Георги К. Петров

Дизайн на цифрови електронни устройства с VHDL и Quartus II

1-во издание

част I - основи на булевата алгебра, основи на програмируемите логически устройства

УЧЕБНО ПОМАГАЛО
София 2010

e-Book първа версия 2010
© 2010 д-р инж. Георги Костадинов Петров
Email: gwtt@abv.bg
http://www.instrumentation.hit.bg/

Авторът не поема отговорности относно повреди, причинени на хора, имущество и собственност по смисъла на безопасността и безотказността на продукти, или всякакъв вид методи, инструменти или идеи съдържащи се в изложения учебен материал.

Потребителите имат право да ползват свободно изложения код и примери изцяло на своя отговорност.

Текстообработката на настоящото учебно помагало е направена с OpenOffice. За упражненията и илюстрациите е използван е софтуерен продукт Quartus II Web Edition, който може да се изтегли безплатно от сайта на компанията производител www.altera.com.

Тази страница умишлено е оставена празна.

СЪДЪРЖАНИЕ

Предговор 5

Увод 6

Цифрови и аналогови системи 7

Глава I Основи на булевата алгебра 8

Преминаване от десетична в двоична бройна система 11

Записване на правилна десетична дроб с двоичен код 12

Представяне на неправилна дроб в двоична бройна система 14

Действия с числа записани в двоична позиционна бройна система 14

Глава II Основи на математическата логика. Основни логически схеми 24

Основни понятия във формалната логика 25

Булева алгебра и логически елементи 27

Основни своиства на логическите функции 31

Логически функции реализуеми от две логически променливи 32

Преобразуване на логически изрази 35

Логически елементи 40

Класификация на логическите електронни схеми 44

Глава IV Основни закони и тъждества в булевата алгебра 46

Функционално пълни системи 47

Връзка между аналитичен запис електронна схема 50

Симулация на елемента на VHDL 51

Нормална и канонична форма на запис на логически функции 54

Минтерм 57

Макстерм 61

Връзка между минтерм и макстерм 63

Преобразуване на логически функции 64

Аналитичен метод 65

Систематичен метод 68

Метода на Куайн-Мак Класки 68

Метод на Карно 71

Глава V Архитектурни особености на CPLD и FPGA 77

Семейства CPLD и FPGA предлагани от Altera 82

Литература 85

Предговор

Настоящото учебно помагало е предназначено за използване от студенти и специалисти работещи в различни области на хардуерното и софтуерно производство. Основното предназначение е да се представи теоретичното съдържание на курсовете свързани с програмируемите логически устройства, които авторът води в департамент Телекомуникации на НБУ. Вниманието е фокусирано върху създаването на цифрови електронни устройства и модули чрез езика за поведенческо описание на цифрови интегрални схеми - VHDL (Very High Speed Integrated Circuit Hardware Description Language). За правилното използване и пълноценен дизайн на електронно оборудване, читателят би следвало допълнително да се запознае с други материали касаещи цифровата и аналогова схемотехника, а също така и да придобие допълнителни познания по програмиране на езици от високо ниво C/C++. Едновременно с това книгата може да бъде четена от съвсем начинаещи в областта, а при необходимост читателят да се обърне към цитираните литературни и Интернет източници. Разглежданите VHDL примери и приложените са оптимизирани за работа с интегрирана среда за разработка Quartus II Web Edition - производство на корпорация Altera. Самите VHDL кодове на могат да се ползват и в други среди за рабработка, като Web ISE производство на корпорация Xilinx.

Организацията е в 3 части:

- В първата част е направен кратък увод в булевата алгебра и цифровите интегрални схеми използвани за дизайн на електронно оборудване, този раздел е максимално олекотен и лесно разбираем дори за начинаещи;
- Втората част включва основните концепции и езикови конструкции на VHDL при реализацията на комбинационни и последователни логически блокове, както и основи за работа с Quartus II;
- Третата част разглежда приложение на VHDL в реализацията на цифрови модули за комуникация, сигнална обработка и програмирането на елементарно процесорно ядро.

Всички изходни кодове на разглежданите проекти могат да бъдат свалени свободно от сайта www.instrumentation.hit.bg. Ще Ви бъдем благодарни, ако изпращате своите препоръки и забележки на електронната поща на автора gpetrov@nbu.bg.

Увод

В част I на тази ще се запознаете с основните понятия и правила на булевата алгебра. Ще се даде обща представа за методите на синтез и дизайн на комбинационни логически елементи, както и основите на оптимизацията на логически функции. Естествено е желанието на всеки начинаещ програмист веднага да напише своята програма. Типично приложение, с което повечето книги за програмиране започват е „Hello World“. Проблемът с VHDL, е че в този случай вие сте човекът който създава компютъра, на чийто екран ще се изпише „Hello World”, а това изисква от вас да имате малко повече познания от простото натискане на бутоните.

Авторът се е опитал максимално лаконично да представи повечето понятия ползвани от булевата алгебра, и без които дизайнът на електронно оборудване е невъзможен. С кратки решени примери ще разберете как цифровите компютри събират, изваждат, умножават и делят. Щ добиете идея за това как се синтезират комбинационни логически устройства, какви основни логически функции съществуват, как се използват техните таблици на истинност, и как се извършва оптимизация на логически функции.

Предвидена е отделна част даваща основните понятия за функционирането на цифровите интегрални схеми и програмируемите логически устройства, тяхната архитектура и някои по-важни особености при дизайна на хардуерни блокове с тях. Фокусът е поставен над този минимум факти, без които дизайнът на цифрови устройства е немислим. Следва да се каже, че читателите трябва да използват посочените в края литературни източници, както и дадените на края на всяко упражнение линкове, за да придобият достатъчно изчерпателни знания.

В част II на книгата ще се запознаете с основните концепции и понятия във VHDL. В различни примери и задачи ще може да добиете умения и разбиране за използването на езикови конструкции и софтуерни инструменти за разработка на електронни блокове, модули и тяхната симулация чрез Quartus II Web Edition.

В част III ще се запознаете с примерни дизайни на 8 битово процесорно ядро, блокове за серийна комуникация, блокове за кодиране на данните, блокове за цифрова сигнална обработка.

Желаем ви приятно четене.

Цифрови и аналогови системи

Цифрова система (дискретна) ще наричаме такава комбинация от устройства специално разработени за обработка на логическа информация или физически величини представени в цифров вид. Тези устройства принципно са електронни, но могат да бъдат и механически, магнитни или пневматически. Най-силно разпределените цифрови системи в световен мащаб днес се явяват цифровите компютри, във всичките си разновидности и приложения, цифровото оборудване за обработка на аудио и видео сигнали и разбирасе телекомуникационните системи. Естествено цифровите системи, колкото и съвършенни взаимодействат с околния свят, а именно с аналогови сигнали. **Аналоговите системи** (непрекъснати) ще наричаме такива, които съдържт устройства опериращи с физически величини представени в аналогова форма. Аналоговите системи са неизменна част на всички съвременни комуникационни, мултимедийни, измерителни и управляващи системи. Макар настоящата книга да се фокусира над принципите за дизайн на цифрови електронни модули, читателят не бива да забрява, че за да се осъществи взаимодействието на цифровите системи със заобикалащият ги смят са нужни цифрово аналогови и аналогово цифрови системи, а също така и редица чисто аналогови устройства, като: усилватели на мощност, преобразуватели на ток в напрежение, модулатори, високочестотни генератори, сензори и други. Книгата касае дизайна на цифрови електронни устройства с VHDL и QuartusII. Нека се спрем над основните преимущества на цифровите системи:

- Цифровите системи за обработка на информация, като правило се разработват по-лесно от аналоговите им конкуренти;
- Относителната цена на цифровите интегрални схеми е далеч по-ниска в сравнение с относителната цена на подобни аналогови интегрални схеми и компоненти, като цената на крайното решение е често несравнима;
- Представянето, оперирането и съхраняването на информация е далеч по-ефективно да става в цифров вид;
- Предаването и приемането на информация е по-оптимално да става в цифров вид, по-лесно се корегират възникналите грешки;
- Цифровите схеми са по-устойчиви на смущения, а ползваните методи за обработка на информацията ги правят по-евтини;
- Голямо количество схемотехнически решения може да се разработи посредством цифрови интегрални схеми и процесори, като времето за разработка е многократно по-малко отколкото разработката на аналогови схеми;

- Цифровите устройства могат да променят своята функционалност лесно, само чрез презапис на програмите които ги управляват, аналоговите устройства изискват сложна промяна на схемата, а често това е невъзможно.

Най-често при разработката на цифрови системи се използват интегрирани процесорни схеми, съвместно с различни по сложност и предназначение интерфейсни схеми, АЦП, ЦАП, комуникационни модули, памети и други интегрални схеми. Съвременният дизайн на оборудване залага на ползването на мощни процесори, като функционалността на устройството в голяма степен зависи от създадения софтуер, а не толкова от фиксираните фабрично възможности на системата. Това позволява бързата смяна на предназначението на стандартни цифрови модули, реализацията на нови функционалности в тях, отдалеченото им наблюдение и препрограмиране. Между използваните интегрални схеми и процесорни модули можем да отделим един специфичен вид, който позволява да се реши дизайнът на цяла интегрирана система използвайки един единствен чип с набор интерфейсни схеми. Това са програмируеми цифрови интегрални схеми с техните разновидности: **PLD** - *programmable logic device***, CPLD** - *complex programmable logic device* **, FPGA** - *field-programmable gate array*, за които ще стане дума по-късно. Възможността на дизайнера да създава цифров хардуер, точно отговарящ на неговите нужди, съчетавайки възможности за интеграцията на стандартни процесорни блокове, памети и комуникационни интерфейси в един завършен дизайн с новите хардуерни блокове, позволява съществена оптимизация в процеса на проектиране и производство на нови цифрови системи. Това снижава крайната цена и подпомага бързото създаване и пускане в продажба на нови електронни системи, които могат чисто софтуерно да бъдат преконфигурирани или модифицирани в последствие. Чрез инсталирането на различни нови конфигурационни файлове, или цялостно препрограмиране на системата е възможно нейното многократно ползване за решаване на различни задачи. Много добър пример е ползването на FPGA при поризводството на WiMax и DVB базови станции, където без възможност за бърза преконфигурация на цифровите блокове въвеждането на нови стандарти за комуникации би било немислимо и крайно скъпо. Изобщо навсякъде където се налага интегриране на стандартни изчислителни процеси, със системи за сигнална обработка, предаване и съграняване на данните, системи за кодиране и разпознаване на сигнали много ефективно е ползването на FPGA и хибридни FPGA-DSP базирани устройства. Естествено уменията за работа с тях надвишават далеч класическите знания на програмистите и схемотехниците. Въпреки, че за дизайн се ползват софтуерни приложения, са нужни и широкоспектърни знания в различни области.

Основи на булевата алгебра

След усвояването на материала в тази глава, ще можете да разбирате защо цифровите компютри използват двоична бройна система, ще разбирате как става преминаването от десетична в двоична бройна система и обратното, и ще разбирате основните действия с двоични числа: събиране, изваждане, умножение и деление. Изложеният материал не може да се счита за максимално изчерпателен, по-скоро има за цел да представи само нужната информация и разбиране относно функционирането на цифровите компютри. В следващите глави по-подробно ще се спрем над различните видове логически функции, начините за тяхната оптимизация. Няма да изясняваме голямото значение на **информационните технологии използвани** при осъществяването, на всички съвременни постижения на човешкото общество. Само ще отбележим, че: болшинството от машините обработващи и предаващи информация работят благодарение на свойствата на електричеството - да протича ток в заворена верига или да не протича в отворена верига, както и възможностите на математиката да представя и извършва математически операции с числата в различни позиционни бройни система. Използваният в компютрите машинен език е базиран върху основата на двоичната позиционна бройна система която използва само два символа: '0' и '1'.

Пример:

Нека е дадено числото 358,992 записано в десетична бройна система, имаща за основа числото 10. За да означим вида на използваната основа, поставяме числото в малки скоби и долу вдясно, след скобата нанасяме, като индекс (в случая) числото 10

ф.1 $(358{,}992)_{10}$, че числото е представено в десетичен код

Всяко число е възможно да бъде представено в различни позиционни бройни системи. Минималният брой символи, достатъчни за запис на числата се постига при двоична позиционна бройна система, където числата се представят като серии от единици и нули. При компютрите често се използва и шеснайсетичната бройна система, а някои компютърни системи обработват числата и в осмична бройна система. *Заб. В позиционните бройни системи тежестта на всяка цифра зависи от мястото в символичния запис, такава е десетичната бройна система. Съществуват и непозиционни системи, така е при римските числа (I, II, III, IV, VI). При тях стойността на цифрата не зависи от мястото, което заема в числото (IV и VI).Тези бройни системи са прекалено трудни за сложни изчисления, не случайно умението да събираш и изваждаш с римски цифри се е считало за наличие на „висок образователен ценз“ от умеещия да го прави.* Всяко число, записано в позиционна бройна

система може да се представи във вида:

ф.2 $A = a_n * q^n + a_{n-1} * q^{n-1} + ... + a_1 * q^1 + a_0 * q^0 \ , \ a_{-1} * q^{-1} + ... + a_{-m} * q^{-m}$

тук $a_n, a_{n-1} ... a_{-m}$ са цифрите заемащи съответна позиция в записа на числото

q - е основата на използваната позиционна система
n - е номер на разряда (бита)
q^n - може да се нарече условно „тегло“ на n-тия разряд
, - разделя цялата от дробната част, както в обикновено число

Пример:

Числото 358,992 в десетичната позиционна бройна система ще изглежда така/или може да бъде представено с **релацията**/:

ф.3 $A = (358,992)_{10} = 3 * 10^2 + 5 * 10^1 + 8 * 10^0 \ , \ 9 * 10^{-1} + 9 * 10^{-2} + 2 * 10^{-3} =$
$= 300 + 50 + 8 \, , \, 0.9 + 0.09 + 0.002$

символът „ **,** “ разделя цялата от дробната част в числото

След края на всеки обособен теоретичен и практически раздел ще бъдат давани задачи за упражнения, характерът на задачите има за цел да даде увереност на четящите за разбиране на нещата, а не за придобиване способности да изчисляват сложни практически задачи самостоятелно. За придобиването на подобни умения следва да се обръщате към други източници, а най-често www.google.com ще даде нужният ви отговор. В обема на тези лекции не се разглеждат общите методи за записване на числата в различните позиционни системи и действията с тях ще се занимаваме с основни елементи от двоичната позиционна бройна система.

Упражнение:

- дайте пример за запис на число в позиционна и непозиционна система;
- означете числото 102,534 че принадлежи на десетичната позиционна бройна система;
- напишете произволно трицифрено цяло число съгласно **ф.2**, развито по степените на основата използвана при записа на числото;
- напишете произволна неправилна дроб съгласно **ф.2**;

Преминаване от десетична в двоична бройна система

Възможността за преобразуване на числа представени в десетична бройна система в двоична, касае реализацията на схеми позволяващи правилното въвеждане на информация в цифровите компютри, а също така и улесняване на работещите при извеждане и разбиране на отработената от компютрите информяция. Много често компютърните системи използват различни видове извеждане и въвеждане на данните, тъй като чисто двоичният им вид е крайно неудобен за хората се ползват двоично шеснайстични кодове и десетични кодове. Съществува цяла гама от специализирани интегрални схеми осъществяващи процесът на преобразуване на числата от една в друга форма.

Алгоритъм за преминаване от десетична в двоична бройна система:

Дадено е числото $(88)_{10}$, започваме да делим по следния начин:
88 : 2 = 44 остатък 0 = a_0 , това е най-нисшият разряд в числото
44 : 2 = 22 остатък 0 = a_1
22 : 2 = 11 остатък 0 = a_2
11 : 2 = 5 остатък 1 = a_3
5 : 2 = 2 остатък 1 = a_4
2 : 2 = 1 остатък 0 = a_5
1 : 2 = 0 (цяло или остатъкът е 1) = a_6 , това е най-висшият разряд в даденото число

Получените остатъци, четени отдолу нагоре са записа на числото 88 записано в десетична система, като двоичното число 1011000, при това съществува еднозначно съответствие мужду коефициентите a_0 до a_6 съгласно таблицата:

a_6	a_5	a_4	a_3	a_2	a_1	a_0
1	0	1	1	0	0	0

Упражнение:

- да се напишат числата от $(0)_{10}$ до $(6)_{10}$ в двоична система по начина даден по-горе

Алгоритъм за преминаване от двоична в десетична бройна система:

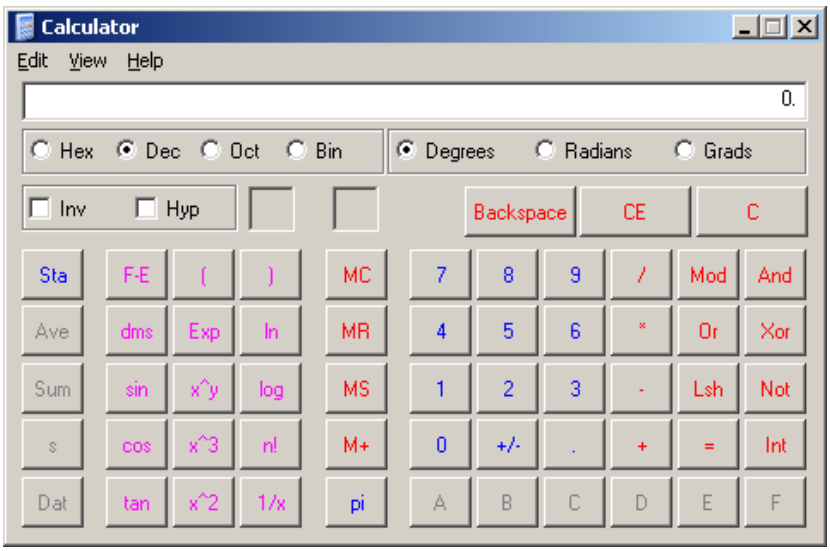
Дадено е числото $(1011000)_2$, преминаването в десетична бройна система става по следния начин: започваме от цифрата в най-младшия разряд на двоичното число (от дясно наляво):

$$0*2^0 + 0*2^1 + 0*2^2 + 1*2^3 + 1*2^4 + 0*2^5 + 1*2^6 =$$
$$= 0 + 0 + 0 + 8 + 16 + 0 + 64 =$$
$$= (88)_{10}$$

Упражнение:

- да се напишат числата $(111)_2$, $(1000)_2$, $(1111)_2$ в десетична система по начина даден по-горе

Заб: Във вашата практика често ще се налага да извършвате конвертиране на стойности от десетична в двоична и шеснайсетична бройни системи. Въпреки че компютърните програми които пишем ще правят това вместо нас, дефакто в практиката се налага интензивна работа с двоични и двоично шеснайсетични формати на числата. За да си улесните работата ползвайте програмите като Calcualtor интегриран в Windows 95, 98, XP, Vista (фиг. 1). Използвайки менюто View изберете режимът на работа да бъде Scientific, за да можете лесно и бързо да преминавате от една в друга бройни системи избирайки радио бутоните – Hex, Dec, Oct, Bin. Предизвикването на „усмивки“ при началното усвояване на този инструмент е крайно неоснователно, тъй като е доказано от практиката, че ползването му спестява много време и нерви в процесът на работа.



Фиг. 1 Калкулатор

Записване на правилна десетична дроб с двоичен код

Алгоритъм за преминаване от десетична правилна дроб в двоична система: Дадено е числото $(0{,}3125)_{10}$, взимаме частта отдясно на десетичната запетая, като нулата пред десетичната запеая нанасяме в ляво от дробната част:

0 3125 , тук 0 = a_0
*2
0 6250 , тук 0 = a_{-1} .Следващия разряд е числото 1, но не сме го прехвърлили и за това пишем 0
*2
1 2500 , в ляво се появява 1, тъй като сме надхвърлили разряд.
тук 1= a_{-2}
*2
0 5000 , тук 0 = a_{-3}
*2
1 0000 , тук 1= a_{-4}

Вертикалната колона се чете отгоре надолу и се записва числото (0,0101) в двоичен вид. Проверката става по **ф.3** за дробната част (след десетичната запетайка в многочлена) съставяме таблица както следва.

a_0	a_{-1}	a_{-2}	a_{-3}	a_{-4}		
0	0	1	0	1		

Заместваме във **ф.3** при основа 2 и получаваме:

$A = a_0 * 2^0$, $a_{-1} * 2^{-1} + a_{-2} * 2^{(-2)} + a_{-3} * 2^{(-3)} + a_{-4} * 2^{-4}$

$A = 0 * 2^0$, $0 * 2^{-1} + 1 * 2^{(-2)} + 0 * 2^{-3} + 1 * 2^{-4}$

0 + 0 + ¼ + 0 + 1/16 =
= 5/16 = 0,3125 в десетична система

Упражнение:

- да се напишат числата от 0.125 0,375 в двоична система (време за изчисляване 5 минути) отговори: 0.001 0.011 в двоична система

Представяне на неправилна дроб в двоична бройна система

Дадена е неправилната десетична дроб $(53,125)_{10}$ в двоичен код. Отделя се цялата част от дробната, като двете части се преобразуват самостоятелно:

53 : 2 остатък **1**
26 : 2 остатък **0**
13 : 2 остатък **1**
6 : 2 остатък **0**
3 : 2 остатък **1**
1 : 2 остатък **1**

Четем отдолу нагоре и записваме цялата част в двоичен код е 110101.

За да преобразуваме дробната част извършваме следното действие:

0 125
*2
0 250
*2
0 500
*2
1 000

Четем от горе на долу, като дробната част в двоична ситема е 0.001.
За да запишем числото в двоичен код, събираме цялата и дробната му части:

110101,000
+
000000,001

110101,001

Упражнение:
- Конвертирайте числото 4.375 в двоичен код - отговор 0100,011
- Конвертирайте 9.345 в двоичен код - отговор 1001,0101100001...

Обикновено при използване на десетична броина система основата 10 не се отбелязва , като индекс.

Действия с числа записани в двоична позиционна бройна система събиране, изваждане, умножение и деление

Ще се занимаваме с реалните числа:

- Естествените 0,1,2,3,4,.....N,...
- Целите ..., -N, ... ,-3-,2-,1, 0, 1 ,2 ,...,N,...
- Рационалните имат вида p/q, където p и q са цели числа и q е различно от 0
- Ирационалните числа: *pi*

В сила остават всички математически закони /свойства/, използвани при извършването на основните действия и сравняването на реалните числа. При работа с числа в двоичен код е добре да се помни, че:

ф.4 $(1)_{10}=(1)_{(2)}=1*2^0=1=01=0001=...$

В двоична бройна система нулите пред цифрата '1' не влияят на изобразената стойност. Броят на нулите се избира по други критерии, например зависещи са от характеристиките на елементите използвани в процесорите или от начина на представяне на цифрите в различните работни регистри. Например числото 984 е записано в десетична и двоично десетична форма, която е удобна при въвеждане на данни от клавиатури и извеждане на информация на дисплеи:

ф.5 $(984)_{10}=(1001)_2\ (1000)_2\ (0100)_2$

Упражнение:

- Запишете числото 498 в двоично десетична форма
- Запишете числото 583 в двоично десетична форма

Събиране на числа записани в двоична бройна система

За да събираме двоични числа може да ползваме таблица за събиране или правила за събиране:

ф.6
0 + 0 = 0
0 + 1 = 1
1 + 1 = 10 (нула и пренос на 1 в следващия разряд)

Пример:
Да представим по-долу дадената сума в двоичен код:

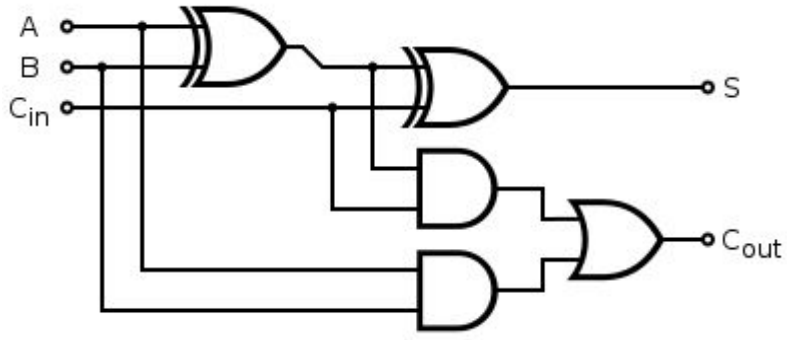
7 + 5 = 12

Извършваме действието събиране /напълно аналогично на събиране в десетичната бройна система/. Реализацията на действието събиране с пренос чрез логически елементи е дадено на Фиг. 2.

0111
+
0101
=
1100

Проверка:

$$0*2^0+0*21+1*2^2+1*2^3=4+8=12$$



Фиг. 2 Схема реализираща действието сумиране на 2 бита с пренос и отчитане на преноса от предния разряд

За онези, които имат бегли познания по цифрова схемотехника Фигура 2 е разбираема. Независимо от това следва да кажем, че реализираната схема е комбинационна логическа схема. Често този тип схеми са прекалено сложни и

затова тяхната функционалност се задава с така наречените таблици на истинност. Таблицата на истинност ви позволява да добиете представа за функционирането на дадено устройство при различни комбинации от '1' и '0' подадени на неговите входове. Таблицата на истинност на пълния суматор е дадена по-долу (Табл. 1).

CARRY IN	*input B*	*input A*	*CARRY OUT*	*SUM digit*
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Табл. 1 Таблица на истинност на пълен еднобитов суматор с флаг за пренос от младши разряд и в старши разряд

Изваждане на числа записани в двоична бройна система

Когато умаляемото е по-голямо от умалителя действието се извършва без проблеми и е аналогично на действието в десетична позиционна бройна система. Необходимо е да се помни, че в десетичната система една единица от по висш разряд съдържа десет единици от по нисшия преди него, а при двоичната система разряда съдържа само две единици вместо десет.

Пример:
5 - 2 = 3 в десетична система и

0101
-
0010

0011

(когато вземем „в заем“ 1 единица от по-висш разряд не трябва да зябравяме, че в по-нисшия тя съдържа 2 единици)

Съществен проблем би възникнал ако спазим същото правило в случаите, когато умаляемото е по малко от умалителя. Разликата трябва да е отрицателно число. Знаците „**+**“ и „**-**“ се записват в разрядната решетка на паметта със символа:

0 за +
1 за **-**

Проблема за получаване на знака се решава с използването на т.н. допълнителен код. Това е допълнението на разглежданото число до най-близката по-голяма цяла степен на основата на бройната система, към която принадлежи числото. Погледнете следващия пример.

Пример:
В десетична бройна система е дадено :

832 - 523 = 309

Най-близката степен, която е по-голяма от умалителя е **1000**
Допълнението на **523** до **1000** е:

1000 – 523 = 477 (нарича се допълнение на 523 до 1000)

Действието изваждане е възможно да се замени с действие събиране, като се събира числото на умаляемото с допълнението на умалителя **- 523**:

832 + 477 = 1 309 (или 1 в това число ще носи знака **- .**

Числото 309 ще представлява големината на търсената разлика. Ако разгледаме числото 523 ,като развито по сепените на на 10 ще получим 5.10.10 + 2.10 +3.1. Тогава допълненията за всеки разряд ще бъдат :

- за разряда на единиците 10 -3 = 7,
- за разряда на десетиците 9 – 2 = 7 и
- за разряда на стотиците 9— 5 = 4 .

Ако четем цифрите отдолу нагоре и ги запишем хоризонтано ще олучим същото число **477**,което е допълнението на **523** до **1000**.

Как е организирано извършването на алгебрическото сумиране при използване на числа записани в двоична позиционна бройна система? Без съмнение правилата, които използвахме за една озиционна системаса в сила иза

всички останали позиционни бройни системи. Нека видим следния пример.

Пример:
2 - 5 = - 3 в десетична бройна система, това може да се запише така

-1 . (5 - 2) = - 3

Записваме абсолютната стойност на числата в двоичен код
0101 $(5)_{10}$
0010 $(2)_{10}$

намираме допълнението на 2 до 2, това става на два етапа, първо инвертираме всички значещи битове и след това прибавяме 1

0010
инвертираме и получаваме
1101
+
0001

1110 това е допълнителния код на **0010** до **2**

извършваме събиране по указания по-долу начин

1 1 1 0
+
0 1 0 1
----
1 0 0 1 1

Първата единица в ляво (от страна на висшите разряди) не се взима предвид. Частта **0011** е точно абсолютната стоиност на разликата 3 или

0011 **- $(3)_{10}$**

Упражнение:
59 – 52 = 7 изчислете израза в двоична бройна система.
52 – 59 = -7 изчислете израза в двоична бройна система.

Заб. Ползвайте следната форма на запис -1(59 – 52) и извършете събирането с допълнителен код на 52 до 2

Умножение на числа записани в двоична бройна система

За да извършваме умножението на числа в двоична бройна система ще ползваме таблица за умножение. Следва да си припомним, че действието умножение е събиране на еднакви множители.

ф.8
0 * 0 = 0
0 * 1 = 0
1 * 1 = 1

Пример 1:
2 * 3 = 6 в десетична система

0010 * 0011
0010
+
0010
00110 $(6)_{10}$

Упражнение:
5 * 3 = 15 получете произведението в двоичен код

Пример 2:
53 * 11 = 583

110101 * 1011
110101
+
110101-
+
000000--
+
110101 ---
1001000111

При реализацията на действие умножение се използват специализирани схеми наричани матрични умножители. Тяхната структура е така направена, че позволяват бързото събиране на всички множители за един процесорен цикъл. В този подход отново се ползват допълнителни кодове за представяне на отрицателни числа преди умножението им. Ако двете числа са отрицателни се представят и двете в допълнителен код, ако само само едното е отрицателно само то се представя в допълнителен код.

Деление на числа записани в двоична бройна система

Деленето на числа записани в двоична бройна система се извършва подобно на делението на две десетични числа. За разбиране на действията следва да помним, че делението е обратно действие на умножението, а умножението е събиране на еднакви множители.

Пример.
ф.9
6 : 2 = 3 това може да се получи и по следната схема:

Тук **2** е **делител**, а **6** е **делимо**. Вадим от 6 числото 2, докато в резултата не получим 0
6 – 2 = 4
4 – 2 = 2
2 – 2 = 0

Преброяваме двойките, които сме отнемали от **6-те**, а те са са **3**. Следователно частното е 3.

Ще разгледаме случай на деление - при делимо и делител с цели числа. Случаят с делимото и делителя съдържащи дробни части, може да се сведе до деление на цели числа. За десетична система например имаме отношението 0,75 / 0,25 ако умножим x100 ще получим цели числа в числителя и знаменателя 75/25. Този „трик“ често се ползва при реализацията на сложни алгоритми за цифрова сигнална обработка, особено когато хардуерното устройство или процесор не притежава оптимизирани хардуерни модули за умножение и деление на дробни числа. При двоичните числа е необходимо, просто да умножим и двете части на отношението с едно и също число, равно на цяла степен на числото 2. /Например: 4, 8, 16,..., това действие се извършва елементарно с преместване на двоичното число в преместващ регистър./

Определяме броя на значещите разряди в частното. Отделяме такъв брой разряди, започвайки от висшия разряд в делимото. При това ако отделената част в делимото е число равно или по голямо от делителя поставяме окончателно една разделителна вертикална линия за визуално представяне на необходимият ни интервал. Ако делимото е по голямо - преместваме вертикалната линия с един разряд в посока на нисшият и сме готови за първото изваждане на делителя. Общото правило е, че броят на разредите в набелязаният от нас интервал се равнява или е с един по голям от броя на разредите в делителя. Ако това условно отделяне е възможно, в старшият разряд на частното се записва 1. В противен случай се записва 0. Ако се окаже, че е възможно отделянето на такава група /интервал/, записваме частното под определения от нас интервал и извършваме действието изваждане. Сваляме следващата двоична цифра от

записа на делимото и отново анализираме разликата, като сравняваме стойността й с тази на частното. Тези действия са аналогични на проведените в началото. Ако стойността на новообразуваното число е по-малка от стойността на делителя, то пишем нула в реда, където се формира числото на частното и сваляме следваща цифра. Процедурите продължават до изчерпване на делимото. Ако след сваляне на последната цифра от делимото получената разлиха не е 0 то до тук свършва цялата част на частното и ние може да продължим делението сваляйки 0 от делителя. Това продължава до достигане та необходимата точност.

ф.10
Пример:
567 : 21 = 27

1000110111 : 10101 – делител
цифрите на частното се записва под делителя
100011 | 0111 : 10101
- 10101 **11011 = 27** (записваме 0 при повторно сваляне на значеща стойност)
011100
- 10101
0011 111
- 10101
010101
- 10101
00000

Делителя има 5 разряда. Отделяме 5 разряда в делимото, започвайки от висшият му разряд. Числото в делителя ще бъде **10001**, но то е по-малко по стоиност от делителя **10101**. Необходимо е да предвидим по голям разряд за да извършим изваждането, заради това увеличаваме избраният интервал с още една цифра в посока на нисшият разряд на делимото т.е. числото в определения от нас интервала ще бъде **100011**. Сега делителя е по малък и е възможно да извършим изваждането. При това записваме и първата цифра в числото на частното '**1**'. Тя лежи в най-висшият му разряд. Числото на умалителя е числото на частното. Записваме го съгласно схемата или низшият му разряд трябва да лежи под низшият разряд на числото в маркирания от нас интервал. Намираме разликата. Тя е **01110** сваляме следващата цифра от делимото '**0**'. Получаваме двоично число **011100**, то е по-голямо число от делителя и пишем нова единица в мястото на частното. Намираме отново разликата. Тя е **00111** и е по-малка от делителя. Пишем '**0**' в частното и сваляме следваща цифра от делимото. По този начин умаляемото става **0011111** и е по голямо число от делителя Записваме по

този повод още една единица в частното и търсим разликата. Получаваме **01010**. Сваляме последната единица от делимото и умаляемото става **010101**, което е равно на делителя **10101**. Това ни позволява да запишем още една единица в частното '**1**'. Тъй като новата разлика е **000000**, то делението е завършено частното ще бъде двоичното число **11011**.

Заб. Деленето на числа в двоична бройна система е крайно неудобно за хората, а също така и за компютрите. Едва малък процент от стандартните процесори и микроконтролери имат интегриран хардуерен блок за делене на цели числа и то предимно 32 битовите контролери. Ето защо деленето е процес отнемащ значителен машинен ресурс. Често се използват специални хардуерни архитектури. Когато се реализират прецизни хардуерни архитектури за обработка на сигнали или изображения ползването на специализирани целочислени и дробни модули за умножение и деление в значителна степен повишава бързодействието на системите. Именно тук активно се ползват т.н. процесори за цифрова сигнална обработка DSP. Болшинството FPGA притежават интегрирани DSP макро блокове.

За повече информация:

„SYNTHESIS OF ARITHMETIC CIRCUITS FPGA, ASIC, and Embedded Systems, JEAN-PIERRE DESCHAMPS“, JEAN-PIERRE DESCHAMPS - University Rovira i Virgil, GE´RY JEAN ANTOINE BIOUL - National University of the Center of the Province of Buenos Aires, GUSTAVO D. SUTTER - University Autonoma of Madrid, A JOHN WILEY NS, INC., PUBLICATION

Основи на математическата логика
Основни логически схеми

Основата на изчислителната техника се крие в съждително смятане /математическа логика/ или наричана още Булева Алгебра. Идеята за създаване на наука, в която понятията биха могли да се означават със символи и чрез математиката да се оперира с тях е изразена от Готфрид Лайбниц /1646-1716г./. Считало се, че така ще стане възможно да се достига до истината без излишни спорове между учените. /И до днес съществува мнетието, че във спора се ражда истината/. Без правилата на логиката трудно би могло да се стигне до истина. Джордж Бул - син на обущар, създава съждителната математика, днес наричана Булева алгебра. Той успява да съчетае точните методи на познанието - присъщи за математиката и чисто описателни методи на логиката. Преоткриването на Булевата алгебра дължим на Клод Шенон през 1938г. **Клод Шенон доказва, че Булевата алгебра е напълно подходяща за анализирането и синтеза на релейни и превключващи /комутиращи/ електрически схеми и вериги.** Много съществени, опасни и тежки, еднообразни операции, извършвани от машини облекчаващи човешките дейности се управляват и извършват благодарение на усилването и регулирането. По своята същност механизмите и устройствата използвани за посочените цели могат да бъдат механически, хидравлични и електрически. В основите си принципа на усилването изисква слаб команден сигнал, който да включи или изключи по наше желание някаква силова верига /без значение от вида и/. Тъй като най-бързодействуващи са електрическите системи, защото можем да приемем, че при тях сигнала се предава със скоростта на разпространение на елекро-магнитното поле в дадената среда, именно този тип устройства се ползват за направа на изчислителни машини и автомати. /Следва да отбележим, че живите организми също функционират благодарение на електрическите сигнали./ Всички видове устройства проявяват инертност при предаване на сигнала: в механичните това е времето за обиране на луфтовете, в хидравлика времето за достигане на определено налягане отпушващо вентилите, при електрическите и електронните устройства това са зоните, в който действуват преходните процеси. Поради тези причини не можем да създадем безкрайно бърза изчислителна система. Тъй като действието на разглежданите по-долу логически схеми ще се обяснява с действието на електромагнитните релета, а именно „включено“ и „изключено“, следва да споменем, че най-старият „усилвател“ в електротехниката е електромагнитното реле. Болшинството електронноизчислителни устройства днес работят с двоичен код, ето защо ползването на булевата алгебра за тяхното обясняване и синтез е оправдано. Не бива да се забравя, че изпълнението на логически операции не е приоритет само на компютрите. С най-обикновени превключватели /механични, хидравлични, оптични, топлинни се осигурява

задействуването на механизмите в самолетите, автомобилите и други химични и физични технологии.

За повече информация:
http://computerscience.jbpub.com/ecoa/2e/Null03.pdf
http://webster.cs.ucr.edu/AoA/DOS/pdf/ch02.pdf

Основни понятия във формалната логика

Логиката е наука за правилността на мисленето.

Формалната логика е наука, която изследва структурата на формите и операциите на правилното мислене или се занимава с елементарните закони и формите на мисленето. /форма-външен вид, структура на нещо, формал произхожда от итал. formal-който се отнася до калъп/ Формален, в смисъл на който става /извършва се/ по законен ред. В математическата логика употребата на термина „формален “, няма нищо общо с преносното значение на думата в говоримият език. Логическите закони отразяват в съзнанието на хората свойствата, връзките и отношенията между заобикалящите ги обекти. Но не се интересуват от съдържанието им. Интересуват се само от тяхната истинност или не истинност ('1' или '0')! Основните форми с които борави нашето съзнание са понятията и твърденията.

Понятие означава форма на мисленето, която отделя съществените признаци на някой предмет или клас предмети, по които се различават от другите предмети.

Пример:

Компютър, човек, куче, писалка. С помощта на понятията се изграждат твърдения /най-често под формата на изречения/.

Пример:

„Лесно се събудих.“ - Това е твърдение, но не можем да отсъдим дали е вярно или не.

„Бягай по-бързо!“ - Това е твърдение, но и за него не е възможно да отсъдим каква истинност има.

Съждението е твърдение за което можем да съдим, дали е вярно или не е вярно. Езиковата форма на съждението е разказвателно изречение. Въпросителните и заповедни изречения не са съждения. Във формалната и

математическата логика не се интересуваме от съдържанието на дадено съждение. Интересуваме се само от неговата истинност или не истинност /вярност или лъжовност, ДА или НЕ/. Съжденията обикновено ще бележим с главни букви, както на дадения по-долу пример:

A, B, C, D, E, F,...

A = 2 x 2 = 4 е **вярно** /истинно/ съждение

B = Паякът е бозайник е **не вярно**, лъжовно или неистинно съждение /паяците спадат към членостеногите/

Упражнение:

Отсъдете истинността на съжденията.

C = професорът е печатещо устройство – не е истина

D = Навън вали дъжд. /отсъдете спрямо момента в който сте прочели това/. Съждението **D** има променлива стойност на истинност за различни дни и часове. Всяко съждение би могло да проявява **истинност** или **не истинност**, т. е. да приема стойности **'1'** или **'0'**. Освен това е възможно да се проявява и като променливо според други обстоятелства, т.е. запазва се като променлива по отношени на истинността или не истинността. Бележи се със символ за променлив алгумент „**X**“ или „**НЕ X**“, последното е отрицанието на „**X**“.

Пример:

Ако отбележим истинността на твърдението, че “**Слънцето грее**“ с „**X“,** то ако слънцето не грее това ще се отбележи с „**НЕ X“**. Истинността им за даден момент от време ще преценим по състоянието в момента.

Булева алгебра и логически елементи

Операциите извършвани в Булевата алгебра са сходни с действията и правилата в елементарната алгебра. Булевата алгебра е алгебрична структура, която съдържа логичните оператори **И, ИЛИ, НЕ**, както и множествените функции **сечение, обединение, допълнение**. Операторите с които се означават различните функции – операции, се срещат често написани по различен начин:

И	**ИЛИ**	**НЕ**
AND	**OR**	**NOT**
∧	∨	¬ математиците често използват тези означения
·	+	**черта** над буквата означаваща съждението за НЕ
&	|	**!=**

В този учебник ще използваме следните означения:
∧ **(И)** - произлиза от от латинското **„Aut“.**
∨ **(ИЛИ) -** произлиза от латинското **„Vel“.**
¬ **(НЕ)**

Дефиниции:
Булева алгебра е множество S със съвкупност от дефинирани логически функции:

Λ (конюнкция И нарича се и логическо произведение),

V (дизюнкция ИЛИ нарича се и логическо сумиране),

¬ (отрицание НЕ),

<=> (еквивалентност),

=> (импликация, размяна замена, транспозиция).

'0' и **'1'** се наричат булеви константи, в този случай всяко съждение се явява функция от променливата наречена „истинност“, която може да взима една от двете константни стойности **'0'** или **'1'**.

Пример:
D = „навън вали“ - Това е съждение с истинността в момента т.е. може да взима '1' или '0' според, това което се случва навън.

Просто съждение - е това нещото, което дава връзка между две понятия. Такива бяха горните разказвателни изречения.

Сложно или съставно съждение - наричаме това, което съдържа две и повече прости съждения.

Упражнение:
Дадени са съжденията:
А: 2 х 2 = 4,
Чете се: Съждението „**А**“ е зададено чрез отношението създадено от израза **2 х 2 = 4.** Съждението „*2 по 2 е равно на четири"* е **истинно** при това функцията е **константа = 1** или „**А**“ = **1**. В Булевата алгебра се отбелязва, че вярността на съждението „**А**“ е **1**.

В: „*Желязото е карбонат*“
или е възможен записът:
В = *Желязото е карбонат*

Чете се: Съждението **В** се задава с отношението между понятията в него /*Желязото е карбонат/.* Каква е неговата истинност? „*Желязото е карбонат*“ е **лъжовно = 0** и се отбелязва, че верността му е **0**.

Множества от съждения

Дадени са група твърдения. Образувайте множество съдържащо само съждения към тях включете и използваните по-горе.

Е = Сложните съждения свързват две и повече прости съждения.

Е1 = Иване ела тука!

F = Настъпи есента и враните долетяха при нас.

G = Дъжда се изля от облака и водата намокри земята.

G1= Захранването падна.

Обединяваме всички използвани до сега съжденията в множеството :

{A, B, C, D, E, E1, F,G}

От гледна точка на математиката между елементите на множеството и всички под множества са възможни функционални връзки дължаши се на тяхната истинност или не истинност. На всяко от съжденията може да се припише една от двете стойности за истинност: вярно '1' или невярно '0'. Открива се възможност да правим умозаключения изучавайки свойствата на новообразуваните от нас съждения - прости или сложни. В математиката примери за умозаключения се явяват и доказателствата на теоремите. Предпоставка за правилни умозаключения могат да бъдат само съжденията имащи истинност '1' или това са само верните и не лъжовни съждения. За всяко съждение е възможно да запишем, че ще може да взима една от двете стойности за истинност и „**вярно**“ или „не **вярно**“, или че конкретно съждение има конкретна истинност: '**1**' или '**0**'

Упражнение :

Създайте подмножество състоящо се от три прости съждения.

Дадено е подмножеството: { A,B,C }, тук A, B , C са прости съждения и където:

A = Електронът има електрически заряд '1'
B = Доматът е плод '0'
C = Стария боб е семе '1'

Ще търсим различни видове съждения, които могат да се констроират от дадените в множеството. Тук имаме 3 твърдения и те са прости съждения, като:

A =1, C= 1, чете се: съжденията имат **истинност '1'** или те са верни и

B = 0 има **истинност '0'** или е не е вярно /лъжовно/

A и **C** са еквивалентни /в смисъл, че имат еднаква истинност = '**1**'

Математически, това може да се означи така:

D = A<=>C защото **A = 1** и **C = 1** /имат еднаква истинност/

B ¬ D защото **B = 0** и **D = 1** /чете се: **B** не е **D**

B ¬ A защото **B = 0** и **A = 1** /**B** е отрицание на **A**/

B ¬ C защото **B = 0** и **A = 1** /**C** е отрицание на **B**/

Чете се: Съждението „**A**“ е еквивалентно на съждението „**C**“, съждението „**B**“ не е еквивалентно на съждението „**D**“ или на съжденията:

„**A**“ или „**C**“, защото **D, A** и **C** са истини =**1**.

B НЕ D
B НЕ A
B НЕ C

В този пример **A, B** и **C** са прости, но новото съждение **D** е сложно съждение. Чрез използване на логическите оператори и съжденията е възможно да се образуват нови различни по сложност съждения. При съставянето на сложни съждения се използват логическите операции „**И**“ (конюнкция), „**ИЛИ**“ (дизюнкция), „**НЕ**“ (отрицание), „**СЛЕДВА**“ (импликация). Най-висок приоритет по ред на изпълнение има отрицанието, следвано от конюнкцията „**И**“ и дизюнкцията „**ИЛИ**“. *Заб: Обикновено за истинност се използва символа 1, а за не истинност 0. Имайте предвид това, когато правите разлика между бита носител на знака на едно число и самото число. Там '1' показва отрицателна стойност на дадено число, а '0' е символ указващ че числото е положително.*

Упражнение:

Дадени са съжденията с истинност: **A = 1, B = 0, C = 1:**

Запишете сложно съждение, като използвате операторите от логическата алгебра.

Вярно ли е съждението **A ^ C = 1** и изпълняват ли се правилата от елементарната математика за произведение?

Намерете стойността за съждението:

A ^ C = ? при **A = 1** и **¬C**

Ако **B** е невярно, то каква стойност ще има **B**?

Каква е истинността на съждението **C**?

Как се нарича операцията използваща оператора „^“ и с какви други символи може да се запише нейният оператор.

Основни своиства на логическите функции

Основни термини:
ЛОГИЧЕСКА ПРОМЕНЛИВА - Променлива приемаща само стойности **„0“** или **„1“.**

ЛОГИЧЕСКА ФУНКЦИЯ - двоична функция, в която аргумента /логическата променлива/, взима само стойност **„0“** или **„1“.**

Забележка: Логическата функция може да зависи и от повече на брой логически аргументи. Възниква естественият въпрос ако „**Y**“ е функция зависеща от една логическа променлива /от един логически аргумент/, то колко броя функции е възможни да бъдат създадени за стойностите, които взима тази логическа променлива? Или колко броя функции има една логическа променлива? (Табл. 2)

Отговорът е 4 функции:

1. **y = f (X) = 0**, при X = 0 и X = 1, y е **константа '0 '**
2. **y = f (X) = X**, където X е **променлива взимаща стойности** '0' и '1' по **някаква закономерност**
3. **y = f (X) = НЕ „X“, инверсия** на **X**.
4. **y = f (X) = 1,** при **X = 0** и **X = 1**, **y** е **константа '1'**

X	**0**	**1**	**Означение**	**Наименование**
f_0	0	0	0	Константа 0
f_1	0	1	X	Променлива y=f(X)
f_2	1	0	¬ X	Инверсия на X
f_3	1	1	1	Константа 1

Табл. 2 Функции на една логическа променлива

Логически функции реализуеми от две логически променливи

y = f(X,Y), където X и Y са логически променливи (Табл. 3).

X	**0**	**0**	**1**	**1**	**Название на функцията y = f(X,Y)**
Y	**0**	**1**	**0**	**1**	
f0	**0**	**0**	**0**	**0**	Константа 0, y= f(X,Y)= 0 = **const.**
f1	**0**	**0**	**0**	**1**	Логическо произведение или y = f(X, Y) = X^У, **AND**
f2	**0**	**0**	**1**	**0**	Забрана по У или y = X ^ ¬Y
f3	**0**	**0**	**1**	**1**	Променлива X, y = X
f4	**0**	**1**	**0**	**0**	Забрана по X, y = ¬X ^ Y
f5	**0**	**1**	**0**	**1**	Променлива У, y = Y
f6	**0**	**1**	**1**	**0**	Сума по модул 2, y = ¬X ^ Y + X ^ ¬Y, **XOR**
f7	**0**	**1**	**1**	**1**	Логическо сумиране / v /, y = X v Y, **OR**
f8	**1**	**0**	**0**	**0**	Операция на Пирс /НЕ-ИЛИ/, y = ¬(X v Y) = ¬ X ^ ¬Y, **NOR**
f9	**1**	**0**	**0**	**1**	Логическа равнозачност, y = (¬X ^ ¬Y) v (X ^ Y), **EQ**
f10	**1**	**0**	**1**	**0**	Инверсия на У, y = ¬Y
f11	**1**	**0**	**1**	**1**	Импликация от У към X, y = X ^ ¬Y
f12	**1**	**1**	**0**	**0**	Инверсия на X, y = ¬X
f13	**1**	**1**	**0**	**1**	Импликация от X към У, y = ¬X + Y
f14	**1**	**1**	**1**	**0**	Операция на Щефер /НЕ-И/, y = ¬(X ^ Y)= ¬X v ¬Y, **NAND**
f15	**1**	**1**	**1**	**1**	Константа 1, y = 1 = **const.**

Таб. 3 Логически функции реализуеми от две логически променливи

При „**n**“ логически променливи броят на функциите ще бъде:

$$\mathbf{f = 2^{2n}}$$

Ако познаваме всички стойности на аргументите и съответстващите им стойности за функцията казваме, че функцията се позната, или че функцията е зададена. По тези стойности се строи таблицата и на истинност. С нея фактически ние задаваме функцията в табличен вид. Набори от променливи са комбинациите от променливи за които се пресмятат стойностите на логическите функции. При логическа функция зависеща от една променлива /един аргумент/ „$\mathbf{X_1}$“ ще са възможни само единичните групи за $\mathbf{X = 0}$ и $\mathbf{X = 1}$. Тук наборите съдържат само по един елемент /'**0**' и '**1**'/. Виж стойностите в Табл. 4 под:

„X_1“

Казваме, че имаме два набора от стойности /макар и съдържащи по един елемент/, които могат да взимат логическите аргументи или логически променливи. При две променливи всяка от стойностите на X_1 ще се комбинира с възможните стойности за X_2 /0 и 1/. При това „наборите“ ще станат 4 броя. При 3 променливи възможни групи или набори от стойности ще са 8 броя.

X_1	X_2	X_3
0	**0**	**0**
		0
	1	**1**
1	**0**	**0**
		1
		0
	1	**1**

Табл. 4

Необходимо е да се прави разлика между броят на възможните логически функции „**f**“ и броят на възможните комбинации от значения „**N**“, които взимат логическите променливи. Или „**N**“ е броят на наборите от стойности, които могат да взимат логическите променливи за функция зависеща от „**n**“ броя логически аргументи. $\mathbf{N = 2^n}$

Упражнение:

- При зададен логически аргумент D, да се зададе таблично функцията равна на D^2 и D^3.
- Да се определи функцията y = f (A, B) = ¬(¬A ^ ¬B). Да се осъществи необходимата логическа схема използвайки означенията на компонентите по стандарта ANSI.
- Да се пресметне таблицата за истинност при логическата функция двойно отрицание.
- Да се зададат в табличен вид логическите функции:
- НЕ, ИЛИ, И, НЕ-ИЛИ, НЕ-И

Упътване:

Таблицата на истинност се получава, като изчислим стойностите за функциите и подредим наборите за променливите със съответно изчислените стоиности на логическите функции.

- Да се определи броят на наборите от стойностите за променливите ако е дадена функция зависеща от 4 броя логически аргументи. Дайте таблично наборите за:

  f = (¬A v A)
  w = A v A ^ B
  z = A ^ A v B
  s = B v ¬B
  u =(A v ¬B) ^ C

- Начертайте съответните логически блокови схеми.
- Определете данните необходими за таблиците на истинност.
- Отговор: 1, A, A, 1, u(0, 0, 0, 0, 1, 0, 1, 1)

Преобразуване на логически изрази

Отделете време за да разпишете разглежданите примери по-долу в текста с молив на лист. Това ще ви даде увереност за това, че разбирате основните теоретични постановки, на които ще се базират реалните упражнения по-долу в книгата. Макар възможностите на съвременните софтуерни системи за дизайн, симулация и синтез на електронен хардуер да оптимизират ползваните от вас блокове и модули, е добре да имате базово разбиране как този процес се осъществява. Още повече, че действието на редица електронни компоненти се симулира чрез детайлно описание на тяхната структура или поведение.

1. **A V ¬*A***
   ако A взима стойност 1,
   1 + 0 = 1
   ако A взима стойност **0**,
   0 + 1 = 1
   следователно винаги,
   A V ¬*A* = A + ¬*A* = 1

2. **A ^ ¬*A* = *?***
   A ^ ¬*A* = *A ¬*A* = 0 x 1 = 1 x 0 = 0**, така е и за двете значения **0** и **1** на съждението **A**

3. **A v A ^ B = ?**
   Както в алгебрата, когато имаме да събираме две величини и тук второто събираемо е съставено от два множителя, длъжни сме да намерим второто събираемо преди да извършим общото сумиране.

   Изчисляване /преобразуване/:
   A V A^B = A + A.B = A (1 + B),

   Израза в скоби е операцията дизюнкция, тази операция има стойност **1** когато сигналите са **1,1 1,0** или **0,1**
   В случаят единят от сигналите е **1** т.е. стойността на B е без значение.

   Заб. Означава ли това, че входа на логичната променлива /ЛЕ - логически елемент/ „B“ в реална схема може да виси във въздуха? /Изходите и входовете на интегралните схеми винаги трябва да бъдат свързан към съответното място, земя или захранване през терминиращ резистор т.к. сигнала може да бъде изкривен от паразитни токове. Болшинството от програмируемите интегрални схеми могат софтуерно да управляват състоянието на изходните буфери, поставяйки ги във високо импедансно състояние. Свободни

входове и изходи не може да бъдат оставяни в електронните схеми. Така те могат да бъдат източник на смущения и грешки в работата на устройството, да предизвика генерация на импулси и шум със случайни параметри, да предизвикат грешки в изчисленията. За терминирането на неизползваните изводи на интегралните схеми ще стане дума по-късно, когато разглеждаме тяхната вътрешна структура, особености и свойства. Винаги когато имате съмнение относно това следва да се обърнете към инструкциите и описанията давани от производителите и да постъпвате съобразно тях.

За израза ще имаме:
1 V B = 1 + B = 1 заместваме в горният израз,

A V A^B = A + A.B = A (1 + B) = A . 1 = A

Правилата на математическата логика и връзките с елементарната алгебра позволяват да изчисляваме изрази от съждения и да получаваме нови форми. Това е важно за логическите елементи, тъй като чрез различни схемни конструкции ще получаваме исканите от нас параметри.

Упражнение:

4. Изчислете стойностите на съждението:

   (C v ¬C) ^ ¬A , тук **A** и **C** са логически променливи

   При какви условия ще се изпълнява конюнкцията?
   От определението за конюнкцията следва: само когато и двете съждения са верни. /отговор при ¬A = 1/

5. **(A v C) ^ (B v C) = ?**
   Изчисляване:
   Изобразяваме логическото съждение в неговата алгебрическа форма и разкриваме скобите.

   (A v C) ^ (B v C) = (A+C) . (B+C) = A.B + A.C +C.B + C.C

   Предварително ще е необходимо да разгледаме, **C.C = C ^ C** , конюкцията се изпълнява само ако и двете логически променливи са верни съждения. При какви стйности на логическите променливи участващи в конюкцията са верни? /отговор при C = 1/

   Тогава:

C.C= C ^ C = 1.1 = 1 = C разглеждаме събираемите **CB + CC = C. (B+C),**

Тъй като получихме, че стойността на логическата променлиа **C** може да е само '**1**', то:

CB + CC = C.(B+C) = C.(B + 1),

този резултат го връщаме обратно във формулата ,

A.B + A.C +C.B + C.C =A.B + A.C + C.(B + 1)

необходимо е да направим справка със задачата **3** . **A v A ^ B = ?** , развитието на логически израз бе:

A v A^B = A + A.B = A (1 + B) = A . 1 = A

тогава за нашият случай ще получим

CB + CC = C.(B+C) = C.(B + 1) = C.(1 + B) = C

връщаме C в сновният израз,

AvC ^ BvC=(A+ C) . (B + C)=A.B +A.C +C.(B + 1)=A.B + AC + C =

= A.B + C (1 + A) = A.B + C = A ^ B v C

Упражнение:

Използвайте блокова схема с квадратчета и свързващи стрелкички, за да начертаете двата израза, като вътре в блокчетата напишете съответната логическа функция. Коментирайте коя от реализациите им, би била икономически по-изгодна при изпълнание в интегрална схема.

AvC ^ BvC
A ^ B v C

6. **Дадено е:**
   ¬(**A * B) =** ¬A + ¬**B** или в логическата им форма:
   ¬*(*A ^ B) = ¬*A V* ¬B

Решението на задачата ще изразим в табличен вид (Табл. 5). Ако всички значения на двете функции са равни, значи фнкциите са равни.

A	**B**	**¬*A***	**¬B**	**¬*A v* ¬B**	**¬*(*A ^ B)**	
0	**0**	**1**	**1**	**1**	**1**	
0	**0**	**1**	**1**	**1**	**1**	
0	**1**	**1**	**0**	**1**	**1**	
0	**1**	**1**	**0**	**1**	**1**	
1	**0**	**0**	**1**	**1**	**1**	
1	**0**	**0**	**1**	**1**	**1**	
1	**1**	**0**	**0**	**0**	**0**	
1	**1**	**0**	**0**	**0**	**0**	

Табл. 5

7. **Дадени са логическите променливи:**
   A, B да се докаже че:
   A ^ B = ¬¬**(A ^ B)** или в алгебричната форма:
   A * B = ¬¬**(A * B)**

Тук двата символа ¬¬ **(НЕ НЕ)** се разбират, като двукратно отрицание (наричано още двойно инвертиране). Схема на решението доказващо или не верността на формулата от примера. /Променливите **A** и **B** взимат само две стойности '**0**' и '**1**'/. Таблица 6 е валидна за получаване на общото доказателство.

A	***B***	***A * B***	***¬(A * B)***	***¬¬(A * B)***
0	**0**	**0**	**1**	**0**
0	**1**	**0**	**1**	**0**
1	**0**	**0**	**1**	**0**
1	**1**	**1**	**0**	**1**
/ a /	**/ b /**	**/c /**	**/d/**	**/e/**

Табл. 6

Следователно :

A ^ B = ¬¬(A ^ B) = A.B = ¬¬(A.B)
за всички възможни стоиности на **A** и **B**.

Упражнение:
За задача 6 (да се докаже равенството **¬*(*A * B) = ¬*A* + ¬B) :**

- Колко колонки и редове са нужни?
- Какви величини ще поставите във всяка колонка?

Аналогично можете да обясните/да направите вярно умозаключение или да докажете верността или не/ на формулите:

¬(C + D) = ¬C * ¬D

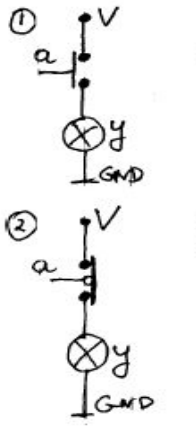
Посочените **до тук** формули се използват доста често за опростяване на сложни съждения или логически изрази от вида:

A = ¬(B + C) + ¬B = ¬B * ¬C + ¬B = ¬B * (¬C + 1)

За повече информация:
„The Calculus of Logic“, George Boole, Cambridge and Dublin Mathematical Journal Vol. III (1848), pp. 183-98

Логически елементи

Логическите елементи са устройства изпълняващи зададените от логическите функции действия. С най-обикновени превключватели /механични, хидравлични, оптични, топлинни/ и други устройства се осигурява задействуването на механизмите в различни сложни машини и съоръжения. Най-бързодействуващи са електрическите системи, при тях сигнала се предава със скоростта на разпространение на елекро-магнитното поле в дадена среда. Контакторно релейни схеми може да посочим, като най-стария електрически логически елемент използван и до днес в практиката. Съвременните електронни логически схеми се изпълняват посредством различни технологии, следва да отбележим, че за реализация на логически функции могат да се ползват всички електрически ключове (Фиг. 3), релета, радио лампи, транзисторни и диодни схеми и други. Конкретната реализация зависи от типовото приложение и физическите параметри на средата в която логическото устройство следва да работи нормално.



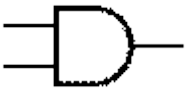
Фиг. 3 Използването на два типа превключватели може да се интепретира, като работата на логическите елементи – повторител и инвертор

Чрез логическите операции И, ИЛИ, НЕ или комбинация от тях могат да се реализират практически всички сложни логически схеми и вериги. Имайки на разположение логически елементи И, ИЛИ, НЕ е възможно да конструираме цифрово електронно устройство с произволна сложност. Например електронната част на всеки компютър е съставена от логически елементи. По-долу са дадени основните означения на логическите елементи И, ИЛИ, НЕ. Следва да обърнете внимание, че повечето схемотехнически редактори могат да ползват единия от двата типа означения на елементите. Пърмият е американски – ANSI (American National Standards Institute) /IEEE (Institute of Electrical and Electronics Engineers) Std 91-1984 и неговия производен ANSI/IEEE Std 91a-1991

дефинира начин за рисуване на логическите елементи особено подходящ при ръчно изчертаване на електронни схеми, този стандарт е приет да се нарича „военен“ поради своя произход. Друг подход е ползването на „правоъгълни форми“ и се базира на IEC (International Electrotechnical Commission) 60617-12, използвайки правоъгълници за изчертаване на всички видове електронни схеми, като при това опростява значително начина за изчертаване от различни устройства, като плотери и принтери налични в момента на създаване на стандарта. Системата за означение IEC е приета за стандарт EN (European Committee for Standardization) 60617-12:1999 в Европа и BS (British Standards) EN 60617-12:1999 във Великобритания. Следва да се обърне внимание, че днес при изчертаване на по-елементарни логически схеми се ползва ANSI стандарта, тъй като получените схеми са по-лесно четими от хора. Обединяването на повече такива схеми в единни модули обаче силно затруднява читаемостта на схемата, и това налага ползването на IEC стандарта при означение на големи логически схеми, чипове и процесори. При работата ви с Quartus II Web Edition ще използваме различни типове означения на логическите елементи и блокове.

Логическа функция	Означение ANSI	Означение IEC	Таблица на истинност

И конюнкция



A . B

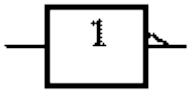
ВХОД		ИЗХОД
A	B	A И B
0	0	0
0	1	0
1	0	0
1	1	1

ИЛИ дизюнкция

A + B

ВХОД		ИЗХОД
A	B	A ИЛИ B
0	0	0
0	1	1
1	0	1
1	1	1

НЕ инвертиране



$\overline{A}$

ВХОД	ИЗХОД
A	НЕ A
0	1
1	0

Табл. 7 Означение на основните логически елементи

В електронната индустрия логическия елемент НЕ е известен с името ИНВЕРТОР. Означението на инвертора в ANSI е едно малко кръгче (балонче) на изхода или входа на съответния логически елемент. Това означение в IEC е наклонена чертичка на изходния или входния порт на логическия елемент. Балончето е по-лесно за възприемане от бърз поглед над схемата, отколкото наклонената чертичка. Буфери и инвертори се ползват активно при реализацията на смесени аналогово цифрови схеми за минимизация на шумовете идващи от цифровите електронни схеми при свързването им с аналогови електронни схеми, АЦП и ЦАП. Група от последователно свързани буфери или инвертори биха могли да се ползват за реализация на линия за закъснение, генератор на импулси и др.

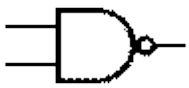
При свързването на логически елементи следва да се имат предвид някои основни техни особености:

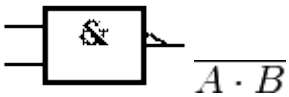
- коефициент на разклонение - колко входа на логически елементи могат да се захранят от един изход на елемент от даден тип;
- закъснение при разпространение на сигнала - какво е времето за което един входящ фронт на импулс се появява като изходящ фронт, което е свързано с изисквания за потребявана мощност при задържането насигнала – speed-power product;
- шумоустойчивост - в какъв интервал от стойности може да бъде входния и изходния сигнал, така че правилно да бъдат декодирани нивата на логическата „**0**“ и „**1**“;
- забранени нива на входни и захранващи напрежения, или в какви граници на работни нива на сигналите схемата остава изправна и правилно декодира постъпващите сигнали.

Технически погледнато процесът на производство на **И-НЕ** логически схеми е по-евтин. Принципно чрез употребата на **И-НЕ** и **ИЛИ-НЕ** схеми може да бъде създадено произволно електронно устройство (Табл. 8).

Логическа функция	Означение ANSI	Означение IEC	Таблица на истинност

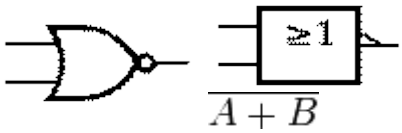
И НЕ





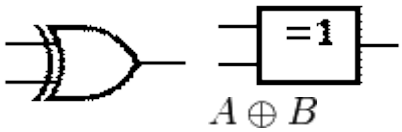
ВХОД		ИЗХОД
A	B	A И НЕ B
0	0	1
0	1	1
1	0	1
1	1	0

ИЛИ НЕ



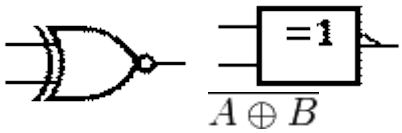
ВХОД		ИЗХОД
A	B	A ИЛИ НЕ B
0	0	1
0	1	0
1	0	0
1	1	0

ИЗКЛЮЧВАЩО ИЛИ



ВХОД		ИЗХОД
A	B	A ИЗКЛЮЧВАЩО ИЛИ B
0	0	0
0	1	1
1	0	1
1	1	0

ИЗКЛЮЧВАЩО ИЛИ НЕ

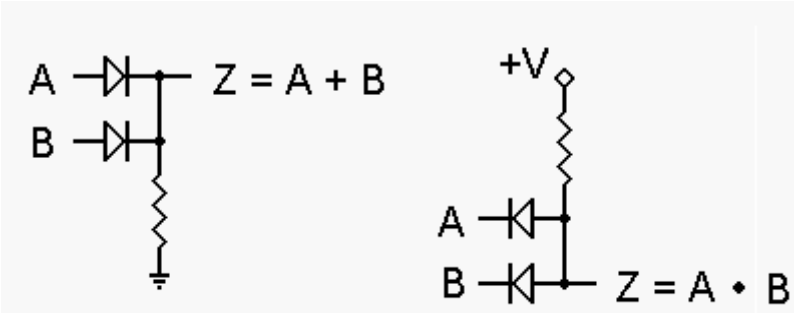


ВХОД		ИЗХОД
A	B	A ИЗКЛЮЧВАЩО ИЛИ НЕ B
0	0	1
0	1	0
1	0	0
1	1	1

Табл. 8 Логически схеми реализирани с И-НЕ и ИЛИ-НЕ компоненти

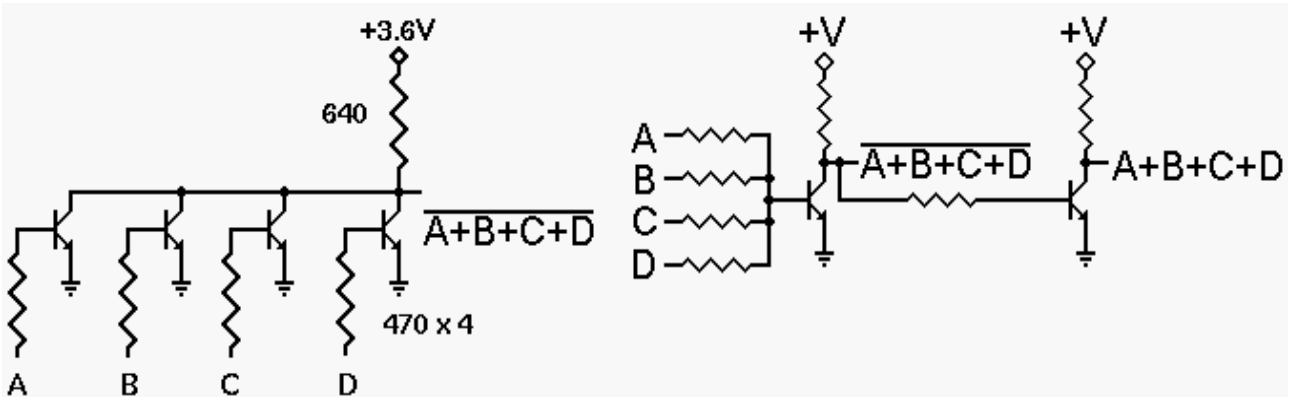
Класификация на логическите електронни схеми

За описание на различните типове логически схеми обикновено се използват началните букви на елементите от които са изградени. Като първи исторически възникнали може да кажем диодно транзисторните схеми ДТЛ (Фиг. 4).



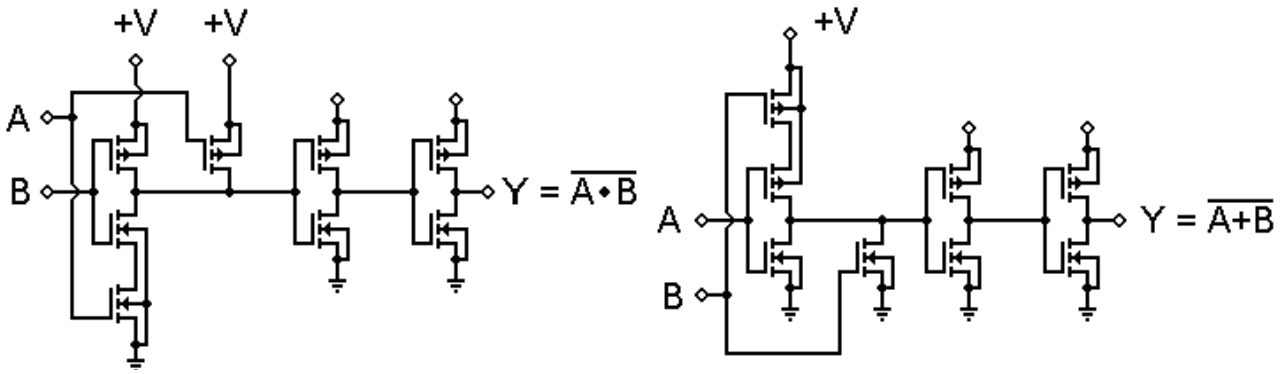
Фиг. 4 Използване на диоди за имитиране действието на логически функции ИЛИ, И

Ако в схемата се използват резистори и транзистори, то използваната в схемата логика се нарича резисторно-транзисторна логика. Бележи се – РТЛ и все още се използва. /Според режима на работа логическите елементи се подразделят на статически и динамически/. Към настоящият момент схемите изпълнени, като РТЛ и ДТЛ се считат за остаряли, големи и скъпи за направа. Прогреса в електрониката доведе до създаването и развитието на схеми от типа, транзисторно-транзисторна логика – ТТЛ, с разновидност емитерно свързана логика. Самите елементи се класифицират и по типа на използваните транзистори.



Фиг. 5 РТЛ имитиране действието на логически функции ИЛИ-НЕ, ИЛИ и НЕ

Понастоящем днес масово се ползват CMOS елементи (Фиг. 6), тяхното производство е по-лесно от ТТЛ схемите, тъй като изцяло се състоят от MOS транзистори. При ТТЛ схемите се налага ползване и на диоди, и резистори което в интегрално изпълнение е скъпо. Друго предимство на MOS схемите, е че консумираният ток е от порядъци по-нисък в сравнение с този от ТТЛ, но ръста на консумирана енергия е по-бърз висока тактовата честота в сравнение с ТТЛ. Като основни недостатъци може да се посочи чуствителността на входовете на схемите към статично електричество, макар и съвременните модели схеми да имат вградени защити, монтъжът им трябва да се извършва при специални условия, също така тези схеми са малко по-бавни от другите по отношение стръмноста на фронтовете на изходните импулси, макар времето за разпространение на сигнала вътре в тях да е по-малко от ТТЛ.



Фиг. 6 CMOS логически елементи И-НЕ, ИЛИ-НЕ

Дадените тук основни сведения за типовете интегрални схеми реализиращи логически функции са само илюстративни. Желаещите да получат повече информация биха могли да потърсят допълнителна информация в интернет и [1], [2]. Следва да кажем, че при свързването на логически схеми е от особено значение те да имат равни захранващи напрежения и съвместими нива за предаване на логически сигнали '1' и '0'. Друга особеност е товарната способност на изходите на една логическа схема, това показва какъв ток може да се черпи от нея без тя да се повреди. За избягване на повреди в програмираното устройство е добре изходите му да бъдат буферирани със стандартни ТТЛ изходни и входни буферни схеми, като: 74FCT2240 и др.

Основни закони и тъждества в булевата алгебра

Комутативни /разместителни/ закони:

A V B = B V A
A ^ B = B ^ A

Асоциативни /съчетателни/ закони:

(A V B) V C = A V (B V C) ,
(A ^ B) ^ C = A ^ (B ^ C)

Диструбтивни /разпределителни/ закони :

A ^ (B V C) = A ^ B V B ^ C ,
A V B ^ C = (A V B) ^ (A V C)

Закони за повторението :

A V A V ... V A = A ,
A ^ A ^ ... ^ A = A

Закони за инверсия / в случая -двойственост /.
Известни са, като закони на „де Морган“

¬(A V B) = ¬A ^ ¬B ,
¬(A ^ B) = ¬A V ¬B

Закони за отрицанието :

A V ¬A = 1 ,
A ^ ¬A = 0

Закон за двойното отрицание :

¬¬A = A

Закон за поглъщането :

A V A ^ B = A
A ^ (A V B) = A

Закон за „слепването“:

A ^ B V A ^ ¬B = A

Правила за операции с константи :

¬0 = 1 , ¬1 = 0 , A ^ 0 = 0 , A V 0 = A , A v 1 = 1

Допълнителни тъждества :

A V ¬A ^ B = A V B ,
A ^(¬A V B) = A ^ B

Пример:

Дадена е формулата за закон на „слепването“:

A ^ B V A ^ ¬B = A , в алгебричната си форма тя има вида:
A.B + A*¬B = A.(B + ¬B),

От закона закони за отрицанието следва:

A V ¬A = 1 , В случая дезюнкцията винаги ще има стойност на истинност равна на 1, защото винаги една от двете логически променливи ще е равна на 1. Така получаваме:

(B + ¬B) = 1, имаме операция ИЛИ

тогава за слепването, като го преобразуваме получаваме:

A ^ B V A ^ ¬B = A.B + A*¬B = A.(B + ¬B) = A .1 = A

Упражнение:

- Напишете горните закономерности в тяхната алгебрична форма.
- Докажете поне три от горните формули (може да използвате таблици на истинност).

Функционално пълни системи

Нека припомним, че функционална зависимост имаме, когато съществуват определени отношения между две или повече величини (променливи), като между съждението и неговата истинност съществува права и обратна функционална зависимост.

Ако „**p**“ е стойността на истинност на съждението,

а „**P**“ е някое съждение.

Тогава истинността „**p**“ е функция от съждението „**P**“ и функцията може да взима една от двете постоянни стойности **1-вярно**, и **0-не вярно**. Истинността може да взима две стойности. Това зависи от верността или не на съждението „**P**“.

p = w(P) = { ако **P** е вярно, то **p = 1**
ако **P** не е вярно, то **p = 0**

Вярно е и обратното съждение: „P“ е функция от аргумента „истинност“ P = z(p). Съждението е вярно или не вярно - това зависи от изменението на неговият аргумент, стойността на неговата истинност „p“ или аргумента „истинност = p“ има две константни стойности :

p = 1 или **p = 0**

при аргумент **p = 1**, функцията или съждението „P“ е вярно
при аргумент **p = 0**, функцията или съждението „P“ не е вярно

Това се записва така:

P = z (p) = { при **p = 1**, стойността на функцията е „**вярно**“
при **p = 0**, стойността на функцията е „**не вярно**“

На всяко съждение, може да припишем стойност на неговата истинност или стойност за вярност „1“ или не вярност „0“. Правата и обратната функции в случаят са интересни с това, че свойствата им могат да се използват за усилване и управление. Но най-съществено, е че те съответстват на цифрите използвани в двоичната позиционна бройна система. С тяхна помощ могат да се създадат устройства реализиращи посочените функции. Тези устройства могат

да са най-разнообразни: електромагнитни, механични, хидравлични и електрически. Всички видове устройства проявяват инертност при предаване на сигнала, дори при съвременните електронни схеми типични стойности на тази инертност са от 10ns но десетки милисекунди, което зависи както от типа устройства, така и от управляващите нива на напрежението, типа товар, консумирания ток от него, мехнична инертност на устройствата и други фактори.

Определение:

ФУНКЦИОНАЛНО ПЪЛНА СИСТЕМА е система от елементарни /прости/ функции, чрез които е възможно да получим всяка друга логическа функция. Функционално пълни системи са следващите пет:

1. **Y = ¬X отрицание (НЕ)**

   Y = X1 . X2 конюнкция (И)

   Y = X1 + X2 дизюнкция (ИЛИ)
2. **Y = ¬X**

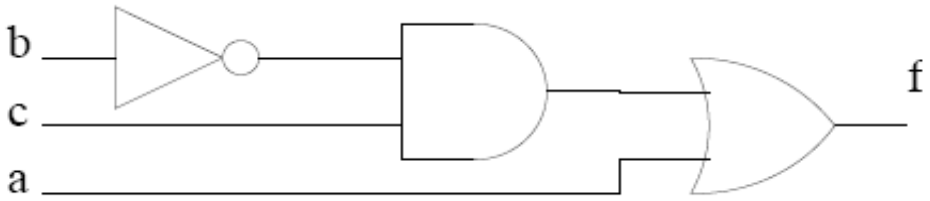
   Y = X1 . X2 конюнкция (И)
3. **Y = ¬X**

   Y = X1 + X2 дизюнкция (ИЛИ)
4. **Y = ¬(X1 . X2) отрицание на конюнкции** (И-НЕ) - операция на **Шеффер**
5. **Y = ¬(X1 + X2) отрицание на дизюнкции** (ИЛИ-НЕ) - операция на **Пирс**

Връзка между аналитичен запис електронна схема

В част от зададените по-горе упражнения се искаше да начертаете дадена логическа схема по аналитично зададена логическа функция (Фиг. 4). Тук по-детайлно ще разгледаме начина по който се извършва дизайна на логически схеми, и начина по който се анализира тяхното действие. Обикновенно задачата е формулирана така: по зададена според Булевата алгебра формула за конкретна логическа функция може да се конструира структурна схема изпълнена с логически елементи.

Например:
y = f (A, ¬B, C) = A + ¬B. C



Фиг. 4 Изчертаване на логическа схема по зададена аналитична форма

При изчертаване на схемата съблюдавайте редът за осъществяване на отделните логически операции по приоритет (инвертиране, логическо произведение и логическо сумиране):

- Инвертирам B, т.е. определяме ¬B
- Извършвам операция И за ¬B. C = D
- Извършвам операция супиране /дезюнкция/ на A + D = f

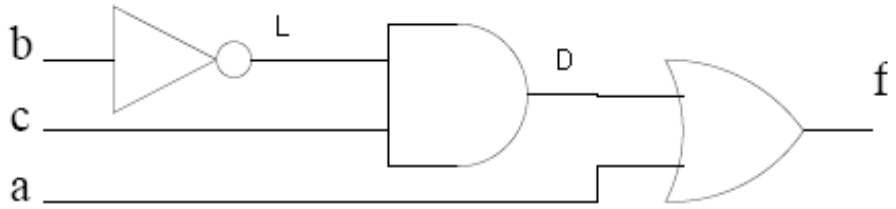
Пояснение:
Всяка логическа функция може да се представи аналитично по един от двата начина:

Дизюнкция от конюнкции /сума от произведения/
y = X.Z + D. P + ...+M. F

Конюнкция от дезюнкции /произведение от суми/
u = (X + Z) . (D + P) . ... . (M + F)

От написаното за двата вида представяне на логически функции в аналитичен вид следва и извода, че на всяка логическа функция записана аналитично по посочените начини съответствува двустъпална логическа схема. Възниква резонният въпрос, как по известна ни схема да можем да запишем

съответната логическа функция, която тя трябва да изпълнява? Това е възможно (Фиг. 5). Започваме от зад напред по схемата от логически елементи.



Фиг. 5 Изчертаване на логическа схема по зададена аналитична форма

По-долу за първи път ще разгледаме описанието на тези логически схеми на езикът за описание на хардуер **HDL** – Hardware Description Language, който по-подробно ще разглеждаме в част II на книгата. Важното нещо което трябва да се осмисли на този етап, е че за обратния анализ на електронните схеми, а и за техният синтез често се налага да се ползва следната техника. Всеки изход на логически елемент свързващ се с друг вход на логически елемент в схемата ще бъде означаван с някакъв индекс - буква. В практиката тези индекси се наричат **сигнали**, а логическите променливи и самата функция f **портове**. Накратко, за да успеем да пресъздадем правилно логическата схема дадена по-горе трябва да въведем 2 нови сигнала, **D** и **L**. Чрез тях по-лесно ще съумеем да запишем аналитичния израз на функцията.Създаването на аналитичен запис на функцията става отзан напред в следния ред:

f = D v A **D = ? , D** се получава след конюкция,

D = C ^ L **L = ? , L** се получава след инвертиране,

L = ¬B,

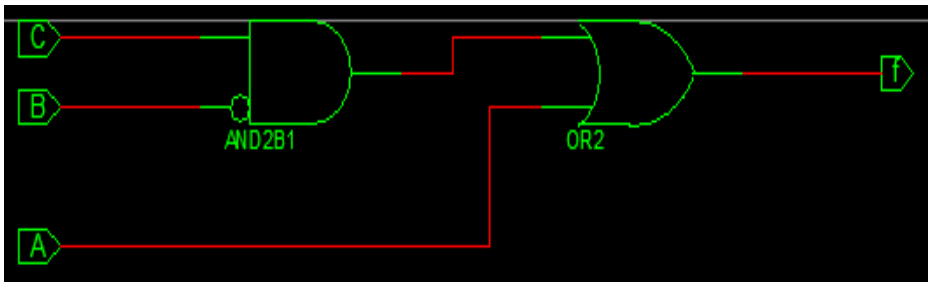
замествам в **D = C ^ L = C ^ ¬B**

f = D v A = C ^ ¬B v A

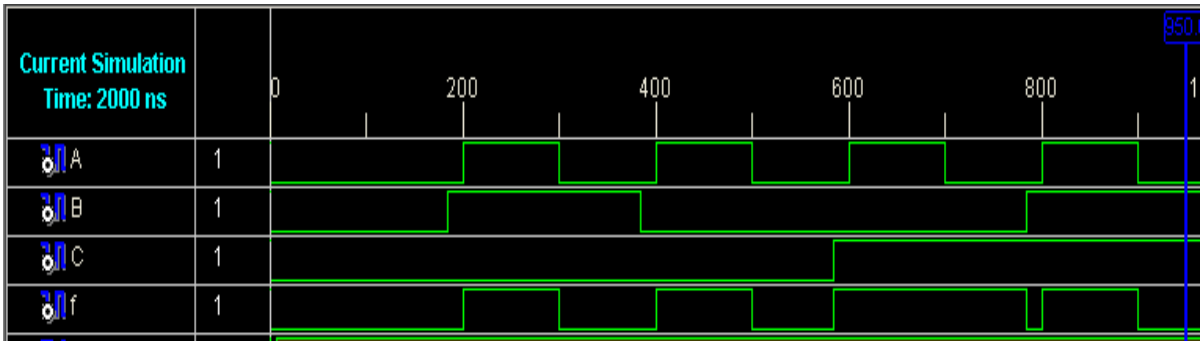
Описанието на съответният логически елемент е дадено по-долу, като за това е ползван езикът **VHDL**. Следва да обърнете внимание на няколко основни неща: описанието на логическите аргументи – променливите е отделено в обособен програмен блок наричан **ентити**, задаването на логическата функция става в друг обособен програмен блок наричан **архитектура**. В следствие на компилиране на така дадения код, софтуерът е генерирал структурната схема на логическата функция зададена в аналитичен вид (Фиг. 6), а след това е симулирал времедиаграма изследваща поведението на схемата при всичките възможни 8 набора от входни сигнали (Фиг. 7).

```
--Симулация на елемента на VHDL
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity module1 is
Port (
        A : in  STD_LOGIC;
        B : in  STD_LOGIC;
        C : in  STD_LOGIC;
        f : out STD_LOGIC
        );
end module1;

architecture Behavioral of module1 is
        begin
        f <= A or ((not B) and C);

end Behavioral;
----------------------------------------------
```

Заб. Кодът тук е даден само за пример.



Фиг. 6 Структурна схема



Фиг. 7 Времедиаграма на симулационна последователност

Понякога дизайнерът на електронно устройство, или програмистът може да иска изпълнението на електронната схема да става по точно определен начин. Или схемата да има точно определена структура. Както казахме по-рано, за точно дефиниране структурата на дадена схема трябва да използваме междинни сигнали вътре в нея. Пример за описание на същата логическа схема, но ползвайки междинни сигнали е даден по-долу.

```
--Симулация на елемента на VHDL с
--дефиниране на междинни сигнали
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity module1 is
   Port (
         A : in  STD_LOGIC;
         B : in  STD_LOGIC;
         C : in  STD_LOGIC;
         f : out STD_LOGIC
        );
end module1;

architecture Behavioral of module1 is
        signal L : std_logic;
        signal D : std_logic;
begin
        --този запис е заместен от следващите 3 реда
        --f <= A or ((not B) and C);
        L <= not B;
        D <= L and C;
        f <= D or A;
end Behavioral;
----------------------------------------------
```

Задача 1: Опитайте да синтезирате по-сложно схема, използвайки още 1 логически елемент и поне още 1 сигнал.

Нормална и канонична форма на запис на логически функции

Аргументите на логическите функции са логическите променливи. Това са Булевите константи и промеливи взимащи стойности '**0**' и '**1**'. Булева константа е величина запазваща стойността си във времето. Булева променлива е параметър, описващ изменението на дадена величина във времето. В различни случай параметъра приема различни значения (Табл. 9).

Логическата 0	**Логическата 1**
ЛЪЖА	**ИСТИНА**
ИЗКЛЮЧЕНО	**ВКЛЮЧЕНО**
НИСКО равнище ва сигнала	**ВИСОКО равнище на сигнала**
КЛЮЧ ОТВОРЕН	**КЛЮЧ ЗАТВОРЕН**
НЕ	**ДА**

Табл. 9 Различните стойности на параметъра, могат да имат различен физически смисъл

Например булевото значение за '**0**' се използва за представяне на всяко напрежение в диапазона от 0V до 0.8V, а за '**1**' е характерно напрежение от 2V до 5V. Това обаче важи само за даден тип интегрални схеми, възможно е друг тип схеми да ползват различни нива за представяне на логическите променливи. За архитектурния дизайнер на електронни устройства булевите '**0**' и '**1**' не бива да се интерпретират, като описвание на реалните величини, било то ток или напрежение. Те описват т.н. логически равнища на сигнала: ниско и високо. Изобщо Булевата алгебра е отличен начин за изразяване на връзката /съотношението/ между входовете и изходите на електрическите схеми без да се интересуваме конкретно за реалната им имплементация. Следва да се помни, че при реален дизайн на електронно устройство винаги следва да се ползват специални съгласуващи интегрални схеми или буфери. Тяхната функция е основно да се предпази от претоварване и повреда програмируемото логическо устройство. Буферите могат да се ползват за управление на различни товари: релета, мотори, бобини и др. Конверторите на логически нива се ползват когато трябва да се съгласуват логическите нива на две интегрални схеми с различни захранващи и работни сигнални напрежения или токове. За изясняване на подобни въпроси следва да се обърнете към странична литература, касаеща основите на цифровите интегрални схеми в частта нива на сигналите, работни токове, начини за галванична изолация и др. [1], [2].

Както казахме по-рано, всяка логическа функция може да се представи аналитично по един от двата начина:

Дезюнкция от конюнкции /сума от произведения/
Примерно:
y = X.Z + D. P + ...+M. F

Конюнкция от дезюнкции /произведение от суми/
Примерно:
u = (X + Z) . (D + P) . ... . (M + F)

Аналитичното задаване на логическата функция се нарича **нормално** и казваме ,че функцията е в нормалната си форма, или функцията е в нормален вид. С помоща на аналитичния запис или нормалната форма на логическата функция е възможно да конструираме структурна схема изградена от основните логически елементи, които ще удоволетворяват конкретната функционална зависимост. Аналитичният запис на логическата функция позволява да изчислим сигнала на изхода при дадени сигнали на входовете. Когато решаваме булеви изрази (аз лично предпочитам да оставя този пролем на специализираните софтуерни продукти, но често се налага да се проверяват на ръка някои прости логически функции, особено когато към програмируемото логическо устройство се прикачват и други логически схеми, като буфери, тристабилни регистри, драйвери за индикатори и др.) се придържаме към следния приоритет на операциите: инвертиране, операции в скоби по реда познат от математиката, операции И преди логическото сумиране ИЛИ, ако над израз е зададена операция НЕ, тогава се изпълняват деиствията вътре в ограниченият от скоби израз ¬(A or B).

Пример:
Зададена е логическата функция:
X = ¬A ^ B ^ C ^ ¬(A v D)

Зададени са сигнали на входните портове:
A = 0, B = 1, C = 1, и D = 1

Търсим сигнала на изхода. За улеснение записвам функцията с обикновените символи за алгебричните действия (.) и (+).

X = ^ B ^ C ^ ¬(A v D) = ¬A . B . C . ¬(A + D) =,
Тук заместваме буквите с цифровите стоиностти на даденият набор от променливи.
= 1 . 1 . 1 . ¬(0 + 1) = 1 . ¬1 = 1 . 0 = 0

Упражнение:

Намерете сигнала на изхода, получаван от структурната логическа схема удоволетворяваща логическата функция:

y = [D v ¬(A + B) ^ ¬C]

За стойностите на набора от аргументи /сигналите на входовете/:

A = 0, B = 0, C = 1, D = 1, и E = 1 /Отговор 1/

От написаното за аналитичното представяне на логически функции следва и извода, че н**а всяка логическа функция записана аналитично, може да съответствува двустъпална електронна логическа схема.**

Упражнение:

Постройте структурните логически схеми за функциите:

y = X.Z + D. P

u = (X + Z) . (D + P)

Колко стъпални са получените схеми?

Как е възможно да се нарекат двете функции?

Възниква резонният въпрос: възможно ли е от таблицата за истинност да се получи, еквивалентна форма на аналитичен запис за конкретна логическа функция? Крайната цел е от „таблицата на истинност“ да се построи структурна схема от основните логически елементи (**И ИЛИ НЕ**). Това е възможно защото табличният начин на представяне на логическа функция я определя напълно. В примера по-долу ще разгледаме решението на този проблем.

Дадена е функцията:

y = A ^ B

Сума от произведения /дизюнкция от конюнкции/,

y = X.Z + D. P + ...+M. F или

Произведение на суми /конюнкция от дезюнкции/.

u = (X + Z) . (D + P) . ... . (M + F)

Тук **X, Z, D, P, ... ,M, F** са логически аргументи

Показаните форми на запис на логическите функции са важни за опростяването на сложните логически схеми. Във връзка с решаването на практически конструктивни въпроси е необходимо да се изяснят понятията и свойствата на **МИНТЕРМ** И **МАКСТЕРМ**.

Минтерм

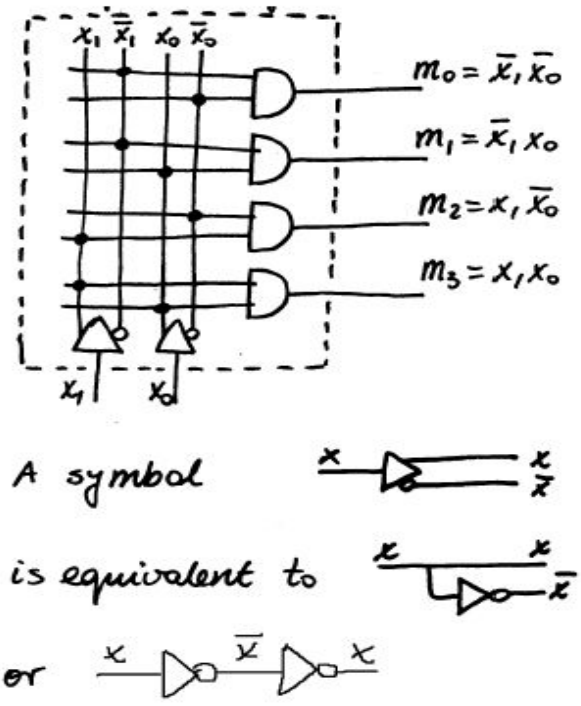
Това е логическа функция приемаща стойност '**1**' само за един единствен набор на променливи. За всички останали комбинации от променливи стойността на функцията е '**0**'. За пояснение ще ползваме таблицата на истинност на логическото произведение (И) дадена по-долу.

X_1	X_2	$y = X_1 \wedge X_2$
0	0	0
0	1	0
1	0	0
1	1	1

Табл. 10 Таблица на истинност на логически елемент И

Определение:

Конюнкциия от вида $\mathbf{m_i = X_{1i} \wedge X_{2i} \wedge ... \wedge X_{ni} = 1}$ наричаме минчлен или минтерм. В тази конюнкция $\mathbf{X_{1i}, ...,X_{ni}}$ са стойности приемани от логическите променливи, където с „**i**“ се означават „**не инвертираната**“ и „**инвертираната стойност**“ на някоя от променливите. $\mathbf{X_1, X_2,...,X_n}$ са логически аргументи на функцията. По-долу (Фиг. 8) е показана структурна схемата за реализация на 4 минтерма. Броят на минтермите за една логическа функция е $\mathbf{N=2^i}$.



Фиг. 8 Структурна схема реализираща 4 минтерма

Пример:
Нека е дадена основната логическата функция **И с** 2 аргумента:
y = f (A, B)

Всяко частно значение на „y_i“ за даден набор от аргументи ще бъде:
$y_i = f_i (A, B) = A_i \wedge B_i$

Умножавам двете страни на уравнението с m_i **(минтерма за набор** $_i$**)**
$y_i.m_i = f_i (A, B) .m_i = A_i . B_i . m_i$

Конюнкцията е равна на '**1**' ако всички променливи участвуващи в нея са равни на '**1**', а минтерма взима стойност '**1**' за един единствен набор на променливите. За нуждите на задачата ще образуваме конюнкции от вида:
$m_i = A_i \wedge B_i = A_i . B_i = 1$ които се съставят по начина:

За да получим '**1**' за рузултат нулите в даден набор трябва да се инвертират, а единиците се запазват. Вижте данните от колони 2, 3 и 7 в таблица 11. За дадената функция са показани стойностите на минтермите за различните набори на функцията. В колона 1 са дадени номерата на наборите за функцията:

y = f (A, B) =A ^ B

1	2	3	4	5	6	7	8
N	Ai	Bi	yi = Ai ^ Bi	¬Ai	¬Bi	mi=Ai . Bi	yi . mi
i = 0	0	0	0	1	1	¬A ^ ¬B = 1	0 . 1 = 0
i = 1	0	1	0	1	0	¬A ^ B = 1	0 . 1 = 0
i = 2	1	0	0	0	1	A ^ ¬B = 1	0 . 1 = 0
i = 3	1	1	1	0	0	A ^ B = 1	1 . 1 = 1

Табл. 11 Таблица на истинност при реализация на 4 минтерма

Пояснения:
За всеки набор от входни стойности инвертираме само нулите, като запазваме единиците. По този начин изчислените конюнкции в колона 7 ще бъдат равни винаги на '**1**'. За всеки ред от таблицата образуваме произведението ($y_i . m_i$), виж колона 8. Сумираме произведенията намиращи се в колона 8. В тази сума влизат всичките стойности, които приема логическата функция.

$$y = m_o . y_0 + m_1 . y_1 + m_2 . y_2 + m_3 . y_3$$

Тази сума е нашата функция **y = f(A, B)** записана е в аналитичен вид, който не винаги е оптимизиран. Възниква въпросът защо

$\mathbf{y = m_o . y_o + m_1 . y_1 + m_2 . y_2 + m_3 . y_3}$ е точно нашата функция?

y = f (A, B) е логическа функция, която има за стойности: **y0, y1, y2** и **y3**. Следователно е възможно да запишем дизюнкцията:

$\mathbf{y = f (A, B) = y_o \vee y_1 \vee y_2 \vee y_3}$

Ако умножим тези равенства на **1** нищо не се променя.

$\mathbf{y = 1 . y_o + 1 . y_1 + 1 . y_2 + 1 . y_3}$

Всяка единица в дясната част е равна на съответната конюнкция наречена минтерм за конкретен набор от променливи. Като заместим с конкретните минтерми получаваме:

$\mathbf{y = mo . y_o + m_1 . y_1 + m_2 . y_2 + m_3 . y_3}$

Горното показва, че сумата представлява аналитичен запис на функцията.

За задачата конкретният запис е:

$\mathbf{y = 0 . 1 + 0 . 1 + 0 . 1 + 1 . 1 = y_3 . m_3}$

$\mathbf{y_3 = f(1, 1) . (A \wedge B) = A . B = 1}$

На практика работим така:

- Дадена е таблицата за истинност т.е. Колони 2, 3, 4;
- Намираме функциите НЕ за всички аргументи;
- Определяме всички конюкции дефинирани от минтермите;
- Образуваме произведенията за всеки ред от таблицата: (y_i . m_i) Виж в таблицата колона 8;
- Сумата от последните произведения дава аналитичният израз; еквивалентен на зададената с таблицата за истинност логическа функция.

Всяка една логическа функция **y** от **n** променливи може да бъде изразена чрез логическа сума от продуктите на минтермите и съответните стойности на функцията. За логическа функция зависеща от „**n**“ логически аргумента са в сила релациите /съотношенията или равенствата/.

$$y = f(x_{n-1}, \ldots, x_0) = \sum_{i=0}^{2^n-1} y_i \cdot m_i$$

Това е чисто еквивалентно на сумата на минтермите имащи стойност **1**:

$$y = f(x_{n-1}, \ldots, x_0) = \sum_{\text{for all } j \text{ such that } y_j=1} m_j$$

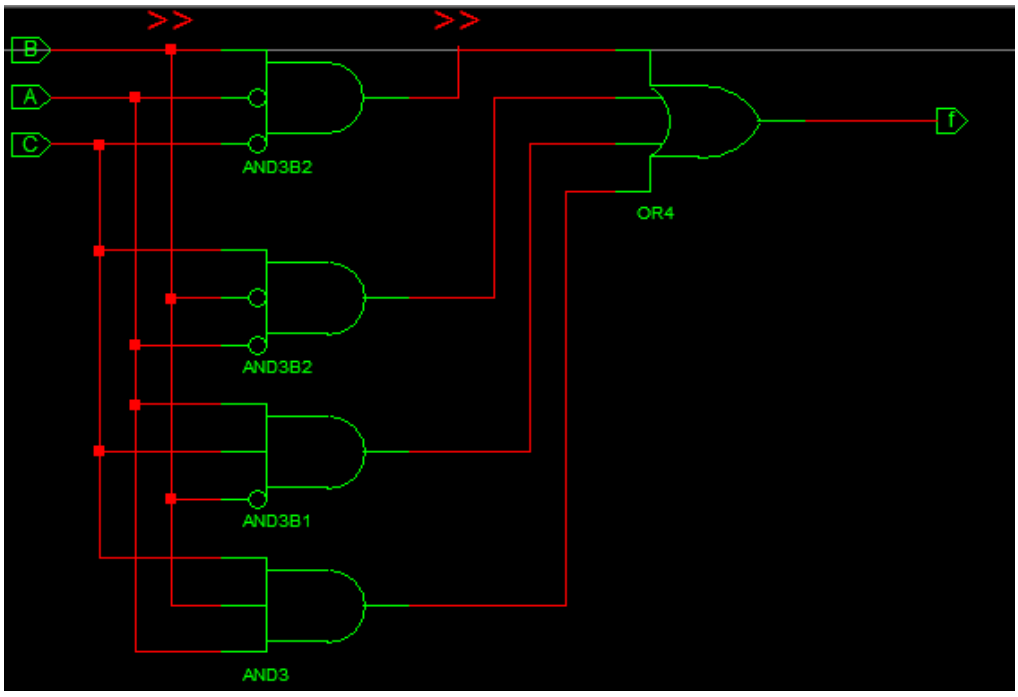
Упражнение:
Дадена е таблица за истинност на функцията **y = f (X1, X2, X3)**:

N	*A*	*B*	*C*	*y*
0	0	0	0	0
1	0	0	1	0
2	0	1	0	1
3	0	1	1	0
4	1	0	0	1
5	1	0	1	1
6	1	1	0	0
7	1	1	1	1

Табл. 12

Запишете функцията в аналитична форма. Реализирайте структурна схема с логически елементи по стандарта ANSI.

Отговор: y(A, B, C) = ¬A.B.¬C + A.¬B.¬C + A.¬B.C + A.B.C (Фиг. 9)



Фиг. 9 Отговор на задачата

Макстерм

За да обясним какво представлява макстерма, ще използваме свойствата на дизюнкцията имаща стойност '**0**' само за един набор от променливи, и свойствата на уравненията за прибавяне на една и съща величина към лявата и дясната част на равенството. Макстерма ще бъде дизюнкции от вида:

Mi= Ai ^ Bi = 0

В таблицата за истинност ще инвертираме само единиците в наборите на функцията. Виж колона 7, на показаната в табличен вид схема подпомагаща решението на задачата (табл. 13). Дизюнкция от посоченият вид се съставя за всеки един от наборите на променливите. Разглеждам таблицата за истинност на същата конюнкция.

y = f(A ^ B)

За всяка дезюнкция :
z = A v B

виж колона 4, където съществува само едно **z = 0**, когато **A = B = 0**

1	*2*	*3*	*4*	*5*	*6*	*7*	*8*
A	**B**	**y = A^ B**	**z = A v B**	**¬A**	**¬B**	**M=Ai + Bi**	**y ^ M**
0	0	0	0	1	1	Mo =A v B = 0 + 0	0 + Mo = A v B
0	1	0	1	1	0	M1= A v ¬B = 0 + 0	0 + M1= A v ¬B
1	0	0	1	0	1	M2 = ¬A v B = 0	0 + M2 = ¬A v B
1	1	1	1	0	0	M3 = ¬A v ¬B = 0	1 + M3 = ¬A v ¬B

Табл. 13 Макстерм

Образуваме логическо произведение от сумите в колона 8:
u = (0 + Mo) . (0 + M1) . (0 + M2) . (1 + M3) =

= Mo . M1 . M2 .(1 + M3) =
От свойствата на булевата алгебра следва, че **(1 + M3) = 1**

= Mo . M1 .M2 . 1 = (A v B) . (A v ¬B) . (¬A v B) =

= (A v B) ^ (A v ¬B) ^ (¬A v B)

Това е една еквивалентна аналитична форма на зададената таблично логична функция, която разглеждаме. С помоща на основните свойства в Булевата алгебра е възможно да извършим и още опростявания:

(A v B) ^ (A v ¬B) ^ (¬A v B) =
= (A v B) . (A v ¬B) . (¬A v B) =
= (A + B) . (A + ¬B) . (¬A +B) =
= (A . A + A . ¬B + B .A + B . ¬B) . (¬A +B) = Тук $A^2 = A \cdot A = A$, $B \cdot \neg B = 0$,
= (A + A . ¬B + B .A + 0) . (¬A +B) =
=[A (1 + ¬B) + B .A] . (¬A +B) = Тук: (1 + ¬B) = 1
= (A . 1 + B . A) . (¬A +B) =
= [A . (1 + B) . (¬A +B) = отново (1 + B) = 1
= A .1 . (¬A +B) =
= A . ¬A + A . B =
= 0 + A .B = A ^ B отново A . ¬A = 0

Работили сме вярно, защото изходната функция беше точно тази конюкция.

За логическа функция зависеща от „**n**“ логически променливи е в сила израза, където всяка една логическа функция „**y**“ от „**n**“ променливи може да се изрази, като логически продукт от сумите на макстермите и респективно стойностите на функцията:

$$y = f(x_{n-1}, \ldots, x_0) = \prod_{i=0}^{2^n-1} (y_i + M_i)$$

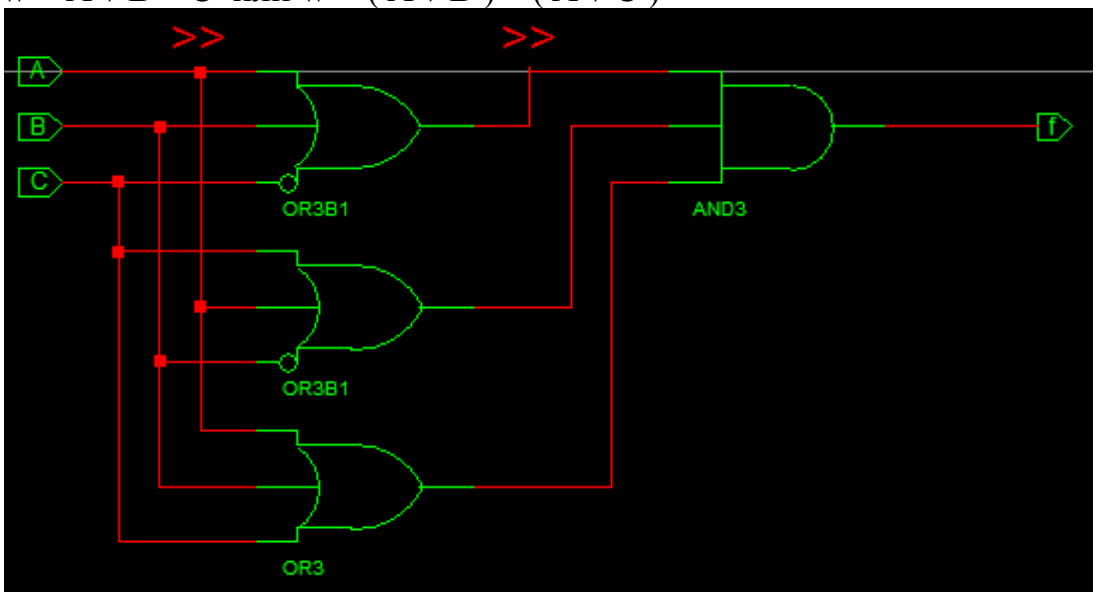
Упражнение:
Дадена е таблица 14 за истинността на функция **w(C, B, A)**.

N	*C*	*B*	*A*	*w*
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	1

Табл. 14

- Запишете функцията в аналитична форма.
- Начертайте електронна схема реализираща функцията.

Отг.
Опростете израза: y(C,B,A) = (C+B+A) . (C+ ¬B+A) . (¬C + B + A)
w = A v B ^ C или w = (A v B) ^ (A v C)



Фиг. 10 Отговор на задачата

Връзка между минтерм и макстерм

Всяка комбинационна логическа функция може да се изрази чрез своите макстерми или инвертирани минтерми съгласно следното правило:

M = ¬m

Използвайки таблиците за истинност докажете това за следните изрази:

y = A.B.¬C.D
y1 = ¬A+¬B+C+¬D

y = ¬y1 ?

Начертайте логическите схеми на двете функции.

Преобразуване на логически функции

Всички методи за преобразуване /опростяване/ на логическите функции произтичат от основните свойства на Булевата алгебра. Избора на един или друг способ зависи от подготовката на програмиста, от начина на задаване на функцията и броят на входящите сигнали. Крайната цел е да се получи най-оптимално минимизирана структурна логическа схема т.е. схема съдържаща минимален брой логически елементи и еквивалентна функция. В процесът на дизайн с програмируеми логически устройства много рядко се налага ръчното оптимизиране на логически функции и то при реализация на макро блокове. Въпреки това при всеки един електронен дизайн се ползват външни логически схеми, като буфери, усилватели, шинни драйвери и други, често тяхното управление се извършва от програмируемото устройство, но минимизацията на броят външни управляващи схеми оказва съществено влияние върху крайния физически размер, цена и качество на устройството. Макар да съществуват редица софтуерни продукти за решаване на тези задачи, е добре програмистите да са запознати с основните принципи за оптимизиране на логически функции, част от които вече използвахме при решаването на примерите по-горе.

Минимизацията на логическите функции се извършва до момента, в който се нарушава несъкратимостта на функцията т.е. до момента, в който премахването на някой „елемент“ донася загуба за стойност „1“ на логическата функцията. Под „елемент“ в случая следва да разбираме някоя конюнкция участваща в общата дизюнктивната форма:

$$y = f(x_{n-1}, \ldots, x_0) = \sum_{i=0}^{2^n-1} y_i \cdot m_i$$

Или в развит вид:

y = f(Xn-1, ... , X0) = y0 .m0 + y1.m1 + y2.m2 + ... + yN-1.mN-1

Тук:

„n“ е броят на логическите променливи;

„i“ е поредния номер на набора от променливи или поредната стойност на функцията, или поредният номер на минтерма от i-тия ред в таблицата за истинност на логическата функция „y“;

„N“ е броят на наборите от променливите за логическата функция и при **N = 2^3** наборите са **8**

„yi “ е значение или стойност на логическата функция, която се получава за i-тия набор от нейни аргументи;

„mi “ е конюнкция имаща стойност равна на единица за даденият i-ти набор от значения на аргументите.

X или **x** са логически аргументи

X = x

Заб. В първата показана общата формула логическите променливи са означени с малки букви, тъй като са взети от различни източници.

Аналитичен метод

Аналитична минимизация на логическа функция е преобразуване на аналитичният израз на логическата функция в съответствие с основните свойства на Булевата алгебра. Минимизираната логическа функция е необходимо да бъде еквивалента на първата. Нека разгледаме едно примерно аналитично преобразуване и опростяване с непосредствени Булеви преобразувания, нека е дадено е:

A v A ^ B

Да се опрости израза ако е възможно.
A v A ^ B = A + A . B = A (1 + B) = A
защото **1 + B** е дизюнкция имаща стойност **1**

Какъв извод можете да направите от написаното?
Отг. 1 + B е винаги равно на 1.

Равенството:
A v A ^ B = A + A . B = A (1 + B) = A изразява важен логически закон, нарича се „**закон за поглъщане**“. Скицирайте навсякъде в задачите съответните структурни логически схеми, преди и след опростяването.

Дадена е дезюнкцията от конюнкции:
A ^ ¬ B v A ^ B

Да се опрости израза.
A ^ ¬ B v A ^ B = A . (¬B + B) = A, защото дизюнкцията **(¬B + B) = 1**, виж правилата за сума в Булевата алгебра. **(¬B + B)** е дизюнкция и тя не може да е нула, тъй като участвуващите събираеми е невъзможно едновременно да бъдат равни на нула.

A ^ ¬ B v A ^ B = A . (¬B + B) = A

Е важен и израза изразява „**закона за слепване“.** Закона за слепването е често използван в методите за опростяване на логически функции. Възможно е да се докаже равенството и по друг начин.

Упражнение: да се докаже верността на равенството с таблица на истинност:

X v 0 = X

Упражнение :

Докажете вярността на основните правила от Булевата алгебра:

X v 1 = 1

X ^ 0 = 0

N ^ N = N

T v T = T

¬ D ^ D =0

A v ¬A = 1

Задача:

Докажете че:

X1 v X2 ^ X3 = (X1 v X2) ^ (X1 v X 3)

За това доказателство ще използваме програмата Excel:

=OR(A2,(AND(B2,C2)))

=AND(OR(A2,B2),OR(A2,C2))

=IF(D2=E2,"OK","--")

Заб. Excel е полезен инструмент за различен тип и сложност изчисления, с възможностите да се ползва Visual Basic for Applications софтуерът е изключително удобен при начално разработване на функции и бърза проверка. С него може да създавате бързо и лесно таблици на истинност на различни по сложност логически функции. Таблиците на истинност са най-лесния начин за проверка правилността на дадена функция за пълния набор входни променливи.

Упътване: Разгледайте алгебричния израз:

(a + b) . (a + c)

Открийте и използвайте правилото от закона за поглъщането:

1 v B = 1

Задача:

Докажете че:

X1 ^ (X2 v X3) = X1 ^ X 2 v X1 ^ X3

Упътване, възможно е да използвате основните свойства на Булевата алгебра или да установите еквивалентността, като изчислите таблиците за истинност на лявата и дясна част на равенството.

Проверете законите на **Де Морган** /използва се за представяне на логически функции в елементна база **И-НЕ** и **ИЛИ-НЕ** / за инвертиране с таблици за истинност.

¬ (X1 ^ X2) = ¬X1 v ¬X2

¬ (X1 v X2) = ¬X1 ^ ¬X2

Покажете верността на **закона за съкращаване** /препоръчваме да се извърши с таблица за истинност/:

X1 v ¬X1 ^ X2 = X1 v X2 и

X1 ^ (¬X1 v X2) = X1 ^ X2

Решение за първото равенство се извършва чрез полагане:

У1= X1 v ¬X1 ^ X2

У2 =X1 v X2

Изчисляваме таблиците за истинност (Табл. 15) и сравнявате стойностите, които могат да взимат **У1** и **У2**. При еднакви стойности за **У1** и **У2** в таблиците за истинност се изпълнява: **y1 <=>** ***y2***

X_1	X_2	$\neg X_1$	$\neg X_1 . X_2$	$y_1=X_1+\neg X_1 . X_2$	$y_2 = X_1 \vee X_2$
0	0	1	0	0 +1.0 = 0	0
0	0	1	0	0	0
0	1	1	1	1	1
0	1	1	1	1	1
1	0	0	0	1	1
1	0	0	0	1	1
1	1	0	0	1	1
1	1	0	0	1	1

Табл. 15 Проверка на y1 <=> *y2*

Задача:
Как се наричат законите:
A (1 + B) = A
(¬B + B) = 1

Отг. закон за поглъщане, закон за слепване

Показаните примери са твърде елементарни. Заради това правилата и свойствата на Булевата алгебра и изчисляването на таблицата на истинност са лесни за прилагане, при доказателства и опростявания. Възможни са и други начини за опростяване на логически функции и логическите схеми за които ще стане дума сега.

Систематичен метод

Ще разглеждаме опростяването на логически функции от вида:
Дезюнкция от конюнкции /сума от произведения /

Примерно:
y = X.Z + D. P + ...+M. F
Тук **X, Z , D, ...** са логически аргументи

От аналитичният израз личи, че той обуславя дву стъпално изпълнение на структурните логически схеми. За дизюнктивно конюнктивната форма /сума от произведения/ за запис на логическите функции в общият случай е в сила :

$$y = f(x_{n-1}, \ldots, x_0) = \sum_{i=0}^{2^n-1} y_i \cdot m_i$$

Или в развитата си форма:

$\mathbf{y = y_0 .m_0 + y_1.m_1 + y_2.m_2 + ... + y_{N-1}.m_{N-1}}$

Метода на Куайн-МакКласки /двама автори/

При този аналитичен метод се счита, че логическата функция е зададена със своята таблица на истинност (фиг. 16). Нека разгледаме следния пример, зададена е таблица за истинност на логическа функция „**Z**“ - зависеща от 3 логически аргументи:

Z = Z(A, B, C) q където **A, B** и **C** са логически аргументи.

N	A	B	C	Zi	
i_0	**0**	**0**	**0**	**1**	**¬A.¬B.¬C**
i_1	**0**	**0**	**1**	**1**	**¬A.¬B.C**
i_2	**0**	**1**	**0**	**1**	**¬A.B.¬C**
i_3	0	1	1	0	
i_4	1	0	0	0	
i_5	1	0	1	0	
i_6	**1**	**1**	**0**	**1**	**A.B.¬C**
i_7	1	1	1	0	

Табл. 16 Таблица на истинност на: Z(A, B, C)

zi е значението на функцията за i-тия набор от значения приемани от аргументите.
Z = (¬A.¬B.¬C) + (¬A.¬B.C) + (¬A.B.¬C) + (A.B.¬C)

За да се извърши оптимизация на аналитичния израз по метода на **Куайн-МакКласки се** зпълняват поредица от операции в следната последователност:

Избираме наборите, за който логическата функция има стойност равна на 1 и ги записваме в колонка:

A	B	C	Zi
0	**0**	**0**	**1**
0	**0**	**1**	**1**
0	**1**	**0**	**1**
1	**1**	**0**	**1**

Табл. 17

Избраните набори се подреждат, според броя на единиците съдържащи се във всеки от изтеглените, от таблицата на истинност набори. Започва се с набор съдържащ най-малък брой единици /0000/. Наборите имащи еднакъв брой единици обособяваме в групи. Именуваме ги с I група, II група и т.н. и ги записваме нов стълб по реда на групите означени с римски числа:

N	A	B	C	групи	сравнение
i_0	**0**	**0**	**0**	**I**	**¬A.¬B -**
i_1	**0**	**0**	**1**	**II**	
i_2	**0**	**1**	**0**	**II**	**- .B.¬C**
i_6	**1**	**1**	**0**	**III**	

Табл. 18

Сравняват се наборите от две съседни групи, като сравненията се извършват между наборите във всички съседните групи. Гледаме група I и група II от ред i0 и i1, те се различават само по един бит. Следователно за този ред можем да запишем обобщен израз даден в колона „сравнение“ **¬A.¬B -**. Там където стойностите се различават в състоянието на един бит записваме чертичка „-“. Повтаряме операциите за следващите групи II и III от редове i2 и i6. Записваме израза **- .B.¬C,** като отново поставяме чертичка „-“ на мястото където стойностите се различават в сътоянието на един единствен бит. На този етап можем да считаме, че оптимизацията на функцията е приключила успешно, като резултатът е:

Z = (¬A.¬B.¬C) + (¬A.¬B.C) + (¬A.B.¬C) + (A.B.¬C) =
(¬A.¬B) + (B.¬C)

Пример 2:

Ще разгледаме пример с четири логически променливи. Следва да се отбележи, че този тип оптимизация е приложим и за функции с повече логически променливи, но в практиката това няма да ви се налага, тъй като тези оптимизационни изчисления се извършват от специализиран софтуерен продукт. Изходните данни са дадени в съответната таблица на истинност (Табл. 19).

N	A	B	C	D	Zi		сравнение
i_0	0	0	0	0	0		
i_1	0	**0**	0	1	1	**¬A.¬B.¬C.D**	**¬A. - .¬C.D**
i_2	0	0	1	0	0		
i_3	0	0	1	1	0		
i_4	0	1	0	0	0		
i_5	0	1	0	1	1	**¬A.B.¬C.D**	
i_6	0	1	1	0	0		
i_7	0	1	1	1	0		
i_8	1	0	0	0	0		
i_9	1	0	0	1	0		
i_{10}	1	0	1	0	0		
i_{11}	1	0	1	1	0		
i_{12}	1	1	0	0	0		
i_{13}	1	1	**0**	1	1	**A.B.¬C.D**	**A.B. - .D**
i_{14}	1	1	1	0	0		
i_{15}	1	1	1	1	1	**A.B.C.D**	

Табл. 19

Аналитичния запис на функцията е:

z = (¬A.¬B.¬C.D) + (¬A.B.¬C.D) + (A.B.¬C.D) + (A.B.C.D)

Ще извършим сравнение в набори i_1 и i_5, а след това в набори i_{13} и i_{15}. Резултатът от оптимизацията е:

z = (¬A.¬C.D) + (A.B.D)

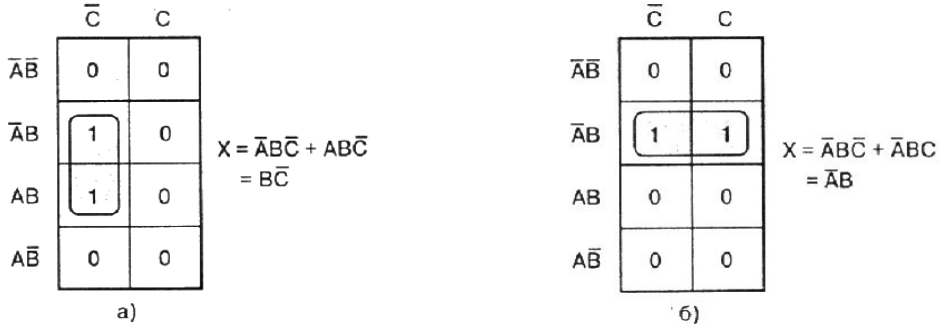
Задача 1: Постройте пълните и оптимизирани логически схеми изпълняващи логическите функции от пример 1 и пример 2.

Задача 2: Кажете колко и какви по вид логически компоненти са спестени при оптимизираната форма на изразите.

Метод на Карно

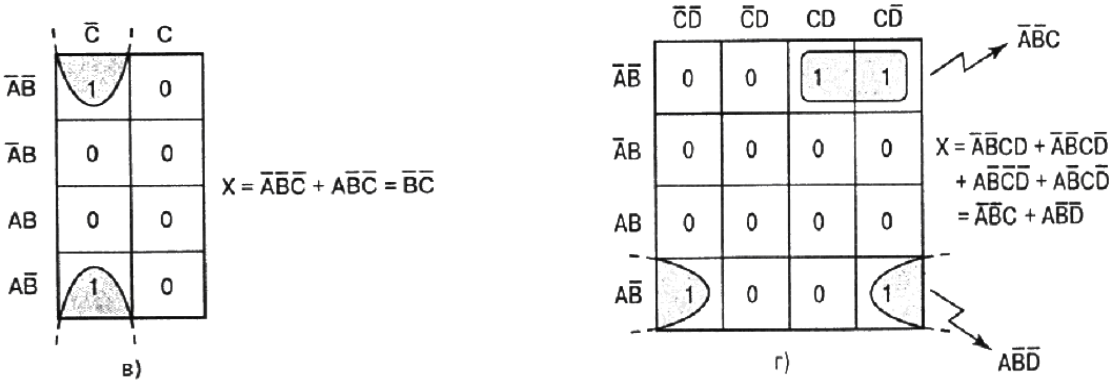
Картата на Карно е графическо средство за опростяване на логически функции и преобразуване на техните таблици на истинност с съответната логическа схема чрез извършване ня последователности от прости операции. При това не е нужно добро познаване на слойствата на булевата алгебра. Едновременно с това ползването на тези карти за опростяване на логически функции с повече от 4 променливи е неудобно. Разглежданите по-долу примери се дават само информативно.

Групиране по двойки



Фиг. 11.а, б

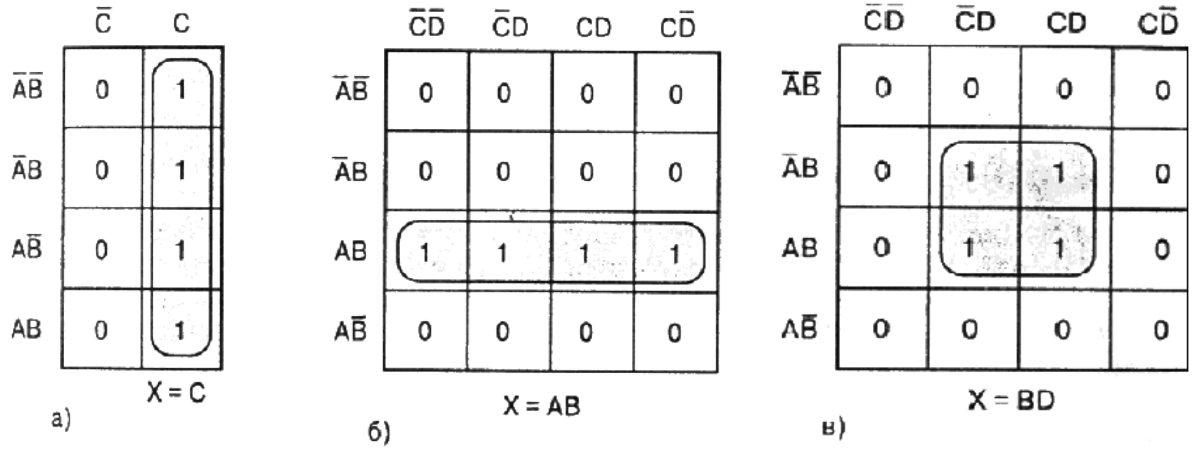
На фиг. 11.а е даден най-елементарен пример за групиране на съседни клетки по вертикала в картата на Карно. От израза отпадат **A** и ¬**A**. На фиг. 11.б е дадено групиране на клетки по хоризонтала, като от израза отпадат **C** и ¬**C**. Резултатите от двете опростявания са дадени вдяскно на картите.



Фиг. 11.в, г

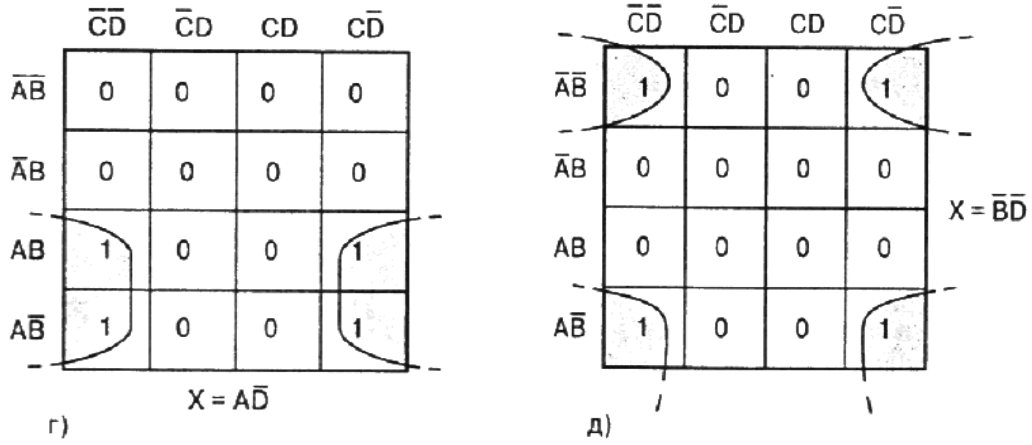
На фиг. в е дадено групиране на кретки стоящи в двата края на един стълб в картата на Карно. След групирането от израза отпада **A** и ¬**A**. На фиг. 11.г е даден по-сложен пример на функция имаща 4 аргумента. Извършени са 2 групирания на клетки по хоризонтала, две съседни и с крайни. В резултат на групирането от израза отпада **D** и ¬**D**, а от другия **C** и ¬**C**. Резултатите от двете опростявания са дадени вдясно на картите.

Групиране по четворки



Фиг. 12. а, б, в

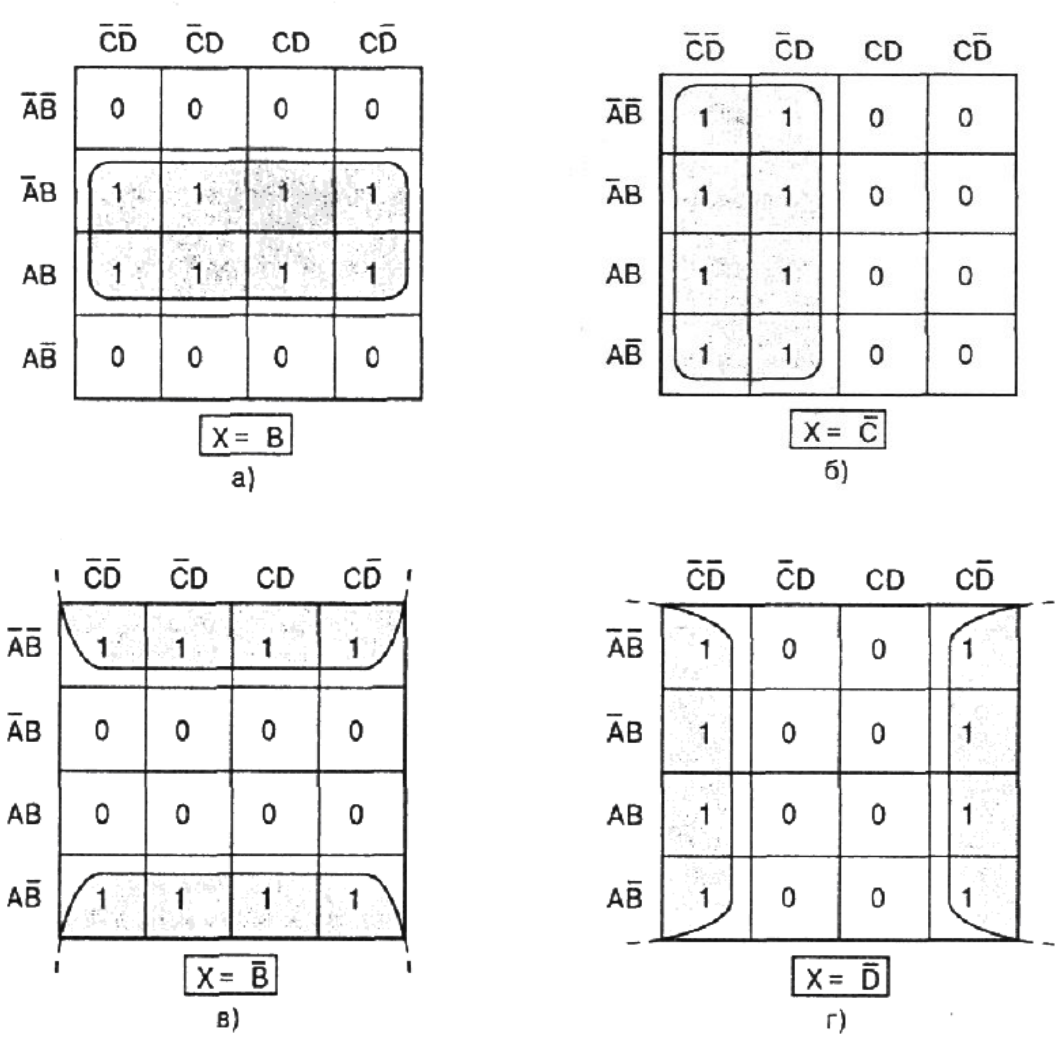
На фигури **12 а, б, в, г** и **д** са дадени примери за едновременно групиране на 4 клетки в картата на Карно.



Фиг. 12. г, д

Задача: Запишете в аналитичен вид изходните функции. Извършете алгебричните опрастявания и докажете истинноста на дадените по-горе резултати от опростяване на изходните изрази с карти на Карно.

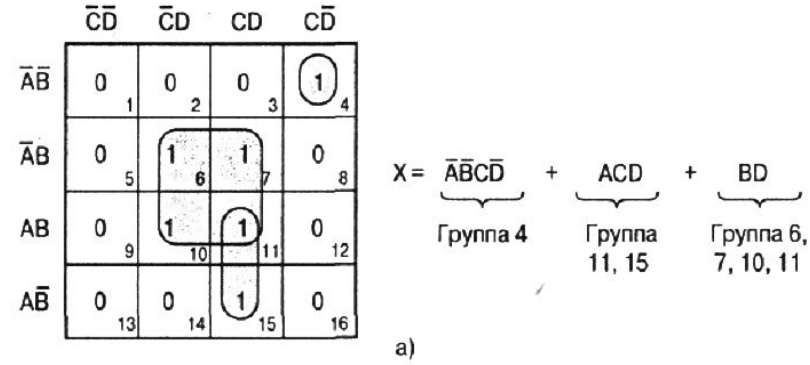
Групиране по осем клетки



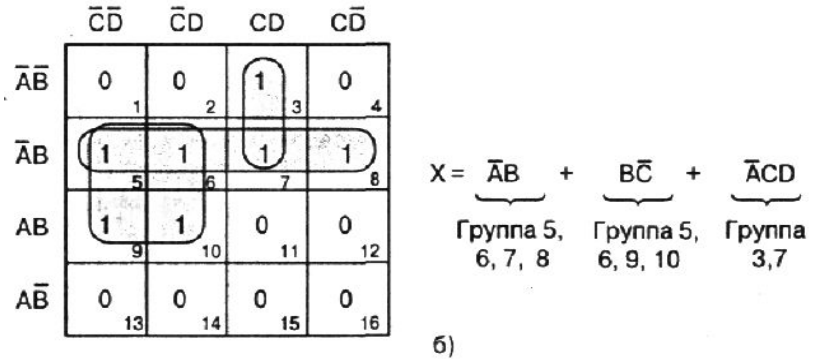
Фиг. 13. а, б, в, г

Задача: Запишете в аналитичен вид изходните функции. Извършете алгебричните опрастявания и докажете истинноста на дадените по-горе резултати от опростяване на изходните изрази с карти на Карно.

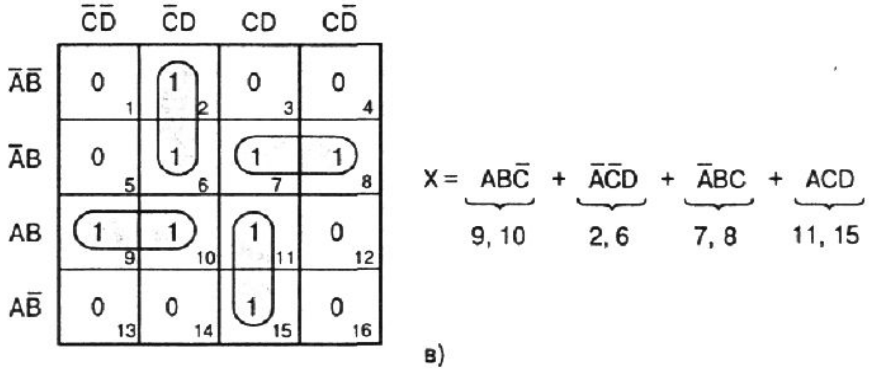
Примери



а)



б)

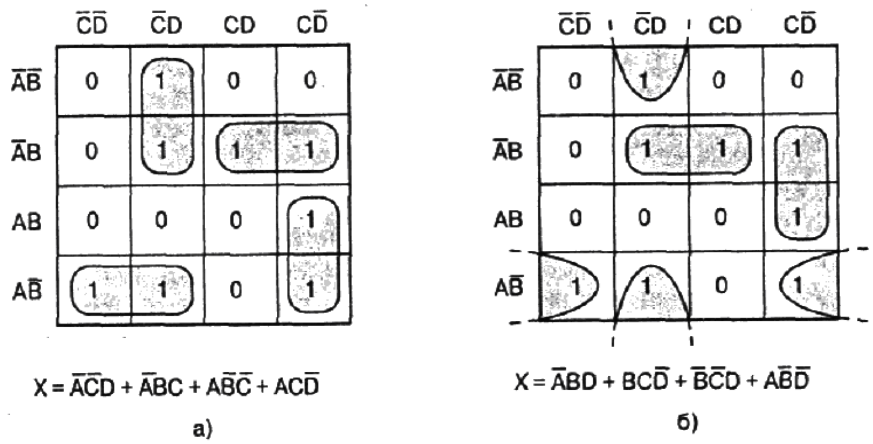


в)

Фиг. 14. а, б, в

Задача: Запишете в аналитичен вид изходните фувкции. Извършете алгебричните опрастявания и докажете истинноста на дадените по-горе резултати от опростяване на изходните изрази с карти на Карно.

Пример за получаване на еквивалентни решения с една и съща карта на Карно



Фиг. 14. а, б

Този пример показва две еквивалентни решения при използване на картите на Карно за оптимизация на логически функции.

Попълване картата на Карно по аналитичен запис на функция

Задача: Да се попълни картата на Карно за следния израз:
y = ¬C(¬A.¬B.¬D + D) + A¬BC + ¬D

Да се извърши опростяване на израза. *Заб. Разклийте скобите,*
y = (¬C.¬A.¬B.¬D) + (¬C.D) + (A¬BC) + ¬D

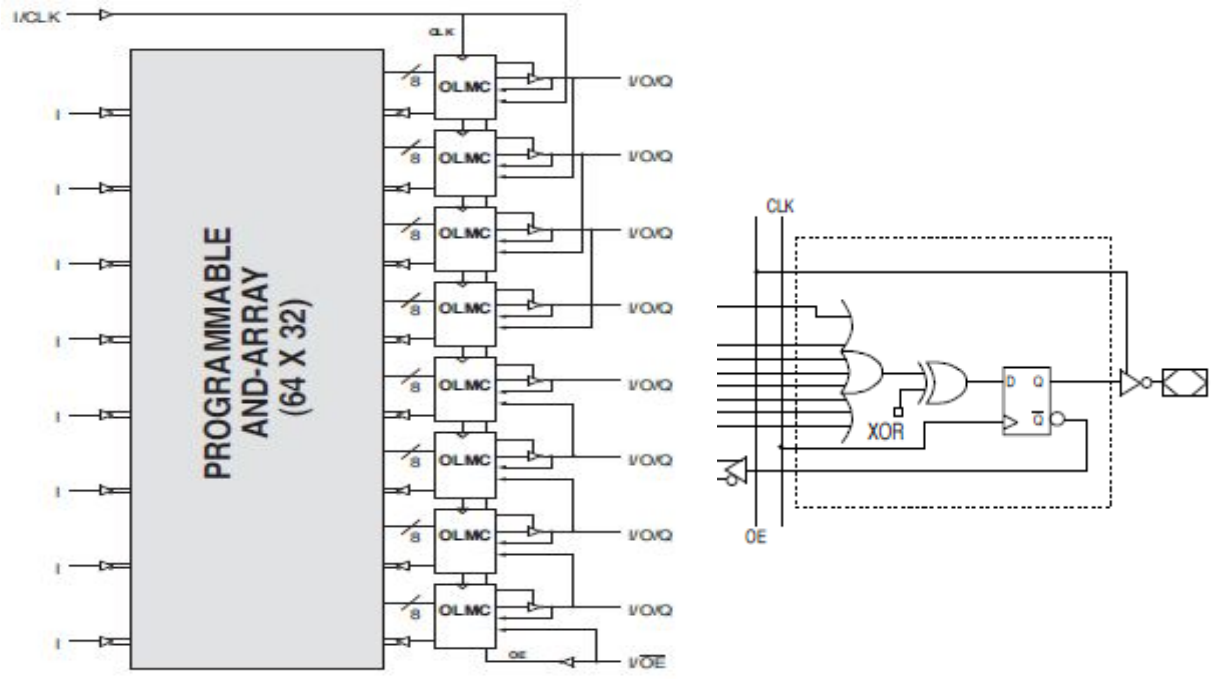
Начертайте картата на карно за 4 променливи, запълнете картата на Карно ползвайки следната последователност:
поставете 1 в позиция **¬C.¬A.¬B.¬D**
поставете 1 в позиции за които **¬C.D** е валидно, това са **¬A.¬B.¬C.D, ¬A.B.¬C.D, A.B.¬C.D, A.¬B.¬C.D**
поставете 1 в позиции за които **A¬BC** е валидно, това са **A.¬B.C.¬D, A.¬B.C.D**
поставете 1 в позициите за които **¬D** е валидно, това са всички леви и десни вертикални стълбове на картата на Карно
номерирайте групите по редове и колони и извършете групиране по 8 – първи и втори стълб, по 4 – последен ред и по 8 ляв и дясен стълб.

Отг. Y = A¬B + ¬C + ¬D

Архитектурни особености на CPLD и FPGA

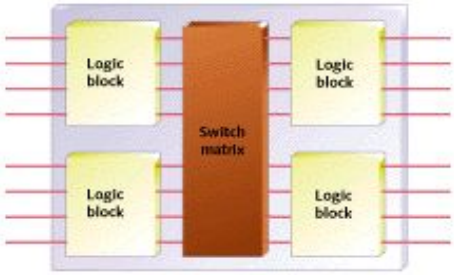
Интегрални схеми с програмируема структура съществуват на пазара над 30 години. Към настоящия момент се предлагат множество серии и семейства от този тип схеми. Нека посочим някои от по-големите производители на CPLD (Complex Programmable Logic Device) устройства: Altera, Atmel, Cypress Semiconductor, Lattice Semiconductor и Xilinx. За реализацията на логически функции първоначално се използват PROM (Programable Read-only Memory) чипове, които имат **адресна шина** с **m входа** и **шина за данни** с **n изхода**. На адресната шина се подава входния вектор, като всеки бит в него се интерпретира за отделна логическа променлива. Изходния сигнал се реализира черз извеждане на единичен бит по шината за данни **n**. Този тип устройства ви позволяват да реализирате множество логически функции от типа **И**, но техен основен недостатък е невъзможността да се синтезират повече от **n** логически функции. Това налага комбинирането на множество PROM чипове, което отнема ненужно много място на платките. Резонно производителите предлагат нов тип устройства на пазара, с общото название PLD (Programmable Logic Device), последвани от SPLD (Simple PLD), след които се въвеждат CPLD.

Архитектурно се различават два типа матлици: PLA (Programmable Logic Array – имащи **И-ИЛИ** програмируема матрица) и PAL (Programmable Array Logic – имаща само програмируема **И** структура), като за PAL е възможно да срещнете и означението GAL (Generic Array Logic) в зависимост от производителя. Следва да се каже, че съгласно закона на де Морган този тип устройства могат да се произвеждат и акот **ИЛИ-НЕ** матрици (Фиг. 15).



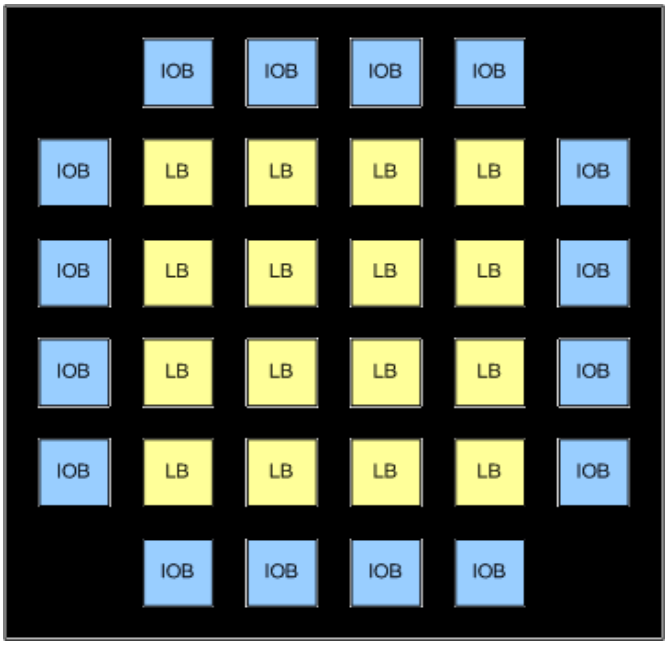
Фиг. 15 PAL (**GAL16LV8** - вляво**)** реализиран с **ИЛИ-НЕ (**в дясно**)** елементи

При CPLD се интегрират множество PLA под формата на макроклетки, като комуникацията между тях става по вградена шина (Фиг. 16). Различните разновидности позволяват свързването и връщането на сигнали от изходите на отделните макроблокове на техните входове.



Фиг. 16 Обобщена CPLD структура

Естествено развитието и темповете на интеграция на тези устройства нарастрат, и като най-сложни предлагани днес устройства можем да посочим FPGA (Field-programmable gate array). (Много от компаниите произвеждат CPLD с висока интеграция, което ги изравнява до някои от FPGA решенията.) Ще изброим основните FPGA производители: Actel, Altera, Aeroflex UTMC, Atmel, Lattice Semiconductor, NEC, QuickLogic и Xilinx. Самите FPGA могат да се разглеждат, като структури имащи специфични входно изходни блокове - IOB (input output block) и логически блокове - LB (logical block), които са свързани помежду си с програмируеми шини. Сложността на тези устройства е много висока, желаещите да научат повече следва да се обърнат към специализираната литература [6].



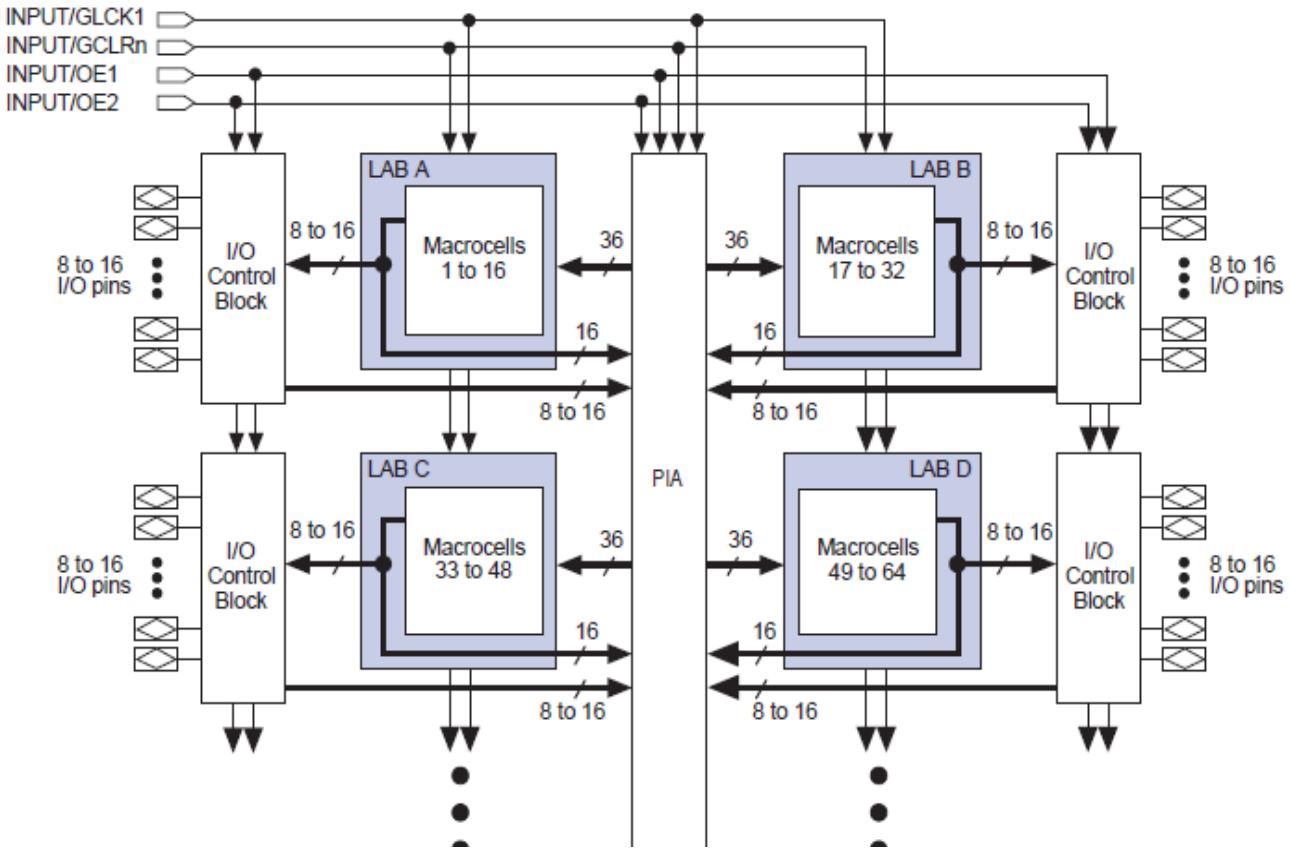
Фиг. 17 Обобщена FPGA структура (IOB-input output block, LB-logical block)

Тъй като използваната в упражненията демонстрационна платка използва MAX7000S чип, ще разгледаме серията чипове от тази фамилия работещи с напрежения от 5V, което ги прави особено удобни за дизайн на всякакъв тип електронно оборудване. Обърнете внимание на най-важните параметри на устройствата дадени в таблица 20 (брой макроклетки, брой логически елементи).

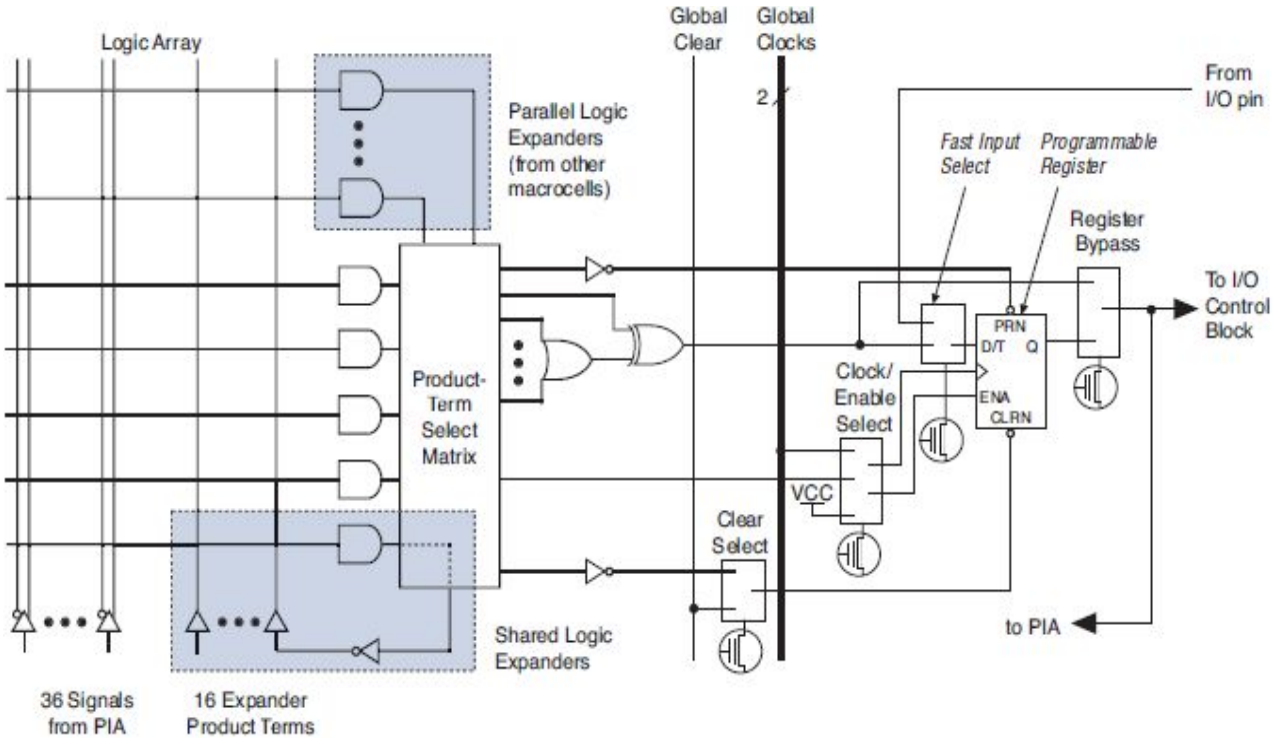
MAX 7000S (5.0 V)						
	EPM7032S	**EPM7064S**	**EPM7128S**	**EPM7160S**	**EPM7192S**	**EPM7256S**
Логически клетки	600	1,250	2,500	3,200	3,750	5,000
Макроклетки	32	64	128	160	192	256
Максимален брой входно изходни портове	36	68	100	104	124	164

Табл. 20 Сравнение на MAX7000S

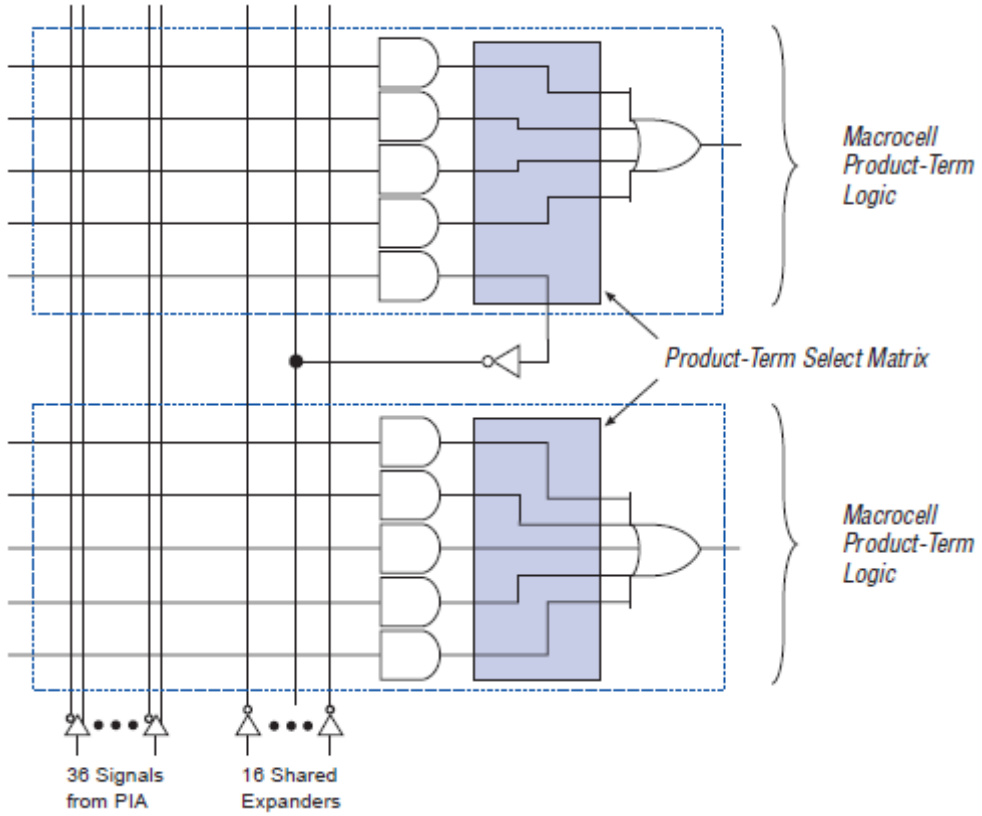
По-доло е дадена блокова схема на EPM7032-64-96 CPLD чип (Фиг. 18), в него са обособени отделни макроклетки (Фиг. 19), входно изходни контролни блокове и вътрешна комутируема шина. Отделните макроклетки имат възможност за използват някои от изходните си сигнали и да ги комутират обратно към входовете (Фиг. 20).



Фиг. 18 Блокова диаграма на EPM7032, EPM7064 и EPM7096



Фиг. 19 Структура на макроклетката на EPM7032, EPM7064 и EPM7096



Фиг. 20 Структура на общи функционални разширители

Общите функционални разширители могат да се ползват за свързване на неползваните сигнални входове на всяка една макроклетка, за да се образуват по-сложни логически функции. Разширителите могат да бъдат еднобитови и паралелни. При това се въвежда допълнително закъснение на сигнала, но за сметка на по-сложна реализирана схема.

Друга важна особеност на серията MAX7000 е вградената система за програмиране, това позволява генерирането на високо програмиращо напрежение за EEPROM паметта в самия чип при загранване от +5V. Възможността чиповете да се програмират след техния монтаж снижава възможностите за повреждане на устройствата при ръчно програмиране и инсталиране в сокетите на платката. За повече справки относно архитектурата на чипа следва да се обърнете към неговото описание от сайта на компанията http://www.altera.com/literature/ds/m7000.pdf.

Семейства CPLD и FPGA предлагани от Altera

ASICs

Тази серия устройства позволява разработката на решения, които ще се произвеждат масово, предлагат се по-ниски цени на един чип, използване на готови комуникационни, процесорни и други вградени блокове за сигнална обработка, кодиране, памет и др. Към настоящия момент се предлагат две фамилии устройства:

HardCopy IV GX - предназначени за реализация на висикоскоростни приме предаватели;

HardCopy IV E – предназначени за създаване на логически схеми, използване на различни памети и сигнална обработка.

CPLDs

Тези базови чипове се произвеждат от 1993г., основното им преимущество е ниската цена отнесена към налични входно изходни портове за един CPLD чип, а също така и малката консумация на енергия. Към настоящия момент не се препоръчва дизайнът на продукция с MAX 7000 серията, но те все още са налични на пазара.

MAX CPLD				
	MAX 7000S	**MAX 3000A**	**MAX II**	**MAX IIZ**
Година на производство	1995	2002	2004	2007
Топологична норма	0.5 µm	0.30 µm	0.18 µm	0.18 µm
Основни параметри	5V	Ниска цена вх. 2.5V, 3.3V, 5.0V изх. 2.5V, 3.3V	Голям брой портове 3.3V	Минимална консумация 3.3V, 2.5V, 1.8V, 1.5V
Максимална честота	60-100MHz	110-220MHz	300MHz	300MHz

Табл. 21 Сравнение на MAX серията

Low-Cost FPGAs

Тази серия FPGA е предназначена за комплексни решения с ниска консумация и цена. Серията предлага най-ниските цени на едно устройство при най-ниска консумация на енергия в целия индустриален бранш към настоящия момент. Устройствата поддържат и процесорно ядро Nios II.

Cyclone FPGA				
	Cyclone	**Cyclone II**	**Cyclone III**	**Cyclone IV**
Година на производство	2002	2004	2007	2009
Топологична норма	130 nm	90 nm	65 nm	60 nm
Основни параметри	Съвързване със стандартна периферия, памет, комуникационни интерфейси, шини	Изчисления и сигнална обработка	За следните приложения: военни, автомобилни, дисплеи, обработка на видео и сигнали	Препоръчва се за реализация на бродкастинг, и други безжични решения

Табл. 22 Сравнение на Cyclone FPGA серията

Midrange FPGAs

Тази фамилия схеми притежават високоскоростни комуникационни интерфейси, а по-новата версия и поддръжка на PCI Express, Gigabit Ethernet, Serial RapidIO, SDI, XAUI и др., което е гаранция за успех на пазара на **DSP** приложенията и разпределените системи.

Сравнение на Arria FPGAs с вградени приемо предавателни интерфейси		
Особеност	**ArRia GX**	**Arria II GX**
Година на производство	2007	2009
Топологична норма	90nm	40nm
Еквивалентни логически елементи	21,580 to 90,227	15,950 to 256,500
Адаптивни логически модули	8,632 to 36,088	6,380 to 102,600
RAM (Kbits)	1,229 to 4,477	783 to 8,550
Вградени умножители 18 x 18	40 to 176	56 to 736
Максимален брой входно изходни портове	235 to 538	250 to 612
Скорост на приемо предавателни интерфейси (Mbps)	600 to 3,125	600 to 3,750
Брой приемо предавателни интерфейси	4 to 12	4 to 16
PCI Express съвместимост с вградени IP блокове	-	1

Табл. 23 Сравнение на Arria FPGA серията

High-End FPGAs

Тази серия устройства е предназначена за приложенията с най-високи нужди от бързодействие и ниска консумация, FPGA притежават висока степен на интеграция и готови комуникационни интерфейси, правейки ги идеални за дизайн на комуникационно и комутационно оборудване от всякакъв клас.

Stratix FPGA					
	Stratix IV E	**Stratix III L**	**Stratix III E**	**Stratix II**	**Stratix**
Еквивалентни логически елементи	228,000 813,050	47,500 338,000	47,500 254,400	15,600 179,400	10,570 79,040
Адаптивни логически блокове	91,200 325,220	19,000 135,200	19,000 101,760	6,240 71,760	N/A
RAM (Kbits)	14,283 23,130	1,836 16,272	5,328 14,688	410 9,163	899 7,253
Умножители 18x18	960 - 1288	216- 576	384 - 896	48 - 384	24 - 88
LVDS (Mbps)	150 to 1,600	124 to 1,600	125 to 1,600	125 to 1,000	840
LVDS канали (Rx/Tx)	56/56 to 132/132	56/56 to 132/132	56/56 to 112/112	38/38 to 152/156	44/44 to 80/80

Табл. 24 Сравнение на Arria FPGA серията

Stratix FPGA с вградени приемо предавателни интерфейси				
	Stratix IV GT	**Stratix IV GX**	**Stratix II GX**	**Stratix GX**
Еквивалентни логически елементи	228,000 - 531,200	72,600 - 531,200	15,600 - 179,400	10,570 - 79,040
Адаптивни логически блокове	91,200 - 212,480	29,040 - 212,480	6,240 - 71,760	N/A
RAM (Kbits)	13.9to 20.3K	6,462 to 20,736	410 to 9,163	899 to 7,253
Умножители 18 x 18	1,024 - 1,288	384 - 1,288	64 - 252	24 - 88
Възможни скорости на приемане и предаване	2.5 - 11.3Gbps	600 - 8,500 Mbps	600 - 6,375 Mbps	500 - 3,875 Mbps
Вградени приемо предаватели	1.12.2024 г.	1.08.1932 г.	1.04.2020 г.	1.04.2020 г.
Скорост (Mbps)	2.5 - 6.5 Gbps	600 - 3,200	N/A	N/A
Брой приемо предавателни интерфейси	Up to 24	0 to 16	N/A	N/A
PCI Express вградени блокове	1	1 to 4	N/A	N/A

Табл. 25 Сравнение на Arria FPGA серията

ЛИТЕРАТУРА

[1] Brian Holdsworth, Clive Woods, “Digital logic design“, 1999
[2] Ronald. J. Tocci, Neal S. Widmer, “Digital Systems Principle and Applications”, 2004, ed. 8
[3] Р.И.Грушвицкии, А.Х.Мурасев, Е.П.Угрюмов, “Проектирование систем на микросхемах с програмируемой структорой – bhv”, 2006
[4] Л.Даковски, Н.Николов, “Ръководство по логика и програмируеми автомати“, 1990
[5] Aires Gustavo D. Sutter, Ge´ry jean Antoine Bioul „Synthesis of Arithmetic Circuits FPGA, ASIC, and Embedded Systems”
[6] Scott Hauck and Andr´e DeHon, "Reconfigurable computing - the theory and practice of FPGA-based computation", 2008

Георги Костадинов Петров има магистърска степен по телекомуникации, получава степен Доктор на Нов български университет през 2008г. Преподавателската му дейност е от 2006г. до сега и е свързана основно с курсове в областта на мултимедийни и широколентови комуникации, телекомуникационен софтуер и конфигурируеми интегрални схеми. Работл е по редица национални и международни проекти в областта на образованието, телекомуникациите и медицински системи. Професионалните му интереси са в области: медицински системи за събиране и анализ на сигнали, системи за телеметрия и мултимедийни системи за обработка и сегментация на цифрово видео.

Дизайн на цифрови електронни устройства с VHDL и Quartus II

част I - основи на булевата алгебра, основи на програмируемите логически устройства

1-во издание

Това учебно помагало е предназначено за студенти изучаващи следните курсове в програма Телекомуникации на НБУ: „Увод в програмируемите логически устройства“, „Съвременни конфигурируеми интегрални схеми“ и „Проектиране с програмируеми интегрални схеми“.

Обобщените примери и коментари, както и последователността на изложението го правят подходящо за всички начинаещи дизайнери на хардуер с VHDL. Книгата е подходяща и за програмисти на C/C++, които желаят да разширят своите познания в областта на хардуера и дизайнът на процесори и електронни блокове. Настоящото учебно пособие е подходящо и изчерпателно начално четиво в областта на разработката на проекти с програмиреуми логически устройства и използването на интегрирани среди за разработка на хардуер.

86 стр.