

DOI: 10.32517/2221-1993-2026-25-3-41-50

**Е. А. Еремин***Пермский государственный гуманитарно-педагогический университет, г. Пермь, Россия***К. Ю. Поляков***Санкт-Петербургский государственный морской технический университет, Санкт-Петербург, Россия;**Школа № 174, Санкт-Петербург, Россия*

ДЕМОНСТРИРУЕМ ШКОЛЬНИКАМ ПРОЦЕСС ПРОЕКТИРОВАНИЯ ВЫЧИСЛИТЕЛЬНЫХ УСТРОЙСТВ

Аннотация

Учителю полезно иметь в своем арсенале комплексные практические задания (проекты), которые требуют использования знаний из нескольких частей изучаемого курса. В статье подробно разработан подобный проект: предлагается пройти основные этапы конструирования одного из важнейших устройств компьютера — сумматора. В ходе выполнения проекта школьникам потребуются применить свои знания по построению соответствующей условию задачи таблицы истинности, замене ее эквивалентной логической формулой, преобразованию последней средствами алгебры логики и, наконец, получению практической схемы устройства на логических элементах. При наличии возможностей и желания можно даже собрать эту схему (например, на платформе Arduino) и посмотреть, как она работает. Существует и «облегченный» вариант — написать моделирующую работу сумматора программу или воспользоваться готовым тренажером, который предлагается авторами.

Статья содержит некоторые дополнительные сведения по теме: понятие о языках описания логических схем HDL (Hardware Description Language), идеи полноты системы используемых логических операций и их влияния на ассортимент выпускаемых микросхем, сведения о построении многоразрядного сумматора и способах ускорения выполнения последовательной по своей сути операции сложения, общие идеи построения АЛУ (арифметико-логического устройства) современного процессора. Весь материал изложен на доступном школьникам уровне.

Проект имеет ярко выраженную профориентационную направленность: в процессе работы школьники получают некоторое представление об основных этапах разработки устройств вычислительной техники.

Ключевые слова: сумматор, логические элементы, алгебра логики, АЛУ, внутрипредметные связи, информатика, школьный курс, образование.

Информация об авторах

Еремин Евгений Александрович, канд. физ.-мат. наук, доцент, доцент кафедры прикладной информатики, информационных систем и технологий, факультет информатики и экономики, Пермский государственный гуманитарно-педагогический университет, г. Пермь, Россия; *e-mail:* evg_eregin@mail.ru

Поляков Константин Юрьевич, доктор тех. наук, профессор кафедры судовой автоматики и измерений, факультет корабельной энергетики и автоматики, Санкт-Петербургский государственный морской технический университет, Санкт-Петербург, Россия; учитель информатики, школа № 174, Санкт-Петербург, Россия; *e-mail:* kpolyakov@mail.ru

E. A. Eremin

Perm State Humanitarian and Pedagogical University, Perm, Russia

K. Yu. Polyakov

St. Petersburg State Marine Technical University, Saint Petersburg, Russia;

School 174, Saint Petersburg, Russia

DEMONSTRATING THE PROCESS OF DESIGNING COMPUTING DEVICES TO PUPILS

Abstract

It is useful for a teacher to have in his arsenal complex practical assignments (projects) that require the use of knowledge from several parts of the course being studied. The article develops such a project in detail: it proposes to pass through the main stages of designing one of the most important computer devices — an adder. During the project, students will need to apply their knowledge of constructing a truth table corresponding to the problem, replacing it with an equivalent logical formula, transforming it using Boolean algebra, and finally, obtaining a practical scheme of a device on logical elements. If you have the opportunity and desire, you can even assemble this circuit (for example, on the Arduino platform) and see how it works. There is also a "light" variant: to write a program that simulates the work of an adder or use the ready simulator proposed by the authors.

The article contains some additional information on the topic: the concept of HDL (Hardware Description Language) for logic circuit description, ideas about the completeness of the system from logical operations and its influence on the range of manufactured microcircuits, information about the construction of a multi-bit adder and ways to speed up the execution of an essentially sequential addition operation, general ideas for constructing an ALU (arithmetic logic unit) of a modern processor. All material is presented at a level accessible to schoolchildren.

The project has a clear career-oriented focus: during their work, students gain some understanding of the main stages of computer device development.

Keywords: adder, logical elements, Boolean algebra, arithmetic logic unit, ALU, inter-subject connections, informatics, school course, education.

Information about the authors

Evgeny A. Eremin, Candidate of Sciences (Physics and Mathematics), Docent, Associate Professor at the Department of Applied Informatics, Information Systems and Technologies, Faculty of Informatics and Economics, Perm State Humanitarian and Pedagogical University, Perm, Russia; e-mail: evg_eremin@mail.ru

Konstantin Yu. Polyakov, Doctor of Sciences (Engineering), Professor at the Department of Ship Automation and Instrumentation, Faculty of Ship Power Engineering and Automation, St. Petersburg State Marine Technical University, Saint Petersburg, Russia; an informatics teacher, School 174, Saint Petersburg, Russia; e-mail: kpolyakov@mail.ru

...Искусство и наука едва ли где-либо передаются не в отрывках, а энциклопедически. Поэтому в глазах учащихся они являются как бы кучей дров или хвороста, и никто не может понять, чем они между собой связаны: один подхватывал одно, другой — другое, и ни у кого образование не было вполне всесторонним, а следовательно, и основательным.

Все, что находится во взаимной связи, должно преподаваться в такой же связи.

Я. А. Коменский. «Великая дидактика»

1. Введение

«Сложившаяся сегодня ситуация в стране требует восстановления производственных процессов и особенно элементной базы радиоэлектронных и цифровых устройств» [7]. России необходимо иметь возможность самостоятельно разрабатывать и производить современные устройства вычислительной техники. Для этого требуются квалифицированные специалисты. Школьный курс информатики может здесь помочь путем ориентации некоторой части способных учеников на профессии в указанных областях.

Данная статья показывает один из возможных путей движения в направлении решения сформулированной стратегической задачи. Идея состоит в том, чтобы на доступном примере дать школьникам представление о том, как выглядит процесс разработки реального вычислительного устройства, в качестве которого выбран сумматор. Проводимое рассмотрение показывает, что для построения схемы сумматора не требуется выходить за пределы программы школьного курса информатики. Конечно, в статье содержатся и некоторые дополнительные сведения, но все они носят описательный характер и доступны пониманию рядового ученика.

Статья будет полезна творчески работающим учителям информатики, а также школьникам, интересующимся компьютерами и планирующим связать свою профессию с информатикой. На основе ее материалов можно реализовать с учениками полезный и поучительный проект.

2. Дидактическое обоснование проекта

Как утверждает классическая педагогика (см. эпиграф), эффективность усвоения учебного материала существенно возрастает, когда его изложение ведется во взаимосвязи, особенно когда демонстрируется *практическое* применение *теоретических* знаний. К тому же систематические знания прочнее, нежели разрозненные сведения, и быстрее вспоминаются.

Речь сейчас идет не столько о *межпредметных* (МПС), сколько о *внутрипредметных* связях (ВПС). По свидетельству Т. Н. Гнитецкой [5], проблема ВПС исследована значительно слабее, чем проблемы МПС. В то же время дидактические требования целостности и систематичности знаний являются общепринятыми. Любой учитель знает, что когда обучение построено на основе запоминания отдельных фактов из отдельных параграфов, то единой картины не получится и, как следствие, не стоит рассчитывать на формирование качественного представления об изучаемом предмете. Так что *усвоение любой учебной дисциплины в значительной мере определяется степенью взаимосвязи ее элементов*.

Многие авторы подчеркивают, что умение выявлять связи в содержании изучаемого материала служит основой его понимания (см., например, [3, 6]). Это важно, поскольку «понимание является наиболее глубоким видом знания и достигается лишь там, где знания приводятся в определенную систему» [1]. «Чем шире круг предметов, явлений, с которыми связывается понимаемое, чем более они существенны, тем глубже понимание» [10].

Простой и наглядный пример необходимости целостности знаний приводится в замечательной статье А. Кзя «Идеям тоже нужна любовь!..» [12]. Большинство студентов крупных университетов правильно отвечали на вопрос о текущем времени года в разных полушариях. Одновременно многие из них с уверенностью утверждали, что Земля летом находится ближе к Солнцу, чем зимой. И только 5–7 % замечали «нестыковку» высказанных утверждений друг с другом.

Имеются и более строгие доказательства необходимости ВПС. Сошлемся на результаты масштабного экспериментального исследования [32], в котором проверялись знания 8232 учащихся в пяти штатах США по четырем естественно-научным предметам. Проведенный анализ показал явные преимущества проверки интеграции знаний по сравнению с традиционным тестом из вопросов со множественным выбором ответов. Под *интеграцией знаний* в статье понимается способность использовать подходящие аргументы, *объединяя* их при объяснении явлений или утверждений.

Еще более далеко идущее положение высказывается в статье Н. И. Заводчиковой, И. А. Быковой [9]: предлагается длину логических цепочек, используемых школьником при связи материала, рассматривать как критерий качества усвоения учебного материала.

Имеются также практические подтверждения того, что систематическое использование ВПС способствует повышению качества усвоенных знаний [14].

В пользу важности взаимосвязанности учебного материала свидетельствует также широкое распространение многочисленных видов карт знаний [4, 31].

О формировании ВПС учитель должен специально позаботиться, поскольку даже если связи и возникнут у учеников стихийно, то совсем не обязательно будут отражать существенные закономерности. Так, в обзоре [33] представлено экспериментальное подтверждение того факта, что новые понятия часто присоединяются у студентов не к важным знаниям, полученным ранее, а к ярким иллюстративным примерам, продемонстрированным на текущем уроке.

Таким образом, *объединение в ходе упражнений материалов из нескольких различных параграфов учебника способствует повышению качества знаний.*

Становится понятным, почему *«содержание, которое в одной задаче является целью действия, должно войти в последующую как способ или часть способов ее решения»* [6]. Именно на такой педагогической основе и построен излагаемый далее учебный материал.

3. Некоторые предварительные сведения

Прежде чем непосредственно перейти к рассмотрению задачи, предварительно приведем некоторые вспомогательные сведения, которые, как окажется в дальнейшем, пригодятся при ее объяснении. При беглом знакомстве с содержанием статьи их можно пропустить.

3.1. Как перейти от таблицы истинности к логической формуле

Договоримся здесь и далее логическое значение «ИСТИНА» кодировать цифрой 1, а значение «ЛОЖЬ» — цифрой 0.

Любое логическое устройство при отсутствии памяти однозначно описывается таблицей истинности, в которой заданы значения выходных сигналов для каждой комбинации сигналов входных. Но для последующей работы по построению и оптимизации логической схемы часто удобнее представление в виде логической формулы. Поэтому важно уметь переходить от первой указанной формы представления логической функции ко второй.

Существуют два стандартных способа выполнить указанное преобразование.

В первом случае обрабатываются только строки таблицы с единичным (истинным) результатом. Чтобы получить логическую функцию, которая выдает единицу для любой из этих строк, их описания должны объединяться логической операцией ИЛИ: она дает единицу при истинности любого из логических слагаемых. Каждое слагаемое, в свою очередь, должно содержать строго определенную комбинацию входных сигналов, поэтому последние необходимо связать логической операцией И: несовпадение хотя бы одного значения приводит к нулевому результату. Осталось добавить, что, если входной сигнал равен 0, его необходимо предварительно проинвертировать (выполнить операцию НЕ), и алгоритм преобразования таблицы в формулу готов. Конкретный пример описан в следующем разделе.

Второй способ, напротив, работает только со строками, где результат равен 0. Он устроен аналогично,

только логические операции ИЛИ и И в нем меняются местами.

Первый способ принято называть *совершенной дизъюнктивной нормальной формой, СДНФ*, а второй — *совершенной конъюнктивной нормальной формой, СКНФ*. Для удобства изложения мы будем пользоваться приведенными аббревиатурами, хотя с точки зрения понимания вводить их не обязательно. Отметим, что оба указанных алгоритма доступно описаны в учебном пособии [2].

Напрашивается интуитивное предположение, что, когда в таблице истинности преобладают единичные результаты, надо всегда выбирать метод СДНФ, а когда преобладают нулевые — СКНФ. Тем не менее, если планировать переход к физической реализации устройства на конкретных микросхемах, полезно анализировать обе возможности. Дело в том, что на практике ассортимент логических элементов может быть ограничен, так что отсутствующие элементы приходится составлять из тех, какие есть. Как следствие, минимальное число требуемых микросхем не обязательно подчиняется указанному предположению.

3.2. Логическая операция XOR и полезные формулы для нее

При проектировании сумматора очень удобно использовать логические элементы, выполняющие операцию «исключающее ИЛИ» (XOR). Не случайно одно из названий этой операции — «сложение по модулю 2». Рассмотрим эту операцию подробнее.

Договоримся операцию XOR обозначать знаком « $\wedge$ », как это делается во многих языках программирования. В дальнейшем окажется, что такой выбор дополнительно обеспечит совместимость с языком описания логических схем (см. раздел 4.4). Будем обозначать операцию НЕ чертой сверху, а операции И и ИЛИ — знаками умножения и сложения соответственно.

Таблица истинности для рассматриваемой операции XOR занимает столбцы 1–3 в таблице 1. Очевидно, что при совпадении значений A и B результат равен 0, а в противном случае — 1.

Выведем СДНФ для XOR.

Рассмотрим строку 2 таблицы, для которой результирующее значение $M = 1$. Для данной строки $A = 0$ и $B = 1$, т. е. одновременно истинны $\bar{A}$ (отрицание A) и B . Указанное выражение занесено в столбец 5 таблицы 1. Аналогичным образом обрабатывается и строка 3. Остается только собрать оба слагаемых через ИЛИ (обозначается здесь и далее знаком «+»). В итоге получаем следующую полезную формулу:

$$M = A \wedge B = \bar{A} \cdot B + A \cdot \bar{B}. \quad (1)$$

Если посмотреть внимательно, то окажется, что смысл формулы (1) достаточно очевиден: результат равен 1, когда A и B различны.

Проделаем аналогичные действия для получения СКНФ.

Здесь все не так наглядно, поэтому просто будем следовать алгоритму. Выделим строки 1 и 4 с нулевым результатом. В каждой из них напишем входные переменные, причем с отрицанием теперь будут те, которые равны 1; соединим переменные логической операцией

Логическая операция XOR

	1	2	3	4	5	6	7	8
	A	B	$M = A \wedge B$	Аргументы		$\bar{M}$	Аргументы	
1	0	0	0	В СКНФ	$A + B$	1	В СДНФ	$\bar{A} \cdot \bar{B}$
2	0	1	1	В СДНФ	$\bar{A} \cdot B$	0		
3	1	0	1	В СДНФ	$A \cdot \bar{B}$	0		
4	1	1	0	В СКНФ	$\bar{A} + \bar{B}$	1	В СДНФ	$A \cdot B$

ИЛИ (результат см. в столбце 5 таблицы 1). Выпишем оба выражения, объединяя их логической операцией И. В результате получим альтернативную формулу для XOR:

$$M = A \wedge B = (A + B) \cdot (\bar{A} + \bar{B}). \quad (2)$$

Формулы (1) и (2), разумеется, описывают одну и ту же функцию. Читатели могут в этом убедиться либо при помощи соответствующих таблиц истинности, либо аккуратно раскрыв скобки в формуле (2).

В дальнейшем нам пригодится еще одна формула — для отрицания M .

Запишем отрицание для формулы (2) и, применяя известные правила де Моргана [2, 16], получим следующее:

$$\begin{aligned} \bar{M} &= \overline{A \wedge B} = \\ &= \overline{(A + B) \cdot (\bar{A} + \bar{B})} = \overline{(A + B)} + \overline{(\bar{A} + \bar{B})} = \bar{A} \cdot \bar{B} + A \cdot B. \end{aligned}$$

Формула примет окончательный вид:

$$\bar{M} = \bar{A} \cdot \bar{B} + A \cdot B. \quad (3)$$

Заметим, что формулу (3) можно было бы вывести более простым способом: взять таблицу истинности для $\bar{M}$ (столбец 6) и построить для нее СДНФ (строки 1, 4).

Смысл формулы (3) также понятен: исходная переменная $M=1$ во всех случаях, *кроме* (из-за отрицания!) тех, где A и B совпадают. Иначе говоря, при совпадении $M=0$.

3.3. Логические элементы в микросхемах

В академической литературе (к упрощенному варианту которой относятся учебники) принято использовать в качестве базового набора логические операции И, ИЛИ, НЕ. Перечисленный набор является *полным*, поскольку его достаточно для реализации *любой* логической функции. В то же время этот классический набор не является *минимальным*, поскольку его можно уменьшить, не нарушая свойства полноты. Об этом свидетельствуют уже упоминавшиеся ранее правила де Моргана, которые (при наличии операции НЕ) фактически позволяют заменять И на ИЛИ и наоборот. В результате минимальной функционально полной системой будет либо построенная на основе операций И, НЕ, либо альтернативная на ИЛИ, НЕ.

Если пойти чуть дальше, то можно объединить эти пары вместе. И+НЕ образуют единую операцию, называемую *штрих Шеффера*, а операция ИЛИ+НЕ известна в литературе как *стрелка Пирса*. Таким образом,

единственный логический элемент И-НЕ (в промышленных микросхемах обычно используется именно эта комбинация) обладает свойством логической полноты, и с его помощью можно построить любую логическую функцию, начиная от «академической тройки» И, ИЛИ, НЕ, до произвольной сколь угодно сложной.

Особое место занимает логический элемент исключающее ИЛИ (XOR). С одной стороны, его можно получить, соединив определенным образом четыре элемента И-НЕ (см., например, [22]). Но, с другой стороны, эта операция в вычислительных устройствах играет настолько большую роль (достаточно назвать суммирование и сравнение), что для XOR выпускаются специальные микросхемы [28].

Имеется большой ассортимент логических микросхем И-НЕ и исключающее ИЛИ. Типичная микросхема содержит четыре логических элемента с двумя входами. Нетрудно сообразить, что на каждый такой элемент необходимо по три вывода (входы и выход), так что всего будет $3 \times 4 = 12$ выводов. Добавив еще два для подключения питания, окончательно получаем микросхему с 14 выводами (*пинами*). Конкретные марки микросхем легко найти в любом справочнике или в интернете. Существует несколько серий, изготовленных по разным производственным технологиям. Наиболее удивительно то, что такие (казалось бы, «древние») 14-пиновые микросхемы до сих пор можно без труда приобрести, в том числе в интернет-магазинах. Подчеркнем, что данное обстоятельство позволяет при желании не ограничиваться теорией, а на практике поэкспериментировать с логическими элементами.

Интересующиеся также могут дополнительно почитать, как выглядят оптические логические элементы [29].

В последнее время большое распространение получили программируемые *микросхемы FPGA* (аббревиатура от *англ.* Field-Programmable Gate Array — программируемая пользователем вентильная матрица). Они состоят из особых *программируемых логических блоков* (*англ.* Configurable Logic Block, CLB). Каждый такой блок является, по сути, универсальным логическим элементом. Не вдаваясь в подробности, отметим только, что для каждого CLB можно задать любую таблицу истинности. Следовательно, для таких микросхем нет ограничений на типы логических элементов.

К сожалению, продемонстрировать школьникам работу с современными FPGA едва ли получится, в то время как поработать со старыми добрыми логическими микросхемами низкого уровня вполне реально.

4. Этапы проектирования устройства

Мы поставили себе цель придумать «сквозную» задачу, объединяющую теоретические параграфы по математической логике и логическим элементам с практической разработкой некоторого реально полезного вычислительного устройства. Тем самым мы хотели продемонстрировать ученикам, как примерно выглядит деятельность разработчика компьютерной техники. Согласно изложенному во введении, данная задача в настоящее время является весьма актуальной.

В качестве проектируемого устройства мы оставили свой выбор на сумматоре.

Во-первых, он включен в программу базового курса информатики X–XI классов, причем все, что необходимо для его разработки, тоже есть в программе. В частности, в Федеральной рабочей программе по информатике [23] записано следующее: «Эквивалентные преобразования логических выражений. <...> Построение логического выражения с данной таблицей истинности. Логические элементы компьютера. <...> Сумматор. Построение схемы на логических элементах по логическому выражению». Все это (плюс некоторый дополнительный материал) и охватит наша задача.

Во-вторых, сумматор является основой арифметико-логического устройства (АЛУ) любого компьютера. Добавив к рассмотренной задаче всего несколько предложений, можно сформировать у учеников первичное и вполне адекватное представление об АЛУ и машинной реализации двоичной арифметики.

В-третьих, рассмотрение объединения нескольких разрядов сумматора в единое устройство позволяет обсудить с учениками важную проблему организации параллельности вычислений на уровне битов. В самом деле, многие даже не задумываются о том, что алгоритм сложения в традиционной арифметике существенным образом *последовательный* и сложение неспешно ведется от младших разрядов к старшим. Для создания быстро работающего АЛУ это подходит очень плохо.

4.1. Формулируем задачу

Задумаемся над тем, как компьютер складывает числа. Предположим, что мы хотим вычислить простейшую функцию $y(x) = x + 123$. Прежде чем процессор сможет выполнить необходимое арифметическое действие, формулу необходимо преобразовать в последовательность понятных процессору инструкций. Один из простых методов — посмотреть результирующую программу в машинных командах — был описан в статье [8]. Следуя описанной методике, легко можно обнаружить, что результат трансляции для 32-битного аргумента x получится следующий:

```
movsxd  rax, ecx
add      rax, 123
```

Первая команда копирует аргумент из регистра *ecx* в 64-битный *rax* (нас сейчас не интересуют детали этого шага). А вот вторая инструкция выполняет собственно сложение (слово *add* именно так и переводится с английского).

Аппаратно операция происходит следующим образом. На один вход сумматора подается копия значения из регистра *rax*, а на второй — двоичное представление константы 123. После выполнения сложения результат по-

мещается обратно в регистр *rax*. Таким образом, *ключевая часть этого процесса выполняется в сумматоре*, который мы и выбрали в качестве проектируемого устройства.

Рассмотрим теперь процесс разработки схемы сумматора для двух целых бинарных чисел без знака. Сначала подробно изучим логику работы для одного отдельно взятого бита, а затем обсудим, как объединить полученные блоки в многоразрядный сумматор.

Обязательно будем учитывать, что в ходе суммирования из младшего разряда в старший может появляться единица переноса.

Итак, рассмотрим отдельно взятый разряд сумматора. Введем обозначения. Пусть A и B есть складываемые биты двух чисел, а C_{in} — входящий перенос из младшего разряда в данный. **Заранее договоримся, что в промежуточных рабочих формулах мы будем C_{in} для краткости обозначать C .** По этим трем входным значениям (аргументам) наша схема должна получить два результата: значение суммы S и выходной перенос в следующий разряд C_{out} .

Из математики известны следующие расчетные формулы:

$$S = (A + B + C) \bmod 2,$$

$$C_{out} = (A + B + C) \operatorname{div} 2.$$

Подчеркнем, что первая формула прозрачно намекает нам на связь получения суммы с логической операцией XOR, которую не случайно часто называют «сложение по модулю 2».

Во многих учебниках сформулированную задачу упрощают за счет разделения операции на два последовательных сложения: $S' = A + B$ и $S = S' + C$. Для каждого из них создается так называемый *полусумматор*. Мы будем решать задачу в полной постановке (не базируясь на полусумматорах): нам кажется, что демонстрация решения общей задачи без специфических приемов лучше соответствует поставленным нами целям.

4.2. Строим таблицу истинности

По приведенным в предыдущем разделе формулам построим таблицу истинности (табл. 2). В ней пять

Таблица 2

Таблица истинности одноразрядного двоичного сумматора

	0	1	2	3	4	5	6
		C_{in}	A	B	сум ₁₀	C_{out}	S
1		0	0	0	0	0	0
2		0	0	1	1	0	1
3		0	1	0	1	0	1
4	g	0	1	1	2	1	0
5		1	0	0	1	0	1
6	p	1	0	1	2	1	0
7	p	1	1	0	2	1	0
8	g	1	1	1	3	1	1

основных столбцов: 1–3 с логическими аргументами и 5–6 с результатами. Дополнительный столбец 4 является десятичным значением суммы. Он добавлен исключительно ради наглядности: результаты в столбцах 5 и 6 есть не что иное, как двоичная запись числа из столбца 4. Наконец, столбец, пронумерованный нулем, понадобится значительно позднее — на него пока можно не обращать внимания.

Верхняя половина таблицы (строки 1–4) описывает случай, когда входной перенос отсутствует, а нижняя (строки 5–8), напротив, когда он равен 1.

4.3. Получаем логические формулы

Напишем логическую формулу, которая будет эквивалентна таблице 2. Будем пользоваться формой СДНФ (см. раздел 2.1).

Начнем с C_{out} . Согласно алгоритму, формула должна содержать четыре слагаемых, поскольку в интересующем нас сейчас столбце 5 именно такое количество единиц. Выпишем комбинации входных переменных для строк 4 и 6–8, помечая инверсией переменные с нулевыми значениями. Полученное выражение упростим, группируя слагаемые в пары:

$$C_{out} = \bar{C}AB + C\bar{A}B + C\bar{A}\bar{B} + CAB = \\ = (\bar{C} + C)AB + C(\bar{A}B + \bar{A}\bar{B}) = AB + C(A\bar{B}).$$

Последняя скобка преобразована согласно формуле (1). Напомним также, что $C = C_{in}$.

После введения обозначений (пока формальных, но в разделе 5.1 мы увидим их смысл):

$$g = A \cdot B, \quad p = A \wedge B \quad (4)$$

получим окончательную формулу для сигнала переноса в старший разряд:

$$C_{out} = g + C \cdot p. \quad (5)$$

Аналогичным образом построим СДНФ для суммы. Единичные значения в столбце 6 содержатся в строках 2, 3, 5, 8. Выпишем для них по тем же правилам комбинации входных переменных и сгруппируем слагаемые попарно:

$$S = \bar{C}\bar{A}B + \bar{C}A\bar{B} + C\bar{A}\bar{B} + CAB = \bar{C}(\bar{A}B + A\bar{B}) + C(\bar{A}\bar{B} + AB).$$

Первую скобку преобразуем согласно формуле (1) в операцию исключающее ИЛИ, а вторую по формуле (3) — в ее отрицание. Используя обозначения для p из (4), получим следующее:

$$S = \bar{C} \cdot p + C \cdot \bar{p}.$$

Применяя формулу (1) еще раз, окончательно имеем:

$$S = C \wedge p. \quad (6)$$

Таким образом, формулы для результатов сложения с учетом обозначений (4) имеют вид (5) и (6).

4.4. Описываем схему на HDL

Все предыдущие действия полностью укладывались в программу школьного курса информатики. Здесь мы впервые хотим выйти за эти пределы ради важного разговора о современных способах описания логических схем с целью их последующей реализации «в железе». Как

ни удивительно, новый материал из достаточно непривычной области практически не потребует от читателей какой-либо предварительной подготовки.

Итак, мы получили весьма компактные формулы (4)–(6), описывающие логику работы одноразрядного двоичного сумматора. Что делать дальше, чтобы перейти от теоретических формул к практической схеме данного устройства из стандартных логических элементов?

Строго говоря, внимательно глядя на формулы, можно построить схему устройства «вручную». Скажем, посмотрев на формулу (4), нетрудно увидеть, что значение g может быть получено с помощью логического элемента И, а значение p — с помощью исключающего ИЛИ. Формула (5) добавит два логических элемента — И и ИЛИ, а формула (6) — еще одно исключающее ИЛИ.

Но можно ли переложить всю техническую работу по подготовке схемы на компьютер? Оказывается, можно! Для описания логических схем существуют специальные языки **HDL** (аббревиатура от *англ.* Hardware Description Language — язык описания аппаратных устройств). Если описать схему на одном из таких языков, то можно немедленно получить ее рисунок в виде соединения логических элементов. Более того, программные системы, как правило, способны предварительно произвести типовые оптимизации разрабатываемой схемы. И, наконец, «вишенка на торте»: имея текст с описанием на HDL, можно попросить компьютер запрограммировать подключенную к нему микросхему FPGA (см. раздел 3.3) и тем самым получить работоспособное устройство. Современные инженеры именно так и делают.

Посмотрите, как выглядит описание одноразрядного двоичного сумматора на одном из самых простых языков под названием SystemVerilog [26]:

```
module fulladder(input logic a, b, cin,
                 output logic s, cout);
    logic p, g;
    assign p = a ^ b;
    assign g = a & b;
    assign s = p ^ cin;
    assign cout = g | (p & cin);
endmodule
```

Не правда ли, это сильно похоже на обычный язык программирования? Процедура со входными и выходными параметрами, описание переменных и операторы присваивания... Сравните приведенный текст с формулами (4)–(6) и вы увидите, что они совпадают с точностью до несущественных перестановок. Ну и обозначения логических операций не совсем привычные.

Оказывается, современный конструктор электронной техники тоже немного программист!

4.5. Рисуем схему устройства

В утверждении о том, что компьютер по тексту на HDL нарисует схему устройства, есть некоторый нюанс: программное обеспечение, как правило, не умеет это делать по отечественным стандартам. А, как известно, принятые за рубежом изображения логических элементов (стандарт **ANSI/IEEE Std 91a-1991**) отличаются от рекомендуемых в нашей стране согласно **ГОСТ 2.743-91**. В частности, схема сумматора на экране дисплея может выглядеть, как показано на рисунке 1.

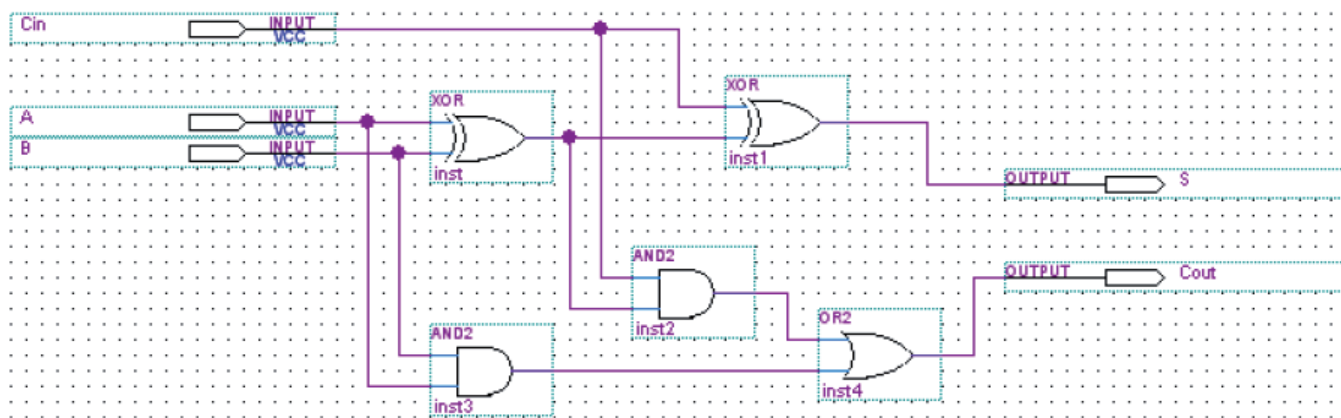


Рис. 1. Компьютерная схема в стандарте IEEE

Большой проблемы это не представляет. Специалисты свободно ориентируются в обеих системах, а на школьном уровне профессиональное ПО с HDL практически не используется.

Поэтому, указав ради справедливости на эту тонкость, возьмем за основу схему сумматора с изображением элементов в отечественном стандарте (рис. 2).

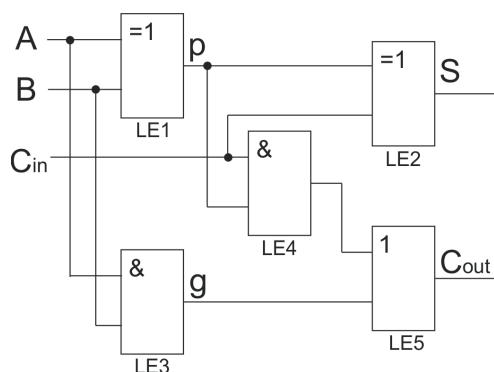


Рис. 2. Схема одноразрядного двоичного сумматора в отечественном стандарте

На схеме легко найти слева три входных сигнала A , B и C_{in} , справа — два выходных сигнала S и C_{out} , а также промежуточные сигналы p и g , определяемые формулами (4).

Логические элементы LE1 и LE2 выполняют операцию исключающее ИЛИ, LE3 и LE4 — логическую операцию И, а LE5 обеспечивает функцию ИЛИ. Предлагаем читателям самостоятельно убедиться, что схема на рисунке 2 полностью соответствует формулам (4)–(6).

Попутно заметим, что LE1 и LE3 образуют первый полусумматор, а LE2 и LE4 — второй. Видно, что к ним дополнительно требуется добавить элемент LE5.

Приведенная схема содержит пять логических элементов. Как нетрудно видеть, время установления выходного сигнала C_{out} (а он появится позднее, чем результат S , из-за наличия LE4) равняется времени срабатывания трех логических элементов. Эта величина может считаться временем суммирования одного разряда.

4.6. Можно ли пойти дальше?

Сначала подведем некоторые предварительные итоги. Мы подробно рассмотрели весь путь от матема-

тической формулы сложения до логической схемы сумматора (через таблицу истинности и упрощение формул с помощью математической логики). Тем самым мы полностью спроектировали реальное вычислительное устройство — одноразрядный двоичный сумматор.

Весь использованный материал (кроме изложенного в разделе 4.4) не выходит за рамки школьной программы. Что же касается языка HDL, то мы считаем, что о нем стоит рассказать на уроке. Во-первых, это даст ученикам некоторое представление о профессиях, связанных с разработкой электронной аппаратуры, что уже само по себе важно. Во-вторых, для понимания этого материала ученикам не требуется никакой дополнительной подготовки. Наконец, в-третьих, если заранее подготовить в том или ином виде готовый текст описания на HDL, приведенный выше, рассказ займет совсем немного времени.

Возможно ли еще чем-то дополнить наш проект сумматора? В принципе да, и даже есть несколько вариантов. Рассмотрим их в порядке возрастания сложности.

4.6.1. Проверить таблицу истинности

Самое простое, что можно сделать, это проверить правильность полученных формул (4)–(6). Для этого достаточно написать несложную программу, которая построит для них на экране таблицу истинности; останется лишь внимательно сравнить ее с исходной таблицей 2.

Программа будет достаточно простой и типовой. Потребуется построить три вложенных цикла (по каждой из входных переменных), внутри которых выполнить действия по формулам и аккуратно вывести данные на экран.

4.6.2. Написать программу-имитатор

Более сложный вариант — это дать ученикам написать программу-симулятор. Тут очень хорошо подошло бы решение в ООП-стиле [17], но, к сожалению, это получится только в XI классе. Возможно, стоит подумать о том, чтобы вернуться к проекту на следующий учебный год.

Есть еще одна альтернатива. Учитель может сам подготовить программную реализацию в качестве модели устройства. В этом случае можно продумать графический интерфейс, вывод текущего состояния каждого логического элемента, построение таблицы истинности и т. д. Словом, предусмотреть все то, что учитель хотел бы обсудить на уроке с учащимися.

В недавно опубликованной статье В. С. Попова [18] подробно описывался симулятор логических схем. В рассмотренном в данной статье ПО есть готовый сумматор (1-Bit Full Adder), который подходит и для наших целей. Необходимо только оговорить, что симулятор базируется на зарубежных обозначениях логических элементов. Можно также воспользоваться тренажером, выложенным на авторском сайте поддержки (<https://kpolyakov.spb.ru/school/probook/online/summator.htm>).

4.6.3. Собрать реальную схему

При наличии у учителя желания и технических возможностей можно попытаться реализовать модель сумматора на реальных микросхемах. Может быть, даже совместно с некоторыми учениками, что было бы еще лучше.

Тут можно пойти несколькими путями.

Если в школе найдутся люди, владеющие паяльником, то можно элементарно собрать приведенную на рисунке 2 схему. Желательно дополнить ее тремя переключателями для задания входных значений и светодиодами для индикации выходных сигналов.

Существует также вариант сборки схемы на специальной монтажной плате, например, из комплекта Arduino. Это проще и, скорее всего, будет выглядеть эстетичнее (см., например, сетевые ссылки [13, 30]).

При сборке электронной схемы неоценимую помощь может оказать прекрасная книга Ч. Платта «Электроника: логические микросхемы, усилители и датчики для начинающих» [15]. В ней в качестве эксперимента под номером 24 подробнейшим образом описан процесс сборки своими руками сумматора из нескольких битов.

В разделе 3.3 говорилось, что в ассортименте выпускаемых микросхем есть не все логические элементы. Для рассматриваемой схемы это также справедливо. Ради ее практической реализации достаточно на рисунке 2 логические элементы LE3—LE5 формально заменить на И-НЕ. Докажем, что при этом логика работы нашей схемы не изменится. Выпишем выражение (5) и добавим к нему двойное отрицание, что, как известно, значения не изменяет. Затем, сохранив одно из отрицаний, для другого применим правило де Моргана. В результате получим:

$$C_{out} = \overline{\overline{g + C \cdot p}} = \overline{\overline{g} \cdot \overline{(C \cdot p)}}.$$

Имеем два элемента И-НЕ (LE3 и LE4), инверсные выходы которых также объединяются через И-НЕ (LE5).

Убедиться в тождественности предлагаемой замены также можно путем построения таблиц истинности и их последующего сравнения.

Учитывая, что в одном корпусе микросхемы собрано по четыре логических элемента, потребуется приобрести всего две микросхемы: одну — с И-НЕ, а другую — с XOR. Типы требуемых микросхем легко найти в справочнике [28] или в интернете (см., например, [19]).

5. От одnorазрядного сумматора к многоразрядному

Итак, мы сумели реализовать простейшее вычислительное устройство — одnorазрядный сумматор. Но реальный сумматор должен содержать множество раз-

рядов: от восьми в самых старых процессорах до 64 — в современных.

Казалось бы, особых проблем быть не должно: соединяем между собой нужное количество разрядов, и задача решена. Тем более что механизм переноса в виде сигналов C_{in} и C_{out} уже предусмотрен. Однако трудности все же есть и достаточно принципиальные. Дело в том, что алгоритм сложения существенным образом последовательный: сложение выполняется поразрядно от младших битов к старшим. Именно из-за этого не удастся выполнять суммирование всех битов параллельно.

Тем не менее ситуация не безнадежна — существуют методы ускорить работу сумматора в разы. Очень кратко познакомимся с ними.

5.1. Ускоренный перенос

Внимательно посмотрим на формулы для вычисления выходного переноса (4) и (5). Обратим внимание на следующие обстоятельства. Перенос выражается через две рабочие (казалось бы, не имеющие самостоятельного значения) переменные — g и p . При этом очевидно, что вырабатываемое данным разрядом значение переноса C_{out} непосредственно подается в качестве входного сигнала C_{in} на следующий разряд. Наконец, вычисление переноса напрямую не связано с получением суммы S ; иными словами, перенос можно определять отдельно и, что еще важнее, до вычисления суммы.

Проанализируем формулы (4) для переменных g и p . Первая равняется 1 в единственном случае — когда оба складываемых бита равны 1. Если вернуться к таблице 2, то такая ситуация имеет место для строк 4 и 8. Обратим внимание, что в столбце 0 эти строки помечены буквой « g ». Смысл в том, что действие $1 + 1$ всегда порождает перенос. Обозначение g как раз и означает *генерацию*, т. е. *возникновение, переноса* (англ. carry generate). Альтернативная возможность $C_{out} = 1$ реализуется в случае, когда складываемые различны ($0 + 1$ и $1 + 0$), но при этом есть входной перенос $C_{in} = 1$. Тогда $p = 1$, а операция И между $C = C_{in}$ и p в формуле (5) также обеспечивает единичный результат. Такая ситуация называется *распространением переноса* (англ. carry propagate). Ей соответствуют строки 6 и 7 в таблице 2. Таким образом, оказывается, что рабочие переменные g и p несут важную информацию для вычисления переноса C_{out} и с учетом C_{in} однозначно определяют результат по формуле (5).

Итак, зная входные значения C_{in} , A и B для данного разряда, удастся сразу вычислить для него выходной перенос C_{out} . Это самое значение одновременно является входным (C_{in}) для следующего разряда, так что теперь можно, снова используя формулу (5), получить значение переноса и для этого разряда. Процедуру можно продолжать, только очевидно, что для каждого последующего разряда длина формулы расчета переноса растет. На практике обычно применяют расчетные формулы для четырех битов (см. [24], [25] или [21]), и именно такую разрядность имели выпускаемые микросхемы.

Интересно отметить, что в одном из первых микропроцессоров Intel 8008, который был 8-битным, в арифметическом устройстве был реализован сквозной перенос для всех восьми битов. Впечатляющий вид увеличенного сегмента кристалла этого процессора с выде-

ленной схемой ускоренного переноса стоит посмотреть в интернете по ссылке [11].

Особенно важно подчеркнуть следующее. После оптимизации схемы расчета переносов для 4-битного сумматора максимальное количество элементов (вентилей), через которые проходит сигнал, никогда не превышает четырех [24]. Сравнивая с рисунком 2, где сигналу требуется пройти три вентиля, получаем, что время вычисления всех четырех переносов примерно равно времени сложения для одного бита (будем обозначать его τ). Зато теперь, зная переносы, можно просуммировать все четыре бита одновременно! Отсюда следует, что в случае четырех битов суммирование с ускоренным переносом выполняется за время примерно 2τ (расчет всех переносов + параллельное сложение разрядов), тогда как при поочередном сложении потребовалось бы 4τ .

Ситуацию можно еще улучшить, если результаты переноса из старших разрядов каждой 4-битовой секции (C_3 , C_7 , C_{11}) тоже ускорять по описанной выше схеме. В результате, как показано в [25], при организации 32-битного сумматора в восемь секций по четыре бита получается ускорение суммирования почти в четыре раза (по сравнению с последовательным). В следующем разделе мы расскажем, как можно улучшить результат еще вдвое.

5.2. Сумматор с выбором переноса

Следуя [20], рассмотрим 32-битный сумматор. Разобьем его на две половинки: младшую L и старшую H . Очевидно, что до вычисления переноса из старшего разряда L провести сложение в H невозможно.

Но вместо 16-битного сумматора H можно сделать два таких же по размерам H_0 и H_1 . Первый заставим вычислять старшие разряды суммы в предположении нулевого переноса, а второй — единичного. Наиболее ценно то, что все три сумматора — L , H_0 и H_1 — теперь могут начинать сложение одновременно! После завершения сложения в L определится значение переноса C_{15} из старшего бита, что позволит правильно выбрать результат: либо из H_0 , либо из H_1 .

Таким образом, увеличив аппаратную часть в 1,5 раза (три сумматора вместо двух), мы ускорили получение суммы вдвое.

Описанный метод можно дополнительно применить к каждому из 16-битных сумматоров, заменяя их на три 8-битных и т. д.

Заметим, что оба описанных метода ускорения суммирования хорошо объединяются друг с другом.

6. Сумматор — основная часть АЛУ

В заключение несколько слов о том, как строится **арифметико-логическое устройство (АЛУ)**, основным элементом которого как раз и является сумматор.

АЛУ, как видно из названия, является устройством для выполнения арифметических и логических операций. Именно они лежат в основе всей обработки данных в компьютере. В качестве типичного примера в таблице 3 приведены базовые инструкции процессора современного стандарта RISC-V.

Умалчивая о некоторых специфических расширениях этой таблицы, можно сказать, что АЛУ такого процессора может выполнять восемь действий, пере-

Таблица 3

Операции АЛУ в процессоре RISC-V

Действие (инструкция)	Мнемоника	Биты инструкции		
		14-й	13-й	12-й
Сложение	ADD	0	0	0
Сдвиг влево	SLL	0	0	1
Сравнение чисел со знаком	SLT	0	1	0
Сравнение чисел без знака	SLTU	0	1	1
Исключающее ИЛИ	XOR	1	0	0
Сдвиг вправо	SRL	1	0	1
ИЛИ	OR	1	1	0
И	AND	1	1	1

численных в первой колонке; их краткие обозначения (мнемоники) приведены в следующем столбце. Каждая операция имеет свой номер от 0 до 7, который, разумеется, кодируется в двоичном виде.

Для тех, кто удивлен отсутствием в таблице логической операции НЕ, сообщим, что вместо нее рекомендовано использовать XOR с константой, состоящей из всех единиц. Предлагаем читателям самостоятельно проверить данную рекомендацию, базирующуюся на еще одной интересной возможности операции исключающее ИЛИ.

Как же работает АЛУ? В него поступают два числа, над которыми требуется выполнить действие; часто их называют **операндами**. (Заметим в скобках, что для команд сдвигов второе число — это количество сдвигов.) Вид требуемой операции для рассматриваемого процессора передается АЛУ в виде 3-битового кода. Поскольку RISC-процессоры устроены максимально просто, то эти три управляющих бита просто копируются из кода инструкции. По полученному управляющему коду специальная логическая схема — **декодер** — выберет одно из имеющихся в арсенале АЛУ устройств, и результат операции будет считан на выходы АЛУ именно с него. Например, при поступлении нулевого кода будут подключены к выходу разряды изучаемого нами сумматора. Заметим, что остальные операции тоже могут в это же время параллельно выполняться, но их результаты просто «окажутся невостребованными».

Декодер АЛУ устроен не очень сложно (по сути, это расширенный элемент И), но здесь мы не будем им интересоваться. Любознательным читателям порекомендуем обратиться за подробностями к книге [20]. Еще более детальное описание для начинающих можно найти в книге [34], но ее последнее издание, к сожалению, пока не переведено на русский язык.

7. Выводы

Итак, мы проследили весь путь реализации схемы для суммирования — от математических формул до логической схемы, которую (при наличии желания и технических возможностей) несложно реализовать непосредственно «в железе».

В ходе проделанной работы нам очень пригодились теоретические знания о таблицах истинности, логических формулах и их преобразованиях, основных логических элементах и их работе. Все эти темы уже включены в программу базового школьного курса информатики.

Мы настоятельно рекомендуем дополнительно познакомить учеников с возможностями одного из используемых в современной промышленности языков описания логических схем. Это потребует совсем немного времени, зато мы продемонстрируем школьникам, как выглядит одна из стратегически важных профессиональных областей. Возможно, под влиянием возникшего от рассказа учителя интереса в недалеком будущем у нас появятся свои Теды Хоффы и Фредерико Фаджины [27].

Список источников

1. Автономова Н. С., Филатов В. П. Понимание как логико-гносеологическая проблема // Вопросы философии. 1981. № 5. С. 164–169. EDN: TKGKTR.
2. Андреева Е. В., Босова Л. Л., Фалина И. Н. Математические основы информатики. Элективный курс: учебное пособие. М.: БИНОМ. Лаборатория знаний, 2005. 328 с. EDN: WLQVUT.
3. Бершадский М. Е. Педагогическая диагностика уровня понимания // Педагогические измерения. 2012. № 3. С. 60–88. EDN: OFVQZU.
4. Босова Л. Л. Об использовании графических схем в курсе информатики и ИКТ // Информатика и образование. 2008. № 5. С. 16–25. EDN: IUKTRP.
5. Гнивецкая Т. Н. Основы теории внутрипредметных связей // Физическое образование в вузах. 1999. Т. 5. № 2. С. 23–39. EDN: HTLKDT.
6. Гнивецкая Т. Н., Резник Б. Л., Плотников В. С. Феномен понимания и метод смысловых структур в ИТ-образовании // Современные информационные технологии и ИТ-образование. Сборник научных трудов II Международной научной конференции и XII Международной научно-практической конференции (Москва, 24–26 ноября 2017 года). М.: Лаборатория Открытых Информационных Технологий факультета ВМК МГУ имени М. В. Ломоносова, 2017. С. 68–75. EDN: YTGIV.
7. Гнивецкая Т. Н., Шутко Ю. Е. Внутрипредметные связи в обучении физике. Владивосток: Изд-во Дальневосточного федерального университета, 2023. 208 с. DOI: 10.24866/7444-5575-0. EDN: EYMQOY.
8. Еремин Е. А. Три этюда по компиляции, или Как использовать язык Julia для изучения генерируемого машинного кода // Информатика в школе. 2023. № 6 (185). С. 18–30. DOI: 10.32517/2221-1993-2023-22-6-18-30. EDN: OJCMNC.
9. Заводчикова Н. И., Быкова И. А. О некоторых психологических аспектах реализации внутрипредметных связей в школьном курсе информатики // Актуальные проблемы методики обучения информатике и математике в современной школе. Материалы Международной научно-практической интернет-конференции (Москва, 22–26 апреля 2024 года). М.: МПГУ, 2024. С. 70–78. EDN: GLERCN.
10. Знаков В. В. Психология понимания: Проблемы и перспективы. М.: Институт психологии РАН, 2005. 448 с. EDN: SURUEZ.
11. Инженерный анализ схемы ускоренного переноса процессора Intel 8008 // Хабр. 15 ноября 2020. <https://habr.com/ru/companies/ruvds/articles/528026/>
12. Кэй А. Идеям тоже нужна любовь!.. // Компьютер в школе. 1998. № 1. С. 11–13.
13. Однобитный полный сумматор на необычных микросхемах // Хабр. 8 июня 2029. <https://habr.com/ru/articles/455312/>
14. Пичугина Г. А. Подходы к реализации внутридисциплинарных связей в обучении // Карельский научный журнал. 2021. Т. 10. № 4 (37). С. 24–27. DOI: 10.26140/knz4-2021-1004-0006. EDN: DUPRGD.
15. Платт Ч. Электроника: логические микросхемы, усилители и датчики для начинающих. СПб.: БХВ-Петербург, 2015. 448 с.
16. Поляков К. Ю., Еремин Е. А. Информатика. Углублённый уровень: учебник для 10 класса. В 2 ч. Ч. 1. М.: БИНОМ. Лаборатория знаний, 2013. 344 с. EDN: TUVVWD.
17. Поляков К. Ю., Еремин Е. А. Информатика. Углублённый уровень: учебник для 11 класса. В 2 ч. Ч. 2. М.: БИНОМ. Лаборатория знаний, 2013. 304 с. EDN: TUZDKH.
18. Попов В. С. Изучение законов алгебры логики в симуляторе логических схем // Информатика в школе. 2025. Т. 24. № 6. С. 45–51. DOI: 10.32517/2221-1993-2025-24-6-45-51. EDN: KOAXDR.
19. Справочник по стандартным цифровым ТТЛ микросхемам. Состав серий ТТЛ ИС и их функциональные аналоги в сериях SN74.../SN54... (простая логика) // Институт ядерной физики имени Г. И. Будкера СО РАН. <https://www.inp.nsk.su/~kozak/ttl/ttlh01.htm>
20. Таненбаум Э. С. Архитектура компьютера. СПб.: Питер, 2003. 704 с.
21. Титце У., Шенк К. Полупроводниковая схемотехника: Справочное руководство. М.: Мир, 1982. 512 с.
22. Токсгейм Р. Основы цифровой электроники. М.: Мир, 1988. 392 с.
23. Федеральная рабочая программа среднего общего образования. Информатика (базовый уровень) (для 10–11 классов образовательных организаций). М.: Минпросвещения России; Институт содержания и методов обучения им. В. С. Леднева, 2025. 46 с. https://edsoo.ru/wp-content/uploads/2025/07/2025_soo_frp_informatika_10_11_baz.pdf
24. Фрике К. Вводный курс цифровой электроники. М.: Техносфера, 2003. 432 с.
25. Хамахер К., Вранешич З., Заки С. Организация ЭВМ. СПб.: Питер, 2003. 848 с.
26. Харрис С. Л., Харрис Д. Цифровая схемотехника и архитектура компьютера: RISC-V. М.: ДМК Пресс, 2021. 810 с.
27. Частиков А. П. Архитекторы компьютерного мира. СПб.: БХВ-Петербург, 2002. 384 с. EDN: RJPSKF.
28. Шило В. Л. Популярные цифровые микросхемы: Справочник. Серия «Массовая радиобиблиотека». Вып. 1111. М.: Радио и связь, 1987. 352 с.
29. Шустов М. А. Оптоэлектронные элементы «Исключающее ИЛИ (HE)» // РадиоЛоцман. 13 июля 2020. <https://www.rlocman.ru/review/article.html?di=619699&srsId=AfmBOorX2xT6sI99wgQ2CNO4eHeIPPOkY02FgPTEMHilDOa4WMTyAljB>
30. 3 Bit Binary Adder: 5 Steps — Instructables. <https://www.instructables.com/3-Bit-Binary-Adder/>
31. Fisher K. M. Overview of knowledge mapping // Mapping Biology Knowledge. Science & Technology Education Library. 2002. Vol. 11. P. 5–23. DOI: 10.1007/0-306-47225-2_2.
32. Lee H.-S., Liu O. L., Linn M. C. Validating measurement of knowledge integration in science using multiple-choice and explanation items // Applied Measurement in Education. 2011. Vol. 24. Is. 2. P. 115–136. DOI: 10.1080/08957347.2011.554604.
33. Luengo R., Casas L. M., Mendoza M., Arias J. Possibilities of “Nuclear concepts theory” on educational research, a review // Proc. Int. Conf. “The Future of Education”, edition 1. https://conference.pixel-online.net/conferences/edu_future/common/download/Paper_pdf/SOE33-Gonzalez,Garcia,Garcia,Masa.pdf
34. Petzold Ch. Code: The hidden language of computer hardware and software. Pearson Education, 2023. 480 p.