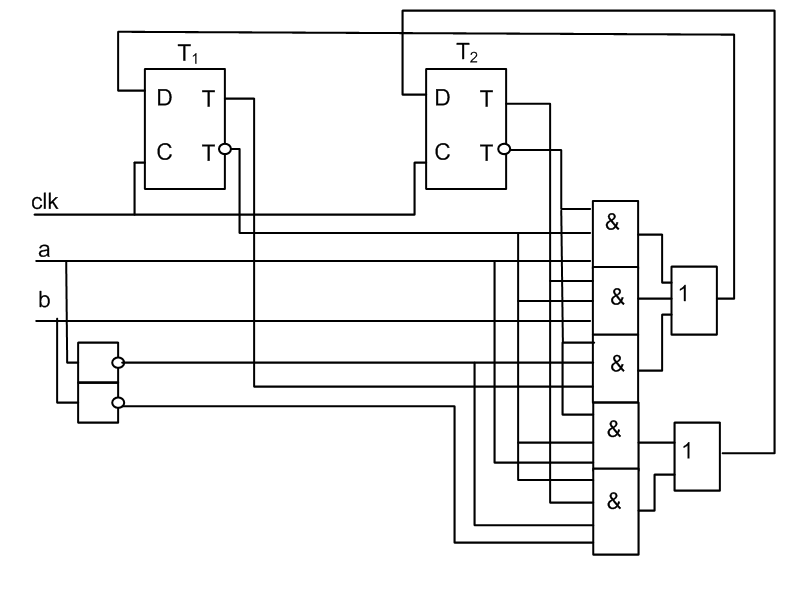
«привязав» их к состояниям. Для триггера Т1 условия перехода в состояния   
(с учетом того, что он принимает состояние «0» в А и В и «1» в «С», будут:



Аналогично для триггера Т2 условия перехода в состояния «0» определяются условиями перехода в состояния А и С, а в «1»- состоянием «B».



Полученные функции при подаче на входы триггеров обеспечивают работу автомата в соответствии с заданным графом. При реализации схемы на D- триггерах на входы D подаются конституенты «1», то есть разработанные нами сигналы Т1-> “1” и Т2-> “1”.



# Арифметические операции

Однобитный полусумматор выдает арифметическую сумму двух однобитных данных:

|  |  |  |
| --- | --- | --- |
|  |  |  |

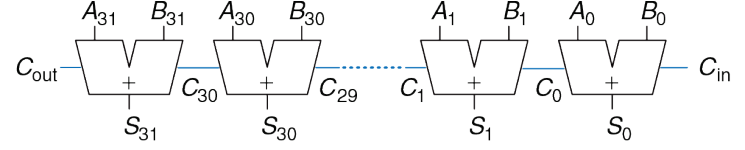
Однобитный полный сумматор выдает арифметическую сумму двух однобитных данных с переносом от младшего разряда:

|  |  |  |
| --- | --- | --- |
|  |  |  |

## Сумматор с последовательным переносом

Строится из однобитных полных сумматоров (в первом разряде - однобитный полусумматор). Обычно в электронных схемах сигналы распространяются слева направо. Арифметические схемы нарушают эти правила, так как перенос идет справа налево (от младшего разряда к старшему).

Приведенная схема называется сумматором с последовательным переносом (ripple-carry adder). При ее проектировании используется принцип модульности и регулярности: модуль полного сумматора многократно используется для формирования большей системы. Такой сумматор имеет недостаток: его скорость падает при увеличении числа. Перенос проходит через всю цепь. Задержка такого сумматора (triple) увеличивается вместе с количеством разрядов



## Сумматоры с ускоренным переносом

Сумматоры с ускоренным переносом (carry-lookahead adder, CLA) – это другой тип сумматоров с распространяющимся переносом, который решает эту проблему, разделяя сумматоры на блоки и реализуя схему так, чтобы определить выходной перенос блока, как только стал известен его входной перенос. Таким образом, мы смотрим вперед через блоки и не ждем прохождения переноса через все полные сумматоры внутри блока. К примеру, 32-разрядный сумматор может быть разделен на 8 четырехразрядных сумматоров.

Сумматоры с ускоренным переносом используют сигналы генерации (G) и распространения (P), которые описывают, как блок (или разряд) определяет выход переноса. i-й разряд сумматора генерирует перенос, если он выдает перенос на своем выходе, независимо от наличия переноса на входе. i-й разряд сумматора генерирует Ci в том случае, если и Ai, и Bi равны 1. Таким образом, сигнал генерации Gi можно вычислить как Gi=AiBi. Разряд называется распространяющим, если выходной сигнал переноса появляется при наличии входного переноса. Разряд будет распространять входной сигнал переноса, Ci-1 , если либо Ai, либо Bi равны 1. Таким образом, Pi = Ai + Bi.

Разряд i сумматора будет формировать выходной сигнал переноса Ci, если он или генерирует перенос Gi, или распространяет входной перенос Pi Ci-1. В виде уравнения это можно записать следующим образом:

Ci = Ai Bi + (Ai + Bi) Ci–1 = Gi + Pi Ci–1.

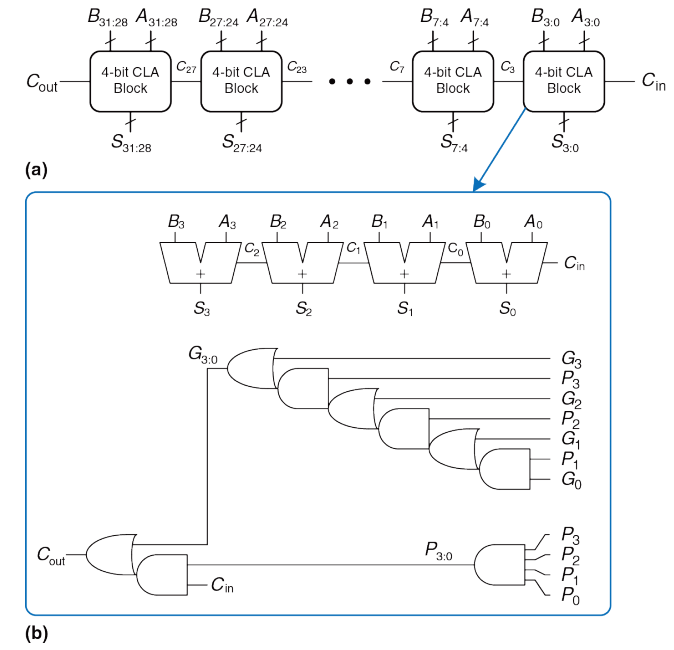
Блок генерирует перенос, если самый старший разряд генерирует перенос или если старший разряд распространяет перенос, сгенерированный предыдущим разрядом, и т. д. Например, логика блока генерации для блока, охватывающего разряды от 0 до 3, будет следующей:

G3:0 = G3 + P3 (G2 + P2 (G1 + P1 G0)).

На VHDL: G30<=G3 or P3 and (G2 or P2 and (G1 or (P1 and G0)));

## 32 разрядный сумматор с ускоренным переносом и его четырехразрядный блок.

Обозначение сигналов то же, что и ранее. На четырех разрядах критический путь составляет семь элементов. При разрядности больше 16 такой сумматор работает быстрее последовательного, но задержка линейно растет с ростом разрядности.



## Сумматоры с ускоренным переносом: префиксный сумматор

Префиксный сумматор развивает логику генерации и распространения сумматора с ускоренным переносом для еще более быстрого выполнения операции сложения. Сначала он вычисляет G и P для пар разрядов, далее для блоков из 4-х разрядов, затем для блоков из 8-ми, 16-ти и т. д. разрядов, пока сигнал генерации не будет известен для каждого разряда. Сумма определяется всеми сигналами генерации.

Стратегия префиксного сумматора заключается в вычислении входного сигнала переноса Ci-1 для каждого разряда так быстро, насколько это возможно. Затем вычисляется сумма:

Si = (Ai ⊕ Bi) ⊕ Ci-1

Определим разряд i =-1 для вычисления Cin: G-1 = Cin и P-1 =0. Следовательно, Ci-1 = Gi-1:-1, так как выходной сигнал переноса i-1-го разряда будет активным, если блок, охватывающий разряды от i-1 до -1, генерирует перенос. Полученный перенос генерируется или в разряде i-1, или в предыдущем разряде и затем распространяется дальше. Таким образом, получим:

Si = (Ai ⊕ Bi) ⊕ Gi-1:-1.

Теперь необходимо быстро вычислить все блоковые сигналы генерации G-1:-1, G0:-1, G1:-1, G2:-1, . . . , GN-2:-1. Эти сигналы вместе с P-1:-1, P0:-1, P1:-1, P2:-1, . . . , PN-2:-1 называют префиксными.

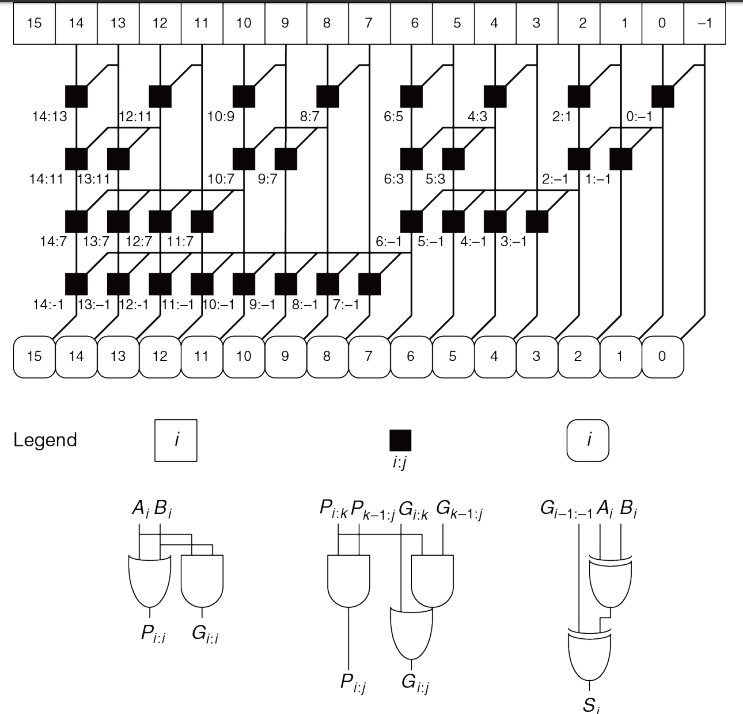
Работа префиксного сумматора начинается с предварительного формирования сигналов Pi и Gi для всех разрядов Ai и Bi с использованием элементов AND и OR. Затем используется log2N = 4 уровня ячеек для формирования префиксов Gi:j и Pi:j. Эти ячейки-формирователи принимают входы из верхней части блока, охватывающего биты i:k, и из нижней части блока, охватывающего биты k-1:j. Затем эти части объединяются для формирования сигналов генерации и распространения всего блока, охватывающего биты i:j.

Gi:j = Gi:k + Pi:k Gk-1:j.

Pi:j = Pi:k Pk-1:j.

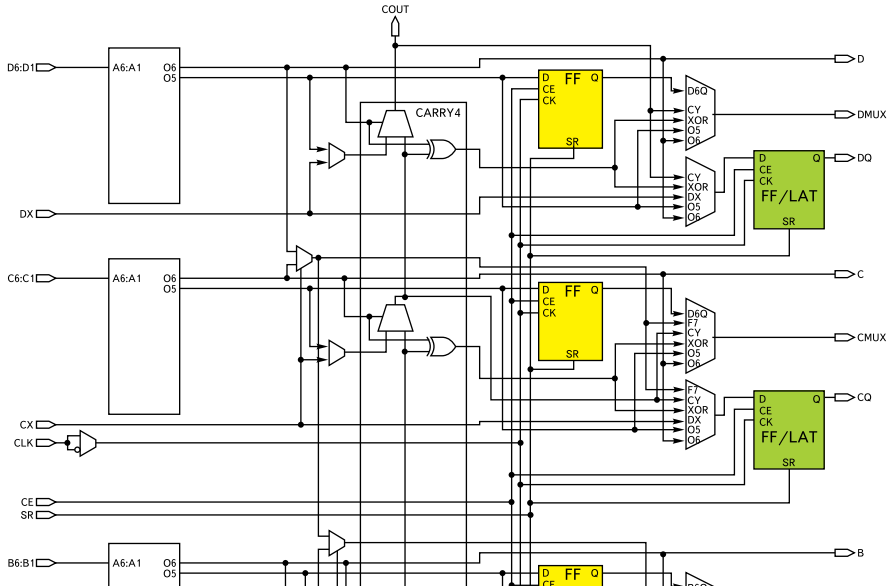
Блок, охватывающий биты i:j, будет генерировать сигнал переноса, если верхняя часть генерирует перенос или если она распространяет перенос, сгенерированный в нижней части. Блок будет распространять перенос, если и верхняя, и нижняя части распространяют его.

16 разрядный префиксный сумматор, длина критического пути растет по log2N от разрядности



В современных FPGA в составе ячейки имеются узлы, повышающие ее эффективность при решении типовых задач цифровой схемотехники. Например, передача бита переноса (carry) в соседний элемент встречается достаточно часто и определяет быстродействие таких элементов, как счетчики и сумматоры.

Выделение для этого сигнала отдельной линии связи (так называемой линии ускоренного переноса) сопряжено с незначительным усложнением ячейки, но существенно повышает рабочую частоту этих стандартных узлов. Поэтому все FPGA имеют в составе ячейки линии ускоренного переноса.



## Описание сумматора на VHDL

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD\_UNSIGNED.ALL;

entity adder is

generic(N: integer := 8);

port(a, b: in STD\_LOGIC\_VECTOR(N–1 downto 0);

cin: in STD\_LOGIC;

s: out STD\_LOGIC\_VECTOR(N–1 downto 0);

cout: out STD\_LOGIC);

end;

architecture synth of adder is

signal result: STD\_LOGIC\_VECTOR(N downto 0);

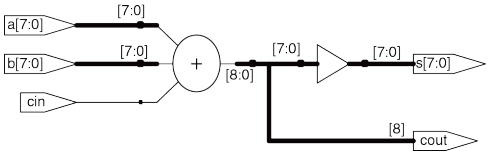
begin

result <= (“0” & a) + (“0” & b) + cin;

s <= result(N–1 downto 0);

cout <= result(N);

end;



## Знак двоичных чисел

Часто для вычислений требуются как положительные, так и отрицательные числа, а это значит, что для знака двоичного числа нам потребуется дополнительный разряд.

Существует несколько способов представления двоичных чисел со знаком (signed). Наиболее широко применяются два: Прямой Код (Sign/Magnitude) и Дополнительный Код (Two’s Complement).

Прямой код

Представление отрицательных двоичных чисел с использованием прямого кода интуитивно покажется вам наиболее привлекательным, поскольку совпадает с привычным способом записи отрицательных чисел, когда сначала идет знак минус, а затем абсолютное значение числа.

Двоичное число, состоящее из N битов и записанное в прямом коде, использует наиболее значимый бит для знака, а остальные N−1 бита - для записи абсолютного значения этого числа. Если наиболее значимый бит «0», то число положительное. Если наиболее значимый бит «1», то число отрицательное.

Ограничение прямого кода: для разрядности N число окажется в диапазоне

[−2N−1 + 1, 2N −1 − 1].

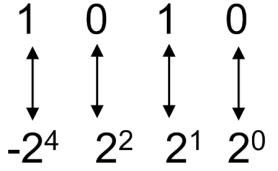
Запишем числа 5 и –5 как четырехбитовые числа в прямом коде.

Решение: Оба числа имеют абсолютную величину 510 = 1012. Таким образом, 510=01012 и –510 =11012.

Проблема прямого кода: попробуем сложить в столбик 5 и -5. Получим 100102, то есть заведомо неверный ответ. Кроме того, возможны ошибки из-за того, что 0 может быть записан двумя разными способами.

Дополнительный код

Двоичные числа, записанные с использованием дополнительного кода, и двоичные числа без знака идентичны, за исключением того, что в случае дополнительного кода вес наиболее значимого бита −2N−1 вместо 2N−1, как в случае двоичного числа без знака.



Правило перекодирования: если число A>0, то [A]доп=[A]пр, если число А<0, то в знаковый разряд кода записывается 1, числовые разряды исходного числа инвертируются, и к младшему числовому разряду добавляется 1.

Если на вход схемы преобразователя четырехразрядного прямого кода в дополнительный код подать дополнительный код, то на выходе получим прямой код.

Дополнительный код гарантирует однозначное представление нуля, допускает сложение чисел по привычной схеме, а значит, избавлен от недостатков прямого кода. В случае дополнительного кода нулевое значение представлено нулями во всех разрядах двоичного числа: 00…0002. Максимальное положительное значение представлено нулем в наиболее значимом разряде и единицами во всех других разрядах двоичного числа: 01…1112 = 2N−1 − 1. Максимальное отрицательное значение имеет единицу в наиболее значимом разряде и нули во всех остальных разрядах: 10…0002 = −2N−1 . Отрицательная единица представлена единицами во всех разрядах двоичного числа: 11…1112.

Пример перевода. Найти представление −210 как 4-битового числа в дополнительном коде. Решение: +210 = 00102. Для получения −210 инвертируем биты и добавим единицу. Инвертируя 00102, получим 11012. 11012 + 1 = 11102. Итак −210 равно 11102.



Дополнительный код

Найдем представление −210 как 4-битового числа в дополнительном коде. +210=00102. Для получения −210 инвертируем биты и добавим единицу. Инвертируя 00102, получим 11012. 11012 + 1 = 11102. Итак, −210 равно 11102.

При использовании дополнительного кода привычный способ сложения работает как в случае положительных, так и отрицательных чисел. Напомним однако, что при сложении N-битных чисел N-й бит (т.е. N + 1-й бит результата) не переносится.

Сложение чисел в двоичном коде – пример

Вычислим (a) −210 + 110 и (b) −710 + 710 с помощью чисел в дополнительном коде: (a) −210 + 110 = 11102 + 00012 = 11112 = −110. (b) −710 + 710 = =10012 + 01112 = 100002. Пятый бит отбрасывается, оставляя правильный 4-битовый результат 00002

Дополнительный код: вычитание чисел

Пусть необходимо вычислить

1. 510 − 310 и (b) 310 − 510

используя 4-разрядные числа в дополнительном коде.

1. 310 = 00112. Вычисляя его дополнительный код, получим −310 = 11012. Теперь сложим 510 + (−310) = 01012 + 11012 = 00102 = 210. Отметим, что перенос из наиболее значимой позиции сбрасывается, поскольку результат записывается в четырех битах.
2. Вычисляя дополнительный код от 510, получим −510 = 10112. Теперь сложим 310 + (−510) = 00112 + 10112 = 11102 = −210.

Таким образом, задачи сложения и вычитания эффективно решаются в дополнительном коде.

|  |  |
| --- | --- |
| Реализация вычитания в VHDL  library IEEE; use IEEE.STD\_LOGIC\_1164.ALL;  use IEEE.NUMERIC\_STD\_UNSIGNED.ALL;  entity subtractor is  generic(N: integer := 8);  port(a, b: in STD\_LOGIC\_VECTOR  (N–1 downto 0);  y: out STD\_LOGIC\_VECTOR(N–1 downto 0));  end;  architecture synth of subtractor is  begin  y <= a – b;  end; |  |

|  |  |
| --- | --- |
| Перевод в дополнительный код | Сложение / Вычитание |

## Сравнение. Компаратор

Компараторы определяют, являются ли два двоичных числа равными или одно из них больше/меньше другого. Для примера рассмотрим два 4-х битных слова А и В, представленных в двоичном виде, причем 3-й разряд будет старшим:

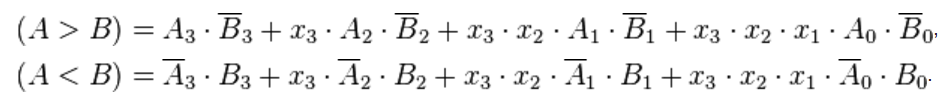
.

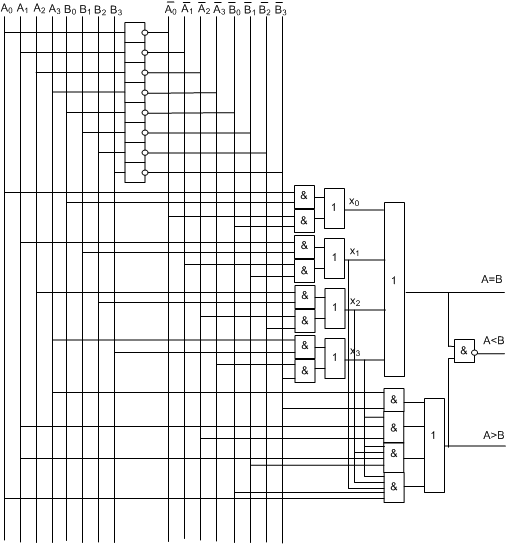
Двоичные числа А и В будут равны, если все пары соответственных битов обоих чисел равны, то есть:



Запишем В таком случае условие равенства \ (A=B) = x_3 \cdot x_2 \cdot x_1 \cdot x_0.

Условия неравенства:





Элемент сравнения без функции вычитания таблица истинности и схема:

|  |  |
| --- | --- |
|  | Однобитный компаратор неравенства |

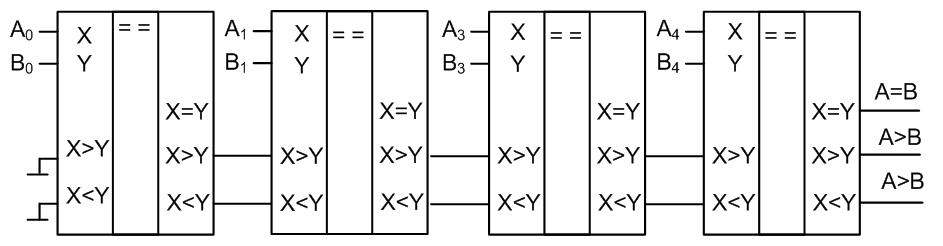
Многоразрядные компараторы обычно выполняют на базе одноразрядных. При этом используется принцип последовательного сравнения разрядов многоразрядных чисел, начиная с их старших разрядов, так как уже на этом этапе, если x1m > x2m, задача решена однозначно, и сравнение следующих за старшими разрядов не потребуется.

Компаратор с входами переноса

У такого компаратора кроме входов соответствующего разряда есть входы для сообщения о результатах сравнения предыдущего разряда. Такие схемы можно объединять последовательно, получая любую нужную разрядность.

|  |  |  |
| --- | --- | --- |
|  |  |  |

Последовательное объединение компараторов. Увеличиваем разрядность до 4.



Реализация сравнения в VHDL

library IEEE; use IEEE.STD\_LOGIC\_1164.ALL;

entity comparators is

generic(N: integer : = 8);

port(a, b: in STD\_LOGIC\_VECTOR(N–1 downto 0);

eq, neq, lt, lte, gt, gte: out STD\_LOGIC);

end;

architecture synth of comparator is

begin

eq <= ‘1’ when (a = b) else ‘0’;

neq <= ‘1’ when (a /= b) else ‘0’;

lt <= ‘1’ when (a < b) else ‘0’;

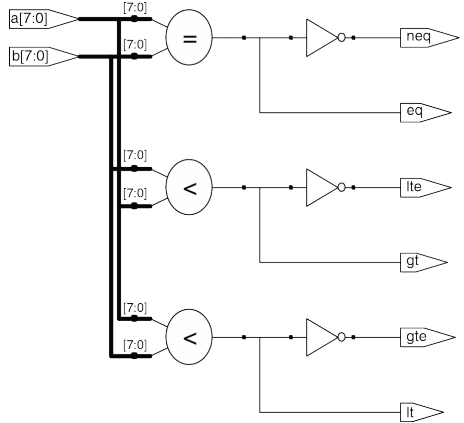
lte <= ‘1’ when (a <= b) else ‘0’;

gt <= ‘1’ when (a > b) else ‘0’;

gte <= ‘1’ when (a >= b) else ‘0’;

end;

На рисунке приведена схема, формируемая САПР из VHDL описания до синтеза и оптимизации.



Операции сдвига и циклического сдвига

Эти операции перемещают биты и, следовательно, умножают или делят число на степень 2. В соответствии с названием схемы сдвига передвигают разряды двоичного числа влево или вправо на определенное число позиций. Существует несколько видов таких схем.

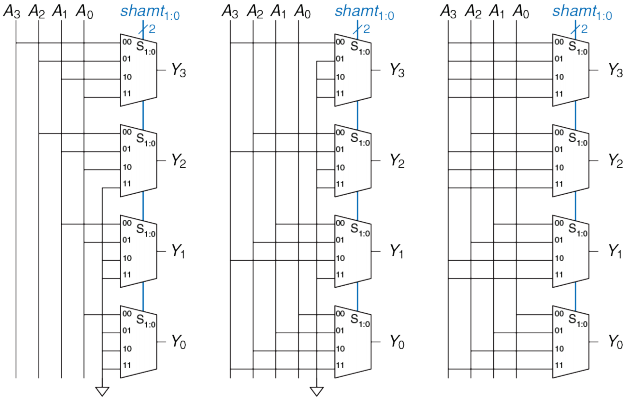
Логические схемы сдвига сдвигают число влево (LSL) или вправо (LSR) и заполняют пустые разряды нулями. Например, 11001 LSR 2 = 00110; 11001 LSL 2= 00100.

Арифметические схемы сдвига действуют так же, как и логические, но при сдвиге вправо они заполняют наиболее значащие разряды значением знакового бита исходного числа. Это необходимо при умножении и делении чисел со знаком.

Например: 11001 ASR 2 = 11110; 11001 ASL 2 = 00100.

Схемы циклического сдвига сдвигают число по кругу так, что пустые места заполняются разрядами, которые выдвинуты из другого конца. Например: 11001 ROR 2 = 01110; 11001 ROL 2 = 00111

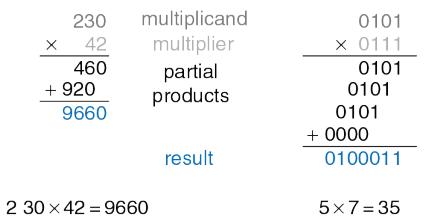
N-разрядная схема сдвига может быть построена из N мультиплексоров N:1. Вход сдвигается на 0 – N-1 разрядов в зависимости от значения log2N линий выбора.



4-разрядные схемы сдвига: сдвиг влево, логический сдвиг вправо, арифметический сдвиг вправо

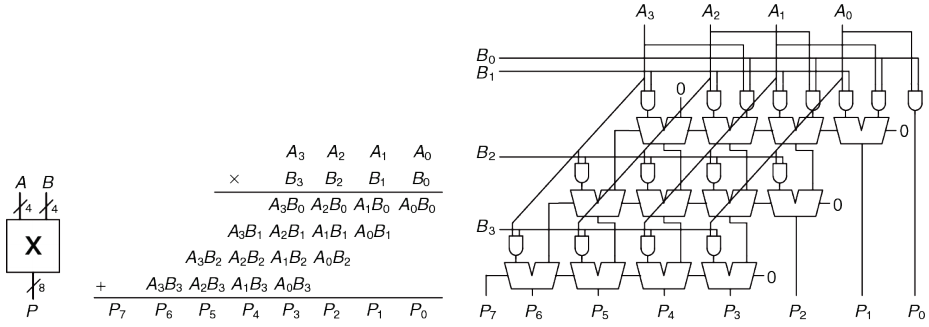
## Умножение

Умножение беззнаковых двоичных чисел подобно десятичному умножению, однако оно оперирует только с единицами и нулями.



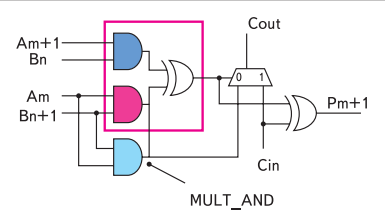
В обоих случаях частичные произведения формируются путем умножения отдельных разрядов множителя на всё множимое. Сдвинутые частичные произведения затем складываются, и мы получаем результат. В общем случае умножитель NхN перемножает два N-разрядных числа и порождает 2N-разрядный результат. Частичные произведения при двоичном умножении равны или множимому, или нулю. Умножение одного разряда двоичных чисел равносильно операции И, поэтому для формирования частичных произведений используются логические элементы И.

Умножитель 4 × 4: условное обозначение, функции, реализация



Так же, как и для сумматоров, существует множество реализаций умножителей с различными компромиссами между скоростью и стоимостью. Инструментальные средства синтеза могут выбирать наиболее подходящую реализацию по заданным временным ограничениям.

Умножение однобитных чисел реализуется вентилем 2И. Но при реализации умножения получается, что шестивходовой LUT задействуется только на реализацию одного разряда. Решением стало расширение логики, использующейся и для ускоренного переноса:



Модуль MULT\_AND, обеспечивающий ускорение операций умножения

Реализация на VHDL и сгенерированный из текста блок

library IEEE; use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD\_UNSIGNED.ALL;

entity multiplier is

generic(N: integer := 8);

port(a, b: in STD\_LOGIC\_VECTOR(N–1 downto 0);

y: out STD\_LOGIC\_VECTOR(2\*N–1 downto 0));

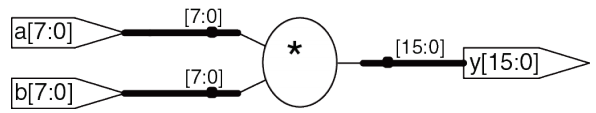
end;

architecture synth of multiplier is

begin

y <= a \* b;

end;



## Деление

Двоичное деление N-разрядных беззнаковых чисел в диапазоне [0, 2N-1] может быть выполнено с использованием следующего алгоритма:

R′ = 0

for i = N–1 to 0

R = {R′ << 1, Ai}

D = R – B

if D < 0 then Qi = 0, R′ = R // R < B

else Qi = 1, R′ = D // R ≥ B

R = R′

Частичный остаток R инициализируется 0. Наиболее значимый разряд делимого A затем становится наименее значимым разрядом R. Делитель В многократно вычитается из частичного остатка, и определяется знак разницы D. Если она отрицательна (т.е. знаковый разряд равен 1), то разряд частного Qi равен 0, и разница отбрасывается. В противном случае – Qi равен 1 и частичный остаток обновляется, он становится равным разнице D. Затем частичный остаток удваивается (сдвигается влево на один разряд), и процесс повторяется. Результат удовлетворяет условию A/B =Q+R/B.

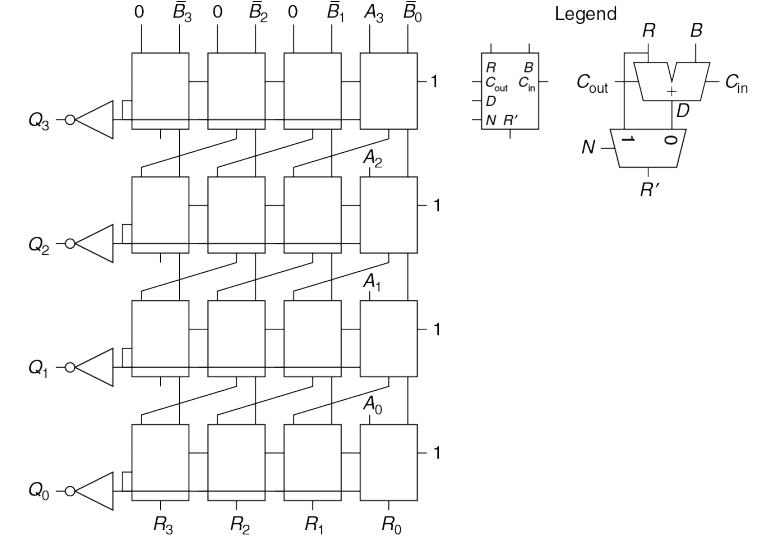


Схема вычисляет A/B и на выход выдает частное Q и остаток R. Сигнал N показывает, является ли результат R–B отрицательным. Это определяется по выходному сигналу переноса Cout самого левого блока в ряду, который является знаком разницы.

Задержка N-разрядной матрицы деления увеличивается пропорционально N2, так как перенос должен пройти через все N каскадов в ряду перед тем, как определится знак и мультиплексор выберет R или D. Это повторяется для всех N рядов. Деление – очень медленная и дорогая операция в аппаратной реализации, поэтому ее следует использовать как можно реже.

При этом существует класс задач, в которых недостатки деления обращаются в достоинства, например задачи факторизации, на которых построена современная криптография.

DSP блок ПЛИС

В ПЛИС семейства VIRTEX (ARTIX-7 и Kintex-7) содержатся многофункциональные блоки, предназначенные для стандартных операций по цифровой обработке сигналов (DSP48E1).

