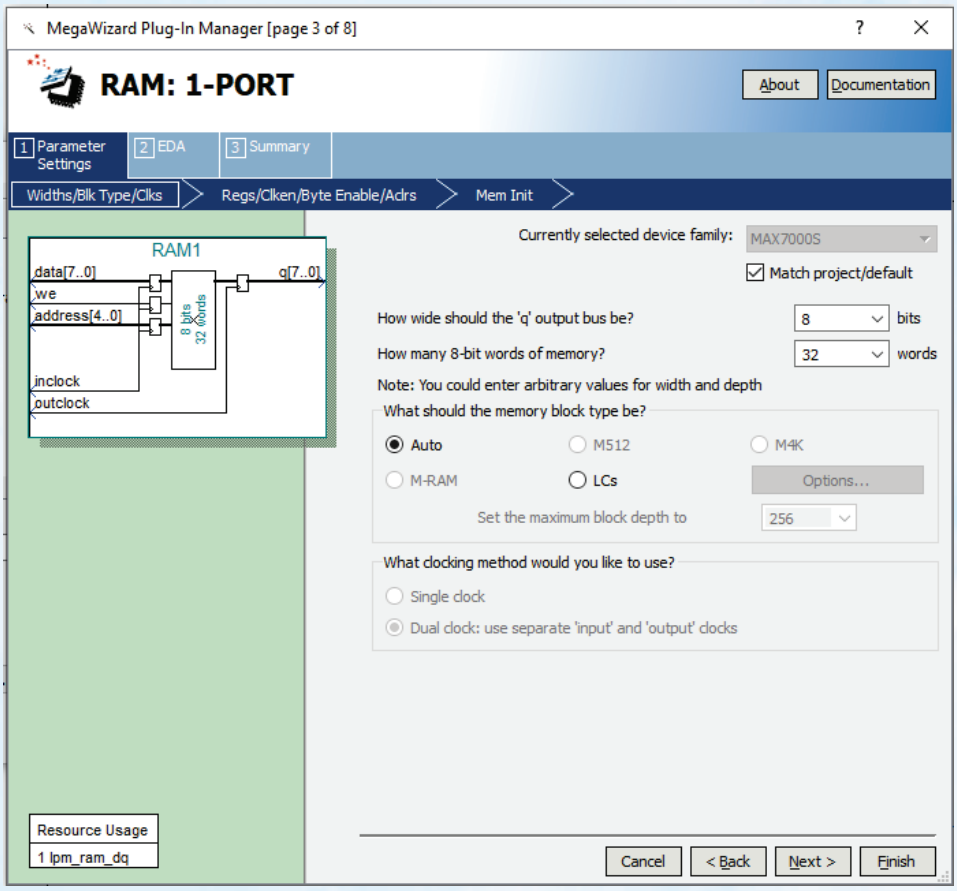
ГОРАН ГОРАНОВ

СХЕМНО ПРОЕКТИРАНЕ В СРЕДА QUARTUS II



Учебно-методическо пособие

УНИВЕРСИТЕТСКО ИЗДАТЕЛСТВО
ГАБРОВО
2024

ГОРАН ГОРАНОВ

СХЕМНО ПРОЕКТИРАНЕ В СРЕДА QUARTUS II

Учебно-методическо пособие

УНИВЕРСИТЕТСКО ИЗДАТЕЛСТВО
"ВАСИЛ АПРИЛОВ"
ГАБРОВО, 2024

Рецензент: доц. Валентина Василева Ранковска

Българска
Първо издание

ISBN: 978-954-683-712-7

Настоящото ръководство е въведение в системата за компютърно проектиране (CAD) Quartus II и предоставя преглед на типичните етапи на проектиране на програмируеми логически интегрални схеми (CPLD, FPGA) от идеята до крайния продукт. Описани са два подхода за проектиране на схеми в CPLD: първият е графичен вход, когато потребителят чертае схеми с помощта на готови шаблони, вторият е кодово описание на логическа схема. Методите за прехвърляне на конфигурационни файлове, съдържащи разработената логическа схема от персонален компютър към CPLD или FPGA чип, са показани с помощта на програматор USB-Blaster. Настоящото ръководство е предназначено за методическо подпомагане на дисциплините "Цифрови схеми и устрой-ства" и "Цифрова схемотехника и микроконтролери в комуникациите" изучавани в бакалавърска степен в катедра ЕЛЕКТРОНИКА.

Използвани съкращения

PLD – Programmable Logic Device

SPLD - Simple Programmable Logic Devices

CPLD – Complex Programmable Logic Devices

LAB - Logic Array Blocks

PIA – Programmable Interconnect Array

IDE - Integrated Development Environment

AHDL - Altera Hardware Description Language

VHDL - VHSIC Hardware Description Language

DSP - Digital Signal Processor

PLL – Phased Locket Loop

MOSFET - Metal-oxide-semiconductor field-effect transistor

CAD - Computer-aided design

IOB – Input/Output Block

CLB - Configurable logic Block

EEPROM – Electrically erasable programmable read-only memory

LUT - Lookup table

ВЪВЕДЕНИЕ

Програмируемо логическо устройство (PLD) е електронен компонент, използван за създаване на цифрови схеми. Функциите на CPLD, за разлика от конвенционалните микросхеми, се задават чрез програмиране (проектиране) и не се определят по време на производството. За да зададете необходимата структура на цифрово устройство, се използват програматор и среда за проектиране (IDE). В резултат на това се формира схема или програма на специални езици за описание на хардуера: Verilog, AHDL, ABEL, VHDL и др.

В момента има много производители на CPLD чипове (Xilinx, Altera, Lattice) и всеки от тях има собствена система за компютърно проектиране (CAD). В курса на обучение работим с два вида CPLD и CAD: (Quartus II 13.0sp1 WEB) от Altera и Xilinx - Xilinx ISE 10.1.

CAD Quartus II е многофункционална среда за проектиране, която съдържа набор от помощни програми, които ни позволяват да разработим, верифицираме и програмираме проект за избрано семейство CPLD чипове.

Логическа диаграма, проектирана в софтуера Quartus II, се нарича проект. Проектът може да има йерархична структура или да се състои от много модулни файлове, като основният модул съдържа няколко допълнителни модула, а всеки допълнителен модул включва още няколко файла. В този случай основният модул се нарича обект на най-горното ниво на йерархията. CAD Quartus II, в допълнение към йерархичната структура, съхранява в един каталог цялата техническа информация, свързана с избраната микросхема (например: присвояване на изводи, предишни опции за логическо проследяване, конфигурация на проекта и др.). Тези файлове се създават в една папка на проекта, което е полезно при копиране и преместване на проекти в други директории или на други компютри.

В настоящото ръководство се описват стъпките за изпълнение на проект, като се използва пример с един файл без йерархично подчинени файлове от по-ниско ниво (сложни проекти с йерархична структура няма да се разглеждат). Разглежда се обща информация за PLD, както и пълна последователност от действия при разработване на проект в среда Quartus II, от създаване на логическа схема до въвеждането й в специфичен чип (CPLD или FPGA конфигурация). За утвърждаване на разглеждания материал се дават контролни задачи, препоръчани за самостоятелно изпълнение.

1. ХАРАКТЕРИСТИКИ НА PLD АРХИТЕКТУРА

PLD се използва за създаване на дадена цифрова интегрална схема. Структурата на PLD се определя чрез програмиране с помощта на програматор и софтуер.

В повечето случаи PLD чипът се състои от набор от регулируеми логически блокове, които изпълняват необходимата логическа функция; програмируеми превключватели, които осъществяват необходимото свързване на логически клетки; програмируеми I/O блокове, които осигуряват комуникация между външния изход на микросхемата и вътрешната структура на PLD. В съвременните CPLD-FPGA често са вградени допълнителни блокове памет, DSP (Digital signal processor) блокове или умножители, PLL (Phased Locket Loop) и други компоненти.

Ако се разработва проект, тогава неговият разработчик може като правило да не вземе предвид характеристиките на елементите, които съставляват PLD. В същото време той описва желаните функции на проектираната микросхема или пише текст в софтуерната система (например на Verilog или VHDL езици за описание на хардуера). Необходимата схема се конструира от компилатора въз основа на известната вътрешна структура на PLD, която сама подрежда елементите на веригата според наличните конфигурируеми логически блокове и свързва тези блокове с помощта на наличните програмируеми комуникационни линии.

1.1 Класификация на PLD по тип на съхранение

За физическото осъществяване на връзки между PLD възли, производителите използват еднократно програмируеми и препрограмируеми устройства.

В първия случай се използва, както MOSFET транзистори с плаващ гейт, така и чрез „джъмпери“ между пътечки - antifuse технология.

При използване на транзистори с плаващ гейт се създава връзка чрез превключване на транзистора в проводимо състояние, което се осъществява чрез прилагане на програмен импулс. Въздействайки върху такъв транзистор с ултравиолетово лъчение, той може да бъде върнат в първоначалното си, непроводимо състояние. Въпреки това, в еднократно програмируемо устройство с тази структура, кристалите са защитени от светлина и програмирането в този случай е възможно само веднъж.

При внедряване на технология antifuse програмирането се извършва чрез свързване (не физически) на специални джъмпери на определени места

на чипа, за да се образува желаната верига. Достойнството на такива PLD е, че те работят достатъчно бързо (могат да работят на високи честоти), помалко податливи са на повреди по време на радиоактивно облъчване.

За създаване на многократно програмируеми токопроводящи връзки в момента се използват както полеви транзистори с двоен гейт от типа MOSFET, произведени по FLASH технология, така и конвенционални.

В микросхемите, базирани на флаш, конфигурацията се съхранява във вътрешна FLASH памет или EEPROM памет. Връзката се създава чрез превключване на полевия транзистор в проводимо състояние, което се осъществява чрез прилагане на програмен импулс. Този подход позволява многократно да променяме конфигурацията на разработеното устройство, като тя се запазва, когато захранването е изключено. Този тип PLD обаче има и недостатъци. При внедряване на FLASH памет в CMOS микросхема се изисква комбиниране на два различни технически процеса - това води до увеличаване на разходите за производство на микросхемите. В допълнение, такива микросхеми, като правило, имат ограничен брой цикли на презаписване на конфигурацията.

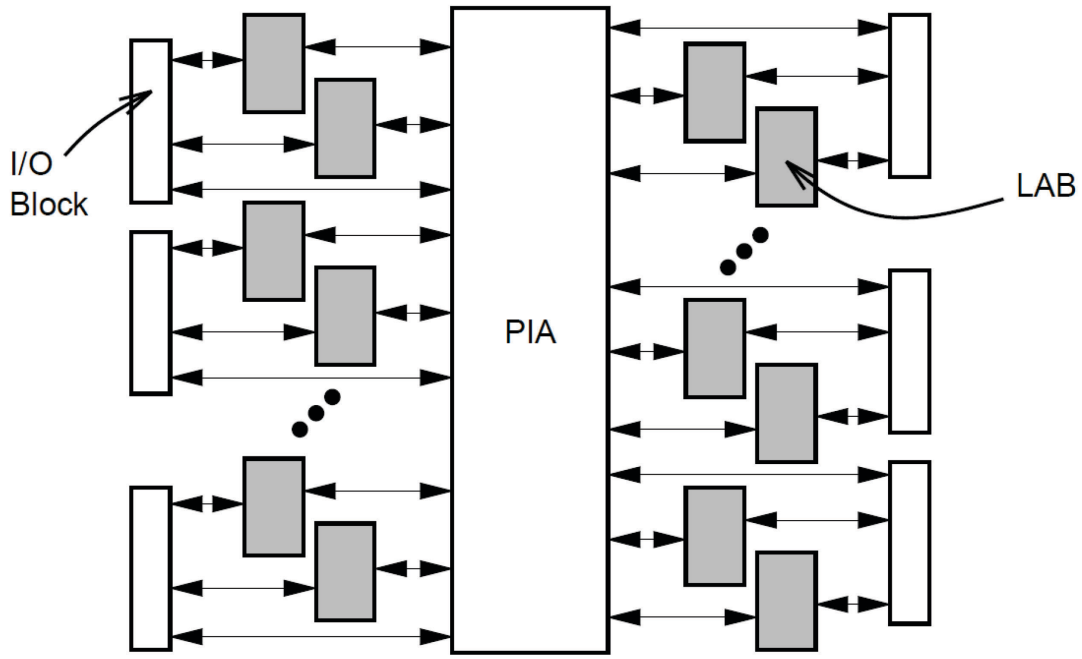
В микросхеми с тригерна памет за съхранение на конфигурация (базирана на SRAM) обикновен транзистор с полеви ефект действа като джъмпер, чиито гейт е свързан към тригера. Високото ниво на сигнала на неговия изход поставя транзистора в проводимо състояние, а ниското го държи изключен. Тригерът се настройва в необходимото състояние по време на зареждане на конфигурационната програма и остава в желаното състояние докато FPGA е включена – под напрежение. Такова онлайн програмиране може да се изпълнява неограничен брой пъти. Въпреки това, в този случай, когато захранването е включено, е необходимо да стартирате процеса на инициализация на конфигурацията, който отнема определено време. В същото време платката трябва да има и bootloader, специален FLASH чип или микроконтролер - всичко това води до увеличаване на цената на продукта.

1.2 Intel-Altera CPLD

Altera разработва три семейства чипове, които се вписват в категорията CPLD: MAX 5000, MAX 7000 и MAX 9000. Тук дискусията ще се фокусира върху серията MAX 7000, защото е широко използван и предлага задоволителен логически капацитет и скоростна производителност. MAX 5000 представлява по-стара технология, която предлага рентабилно решение, а MAX 9000 е подобен на MAX 7000, с изключение на това, че MAX 9000 предлага по-висок логически капацитет (най-високият в индустрията за CPLD).

Общата архитектура на серията Altera MAX 7000 е изобразена на фигура1. Тя включва масив от блокове, наречени логически масиви (LAB), и междусвързващи проводници, наречени програмируеми междусистемни

масиви (PIA). PIA е в състояние да свърже всеки вход или изход на LAB с всяка друга LAB. Също така, входовете и изходите на чипа се свързват директно към PIA и към LAB. LAB може да се разглежда като сложна структура, подобна на SPLD, и така може да се разглежда целият чип да бъде масив от SPLD. MAX 7000 устройства се предлагат, както на базата на EPROM, така и на EEPROM технология. Доскоро, дори с EEPROM, MAX 7000 чипове можеха да бъдат програмирани само „извън платката“ в програмен блок със специално предназначение, обаче през 1996 г. Altera пуска серия 7000S, която е препрограмируема „в системата“.

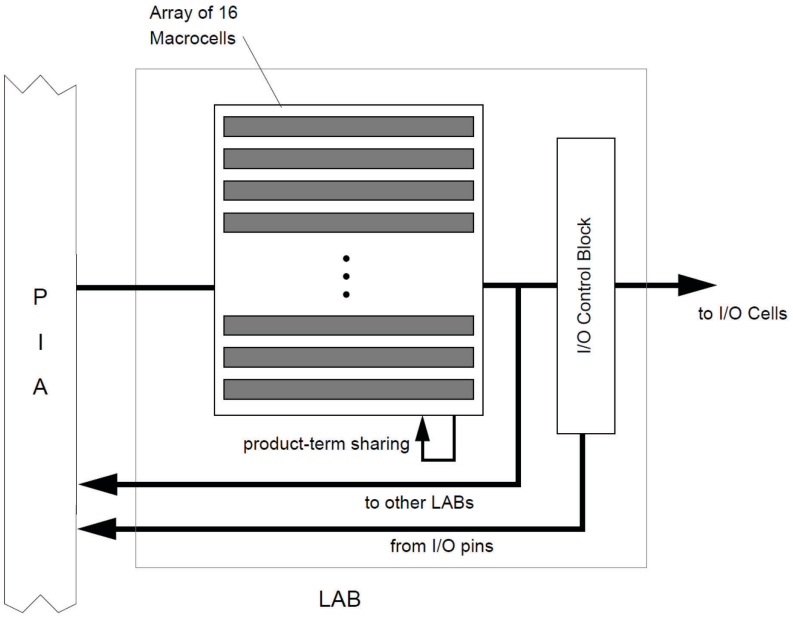


Фиг.1. *Intel (Altera) MAX7000*

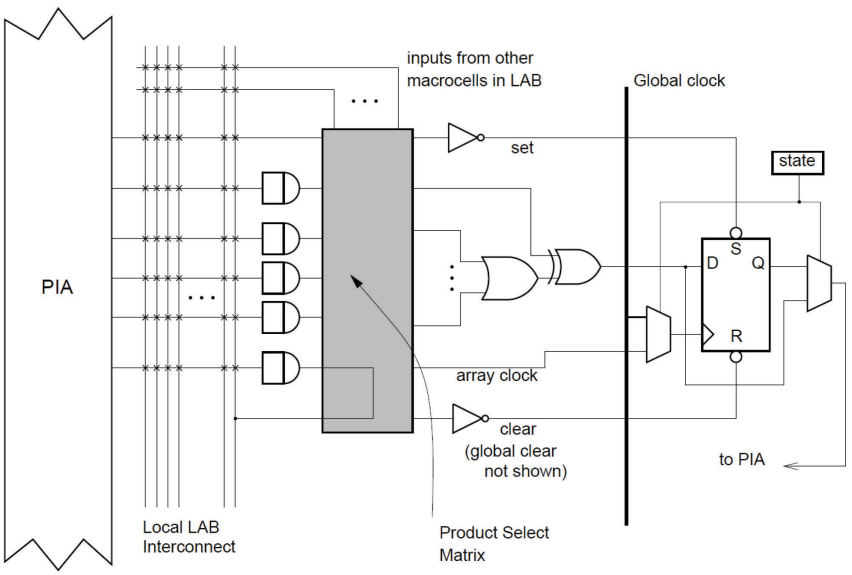
Структурата на LAB е показана на фигура 2. Всяка LAB се състои от два комплекта от осем макроклетки (показан на фигура 3), където макроклетка включва набор от програмируеми продуктови термини (част от И-равнина), която захранва елемент ИЛИ и тригера. Тригерите могат да бъдат конфигурирани като тип D, JK, T, SR или могат да бъдат „прозрачни“. Както е илюстрирано на фигура 3, броят на входовете към елемента ИЛИ в макроклетка е променлив; OR-gate може да се захранва от всеки или всичките пет продуктови термини в макроклетката и в допълнение може да има до 15 допълнителни продуктови термини от макроклетки в същата LAB. Тази гъвкавост на термините на продукта прави LAB от серия MAX 7000 по-ефективен термини на площта на чипа, тъй като типичните логически функции

не се нуждаят от повече от пет продуктови термини и архитектурата поддържа по-широки функции, когато са необходими. Интересно е да се отбележи тази променлива ИЛИ с размери от този вид не са налични в основните SPLD. Подобни характеристики от този вид се намират в други CPLD архитектури.

Освен Altera, няколко други компании произвеждат устройства, които могат да бъдат категоризирани като CPLD. Например, производителите на AMD от семейството Mach, Lattice има серия (i)pLSI, Xilinx произвежда серия CPLD, която те нарича XC9500, а ICT разполага с PEEL масив.



Фиг.2. *Altera LAB*



Фиг.3. *MAX7000 макроклетка на Altera*

1.3 Конфигурируеми логически блокове на FPGA

Документацията на Altera, както вече знаем, използва израза Logic Array Block (LAB) за позоваване на конфигурируеми логически блокове. Xilinx има подобни блокове в FPGA чиповете - Configurable Logic Block (CLB). Конфигурируемият логически блок е основен елемент в FPGA; в него може да се изпълнява някаква проста логическа функция или резултатът от изчислението може да се съхранява в регистри (тригери). FPGA се състои от правоъгълен масив от конфигурируеми логически блокове (CLB), заобиколени от входно/изходни блокове (IOB). Програмируемите трасиращи линии са разположени между CLB. Отделните CLB нямат отделни изходи, свързани към външните FPGA изводи. Вместо това функциите за преобразуване на сигнали изпълняват специални ресурси - IOB. Между CLB матрицата и I/O блоковете има отделни връзки, които осигуряват свързването на външни сигнали.

Сложността и структурата на CLB се определят от производителя. CLB може да бъде прост като един транзистор или сложен като процесор (това са екстремни точки на изпълнение).

В първия случай са необходими голям брой програмируеми връзки, за да се сглоби необходимата схема от отделни транзистори. Във втория случай не са необходими толкова много връзки, но се губи гъвкавостта при проектиране на персонализирана схема.

На практика конфигурируемият блок обикновено е достатъчно сложен, за да приложи някаква функция, но достатъчно малък, за да побере много такива блокове вътре в FPGA, което позволява гъвкавост на дизайна.

По този начин изборът на структурата на конфигурируем логически блок от производителя на FPGA винаги е търсене на компромис по отношение на площта на чипа, скоростта, консумацията на енергия и т.н.

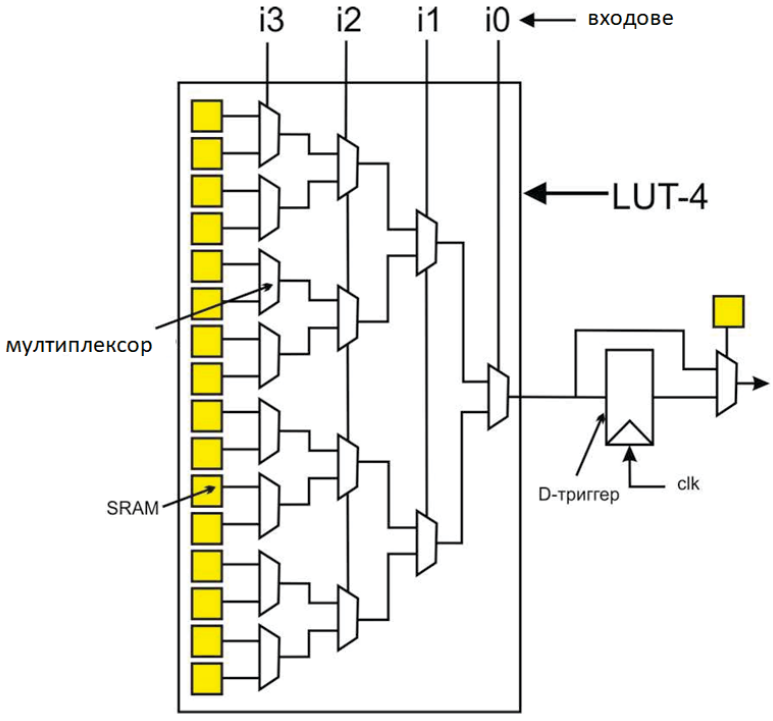
Конфигурируемият логически блок може да се състои от един или повече основни логически елементи. В английската литература това е Basic Logic Element (BLE) или просто Logic Element (LE). FPGA обикновено използват така наречените LUT-базирани базови логически елементи. Съкращението LUT означава Look-Up Table, което буквално може да се преведе като "референтна таблица" или "таблица за търсене". LUT е метод за изпълнение на функция, при която директното изчисление се заменя с търсене в таблица с готови решения. По отношение на FPGA, това ви позволява да реализирате всяка логическа функция под формата на SRAM памет, където адресът е аргумент, а съдържанието на клетката е стойност. Необходимата таблица за истинност се съхранява като маска (LUT-маска) в съответната SRAM клетка. С помощта на мултиплексори се избира желаната стойност. Мултиплексорите се управляват от сигнали на входния порт. Изграждането

на k-входен LUT (k-LUT), който реализира всяка логическа функция от k променливи, изисква 2k бита SRAM и 2k-1 мултиплексори. С този подход е възможно точно да се предвиди времето за преминаване на сигнала и то няма да зависи от изпълняваната логическа функция. Тази важна характеристика прави възможен времевия анализ на веригата.

Например фиг. 4 показва четирибитов LUT като част от основен логически блок. Тук на четирибитово число на входа на логическа функция се присвоява еднобитов резултат. Квадратите на фиг. 1 обозначават програмируем елемент (SRAM) или регистър, където се съхранява фърмуерът за FPGA. От фиг. 1 показва, че за конфигуриране на 4-битов LUT са необходими 16 конфигурационни регистъра. Съдържанието на тези регистри определя логическата функция, реализирана в основния логически елемент.

Друг конфигурационен регистър (на фиг. 1 е единичен квадрат вдясно), който определя дали е необходимо да се изведе стойността от LUT директно към изхода на основния логически елемент или е необходимо да се изведе стойността от LUT, фиксиран в D-тригера. Записването и съхраняването на данни в цифрови схеми е необходимо в почти всеки проект.

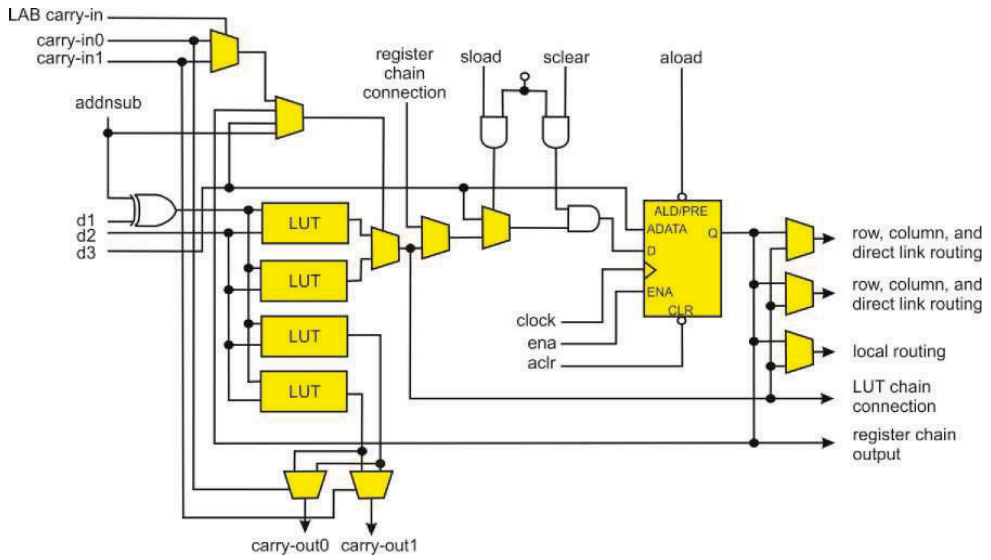
В крайна сметка конфигурационните регистри (квадрати) не са директно достъпни за използване в цифров проект. Те служат само за формиране на персонализирана функция. Един D-тригер в потребителски проект изисква повече от 16 (понякога много повече) тригера за съхраняване на конфигурацията на FPGA.



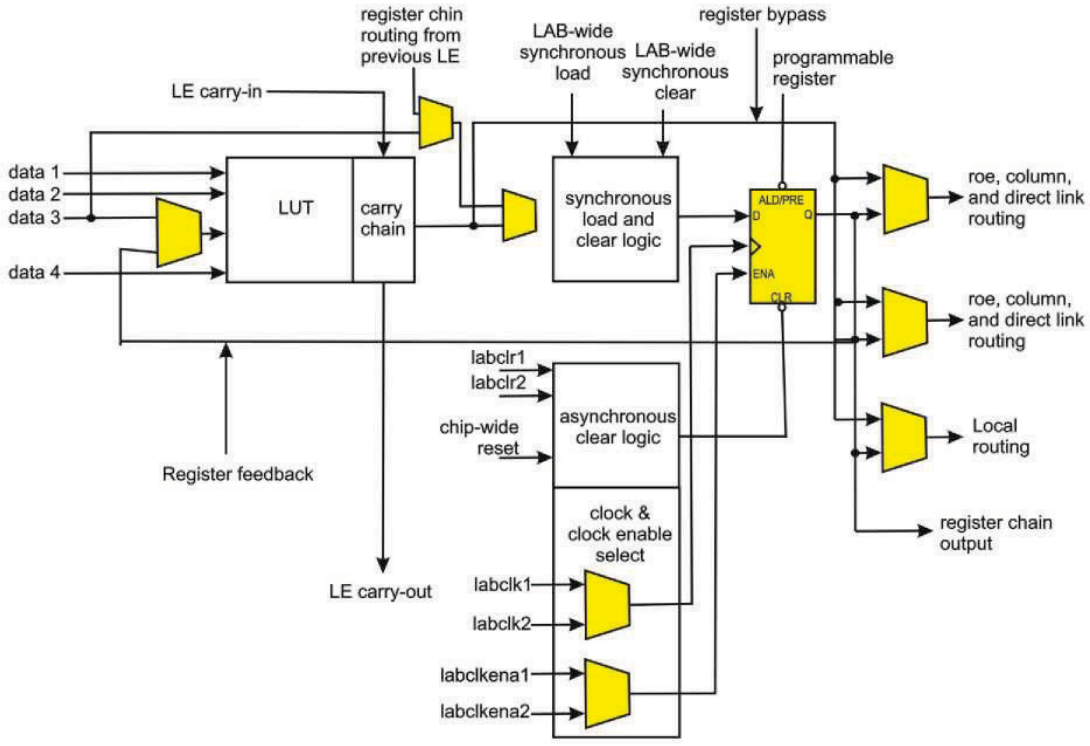
Фиг. 4. *Четирибитов LUT*

Понякога основният логически елемент в други FPGA е много по-сложен, отколкото е показано на фиг. 4. По-долу са дадени няколко примера от документацията за различни видове FPGA. На фигура 5 се показва основния логически елемент на чипа CPLD MAX II на Altera. Тук ясно се виждат LUT и D-тригерът за съхранение на резултата.

По-долу, на фиг. 6, е показан основният елемент на чипа Cyclone III.



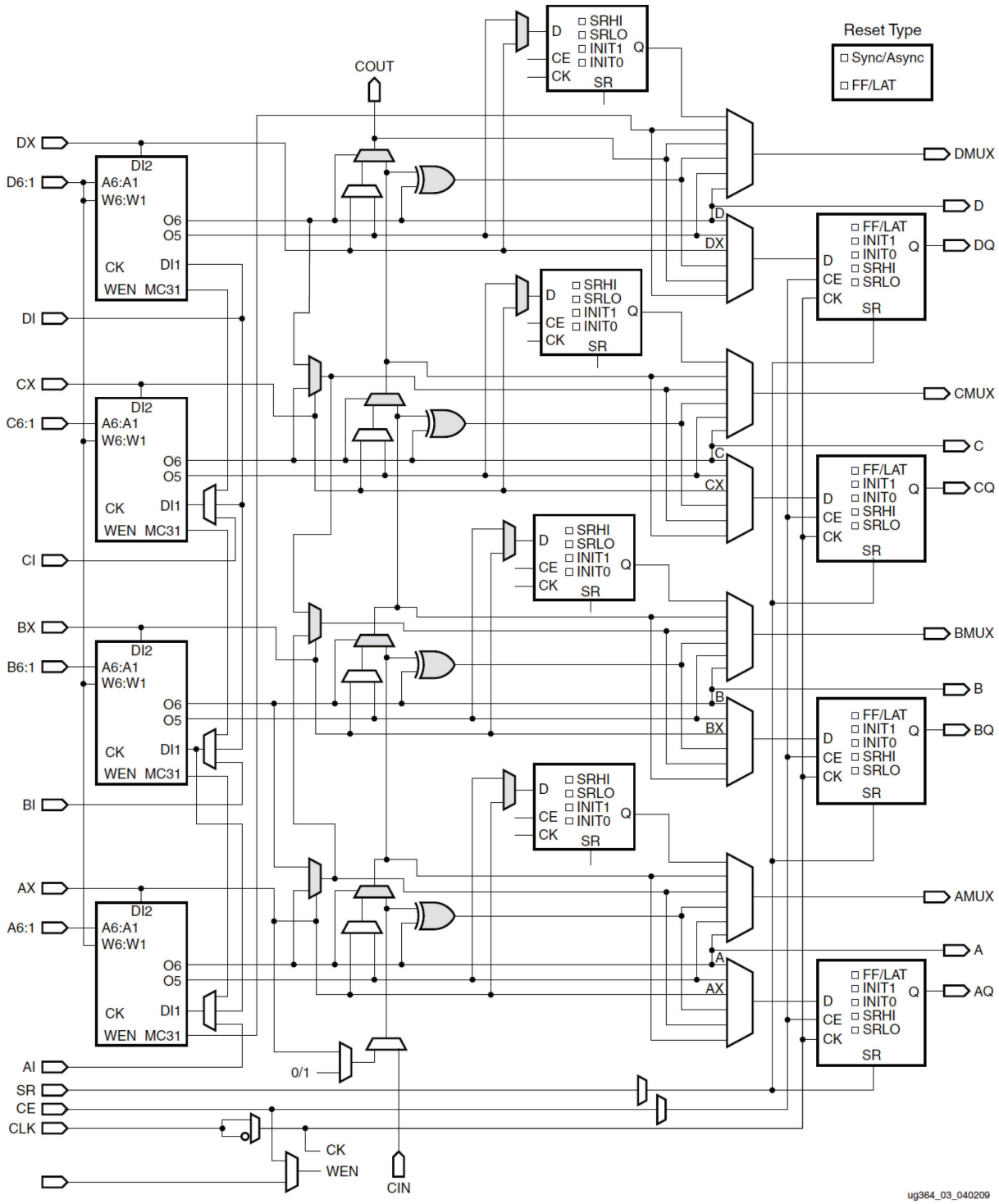
Фиг. 5. *Основен логически елемент CPLD MAX II от Intel-Altera*



Фиг. 6. *Базов логически елемент на FPGA Cyclone III на Intel-Altera*

В чиповете на Altera един LAB може да съдържа 1016 логически елемента.

В микросхемите Xilinx Virtex-6 основният логически елемент е т. нар. Slice. В едно CLB има два Slice. От друга страна, един Slice е доста сложно устройство (фиг. 7), като в един CLB Virtex-6 има 8 LUT и 16 D-тригера.



Фиг. 7. *Базов елемент AMD-Xilinx Virtex-6 Slice*

1.4 Софтуер за проектиране в CPLD и FPGA

CAD (Computer Aided Design) за проектиране на CPLD и FPGA, а именно компилатор (логически синтезатор, фитер и асемблер) е може би най-сложната част от цялата FPGA технология. Компилаторът трябва да анализира потребителския проект (схеми и текстови описания във Verilog HDL или VHDL) и да генерира netlist (netlist) - списък на всички елементи на веригата и връзката между тях. Netlist трябва да бъде оптимизиран - логическите функции трябва да бъдат сведени до минимум, възможните дублиращи се регистри трябва да бъдат премахнати. След това компилаторът трябва да вмести цялата логика от netlist в съществуващата CPLD или FPGA архитектура. Това прави фитера. Той поставя логическите елементи и проследява връзките между тях (процес на място и маршрут). Трудността се крие във факта, че един и същ проект може да бъде поставен в FPGA по различни начини и има милиони от тези начини. Някои разположения и маршрути са по-добри от други. Основният критерий за качество на получената система е максималната честота, с която проектът може да работи при дадено разположение на елементите и при дадено трасиране на връзки. Тук влияят дължината на връзките между логическите блокове и броя на програмируемите превключватели между тях.

Компилаторът, познавайки архитектурата на CPLD или FPGA, въз основа на резултатите от работата допълнително издава отчет за времето, необходимо за преминаване на сигналите от регистър към регистър. Тази информация често е полезна за проектантите на високопроизводителни системи. Разработчикът на CPLD и FPGA има възможност да даде някои съвети на компилатора: къде, на какво място в кристала е по-добре да поставите един или друг модул на проекта.

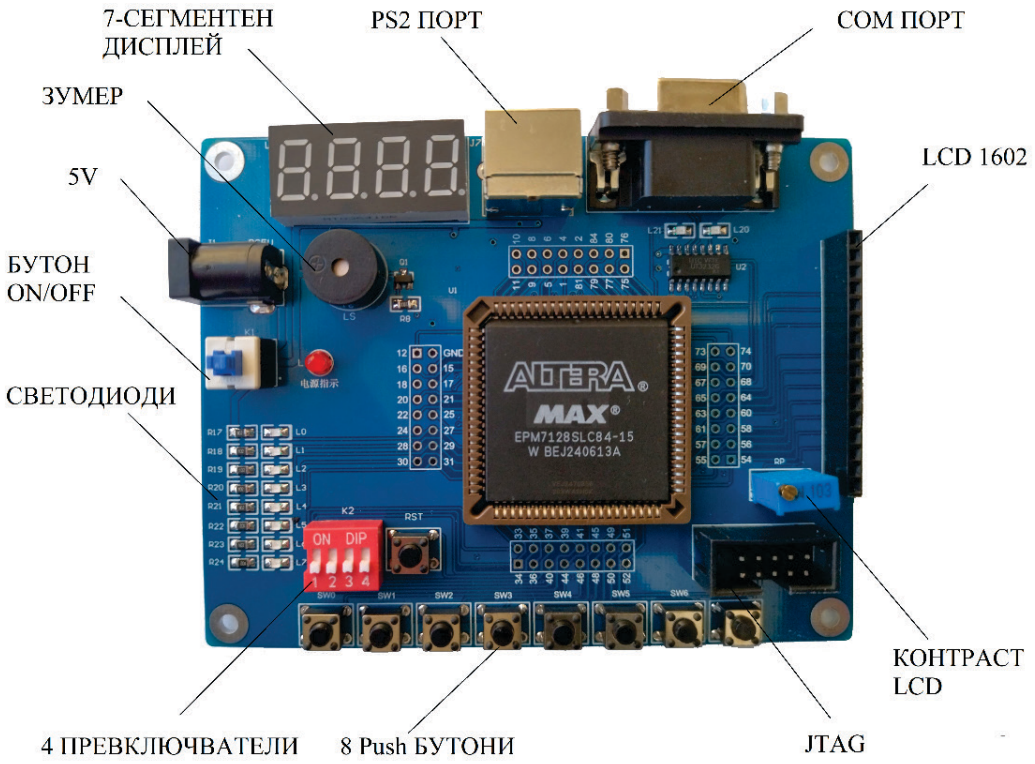
Избирайки конкретен модел CPLD или FPGA чип за своя проект, разработчикът до известна степен става зависим от производителя на тази CPLD или FPGA, тъй като той трябва да използва софтуер от същия производител в работата си.

Intel софтуера е наречен Quartus II. Софтуер за проектиране на Xilinx FPGA е ISE Suite или Vivaldo Design Suite. На Microsemi софтуера е Libero IDE или Libero SoC. Софтуерът и компилаторите за CPLD или FPGA са един от най-важните компоненти на интелектуалната собственост на компаниите производители на FPGA. В нашата работа ние разглеждаме MAX7000 CPLD на Intel (чип EPM7128SLC84-15 фиг.8), а проектирането се извършва в Quartus II V13.0 CAD.

Развойната система Intel (Altera) MAX7000S серия CPLD се програмира директно на платката чрез USB-Blaster (Altera).

2. Развойна система

Развойната платка притежава 1 захранващ интерфейс DC5V, 1 превключвател за захранване, 1 индикатор за захранване, 1 бутон за нулиране, 8 независими бутона, 4 DIP превключвателя, 1 JTAG интерфейс, 8 LED индикатора, 4 – 7 сегментни индикатора, 1 LCD1602 LCD интерфейс, 1 зумер, 1 интерфейс за клавиатура PS/2, 1 сериен порт и всички I/O портове са изведени, което е много удобно за допълнително използване (фиг. 8).



Фиг. 8. *Развойна платка CPLD-Altera CSE1*

Потребителят може да използват тази платка, за да прави следните експерименти и разработка на схеми, като например: вход с DIP превключвател – изследване на функция с 4 променливи, LED светлинни ефекти – бягаща светлина (изследване на броячи и регистри), управление на динамичен/статичен цифров 7 сегментен дисплей, управление на LCD дисплей, управление на зумер с различна честота, вход от PS/2 клавиатура – визуализиране на ASCII код, серийна комуникация и т.н. и всички налични I/O портове са изведени, което е удобно за вторичната разработка на потребителя и улеснява обучението и развитието на потребителите в най-голяма степен.

Номерата на изводите на CPLD свързани с всички външни компоненти са, както следва:

Табл. 1.

Означение	Номер на извод на CPLD	Вход/Изход
50MHz CLK	GCLK1 – PIN_83	ВХОД – активно ниво CLK
Push-button- RST	PIN_36	ВХОД – активно ниво 0
Push-button- SW0	PIN_44	ВХОД – активно ниво 0
Push-button- SW1	PIN_45	ВХОД – активно ниво 0
Push-button- SW2	PIN_46	ВХОД – активно ниво 0
Push-button- SW3	PIN_48	ВХОД – активно ниво 0
Push-button- SW4	PIN_49	ВХОД – активно ниво 0
Push-button- SW5	PIN_50	ВХОД – активно ниво 0
Push-button- SW6	PIN_51	ВХОД – активно ниво 0
Push-button- SW7	PIN_52	ВХОД – активно ниво 0
Button-SWDIP4(1)	PIN_41	ВХОД – активно ниво 0
Button-SWDIP4(2)	PIN_40	ВХОД – активно ниво 0
Button-SWDIP4(3)	PIN_39	ВХОД – активно ниво 0
Button-SWDIP4(4)	PIN_37	ВХОД – активно ниво 0
PS2/DATA	PIN_6	ВХОД - DATA
PS2/CLK	PIN_8	ВХОД – CLK
RS232-RXD	PIN_69	ИЗХОД – RXD
RS232-TXD	PIN_68	ВХОД – TXD
Зумер	PIN_12	ИЗХОД – активно ниво CLK
LED-L0	PIN_27	ИЗХОД – активно ниво 0
LED-L1	PIN_28	ИЗХОД – активно ниво 0
LED-L2	PIN_29	ИЗХОД – активно ниво 0
LED-L3	PIN_30	ИЗХОД – активно ниво 0
LED-L4	PIN_31	ИЗХОД – активно ниво 0
LED-L5	PIN_33	ИЗХОД – активно ниво 0
LED-L6	PIN_34	ИЗХОД – активно ниво 0
LED-L7	PIN_35	ИЗХОД – активно ниво 0
7-Seg M1	PIN_11	ИЗХОД – активно ниво 1
7-Seg M2	PIN_9	ИЗХОД – активно ниво 1
7-Seg M3	PIN_10	ИЗХОД – активно ниво 1
7-Seg M4	PIN_15	ИЗХОД – активно ниво 1
7-Seg – A,B,C,D,E,F,G,H	PIN_22, 24, 17,20,21,25,26,18	ИЗХОД – активно ниво 0
LCD1602 – RS, RW, E, D0-D7	PIN_54, 55, 56, (57,58,60,61,63,64,65,67)	ВХОД / ИЗХОД

3.Основни етапи на проектиране в QUARTUS II CAD

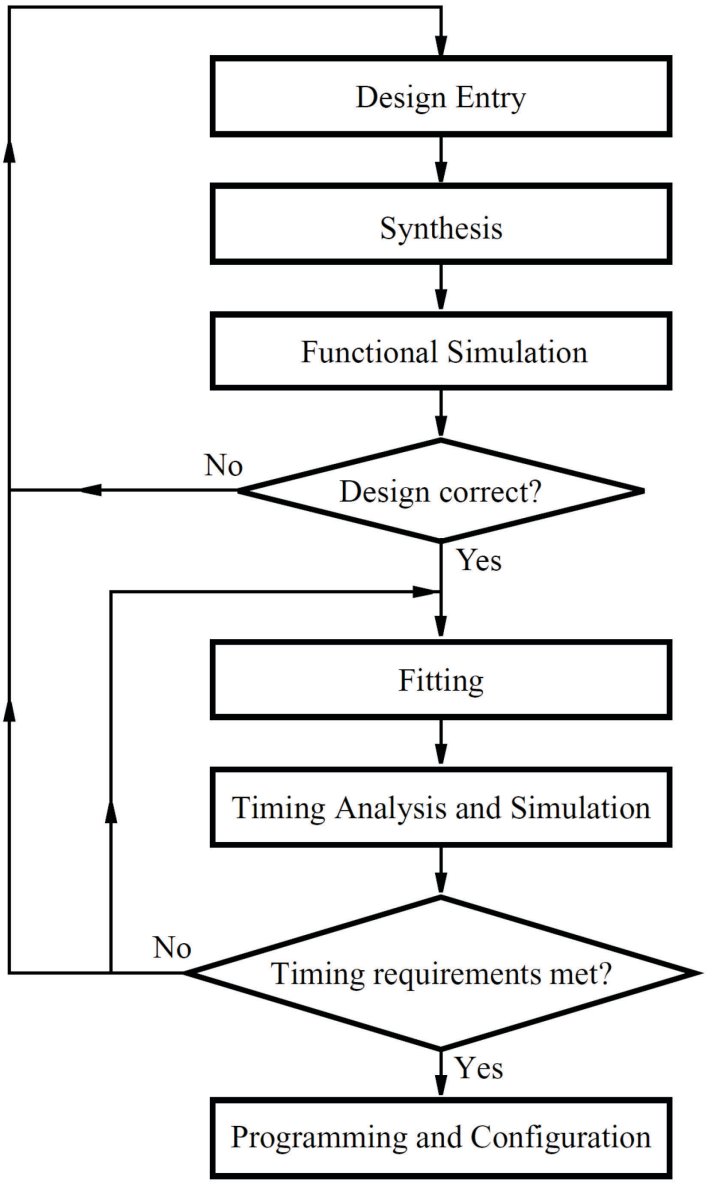
Quartus II улеснява прилагането на необходимата логическа схема в CPLD и FPGA. Основните етапи на проектиране в тази среда за разработка са показани на фиг. 9.

Проектирането в CAD Quartus II включва следните стъпки:

• Въвеждане на проекта (**Design Entry**) - желаното оформление на дизайна се посочва или графично, или с помощта на езици за описание на хардуера като Verilog HDL, VHDL и др.

• Синтез (**Synthesis**) – входният проект се синтезира във верига, която се състои от логически елементи (LE) и логически блокове (LB) в CPLD и FPGA чипа.

• Функционална симулация (**Functional Simulation**) - синтезираната схема се тества за правилно функциониране във вградения симулатор, който моделира зависимостта на състоянието на сигналите на веригата (избрани от разработчика) от времето. Тази симулация не взема предвид времеви закъснения на сигнала (наречени логически състезания) между LE и LB на CPLD и FPGA чипа.



Фиг.9.

• Трасировка (**Fitting**) – инструментът за проследяване, изчислява оптималното разположение и свързване на LE/LB, дефиниран в netlist на действителния CPLD или FPGA чип. Трасирането също така избира проводниците или „маршрута на сигналите“ в чипа, за да реализира необходимите връзки между дадения LE/LB.

• Анализ на времето (**Timing Analysis**) - анализира закъсненията при разпространението на сигнала по различни пътища в проследена логическа верига, за да изчисли наличието/отсъствието на логически надпревари, така че сигналите от различни LB да пристигат едновременно до един или друг краен възел на веригата.

• Симулация на времето (**Timing Simulation**) – Веригата се тества за функционалност и ограничения във времето, но за разлика от функционалната симулация, действителните закъснения на избраните сигнали се вземат предвид, за да се открие наличието или отсъствието на логически състезания.

• Програмиране и конфигуриране **(Programming and Configuration)** - разработената схема се поставя в CPLD или FPGA чипа чрез програмиране на електронни връзки между конфигурируемите LE / LB, което се реализира чрез прехвърляне на конфигурационния файл от компютъра или към FPGA(CPLD) чипа, или към допълнителни (не вградени в FPGA) памет, ако такава е налична в комплекта.

Освен това приложението на всеки етап ще бъде разгледано отделно, като се използва примерът за проектиране на проста логическа схема. За да направим това, трябва да извършим следната последователност от действия:

• Създаваме проект

• Въвеждаме логическата схема

• Синтез на връзките във веригата

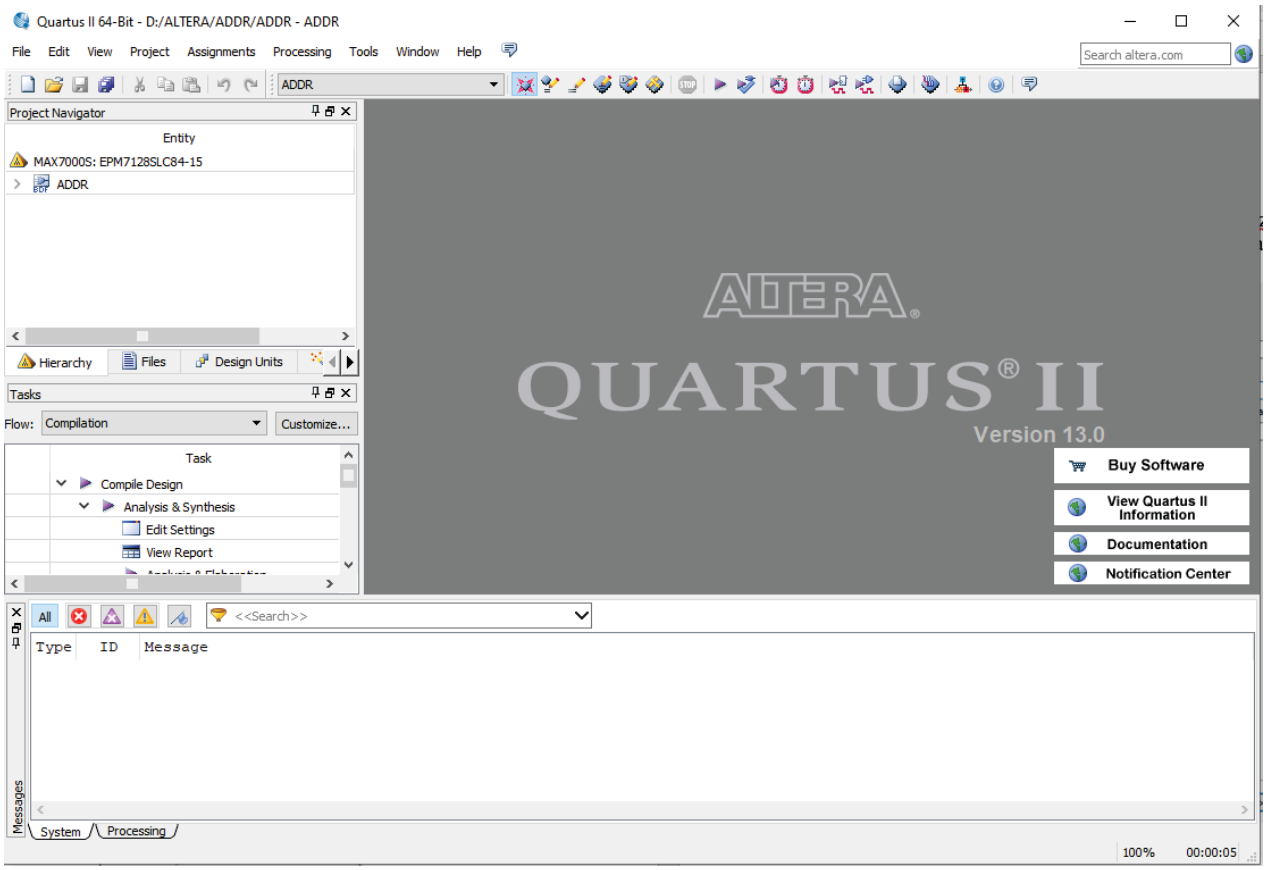
• Задаване на входове и изходи на схемата на избрания CPLD чип

• Моделиране на разработената логическа схема

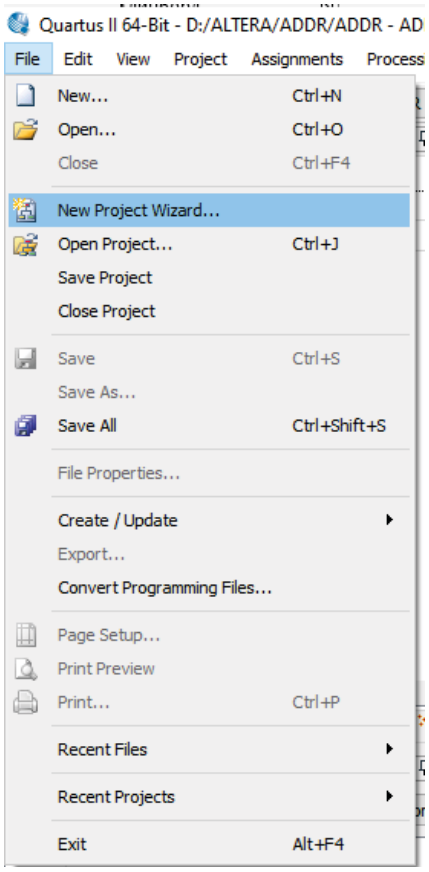
• Програмиране и конфигуриране на CPLD чипа.

3.1 Първи стъпки в средата на Quartus II

Първата стъпка при внедряването на нов проект с логическа диаграма е да се създаде директория за съхранение на проектните файлове. Стартирайте програмата Quartus II, прозорец, подобен на фиг. 10, който се състои от няколко прозореца, които осигуряват достъп до всички функции на софтуера Quartus II. Повечето от командите могат да бъдат достъпни чрез набор от менюта, намиращи се в лентата с инструменти. Избирането на бутона с име "Файл" отваря диалоговия прозорец, показан на фиг. 11.



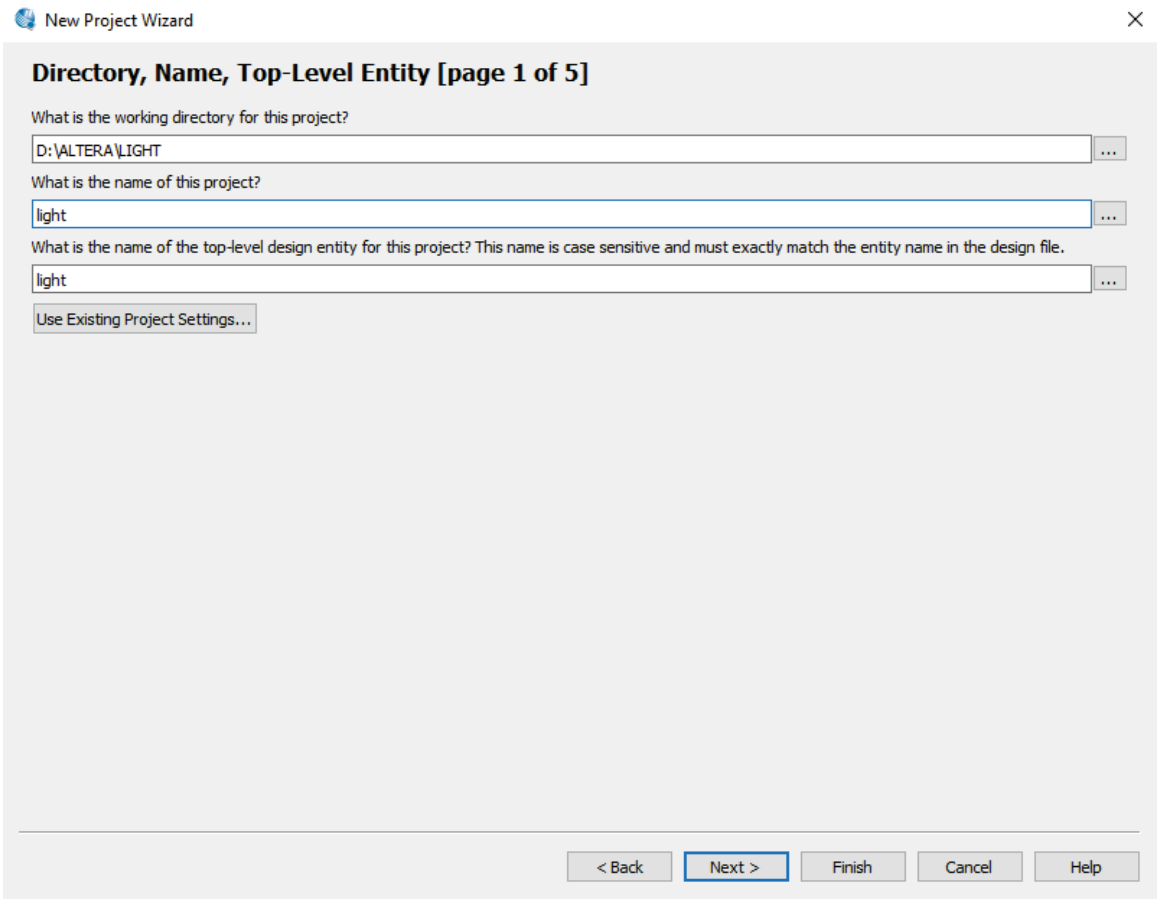
Фиг. 10. *Основен изглед на Quartus II*



Фиг. 11. *Меню „Файл“*

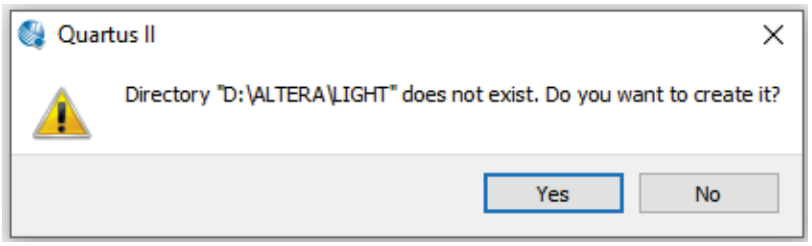
За да започнете, трябва да създадете нов проект:

1. От менюто "Файл" изберете "New Project Wizard", прозорец, подобен на фиг. 12, който пита за името и директорията на проекта.

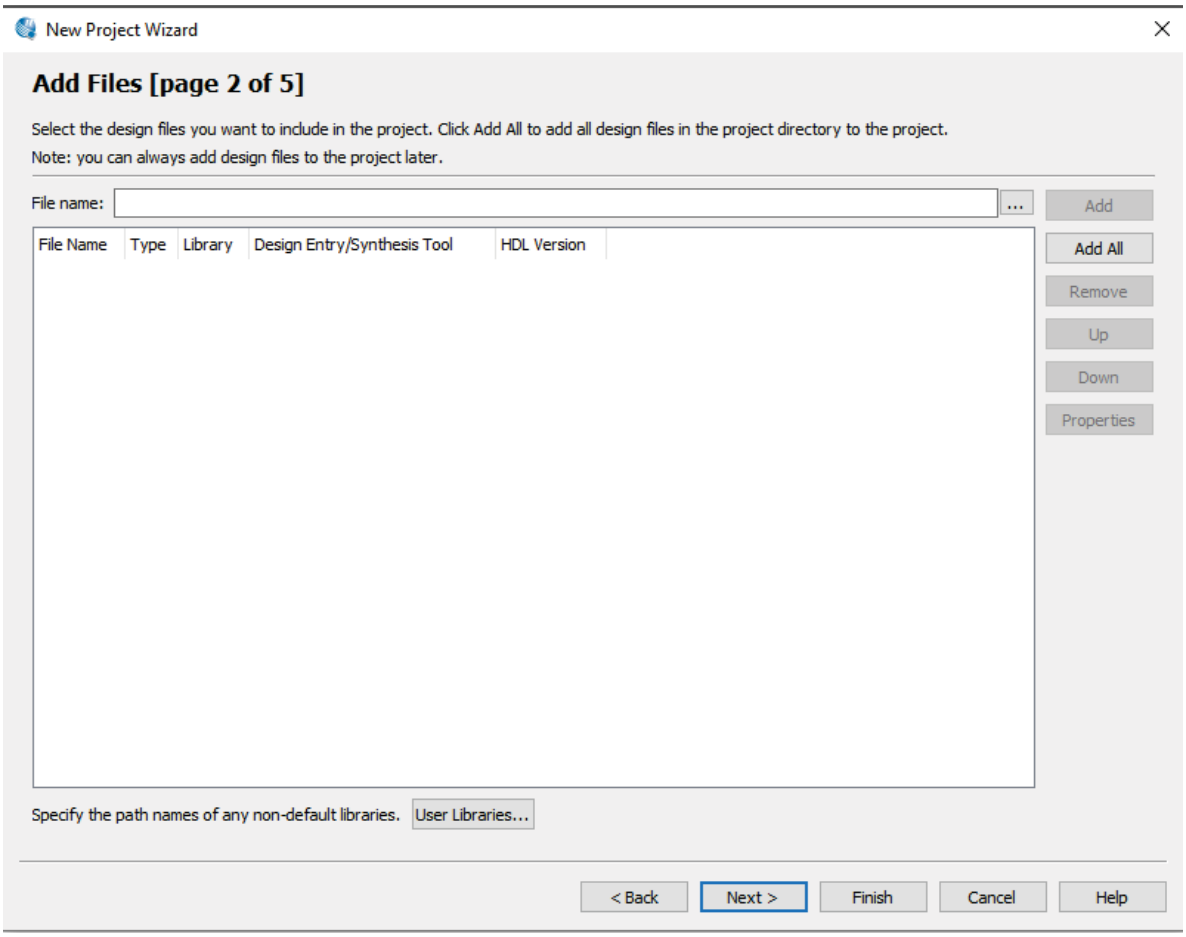


Фиг. 12. *Задаване име на проекта*

2. Изберете работна директория, например: introtutorial. Удобно е, когато името на проекта има същото име като обекта на проекта от най-високо ниво. В редовете „Какво е името на този проект“ и „Какво е името на обекта за проектиране от най-високо ниво за този проект“, въведете „light“ като име на самия проект и името на най-високото ниво обект (виж фиг. 12). След натискане на бутона "Напред", ще се появи изскачащ диалогов прозорец (фиг. 13), който ви подканва да създадете необходимата папка. След натискане на бутона "Да" излиза прозорецът, показан на фиг. 14.

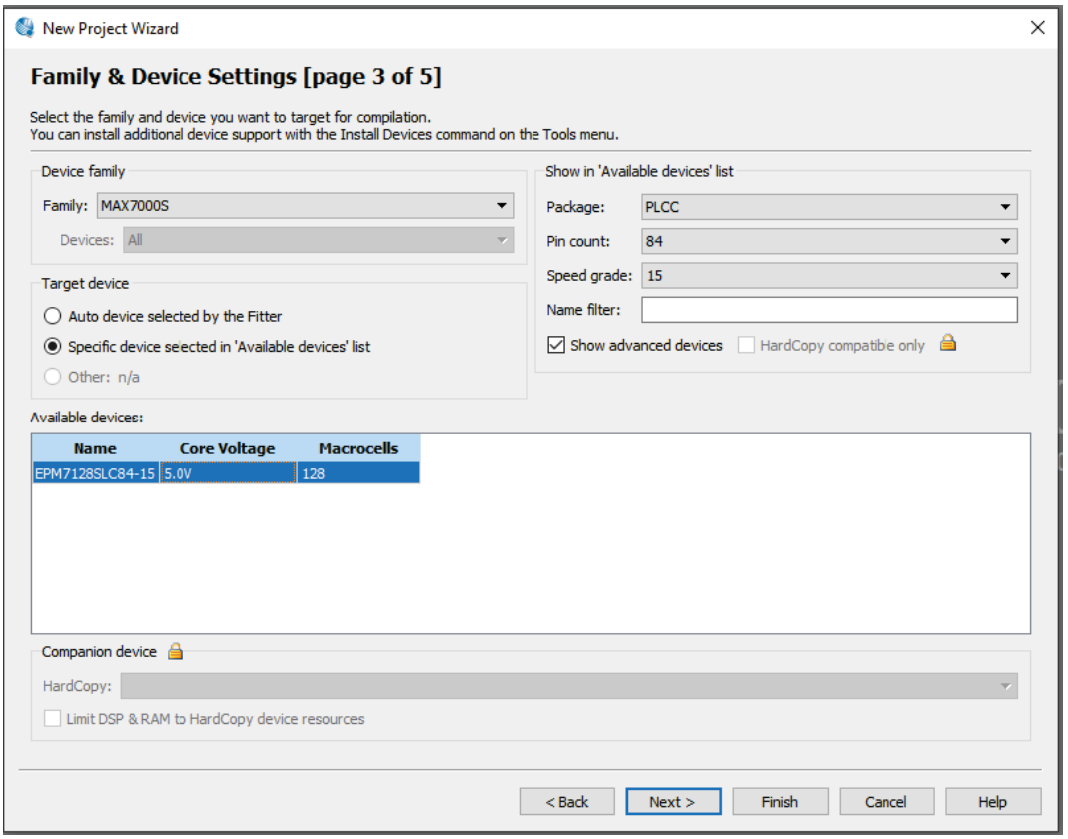


Фиг. 13. *Създаване на директория*

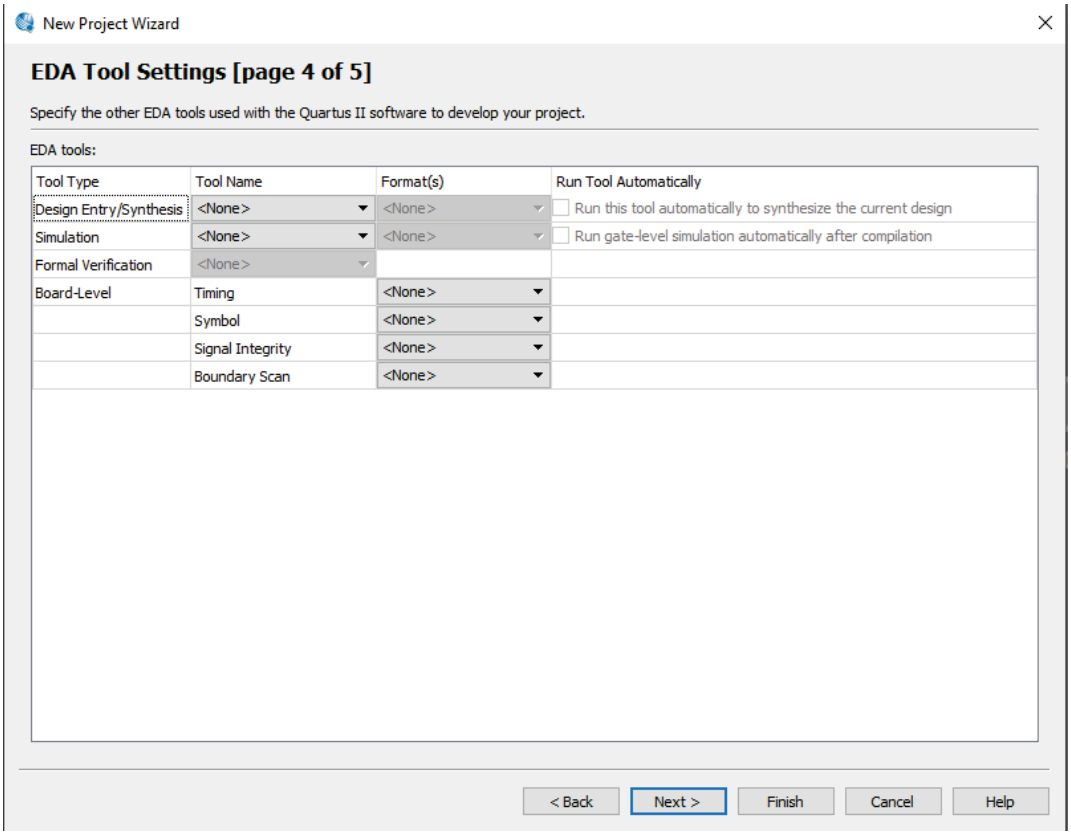


***Фиг. 14**. Възможност да се добавят специфични файлове на потребителя*

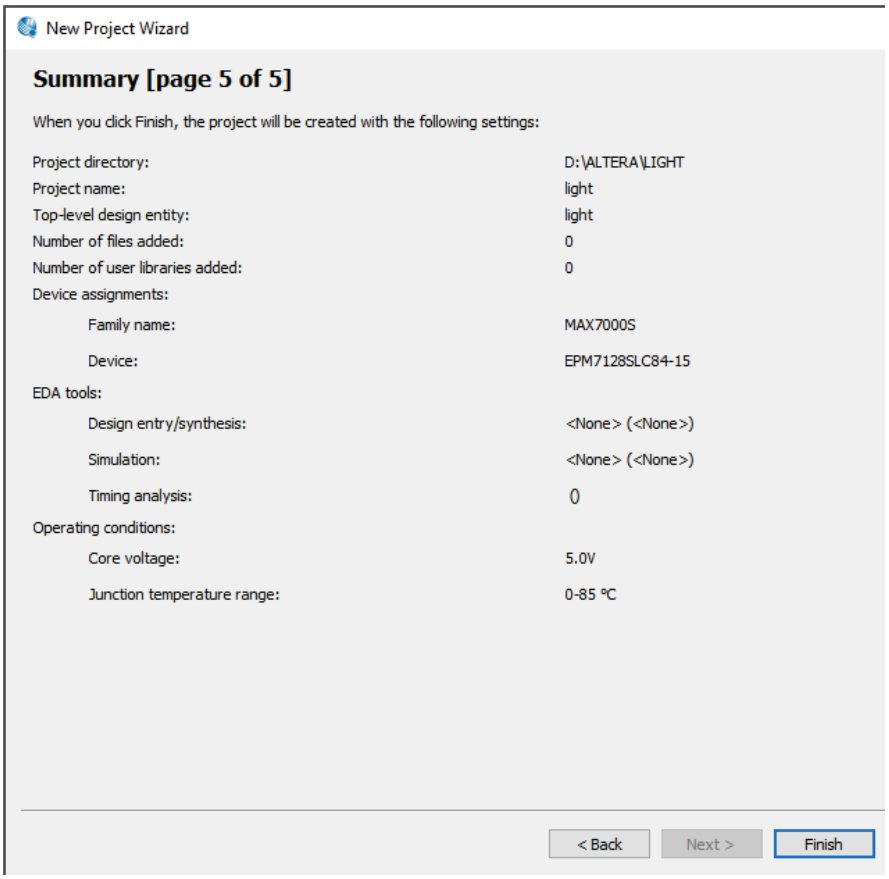
3. Ако разработчикът има опит със CPLD и преди това е създал някакви дизайнерски файлове, тогава прозорецът на фиг. 14 улеснява определянето и включването на такива файлове в проекта. Ако няма създадени по-рано проектни файлове, щракнете върху Напред.
4. В следващия диалогов прозорец (фиг. 15) трябва да определите вида на микросхемата, в която ще бъде реализиран разработеният проект. Комплектът за развойна система използва CPLD чипа EPM7128SPC84-15, за да го изберете в секцията "Фамилия устройства", изберете "MAX7000". За да ускорите търсенето, изберете параметрите на необходимия чип (Package - тип пакет, Pin_count - брой пинове, Speed grade - оценка на скоростта), както е показано на фиг. 15. Изберете чипа EPM7128SPC84-15. След като щракнете върху ”Напред”, ще се покаже прозорецът, показан на фиг. 16.
5. В прозореца "EDA Tool Settings" (EDA - Electronic Design Automation,) на фиг. 16, потребителят може да посочи всякакви помощни програми за симулация на трети страни (напр. ModelSim). В рамките на този урок няма да се използват инструменти за моделиране на трети страни, а ще се използва само вграденият инструмент за симулиране в Quartus II 13 WEB. Щраква се върху „Напред“.



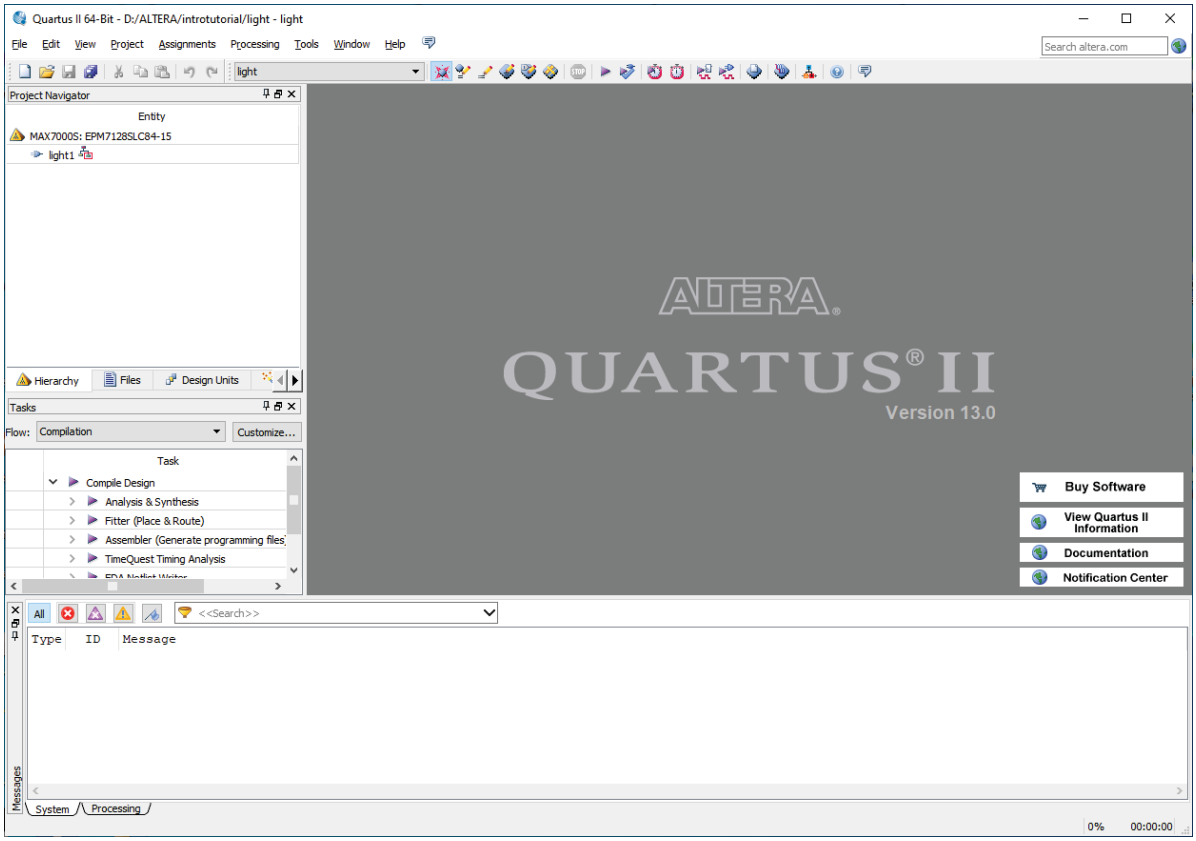
Фиг. 15. *Избор на чип*



Фиг. 16. *Други възможности за задаване*



Фиг.17. *Всички настройки на проекта*

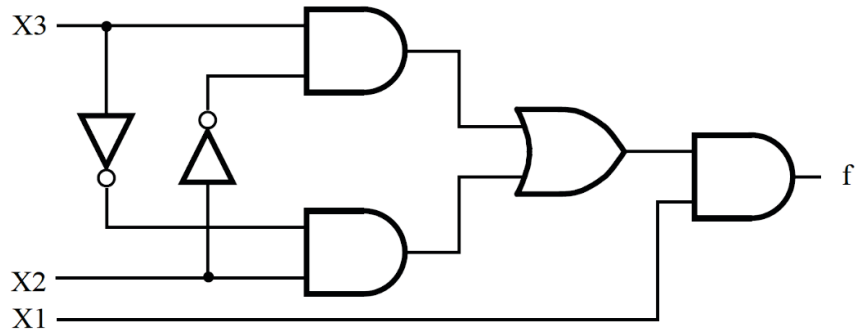


Фиг. 18. *Създаден проект*

6. Кратка информация за избраните настройки се показва в прозореца, показан на фигура 17. След щракване върху бутона Finish се връщате към главния прозорец на Quartus II, с основните настройки на новосъздадения проект (фигура 18).

3.2. Въвеждане на проект с помощта на графичен редактор

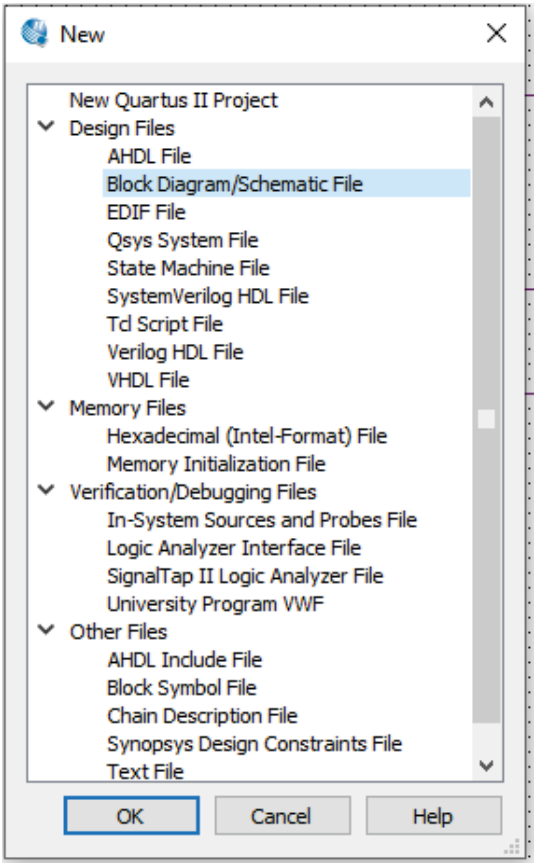
Като пример за проектиране ще използваме три-входова схема на контролера на светлина, показана на фигура 19. Схемата може да се използва за управление на една светлина от всеки от двата ключа, x2 и x3, където затворен ключ съответства на логическата стойност 1. Таблицата на истинността за схемата също е дадена на фигурата. Имайте предвид, че това е функцията Exclusive-OR на входовете x2 и x3, реализирана с логически елементи И-ИЛИ, плюс вход за разрешаване x1.



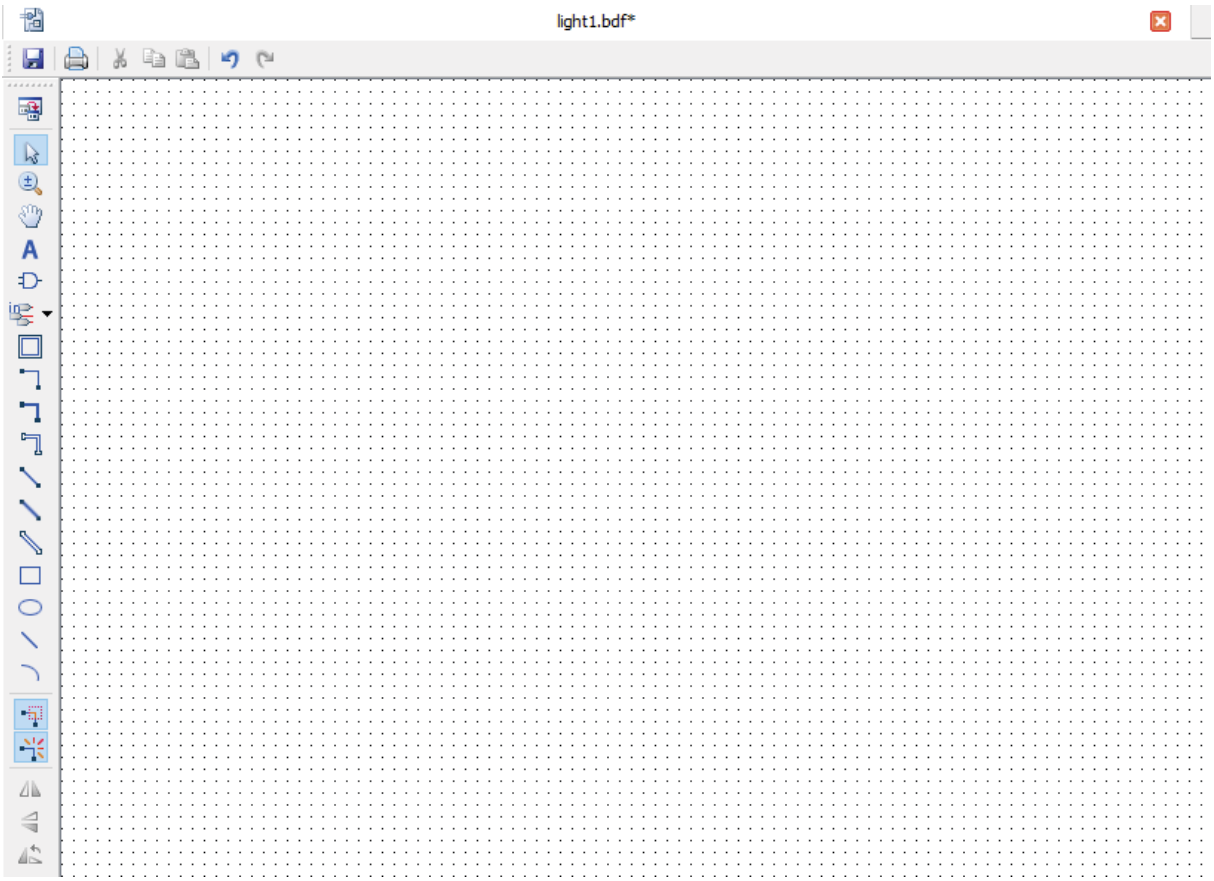
X1	X2	X3	*f*
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Фиг.19. *Примерна логическа схема*

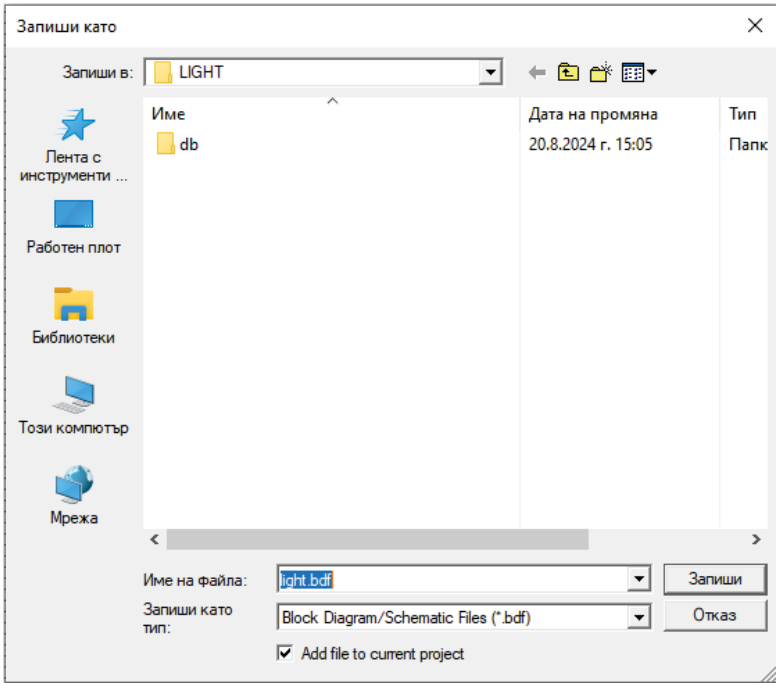
За да започнете да изпълнявате диаграмата, щракнете върху бутона "Ново" в главния прозорец на Quartus II, в диалоговия прозорец, който се появява (фиг. 20), изберете "Блокова диаграма / Схематичен файл", след което щракнете върху "OK". Ще се отвори прозорецът на графичния редактор (фиг. 21). Първо, трябва да посочите името на файла, за това изберете „Запиши като“ в менюто „Файл“, ще се появи прозорец (фиг. 22), където в реда „Име на файл“ въвеждаме Light, след това щракваме върху „Запазване“.



Фиг. 20. *Избор за въвеждане на ЛС*



Фиг. 21. *Графичен редактор*

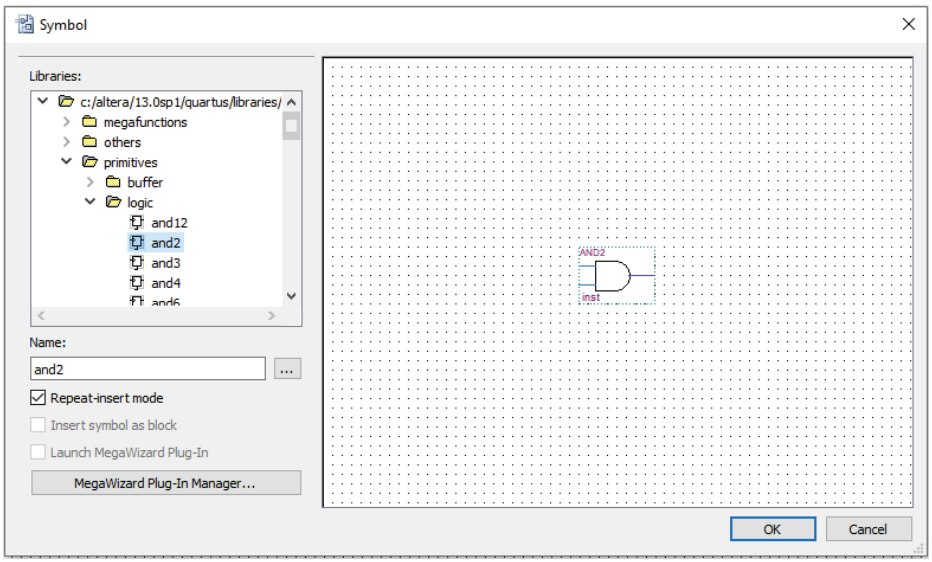


Фиг. 22. *Име на файла*

3.3 Импортиране на логически символи

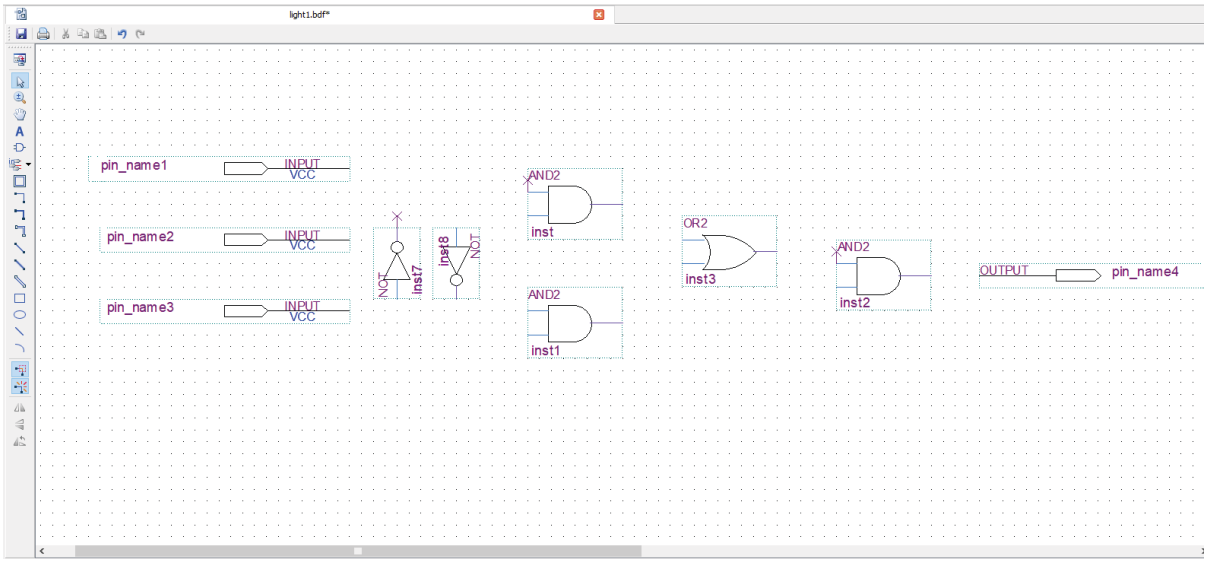
Графичният редактор предоставя набор от библиотеки с основни логически елементи, които могат да бъдат импортирани в схема. Щракнете двукратно върху празно място в прозореца на графичния редактор или щракнете върху бутона в лентата с инструменти, който изглежда като логическо И. Ще се появи изскачащ прозорец (фигура 23), изберете логическия елемент and2 от йерархията на примитивите/логическата библиотека и щракнете върху бутона OK.

Сега елементът ще се появи в прозореца на графичния редактор. Щракнете, за да поставите елемента навсякъде в прозореца на графичния редактор.



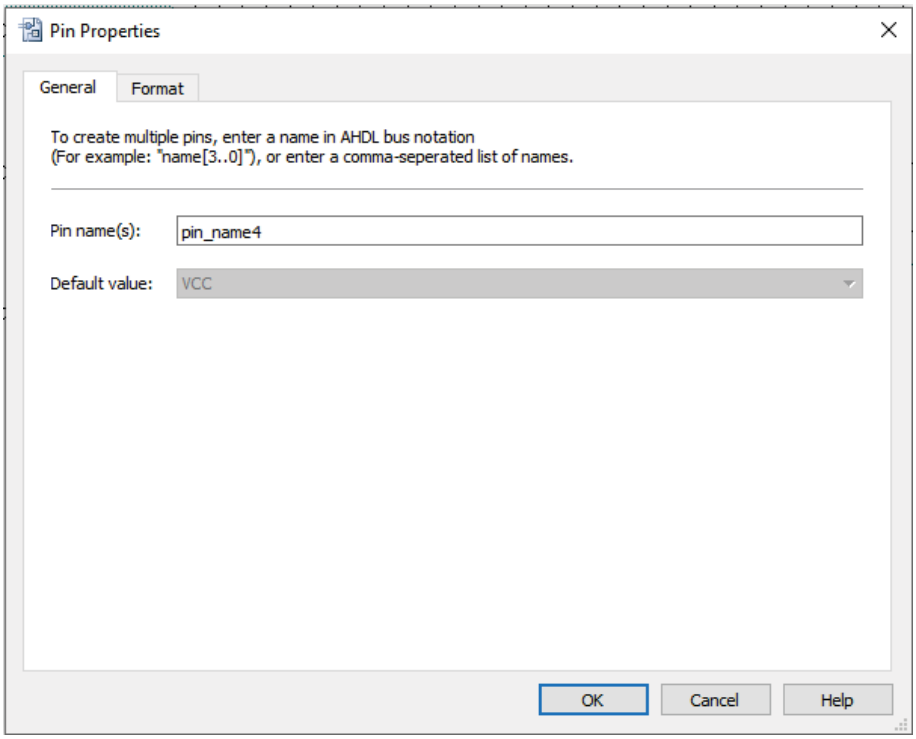
Фиг. 23. *Избор на логически символ*

По същия начин изберете елемента or2. След това от библиотеката primitives / pin намерете входно-изходните елементи „вход“ и „изход“ (фиг. 24). Ако е необходимо, елементите могат да бъдат завъртени и показани с помощта на съответните инструменти.



Фиг. 24. *Импортиране на I/O*

След двукратно щракване върху I/O елемента се появява прозорецът със свойства на извода (фиг. 25), където имената на I/O елементите се задават в реда Pin name(s). Въведете имената на входните/изходните елементи в съответствие с фиг. 19 (x1, x2, x3, f).

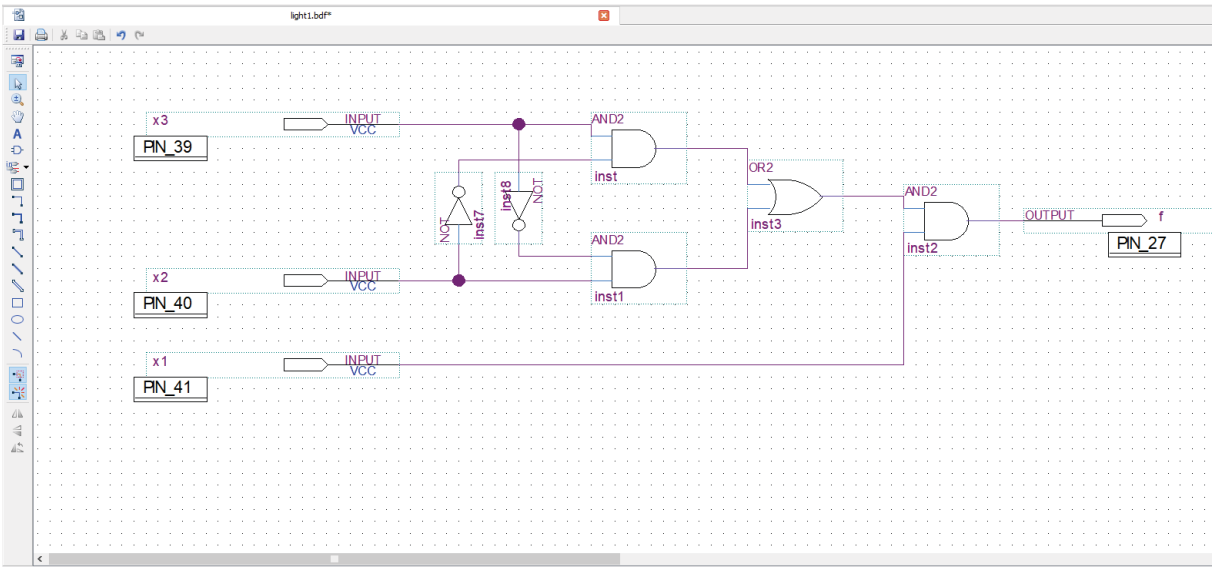


Фиг. 25. *Свойства на I/O порт*

3.4. Свързване на възли с проводници

Символите на диаграмата трябва да бъдат свързани чрез чертане на линии (проводници). Щракнете върху иконата в лентата с инструменти, за да активирате инструмента за ортогонален възел. Позиционирайте показалеца на мишката над десния край на входния извод x1. Щракнете върху и задръжте бутона на мишката и плъзнете мишката надясно, докато начертаната линия достигне извода на горния вход на елемента И. Освободете бутона на мишката, който оставя линията, свързваща двата извода. След това начертайте връзка от входния извод на инвертора към изчертания вече проводник, ще се появи точка, показваща връзка между двата проводника.

Използвайте същата процедура, за да нарисувате останалите проводници във схемата. Ако се направи грешка, може да се маркира проводник като щракнете върху нея и премахнете чрез натискане на клавиша Delete на клавиатурата. След като завършите диаграмата може да щракнете на иконата, за активирате на инструмента за избор. Сега промените във външния вид на диаграмата могат да бъдат направени чрез избиране на конкретен символ или проводник и или преместването му на друго място, или изтриването му. Крайната диаграма е показана на фигура 26.



Фиг. 26. *Въведената логическа схема.*

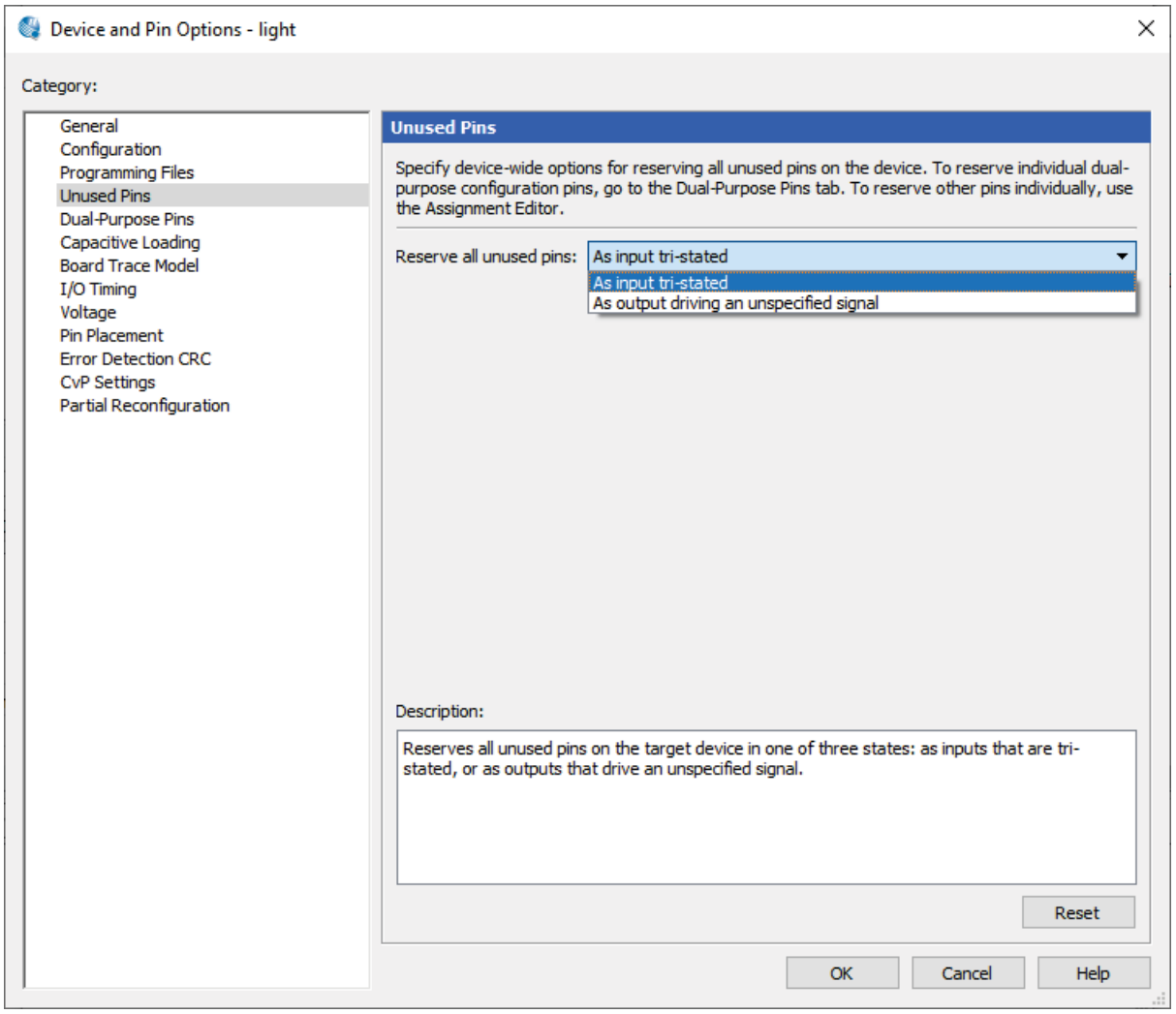
3.5. Компилиране на създадената схема

Въведеният файл със схематична диаграма, light.bdf, се обработва от няколко инструмента на Quartus II, които анализират файла, синтезират веригата и генерира файл за програмиране в целевия чип. Тези инструменти се контролират от приложна програма, наречена компилатор.

Стартирайте компилатора, като изберете Processing > Start Compilation или като щракнете върху иконата на лентата с инструменти, която прилича на лилав триъгълник. Тъй като компилацията преминава през различни

етапи, нейният напредък се отчита в прозорец от лявата страна на дисплея на Quartus II.

Преди това обаче, е необходимо да зададем какви ще бъдат неизползваните изводи. Поради факта, че вече има създадени връзки с други елементи в използваната платка, е удачно да се зададат като входове с високо импедансно състояние. Чрез двойно щракване върху избрания чип в “Project Navigator” се появява менюто за избор на чип. Натиска се бутона „Device an Pin option“ (фиг.27) и се избира съответната опция за неизползваните изводи.

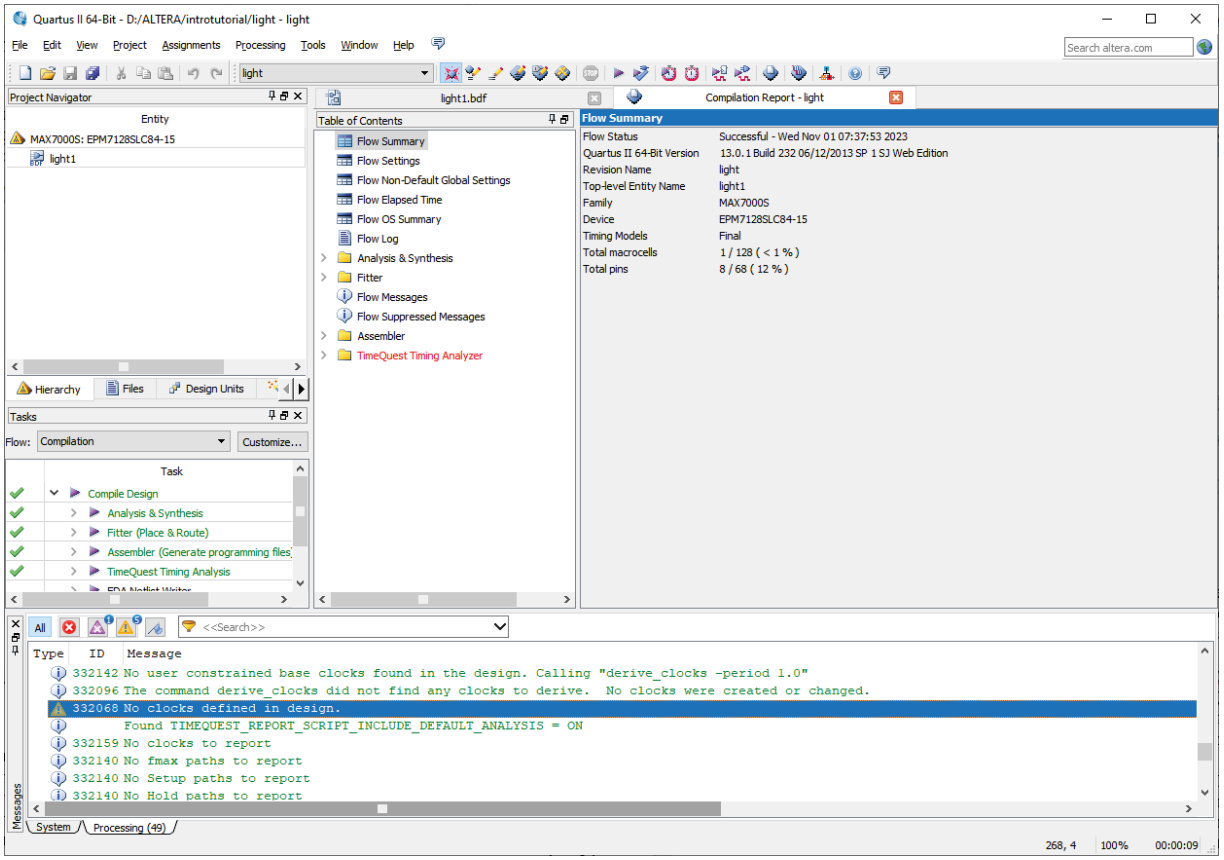


Фиг. 27. *Задаване на функция на неизползваните изводи*

Успешната (или неуспешна) компилация се показва в изскачащ прозорец. Потвърдете го, като щракнете върху OK, което води до дисплея на Quartus II на фигура 28. В прозореца за съобщения, в долната част на фигурата се показват различни съобщения. В случай на грешки ще бъдат дадени подходящи препоръки. Когато компилацията приключи, се изготвя отчет за компилацията. Отваря се прозорец, показващ този отчет автоматично, както

се вижда на фиг.28. Прозорецът може да бъде преоразмерен, разширен или затворен по нормалния начин и може да бъде отворен по всяко време, като изберете Processing > Compilation Report или като щракнете върху иконата приличаща на файл.

Отчетът включва редица секции, изброени в лявата част на прозореца му.



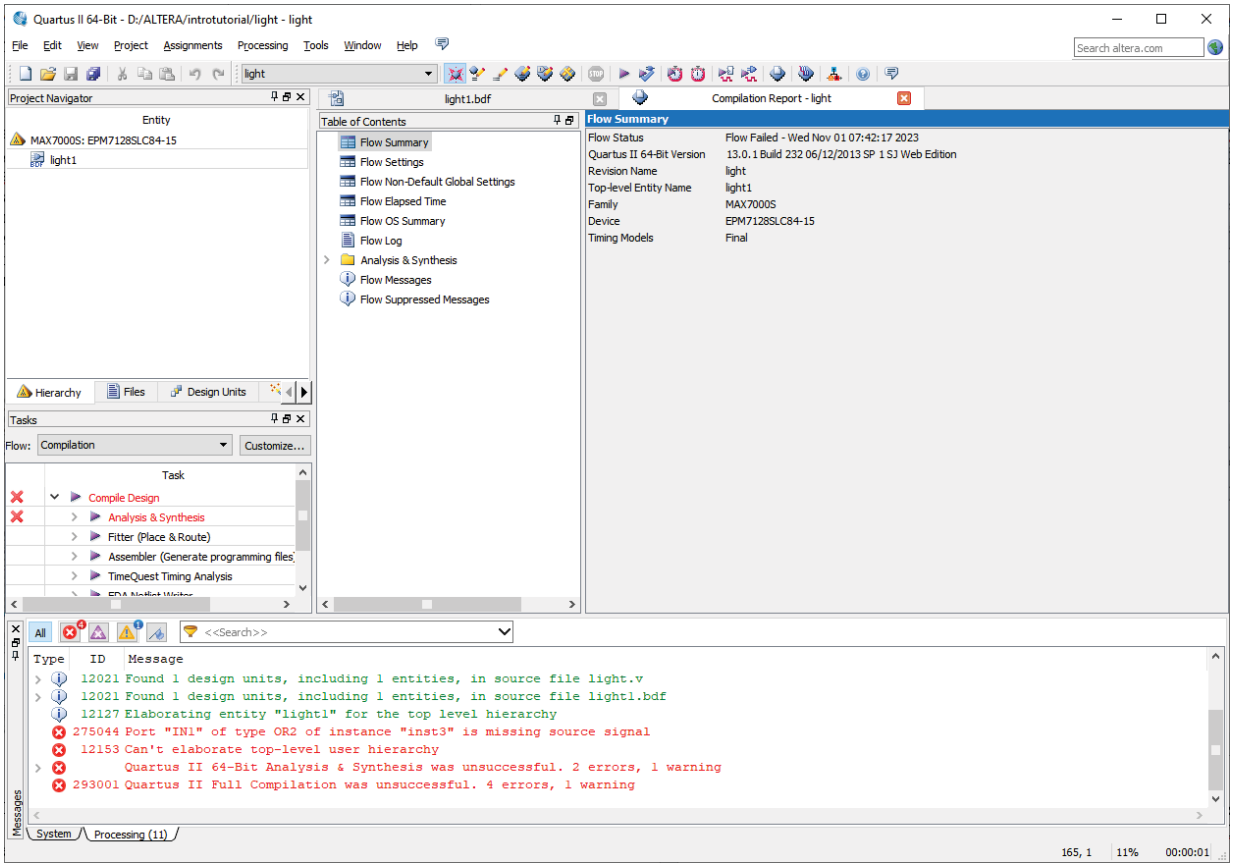
Фиг. 28. *Успешна компилация*

3.6 Грешки

Софтуерът Quartus II показва съобщенията, генерирани по време на компилацията, в прозореца Messages. Ако блоковата диаграма - дизайн файлът е правилен, едно от съобщенията ще гласи, че компилацията е била успешна и че няма грешки. Ако компилаторът не отчита нула грешки, значи има поне една грешка в записа на схемата. В този случай в прозореца Messages ще се покаже съобщение, съответстващо на всяка открита грешка. Двойно щракване при съобщение за грешка ще освети грешната част от веригата в прозореца на графичния редактор. По същия начин компилаторът може да покаже някои предупредителни съобщения. Техните подробности могат да бъдат проучени по същия начин, както в случая със съобщения за грешки. Потребителят може да получи повече информация за конкретна грешка или предупредително съобщение, като избере съобщението и натискане на функционалния клавиш F1.

За да видите ефекта от грешка, отворете файла light.bdf. Отстранете проводника, свързващ изхода на единия И елемент към елемента ИЛИ. За да направите това, щракнете с мишката върху проводника, който трябва да бъде премахнат (за да го изберете) и натиснете Delete. Компилирайте грешния дизайн отново. Изскачащ прозорец ще попита дали промените ще бъдат запазени във файла light.bdf -трябва да бъдат запазени; щракнете върху Да. След като се опитате да компилирате схемата, софтуерът Quartus II ще покаже изскачащ прозорец показващ, че компилацията не е била успешна. Потвърдете го, като щракнете върху OK.

Резюмето на отчета, дадено на фигура 29, потвърждава неуспешния резултат. Разширете частта Анализ и Синтез на отчет и след това изберете Messages, за да се показват съобщенията, както е показано на фигура 29. Щракнете двукратно върху първото съобщение за грешка, което гласи, че на един от възлите липсва източник. Софтуерът Quartus II реагира чрез показване схемата на light.bdf и подчертаване на елемента ИЛИ, която е засегната от грешката, както е показано на фигура 30. Коригирайте грешката и прекомпилирайте дизайна.

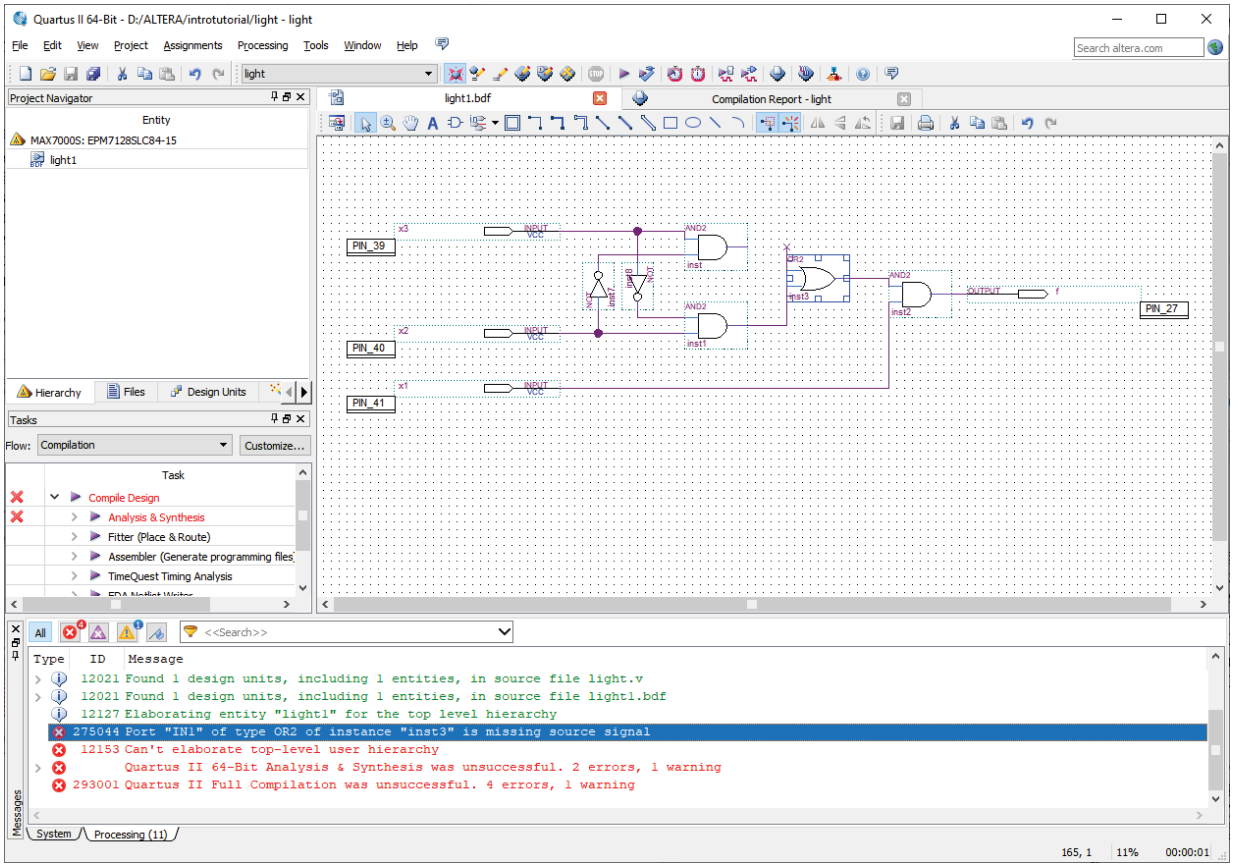


Фиг. 29. *Съобщения за грешки*

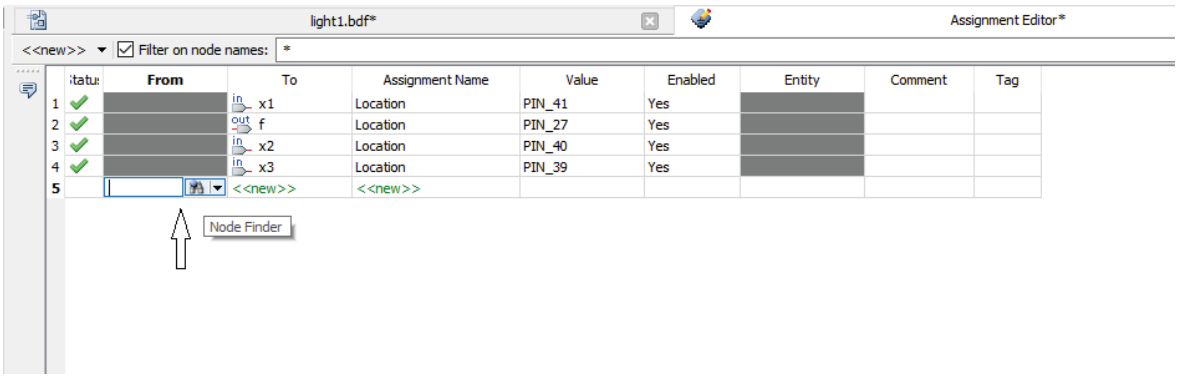
3.7 Присвояване на изводи

По време на компилацията по-горе компилаторът Quartus II беше свободен да избере всякакви изводи на избраната CPLD, които да се използват като входове и изходи. Въпреки това, платката CSE1 има кабелни връзки между CPLD изводите и другите компоненти на платката. Ще използваме три превключвателя, обозначени като SW-DIP4(1), SW-DIP4 (2) и SW-DIP4(3) за да предоставим външни входове към x1, x2 и x3, към нашата примерна схема. Тези превключватели са свързани съответно към изводите на CPLD 41, 40 и 39. Ще свържем изхода f към малкия светодиод с етикет L0, който е твърдо свързан към CPLD извода 27.

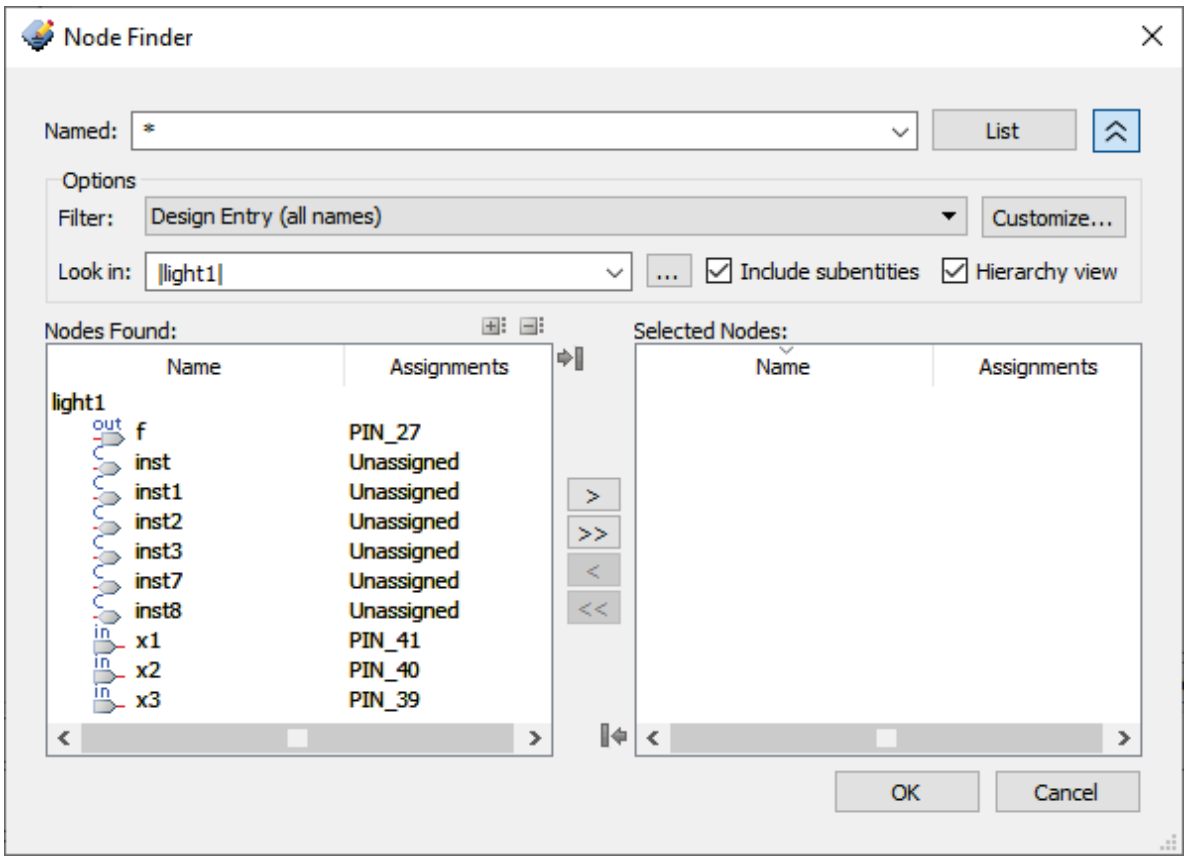
Присвояването на изводи се извършва с помощта на редактора на задания. Изберете Assignments > Assignment Editor за стигане до прозореца на фигура 31. Щракнете двукратно върху записа <<нов>>, който е маркиран в синьо в колоната с етикет “To”. Ще се появи падащото меню и избираме Node finder. Избираме наименуваните входове и изходи и ги добавяме в дясно – фиг.32 .



Фиг. 30. *Локация на грешката*

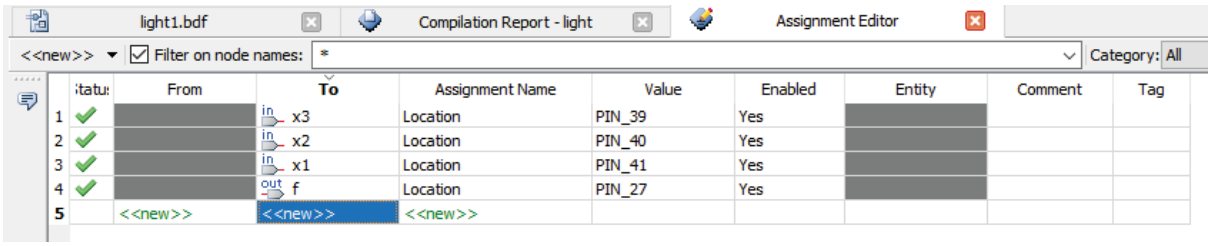


Фиг. 31. *Присвояване на щифтове*



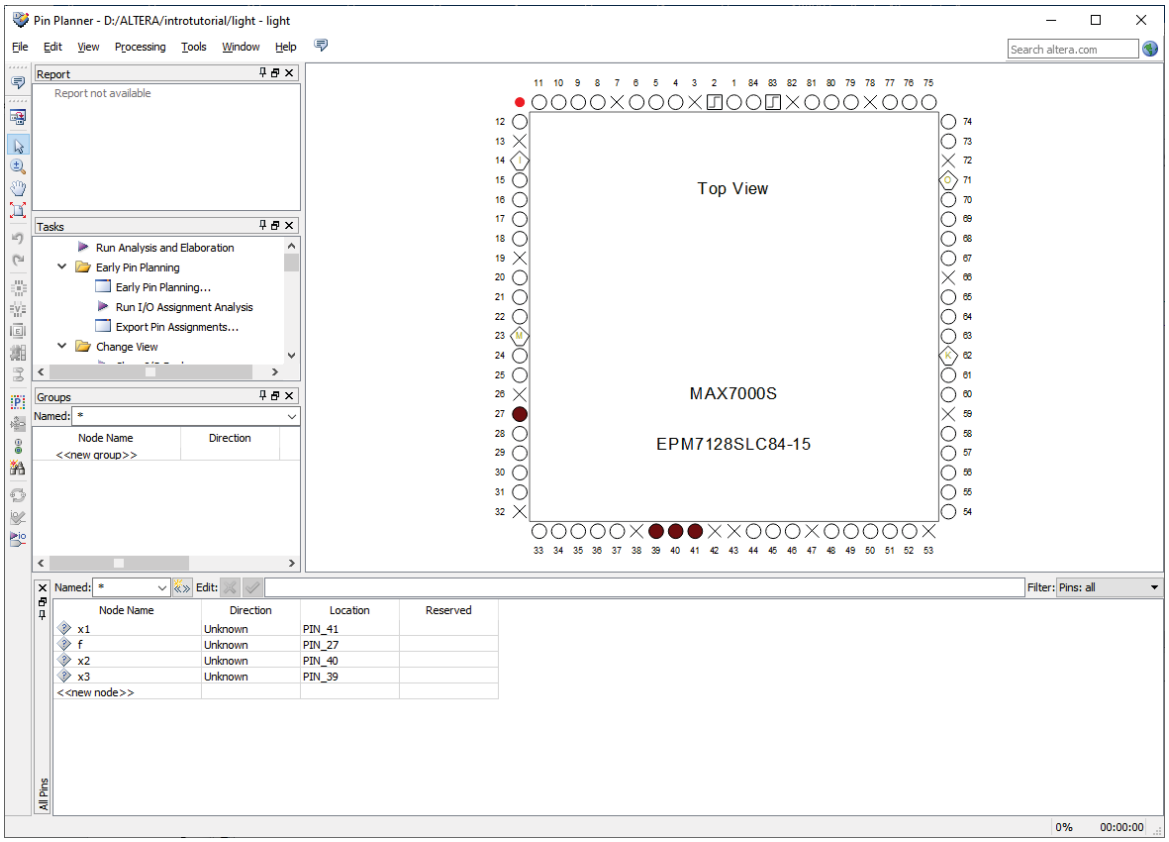
Фиг. 32

След това, като щракнете двукратно върху полето вдясно от този нов запис x1, в колоната с етикет “Assignment Name” избираме „Location“. В колоната „Value“ въвеждаме името на щифта PIN_41. Използвайте същата процедура, за да присвоите вход x2 към извод 40, x3 към извод 39 и изход f към извод 27 (фиг. 33). За да запазите направените задания, изберете File > Save. Можете също просто да затворите прозореца на редактора на задания, в който случай изскачащ прозорец ще попита дали искате да запазите промените. Прекомпилирайте схемата, така че да бъде компилирана с правилните назначения на щифтове.



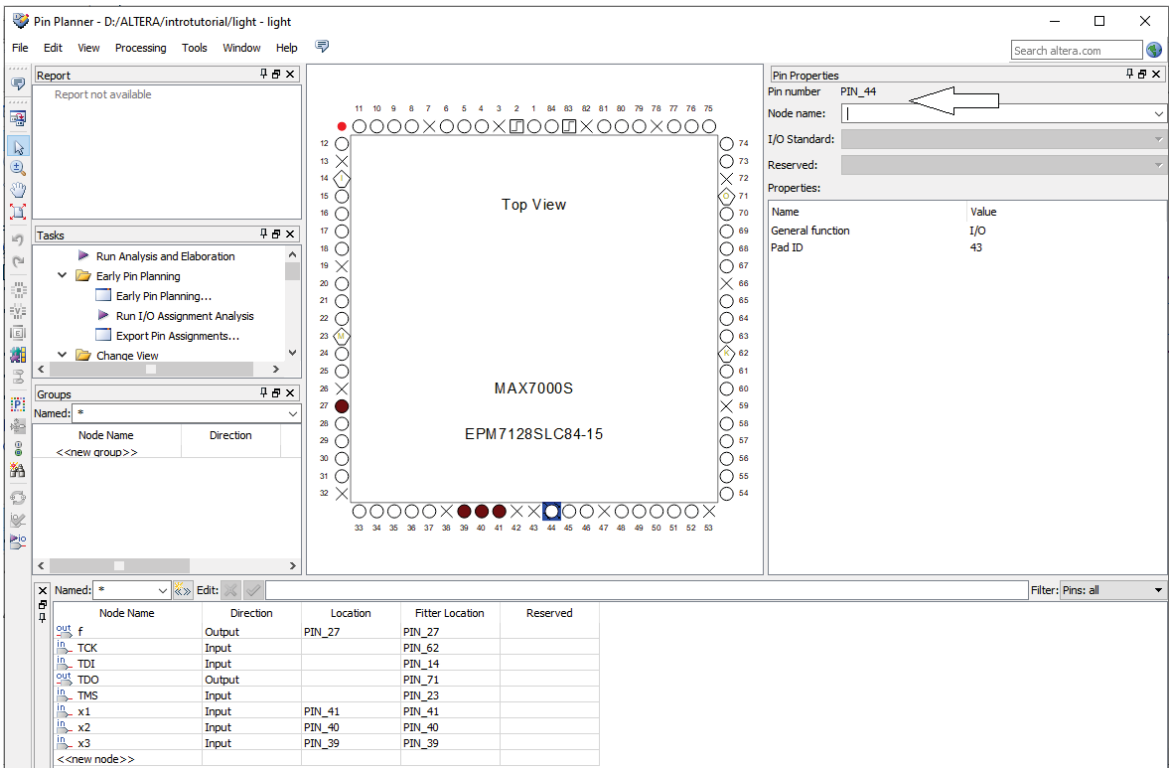
Фиг. 33

Друга възможност е използването на „Pin planner“. Визуално се показват изводите на използваната интегрална схема CPLD –фиг. 34.



Фиг. 34. *Изглед на CPLD от Pin planner*

Щракваме два пъти върху избрания номер извод и присвояваме логическо име написано от нас в схемата (x1, x2, x3 и f) – фиг.35. Пишем името в “Node name” потвърдено с “Enter”.

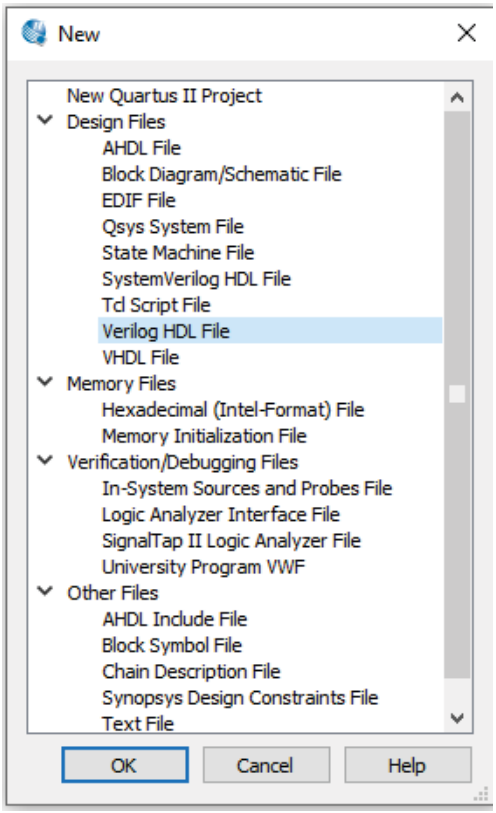


Фиг. 35 Присвояване на щифт към лаически вход или изход

Платката CSE1 има фиксирани назначения на изводи. След като завърши един дизайн, потребителят ще иска да използва същото задаване на изводи за последващи проекти. Преминаването през описаната по-горе процедура става досадно, ако има много изводи, използвани в дизайна. Полезна функция на Quartus II позволява на потребителя да експортира и импортира заданията на присвояване на изводи в специален файлов формат. Файловият формат, който може да се използва за тази цел, е форматът на стойностите, разделени със запетая (CSV), който е често срещан текстов файлов формат. Този файлов формат често се използва във връзка с Microsoft Excel програмата за електронни таблици, но файлът може да бъде създаден и на ръка, като се използва всеки обикновен ASCII текстов редактор.

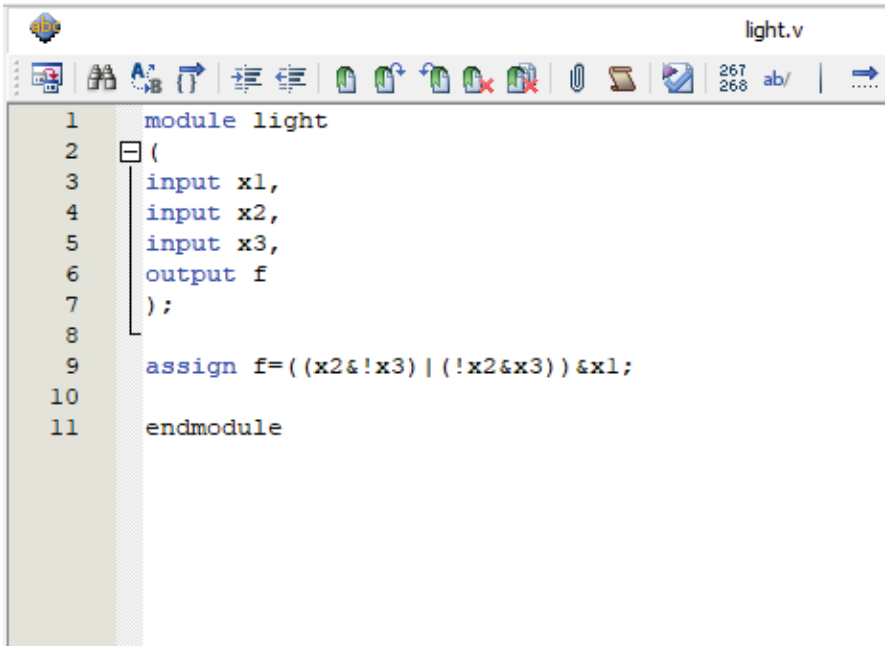
3.8 Програмно въвеждане на проекта

За да въведете проект програмно на базата на езика Verilog HDL, в менюто на Quartus II, щракнете върху бутона “Нов”, изберете “Verilog HDL File” (фиг. 36). В текстовия редактор, който се показва, въведете Verilg кода, както е показано на фиг. 37. Verilog HDL кодът започва с ключовата дума "module", последвана от името му, в нашия пример "light", завършва файла с ключовата дума "endmodule". След декларацията на модула в скоби, разделени със запетаи, има последователна декларация на входни/изходни портове; препинателните знаци не се поставят преди затварящата скоба.



Фиг. 36. *Програмно въвеждане с Verilog HDL*

Скобите се затварят с точка и запетая. По-нататък в конструкцията на непрекъснато "присвояване" е описан кодът на функцията на логическата схема на един изход. След като приключите с писането на кода, модулът трябва да бъде запазен - имайте предвид, че името на записания файл трябва да съвпада с името на декларирания модул.

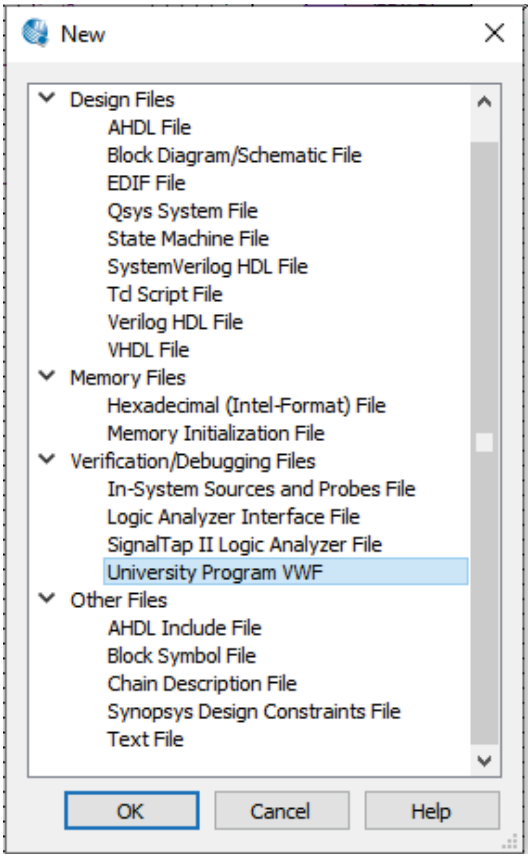


Фиг. 37. *Пример за Verilog описващ логическата схема.*

3.9 Моделиране на разработената схема

Преди да продължите с конфигурацията на CPLD чипа на платката CSE1, препоръчително е да извършите симулация на разработената схема, за да проверите правилната работа. Quartus II CAD включва инструмент за симулация, който може да се използва за симулиране на поведението на проектираната схема. Преди моделирането на веригата е необходимо да се създадат необходимите времеви диаграми, наречени тестови вектори. За да направите това, трябва да посочите входните / изходните елементи, както и възможните вътрешни точки на веригата, които разработчикът иска да анализира. Инструментът за моделиране или симулаторът използва тестовите вектори във внедрения модел на схемата и изчислява очакваните отговори. За въвеждане на тестови вектори в Quartus II 13WEB се използва редакторът Waveform:

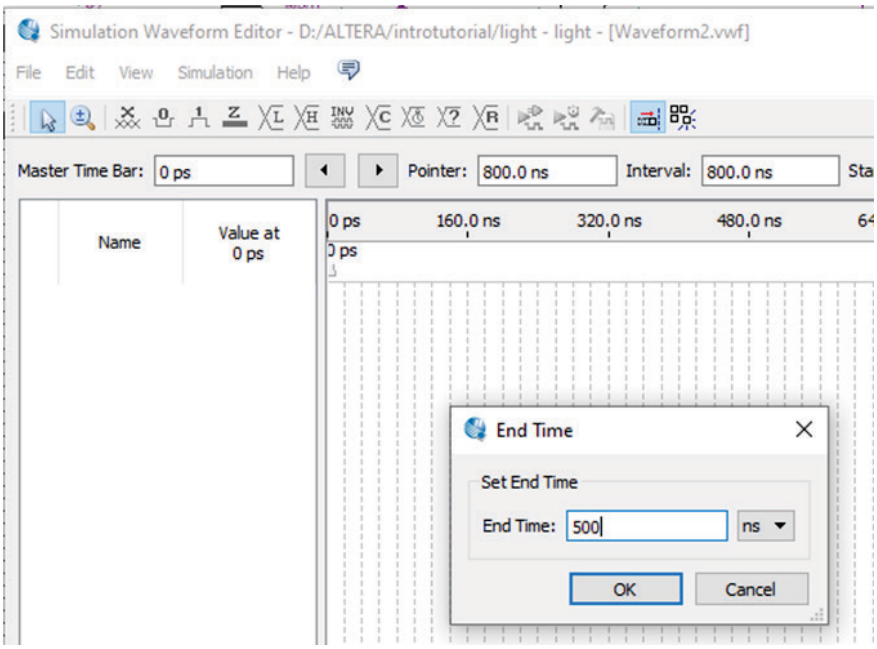
- От менюто File/New изберете "University Program VWF" (Фигура 38).



***Фиг. 38**. Избор на файл за симулация*

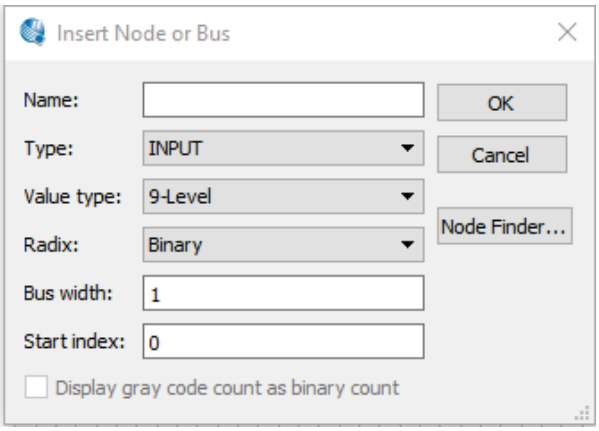
- Прозорецът Waveform Editor е изобразен на фигура 39. Запазете файла под името light.vwf; имайте предвид, че това променя името в показания прозорец. Задайте желаната симулация да работи от 0 до 500 ns, като изберете Edit > Set End Time и въвеждане на 500 ns в диалоговия прозорец, който се появява. Избиране на View > Fit in Window показва целия диапазон на

симулация от 0 до 500 ns в прозореца, както е показано на фигура 39. Прозореца може да се преоразмерява.

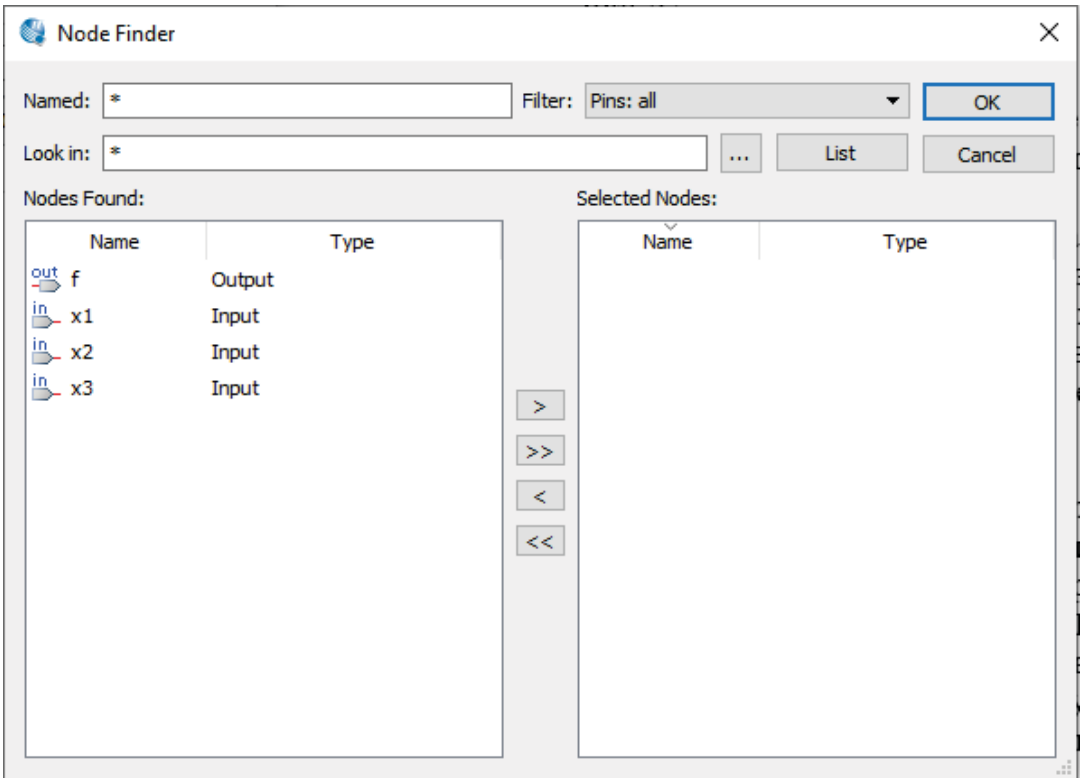


Фиг.39. *Екран за симулация*

След това искаме да включим входните и изходните възли на схемата, която ще бъде симулирана. Щракнете върху Edit > Insert > вмъкнете възел или шина, за да отворите прозореца на фигура 40. Възможно е да въведете името на сигнал (пин) в полето Name, но е по-лесно да щракнете върху бутона с надпис Node Finder, за да отворите прозореца на фигура 41. Помощната програма Node Finder има филтър, използван за да посочи какъв тип възли трябва да бъдат намерени. Тъй като се интересуваме от входни и изходни щифтове, задайте филтъра на Pins: all. Щракнете върху бутона List, за да намерите входа и изходните възли, както е показано от лявата страна на фигурата.



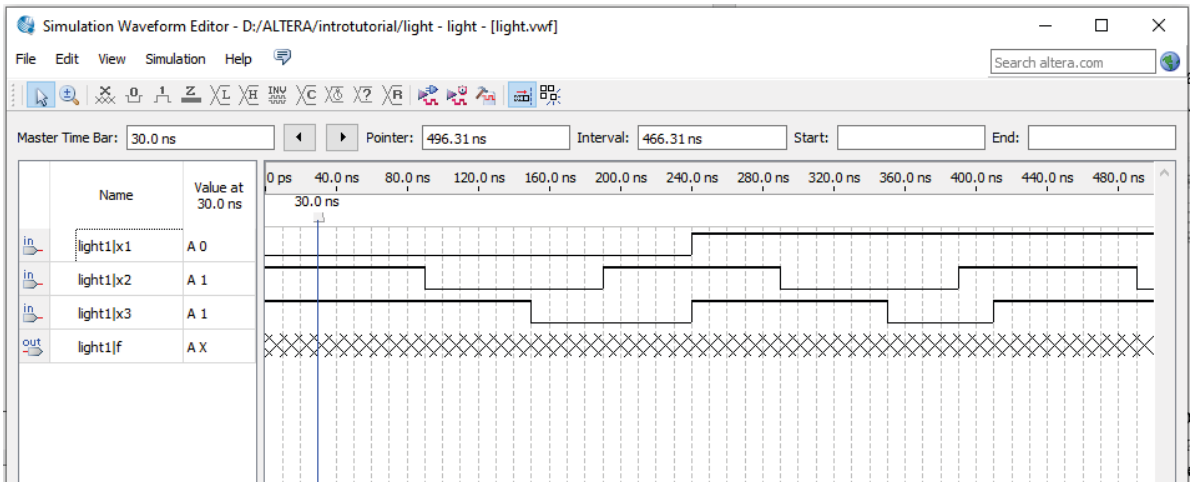
Фиг. 40. *Добавяне на вход за симулация*



Фиг. 41. *Добавяне на изследваните входове и изходи*

Ако не сте избрали възлите в ред удобен за вас, възможно е да ги пренаредите. За да преместите стимулационна форма нагоре или надолу в прозореца Waveform Editor, маркирайте я. Щракнете отново върху формата на сигнала и я плъзнете нагоре или надолу в редактора на сигнална форма.

- Посочваме логическите стойности, които да се използват за входните сигнали x1, x2 и x3 по време на симулацията. Логическата стойностите на изхода f ще бъдат генерирани автоматично от симулатора. За да е по-лесно, редакторът на вълнови форми позволява да нарисувате желаните вълнови форми като използвате бутона с пресечени вълнови линии – фиг. 42-1.



Фиг. 42-1.

За да се симулира поведението на схемата, е необходимо да се приложи достатъчен брой входни сигнали, като се има в предвид и закъснението от превключване на логическите елементи. В голяма схема броят на възможните входни стойности може да бъде огромен, така че на практика ние избираме относително малка (но представителна) извадка от тези входни сигнали. Въпреки това, за нашата малка схема можем да симулираме всичките входни стойности, дадени на фигура 19. Ще използваме по-големи от 100-ns времеви интервали за прилагане на тестовите вектора.

3.9.1 Извършване на симулацията

Проектирана схема може да бъде симулирана по два начина. Най-простият начин е да се приеме, че логически елементи и взаимовръзката с проводниците в CPLD са перфектни, като по този начин не причиняват забавяне при разпространението на сигнали през схемата. Това се нарича функционална симулация. По-сложна алтернатива е да се вземат предвид всички закъснения на разпространение, което води до времева симулация. Обикновено функционалната симулация се използва за проверка на функционалната коректност на схемата. Това отнема много по-малко време, тъй като симулацията може да се извърши просто с помощта на логически изрази, които дефинират схемата.

3.9.2. Инсталиране на ModelSim-Altera

ModelSim-Altera е симулационен инструмент, който е необходим за извършване на симулацията и има по-ниска производителност в сравнение с този на Mentor Graphics. Той е модифициран от Altera и се предлага в две издания:

- ModelSim-Altera Starter Edition (MSSE)
- ModelSim-Altera Edition (MSAE)

Основните разлики между ModelSim-Altera Starter Edition и ModelSim-Altera Edition са:

1. MSSE не можете да компилирате повече от 10 000 реда код - безплатен.

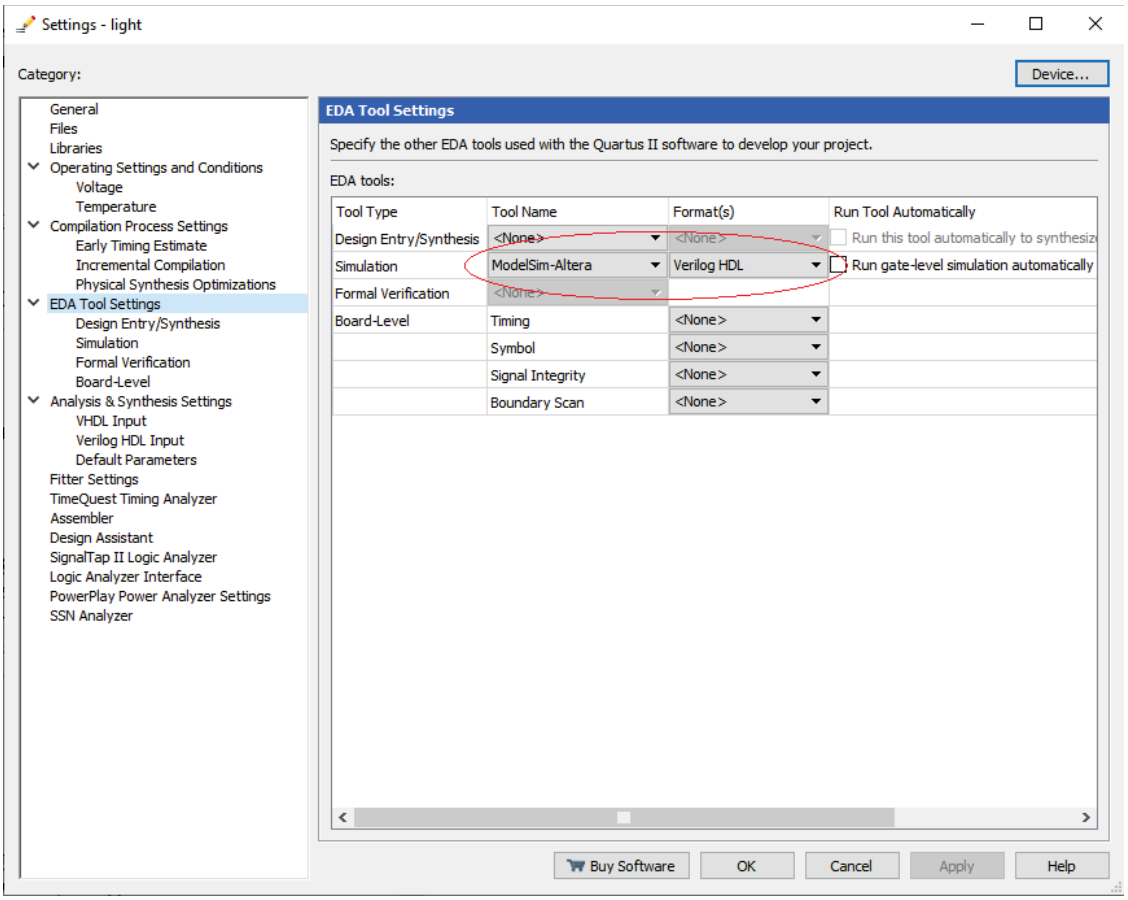
2. MSAE има по-добра производителност при симулация - платен.

- Инсталация

Изтегляме от сайта на Intel, ModelSim-Altera версия 13.01.232. След като го инсталираме, има две важни настройки, които трябва да направим, преди да започне да използваме ModelSim-Altera Starter Edition.

- Избор на правилния ModelSim инструмент.

Избираме от Quartus софтуера: Assignments > Settings... > EDA Tool Settings. В таба отдясно има таблица – фиг. 42-2.



Фиг. 42-2.

В таблицата с EDA инструменти избераме на кръстопътя на реда Tool Type Simulation и колоната Tool Name > ModelSim-Altera от падащото меню. След това изберете от колоната Format(s) > VHDL, Verilog или SystemVerilog, в зависимост от езика за описание на хардуер. Завършете с натискане на OK.

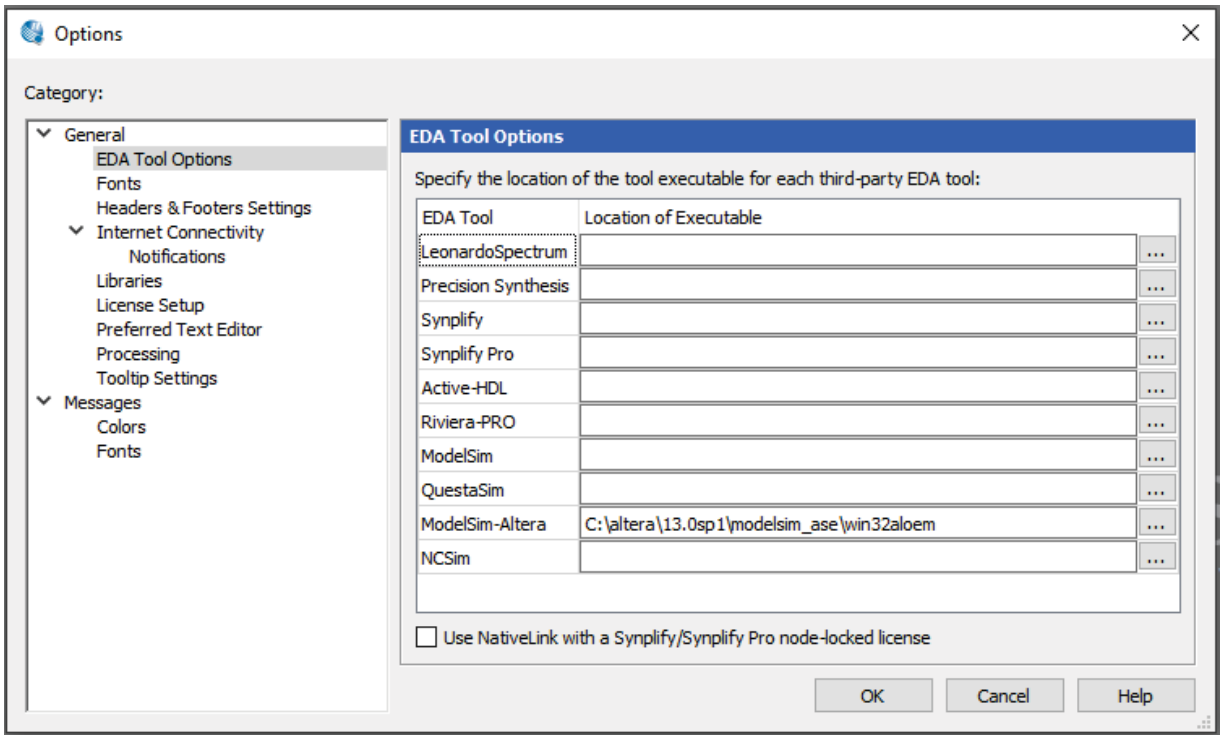
- Настройка на пътя

За да избегнете тази грешка:

"Cannot launch the ModelSim-Altera software because you did not specify the path to the executables of the ModelSim-Altera software", следвайте тези стъпки:

Избираме от Quartus софтуера: Tools > Options... > General > EDA Tool Options. Отдясно има таб с местоположението на изпълнимите файлове за няколко софтуера. Внимание, има два различни симулационни инструмента: ModelSim и ModelSim-Altera. Избираме втория и намираме изпълнимия файл modelsim.exe там, където е инсталиран – фиг.42-3. Например, в Windows може да бъде:

- C:\altera\quartus\13.0sp1\modelsim_ase\win32aloem



Фиг. 42-3.

При Linux е същото, освен крайното име на директорията:

- /soft/fpga/altera/quartus/13.0sp1/modelsim_ase/linuxaloem

Натискаме OK и ModelSim-Altera Starter Edition симулационен инструмент е готов за използване. В папката „modelsim_ase“ се намира файл „modelsim.ini“ в него се добавят редовете:

Under VHDL

max7000s = $MODEL_TECH/../altera/vhdl/max

Under Verilog

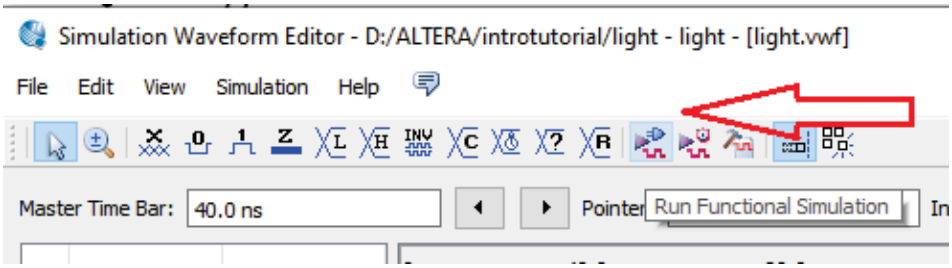
max7000s_ver = $MODEL_TECH/../altera/verilog/max

Това се налага защото серията MAX7000S е по стара и е нужен път до библиотеките за функционална симулация.

3.9.3 Функционална симулация

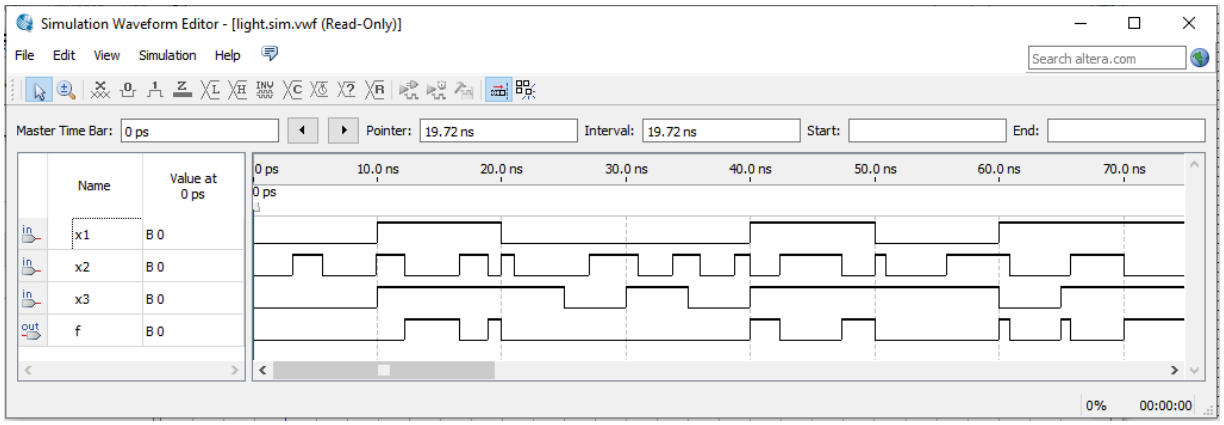
Преди да стартираме функционална симулация, е необходимо да изпълним Analysis and Synthesis на нашия проект, като избираме иконата в главния прозорец на Quartus II. Да се има предвид, че Analysis and Synthesis се изпълнява като част от основния процес на компилация. Ако вече сме компилирали дизайна си, тогава не е необходимо да изпълнявате Analysis and Synthesis отново.

За да извършите функционалната симулация, изберете Simulation > Run Functional Simulation или изберете иконата - фиг.43. Ще се появи изскачащ прозорец, който ще покаже прогреса на симулацията и автоматично ще се затвори, когато тя приключи. В края на симулацията ще се отвори втори прозорец на Waveform Editor, който ще покаже резултатите от симулацията, както е илюстрирано на Фигура 44.



Фиг. 43

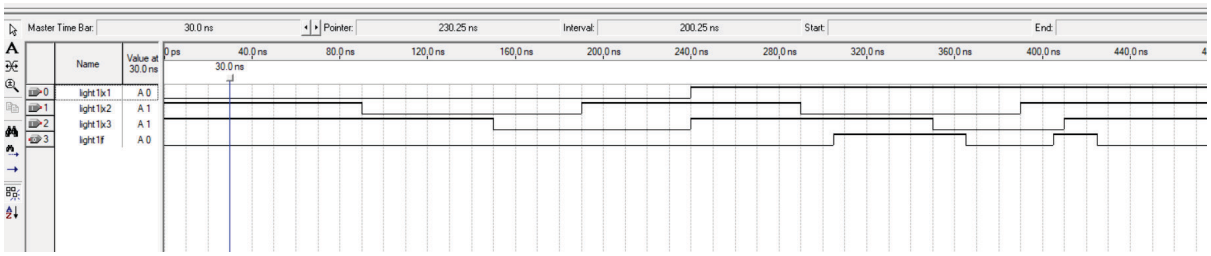
Ако прозорецът на вашия отчет не покажете целия времеви диапазон на симулацията, щракнете върху прозореца на отчета, за да го изберете и изберете View > Fit in Window.



Фиг. 44. *Функционална симулация*

3.9.4 Времева симулация

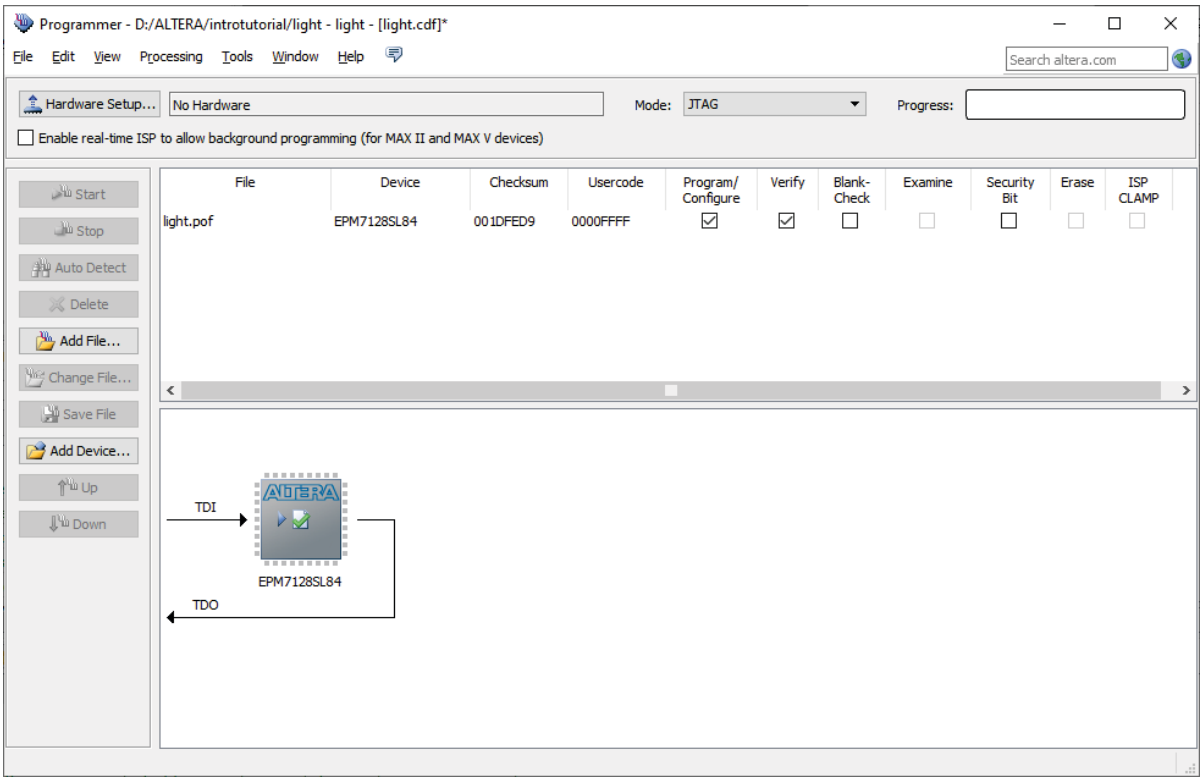
След като се уверим, че проектираната схема е функционално правилна, сега трябва да извършим симулация на времето, за да видите как ще се държи, когато действително бъде внедрен в избраното програмируемо устройство. Изберете Assignments >Settings > Simulator Settings, за да стигнете до прозореца на фигура 43, изберете Timing за режим на симулация и щракнете върху OK. Стартирайте симулатора, който трябва да генерира времедиаграмите на фигура 45. Обърнете внимание, че има забавяне от около 15 ns при произвеждане на промяна в сигнала f от момента, в който входните сигнали, x2 и x3, променят своите стойности. Това забавяне се дължи на закъсненията на разпространение в логическия елемент и проводниците в CPLD устройството.



Фиг. 45. *Времева симулация*

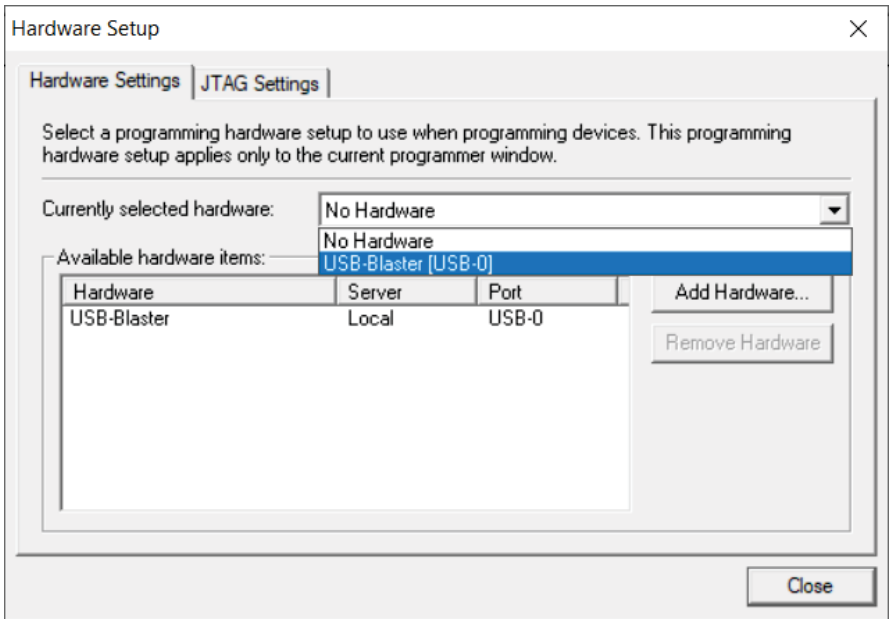
3.10 JTAG програмиране

Задачата за програмиране и конфигуриране се изпълнява по следния начин. Натиснете ключа RUN в положение RUN позиция. Изберете Инструменти > Програматор, за да стигнете до прозореца на фигура 46. Тук е необходимо да посочите хардуер за програмиране и режима, който трябва да се използва.



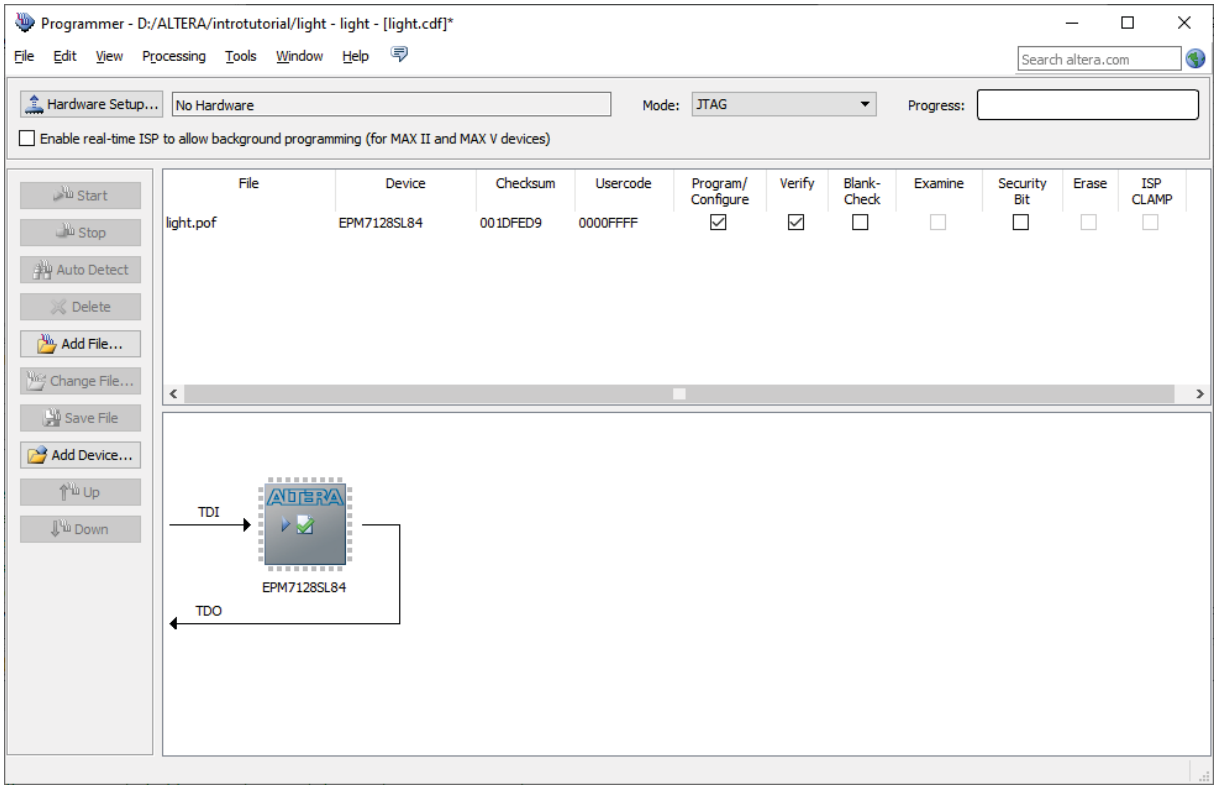
Фиг. 46. *Програматор*

Ако все още не е избран по подразбиране, изберете JTAG. Освен това, ако USB-Blaster не е избран по подразбиране, натиснете бутона Hardware Setup... и изберете USB-Blaster в изскачащия прозорец, както е показано на фигура 47.



Фиг. 47. *Избиране на USB-Blaster*

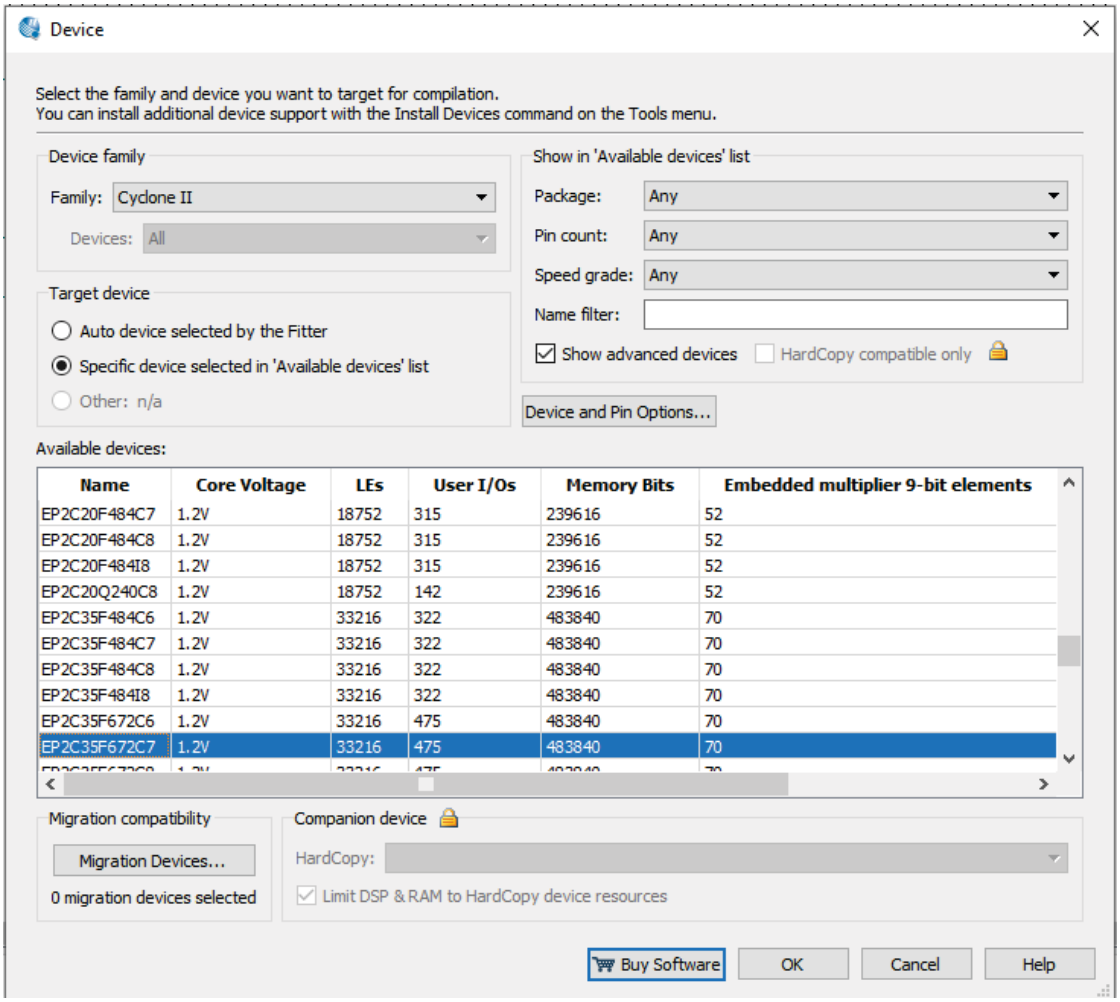
Сега натиснете Старт в прозореца на фигура 48. Светодиод на платката ще светне, когато данните за конфигурацията са изтеглени успешно. Ако видите грешка, докладвана от софтуера Quartus II, показваща това програмиране неуспешно, проверете дали платката е правилно включена.



Фиг. 48. *Правилно конфигуриран програматор*

3.11 Активно програмиране в сериен режим

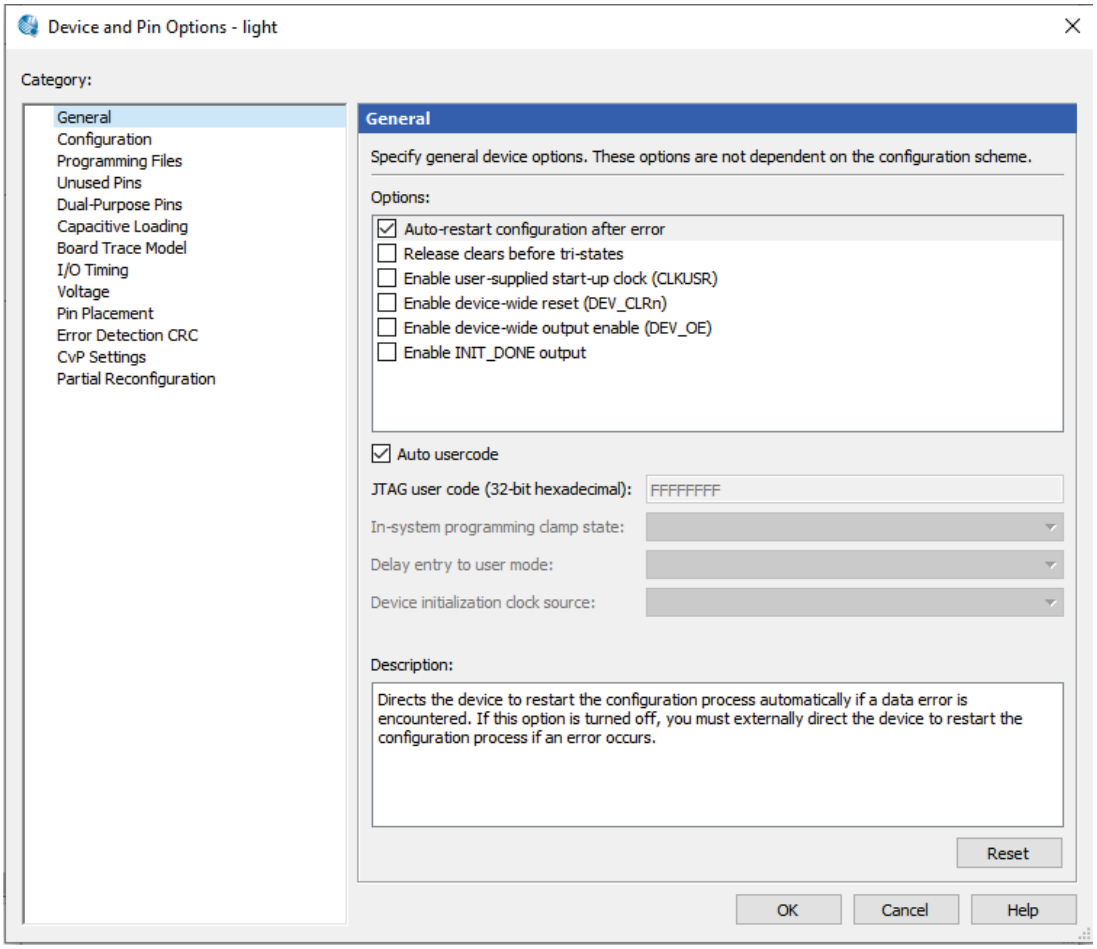
Такъв тип програмиране се налага при конфигуриране на памет за FPGA, примерно Cyclone II. Развойната система може да няма допълнителна серийна памет, тя не се и изисква. В случай, че програмираме FPGA, конфигурационните данни трябва да бъдат заредени в конфигурационното устройство на платката, което е идентифициран с примерно името EPCS4 или EPCS16. Това е енергонезависима памет. За да посочите необходимото устройство за конфигурация, изберете Assignments > Device, което води до прозореца на фигура 49.



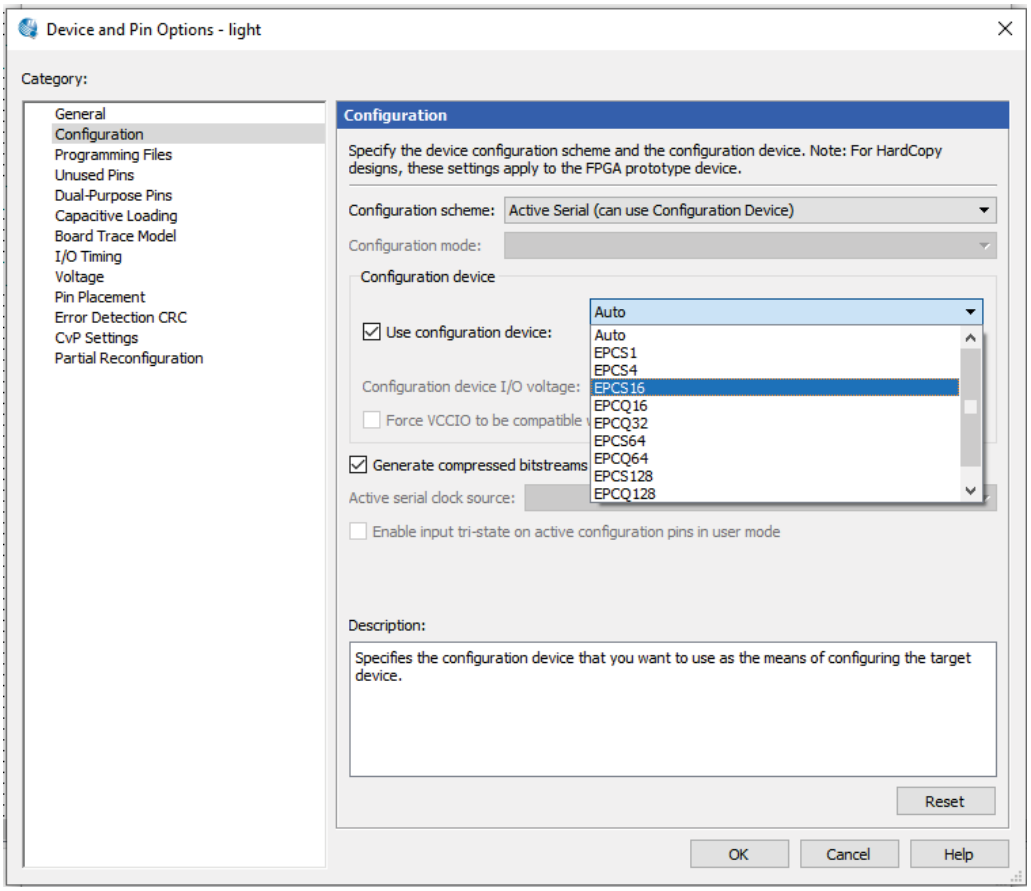
Фиг. 49. *Настройки на FPGA*

Щракнете върху бутона Device and Pin Options, за да стигнете до прозореца, показан на фигура 50. Сега щракнете върху раздела Configuration, за да получите прозореца на фигура 51. В устройството за конфигуриране, поле (което може да бъде настроено на Auto) изберете EPCS16 и щракнете върху OK. Прекомпилирайте проектираната схема. По-нататъшните стъпки

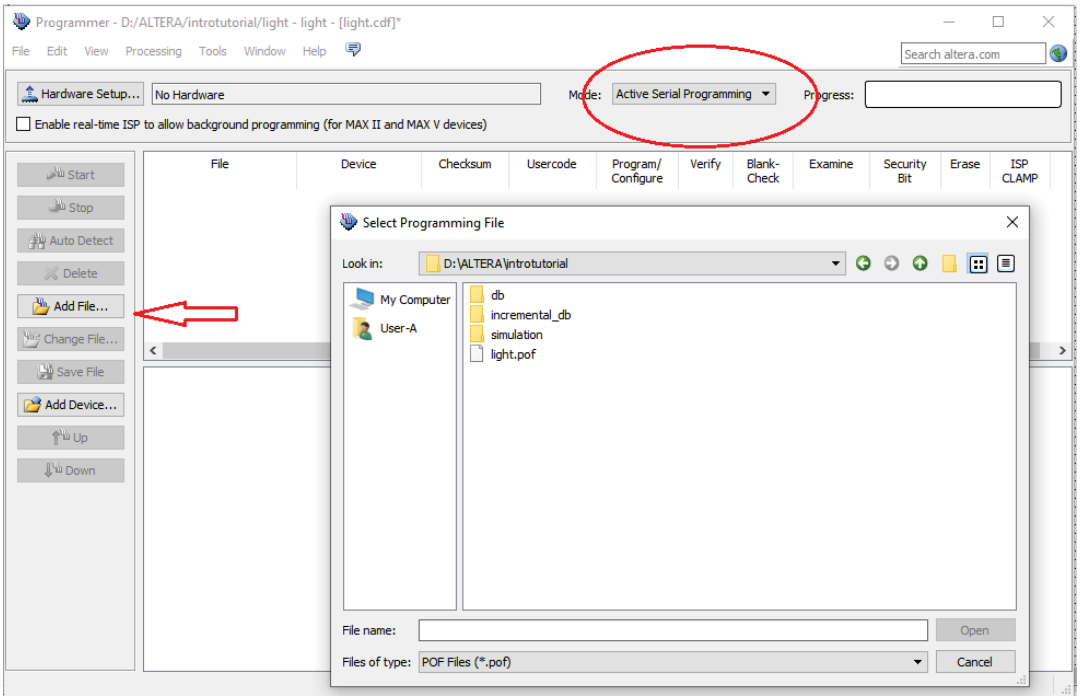
от процедурата за програмиране на FPGA са подобни на описания по-горе режим JTAG. В горния панел на главния прозорец на QuartusII щракнете върху бутона “ Programmer”, ще се появи прозорец за конфигурация. В секцията " Mode" променете режима JTAG на " Active Serial Programming", след което ще се появи нов прозорец. Този прозорец предупреждава за необходимостта от нулиране на предварително посочените устройства за промяна на режима на програмиране. Щракнете върху бутона "Да" или "Ок". След това, като щракнете върху бутона "Add file" фигура 51-2, изберете конфигурационния файл "xxxx.pof" и поставете отметка в квадратчето "Програмиране/Конфигуриране".



Фиг. 50



Фиг. 51. *Избиране на памет за програмиране*



Фиг. 51-2. *Добавяне на файл за програмиране*

Сега, като превключите превключвателя "Run/Prog" на платката в позиция "Prog", стартирайте процеса на програмиране на ROM, като щракнете върху бутона "Start" в прозореца за конфигурация на устройството.

След като програмирането приключи, когато полето за напредък показва 100%, изключете захранването на платката, превключете ключа Run/Prog в положение Run и включете отново захранването на платката. В този случай FPGA чипът ще бъде автоматично програмиран всеки път, когато платката се включи.

3.12. Лабораторни упражнения

I. Синтез на логически схеми.

1. Цел на упражнението

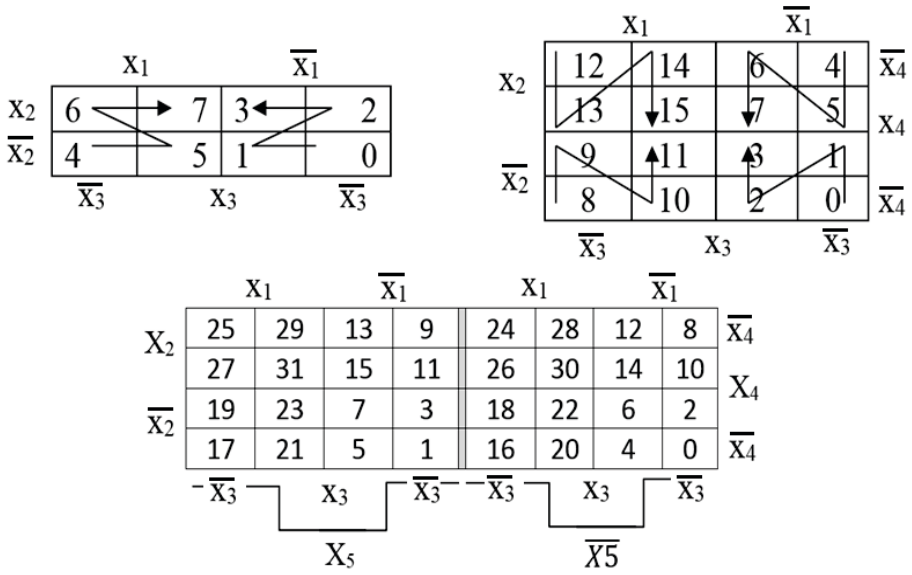
Целта на работата е запознаване с CAD (computer-aided design) за CPLD Intel QuartusII, както и изучаването на методи за синтез на логически схеми и минимизиране на логическите функции.

2. Кратка теоретична информация

2.1. Методи за синтезиране на логически схеми

Всяка логическа схема без памет се описва напълно чрез таблица на истинност. Тази таблица е първоначалната информация за синтез на схема въз основа на логически елементи "И", "ИЛИ", "НЕ". За да разработите необходимото цифрово устройство, първо въз основа на таблицата на истинността се записва нейният логически израз. След това, за да опростите цифровото устройство, го минимизирайте. От получения логически израз разработваме схема, която реализира получената минимална форма. Сложността на цифровите логически схеми за реализиране на булева функция е пряко свързана със сложността на алгебричния израз. Освен това, увеличаването на броя на променливите води до увеличаване на сложността. Въпреки че представянето на булева функция чрез таблица на истинността е уникално, нейният алгебричен израз може да има много различни форми. Методът на картата, предложен от Вейч/Карно, осигурява прост и ясен процес за опростяване на булеви функции. За 2, 3 и 4 входни променливи са показани на фиг.U1.0.

Картата на Вейч предоставя систематичен метод за опростяване и манипулиране на булев израз. Тя представлява диаграма, състояща се от квадрати. За n променливи в карта на Вейч има 2n на брой квадрата. Всеки квадрат или клетка представлява един от минтермите, изразен като сума от минтерми, и е възможно графично да се разпознае булева функция върху картата чрез зоната, която обхваща тези квадрати, чиито минтерми се появяват във функцията.



Фиг. U1.0. *Карти на Вейч – правилото на стрелките*

Чрез анализиране на различни модели е възможно също така да се извлекат алтернативни алгебрични изрази или да се опрости изразът с минимален брой променливи или литерали и сумата от произведения или произведения от суми. Всъщност, картата представлява визуална диаграма на всички възможни начини, по които функцията може да бъде изразена в стандартна форма, и може да бъде избран най-простият алгебричен израз, състоящ се от сума от произведения или произведение от суми. Забележете, че изразът не е задължително уникален.

Булевите изрази могат да се опишат по два начина:

- въз основа на дизюнктивна нормална форма;
- въз основа на конюнктивна нормална форма.

Канонично дизюнктивна нормална форма (КДНФ) се представя от сбора на продуктови термини. Всяка група термини представлява минчлена (резултат лог.1) на функцията.

Например,

$$F(A,B,C) = \overline{A} \cdot B \cdot C + A \cdot \overline{B} \cdot C + A \cdot B \cdot \overline{C}$$

Канонично конюнктивна нормална форма (ККНФ) се представя като умножение на суми. Всяка група представлява максчлен (резултат лог.0) - сумата, която включва всички променливи.

Например,

$$F(A,B,C) = (\overline{A} + B + C) \cdot (A + \overline{B} + C) \cdot (A + B + \overline{C})$$

Ако веригата има няколко изхода, тогава всеки изход се описва и определя от неговата функция. Такава система от функции се нарича система собствени функции.

Таблица L1-1 истинност на функция с три променливи

A	B	C	Y	Минтерм	Макстерм
0	0	0	0	$\overline{A}\,\overline{B}\,\overline{C}$	$A+B+C$
0	0	1	1	$\overline{A}\,\overline{B}C$	$A+B+\overline{C}$
0	1	0	0	$\overline{A}B\,\overline{C}$	$A+\overline{B}+C$
0	1	1	1	$\overline{A}BC$	$A+\overline{B}+\overline{C}$
1	0	0	0	$A\,\overline{B}\,\overline{C}$	$\overline{A}+B+C$
1	0	1	0	$A\overline{B}C$	$\overline{A}+B+\overline{C}$
1	1	0	1	$AB\,\overline{C}$	$\overline{A}+\overline{B}+C$
1	1	1	1	ABC	$\overline{A}+\overline{B}+\overline{C}$

За функцията Y - КДНФ се компилира от таблицата на истинност (L1-1) чрез сумиране (логическо "ИЛИ") на всички онези минтерми, за които изходът Y е 1.

$$Y = \overline{A}\cdot\overline{B}\cdot C + \overline{A}\cdot B\cdot C + A\cdot B\cdot\overline{C} + A\cdot B\cdot C$$

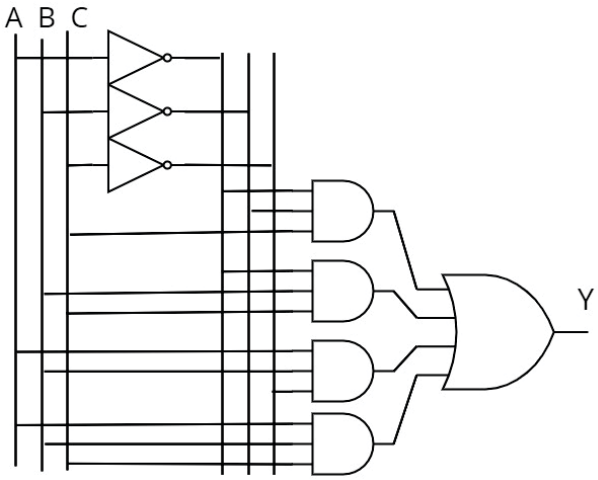
ККНФ се компилира на базата на таблицата на истинността чрез умножаване (логическо И) на всички макстерми, за които изходът Y е 0.

$$Y = (A+B+C)\cdot(A+\overline{B}+C)\cdot(\overline{A}+B+C)\cdot(\overline{A}+B+\overline{C})$$

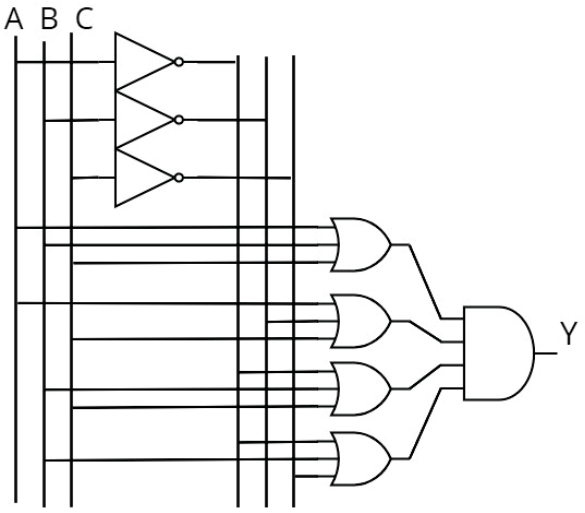
Въз основа на получените изрази е възможно да се състави схема на устройство, което изпълнява дадената функция. Схемата на устройството, получена на базата на КДНФ, е показана на фиг. U1.1, а на базата на ККНФ на фиг. U1.2.

3. Задача за изпълнение

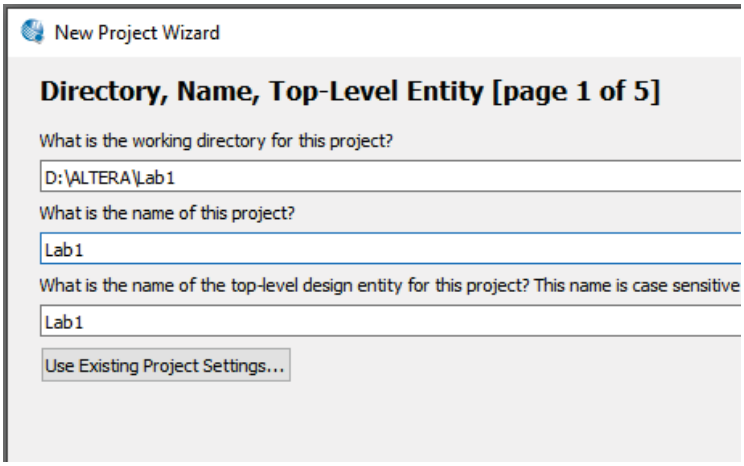
1. Синтезирайте логическа схема на базата на две нива логика (КДНФ и ККНФ) според таблицата на истинността на избрана версия (виж Таблица L1-2).
2. Стартирайте средата Intel Quartus II и създайте нов проект (меню File → New Project Wizard…). Съветникът за създаване на проект съдържа 5 стъпки:
   - Стъпка 1 - Името на проекта и директорията за запазването му (фиг. 1.3). Внимание! Не допускайте имена на файлове с знаци на кирилица. За записване на всеки проект, използвайте отделна папка на диска.



Фиг. U1.1

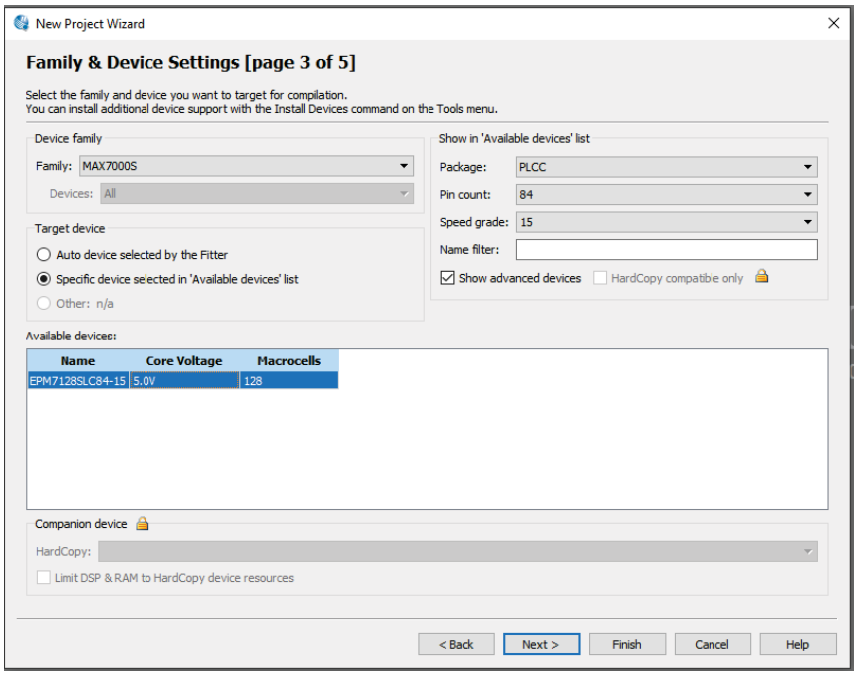


Фиг. U1.2



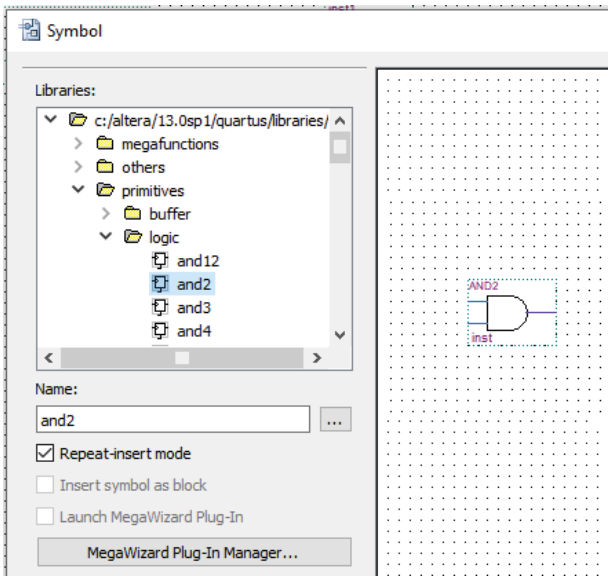
Фиг. U1.3

- Стъпка 2 - Добавяне на други необходими файлове към проекта. В тази работа тази стъпка трябва да бъде пропусната.
- Стъпка 3 – Избиране на семейството и името на CPLD, под която се създава проектът. Семейството MAX7000 трябва да бъде посочено, обозначение EPM7128SLC84-15. За ускоряване на процеса избор на тип CPLD, можете да използвате филтри върху корпуса, брой щифтове, скоростен клас, както и по име чрез въвеждане на символ, включени в името (виж фиг. U1.4).
- Стъпка 4 - Избор на инструменти за синтез, отстраняване на грешки и симулация. В тази работа тази стъпка трябва да бъде пропусната.
- Стъпка 5 – Последната стъпка показва обобщена информация за проекта. За да завършите създаването на проекта, щракнете върху бутона "Край".



Фиг. U1.4

3. Добавете нов файл към проекта (меню File → New…) от типа „Block Diagram/Schematic File“ и начертайте синтезираната схема с помощта на „Orthogonal Node Tool“ (свързващи проводници), „Pin Tool“ (входни и изходни портове), „Symbol Tool“ (логически елементи). Intel Quartus II вече съдържа в своята библиотека стандартни символи за прости логически елементи ("and3", "or4", "not" и т.н.) (виж Фигура U1.5).
4. Запазете файла, без да променяте името на файла и извършете анализ и синтез на проекта (Processing → Start → Start Analysis & Synthesis). Ако няма грешки, направете присвояването на щифтове (меню Assignments → Pin Planner на щифтове) в полето Location в долната част на прозореца на Pin Planner в съответствие с първия и втория ред на Таблица. L1-2.



Фиг. U1.5

5. Пълна компилация на проекта (Processing → Start Compilation). Процесът на компилация създава *.sof конфигурационен файл в папката на проекта в подпапката "output_files". Използва се в програмирането на CPLD.

Таблица L1-2 Назначаване на изводи на схемата

Сигнал	A	B	C	Y
Извод на CPLD	PIN_41	PIN_40	PIN_39	PIN_27
Наименование на стенда	Button-SWDIP4(1)	Button-SWDIP4(2)	Button-SWDIP4(3)	LED_L0

6. В присъствието на учител свържете лаборатория стенд към компютъра и включете захранването на стенда.

7. Програмирайте тренировъчната платка (меню Tools → Programmer) и настройте на входовете на веригата с помощта на ключове SWDIP4(1)…(4) всички възможни кодови комбинации от таблицата на истината и гледайки LED_L0, проверете работата на веригата.

8. Съставете минимизирана логическа функция според таблицата на истинността на вашата версия (вижте Таблица L1-3), като използвате картата на Вейч. Синтезирайте веригата от минимизираната функция.

9. Начертайте новата схема в Intel Quartus II в същия файл и извършва анализ и синтез на проекта. След това задайте номера на изводите и завършете компилацията на проекта.

10. Програмирайте тренировъчната платка и проверете работата на веригата за нейното съответствие с дадената таблица на истинността.

11. Изключете захранването на лабораторната развойна система и я изключете

от компютър.

12. Направете изводи.

Съдържание на доклада

1. Целта на работата.
2. Дадена таблица на истинността и синтезирана схема.
3. Карта на Вейч, минимизирана логическа функция и схема.
4. Изводи

Варианти на задачи

Таблица L1-3

Вариант			1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	B	C	Y														
0	0	0	0	1	0	0	1	0	1	0	1	0	1	0	0	0	1
0	0	1	1	0	0	1	0	0	0	1	1	1	0	1	0	1	1
0	1	0	0	1	1	1	1	0	1	0	0	1	0	0	1	0	0
0	1	1	1	1	1	0	0	1	0	1	0	0	1	0	0	1	0
1	0	0	1	0	1	0	1	1	1	0	0	0	1	1	1	0	1
1	0	1	1	0	0	1	0	1	1	1	0	0	1	0	1	0	1
1	1	0	0	1	1	0	0	0	0	1	1	1	0	1	1	1	0
1	1	1	0	0	0	1	1	1	0	0	1	1	0	1	0	1	0

Контролни въпроси

1. Какво е КДНФ/ККНФ?
2. Как да напиша КДНФ/ККНФ с помощта на таблица на истинност?
3. Как най-добре да се синтезира логическо устройство (въз основа на КДНФ или ККНФ), ако стойността на функцията в таблицата на истината има повече нули от единици?
4. Как да разработим логическо устройство, ако то има няколко изхода?
5. Какво е минимизирането на логически израз и какви са начините за минимизиране?
6. Кажете как се определя таблицата на истинността на логиката експериментално с помощта на лабораторната платка.
7. Напишете таблици на истинността за главните логически елементи (И, ИЛИ, НЕ, XOR).
8. Какво е проект в Intel Quartus II? Защо е необходимо компилиране на проект в Quartus II и как да го направим?
9. За какво е операцията за присвояване на изводи в Intel Quartus II?

II. Изследване на комбинационни схеми.

1. Цел на упражнението

Целта на работата е изучаване на принципите на действие на комбинационните схеми: дешифратор, шифратор, преобразувател на код за седемсегментен индикатор, мултиплексор и суматор.

2. Кратки теоретични сведения

При практическата реализация на КС се използват типови логически елементи, имащи различна структура. Към едностъпалните ЛЕ се отнасят инверторите, схемите И, ИЛИ, И-НЕ, ИЛИ-НЕ с брой на входовете най-често 2, 3, 4 или 8. Няколко такива схеми са обединени в един интегрален модул. Двустъпални са схемите“ сума по модул 2”,“равнозначност”,структури от вида 2И/2И-2ИЛИ-НЕ, 2И/2И/2И/3И-4ИЛИ-НЕ, мултиплексори, програмируеми логически матрици и др. На тази база могат да се получат различни варианти на реализация на дадена логическа зависимост. Част от тях се основават на минималното представяне на тази зависимост, а друга – на таблицата й на истинност.

2.1. Дешифратор (декодер)

Дешифраторът (декодерът) служи за преобразуване на n-разреден позиционен двоичен код в единичен изходен сигнал на един от 2^n изходи. При всяка входна комбинация на един от изходите се появява 1. По този начин, по единичен сигнал на един от изходите може да се съди за входната кодова комбинация. Таблицата на истинността за декодер с два входа е показана в таблица L2-1.

Таблица L2-1 – Таблица на истинността на двуразреден дешифратор с прави изходи.

x1	x2	y0	y1	y2	y3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

За построяването на схемата на декодера по таблицата на истинността ще се използва методиката за работа с Quartus, изложена по-горе, изпълнявана на лабораторния стенд. Например, устройството трябва да има 4 изхода. За всеки изход записваме логическо уравнение. На основата на КДНФ:

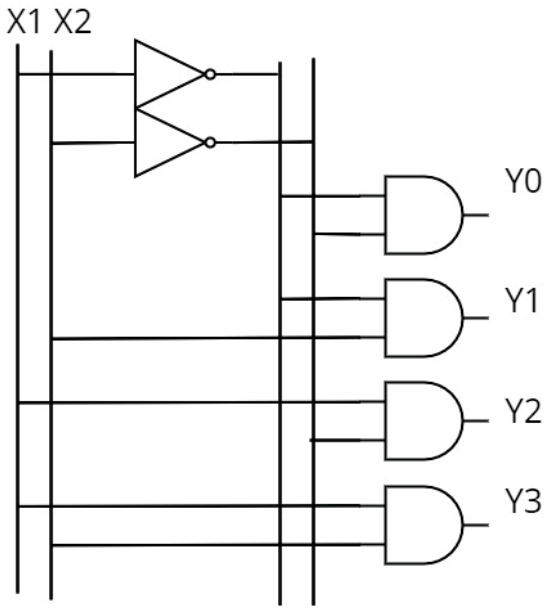
$$y0 = \overline{x1} \cdot \overline{x2}$$

$$y1 = \overline{x1} \cdot x2$$

$$y2 = x1 \cdot \overline{x2}$$

$$y3 = x1 \cdot x2$$

По тази система от изрази не е трудно да се построи схемата на изискванията дешифратор (фигура U2.1).



Фиг. U2.1. *Схема на дешифратора*

2.2 Шифратор

Шифраторът изпълнява функция, обратна на декодера (дешифратора), тоест преобразува непозиционен (унитарен) двоичен код с 2^n разреда в n-разреден позиционен код. При подаване на единичен сигнал на един от входовете на изхода се формира съответстващ двоичен код. Ще съставим таблица на истинността на шифратора при n = 2.

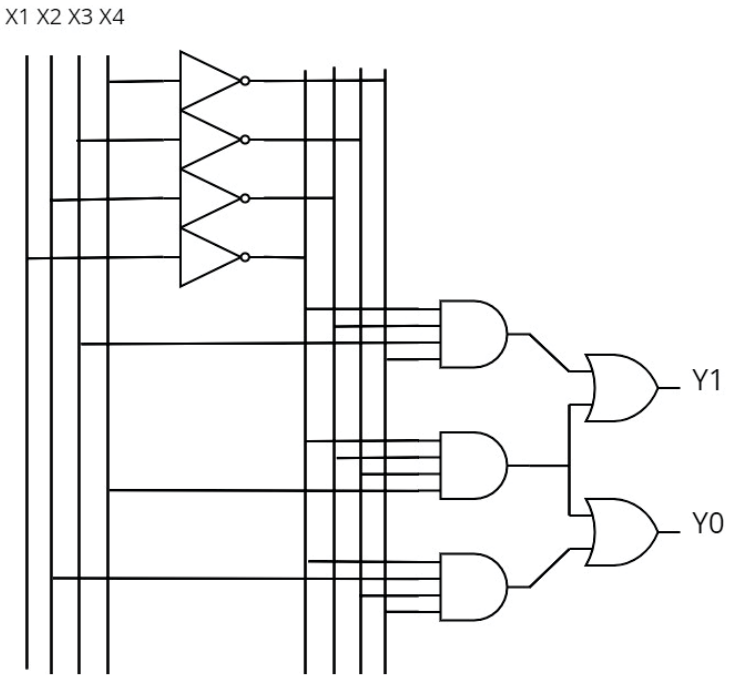
Таблица L2-2 – Таблица на истинността на шифратора при n = 2

x1	x2	x3	x4	y1	y0
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1

Синтезираме шифратор. За целта ще запишем системата на неговите собствени функции:

$$y1 = \overline{x1} \cdot \overline{x2} \cdot x3 \cdot \overline{x4} + \overline{x1} \cdot \overline{x2} \cdot \overline{x3} \cdot x4$$

$$y0 = \overline{x1} \cdot x2 \cdot \overline{x3} \cdot \overline{x4} + \overline{x1} \cdot \overline{x2} \cdot \overline{x3} \cdot x4$$



Фиг. U2.2. *Схема на шифратор*

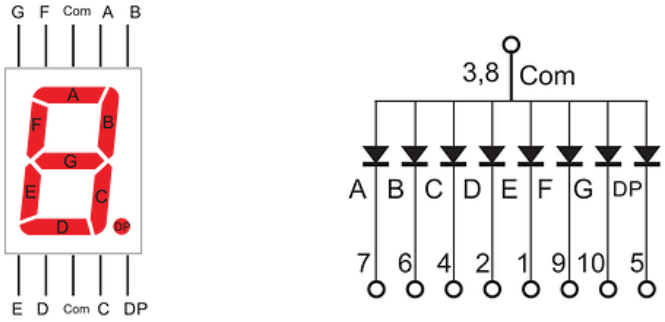
Видно е, че така получените функции Y1 и Y0 зависят само от променливите X3 или X4, както и X2 или X4, т.е. резултата Y ще е лог.1 когато поне един от тези входове е лог.1. Уравненията ще бъдат опростени. Това при условие, че входните комбинации са тези от таблица L2-2

$$y1 = x3 + x4$$

$$y0 = x2 + x4$$

2.3 Преобразувател на код за седемсегментен индикатор

Най-широко преобразувателите на кодове са известни по отношение на цифровите индикатори. Например, преобразувател на 4-разреден позиционен двоичен код в десетични цифри. Разполага се със седемсегментен индикатор, с помощта на който е необходимо да се изобразят десет цифри.



Фиг. U2.3 *– Седемсегментен индикатор*

Очевидно е, че двоичният код трябва да има най-малко 4 разреда ($2^4 = 16$, което е повече от 10). Ще съставим таблица на истинността за работата на такъв преобразувател. Прието е, че даден сегмент свети при подадена към него логическа нула.

Таблица L2-3 – Таблица на истинността на преобразувателя.

вх. код	изх. код
$x_1x_2x_3x_4$	$y_ay_by_cy_dy_ey_fy_g$
0 0 0 0	0 0 0 0 0 0 1
0 0 0 1	1 0 0 1 1 1 1
0 0 1 0	0 0 1 0 0 1 0
0 0 1 1	0 0 0 0 1 1 0
0 1 0 0	1 0 0 1 1 0 0
0 1 0 1	0 1 0 0 1 0 0
0 1 1 0	0 1 0 0 0 0 0
0 1 1 1	0 0 0 1 1 1 1
1 0 0 0	0 0 0 0 0 0 0
1 0 0 1	0 0 0 0 1 0 0
1 0 1 0	* * * * * * *
----------	----------
1 1 1 1	* * * * * * *

x_2

y_a

x_1		x_3	
*	*		1
*	*		
	*		1
	*		

x_4

y_b

*	*	1	
*	*		1
	*		
	*		

y_c

*	*		
*	*		
	*		
	*	1	

y_d

*	*		1
*	*	1	
	*		1
	*		

y_e

*	*		1
*	*	1	1
1	*	1	1
	*		

y_f

*	*		
*	*	1	
	*	1	1
	*	1	

*	*		
*	*	1	
	*		1
	*		1

y_g

$y_a = \bar{x}_1 \bar{x}_2 \bar{x}_3 x_4 \vee x_2 \bar{x}_3 \bar{x}_4;$

$y_b = x_2 x_3 \bar{x}_4 \vee x_2 \bar{x}_3 x_4;$

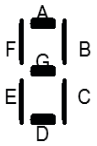
$y_c = \bar{x}_2 x_3 \bar{x}_4;$

$y_d = y_a \vee x_2 x_3 x_4;$

$y_e = x_4 \vee x_2 \bar{x}_3 \bar{x}_4;$

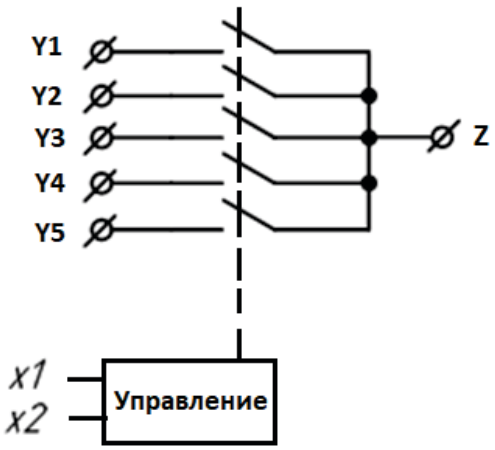
$y_f = x_3 x_4 \vee \bar{x}_2 x_3 \vee \bar{x}_1 \bar{x}_2 x_4;$

$y_g = \bar{x}_1 \bar{x}_2 \bar{x}_3 \vee x_2 x_3 x_4.$



2.4 Мултиплексор

Мултиплексорът е устройство, което позволява комутация на един от 2^n информационни входове Y към един изход Z под действието на n управляващи (адресни) сигнали. На фигура U2.4 е изобразена опростена функционална схема на мултиплексора с идеализирани електронни ключове.



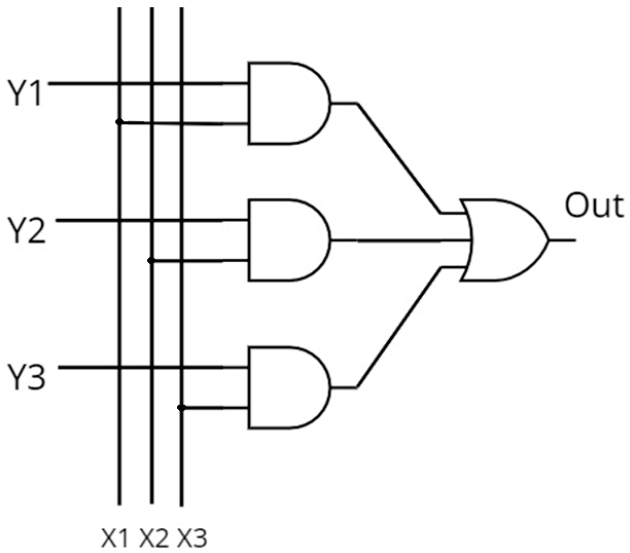
Фиг. U2.4 *–Схема на мултиплексора с идеализирани електронни ключове.*

В цифровите схеми е необходимо да се управляват ключовете с помощта на логически нива. Поради това е желателно да се избере устройство, което да изпълнява функциите на електронен ключ с управление чрез цифров сигнал. Ще опитаме да използваме вече познатите ни логически елементи като електронен ключ. Нека разгледаме логическия елемент "И". Един от входовете на логическия елемент "И" ще разгледаме като информационен вход на електронния ключ, а другият вход – като управляващ. Тъй като двата входа на логическия елемент "И" са еквивалентни, не е важно кой от тях ще бъде управляващ вход. Нека вход X бъде управляващ, а Y – информационен. За улеснение на разсъжденията ще разделим таблицата на истинността на две части в зависимост от нивото на логическия сигнал на управляващия вход X.

Таблица L2-4 – Таблица истинност

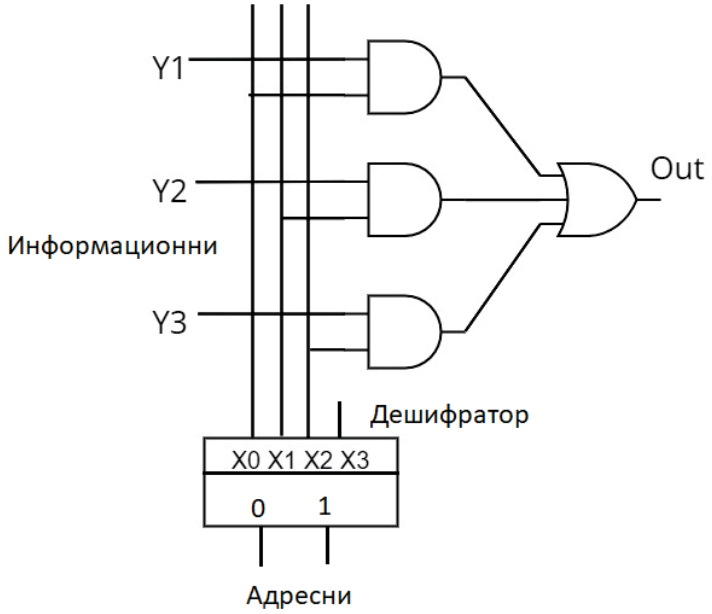
y	x	Out
0 0	0 1	0 0
1 1	0 1	0 1

По таблицата на истинността ясно се вижда, че ако на управляващия вход X се подаде нулево логическо ниво, сигналът, подаден на вход Y, не преминава на изход Out. При подаване на логическа единица на управляващия вход X, сигналът, постъпващ на вход Y, се появява на изхода Out. Това означава, че логическият елемент "И" може да се използва като електронен ключ. При това не е важно кой от входовете на елемента "И" ще се използва като управляващ вход и кой – като информационен. Остава само да се обединят изходите на елементите "И" на един общ изход. Това се прави с помощта на логическия елемент "ИЛИ", точно както при построяване на схема по произволна таблица на истинността. Полученият вариант на схемата на комутатора с управление чрез логически нива е показан на фигура U2.5.



Фиг. U2.5. *Мултиплексор, реализирана с логически елементи*

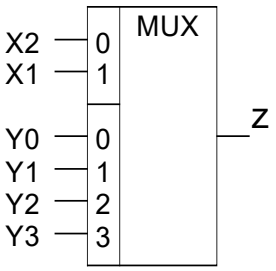
В схемата, показани на фигура U2.5 може едновременно да се включват много входове на един изход, това обикновено води до непредсказуеми последици. Освен това, за управление на такъв комутатор са необходими много адресни входове, затова в състава на мултиплексора обикновено се включва двоичен дешифратор, както е показано на фигура U2.6.



Фиг. U2.6. *Мултиплексор, управляван от двоичен код*

Такава схема позволява управление на превключването на информационните входове на мултиплексора чрез двоични кодове, подавани на неговите управляващи входове. Броят на информационните входове в тези схеми 2^n, където n е броя на адресните входове.

Условното графично обозначение на 4-входов мултиплексор с управление чрез двоичен код е показано на фигура U2.7. Входовете X1 и X2 са управляващи входове на мултиплексора, които определят адреса на информационния входен сигнал, който ще бъде свързан с изходния извод на мултиплексора Z. Информационните входни сигнали са обозначени като: Y0, Y1, Y2 и Y3.



x_1x_2	Изход Z
0 0	$z = y_0$
0 1	$z = y_1$
1 0	$z = y_2$
1 1	$z = y_3$

$$Z = \mathbf{y_0}\overline{x}_1\overline{x}_2 + \mathbf{y_1}\overline{x}_1x + \mathbf{y_2}x_1\overline{x}_2 + \mathbf{y_3}x_1x_2$$

Фиг. U2.7. *Условно графично обозначение на 4-входов мултиплексор*

2.5 Суматор

Суматорът е устройство в компютъра, предназначено за събиране на двоични числа. Проектирането на двоични суматори обикновено започва със суматор по модул 2.

Схемата на суматора по модул 2 съвпада със схемата на изключващото "ИЛИ" (XOR).

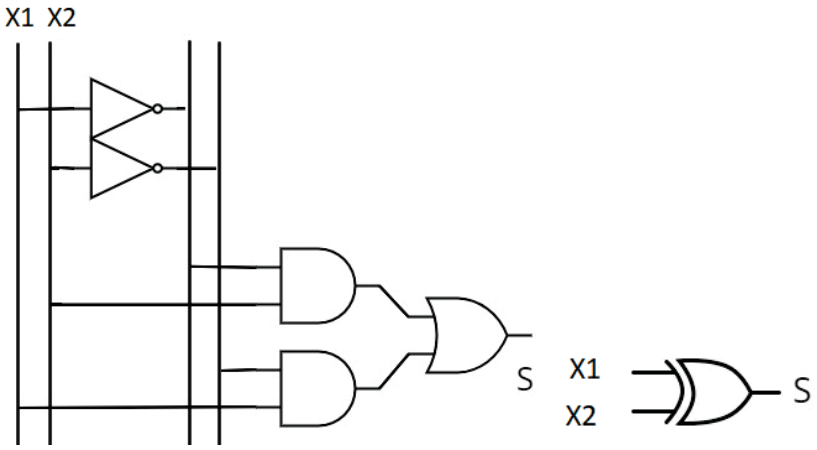
Таблица L2-5 – Таблица на истинността на суматора по модул 2

x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0

Логическото изражение, което описва суматора по модул 2. Тази операция е типична за сумиране на два битови входа без пренасяне.

$$y = \overline{x1} \cdot x2 + x1 \cdot \overline{x2} = x1 \oplus x2$$

На основата на логическото уравнение, което описва суматора по модул 2 (или изключващо ИЛИ - XOR), може да се синтезира схема.



***Фиг. U2.8** – Схема сума по модул 2*

Суматорът по модул 2 извършва сумирането, без да отчита пренасянето. Един конвенционален двоичен суматор трябва да отчита пренасянето,

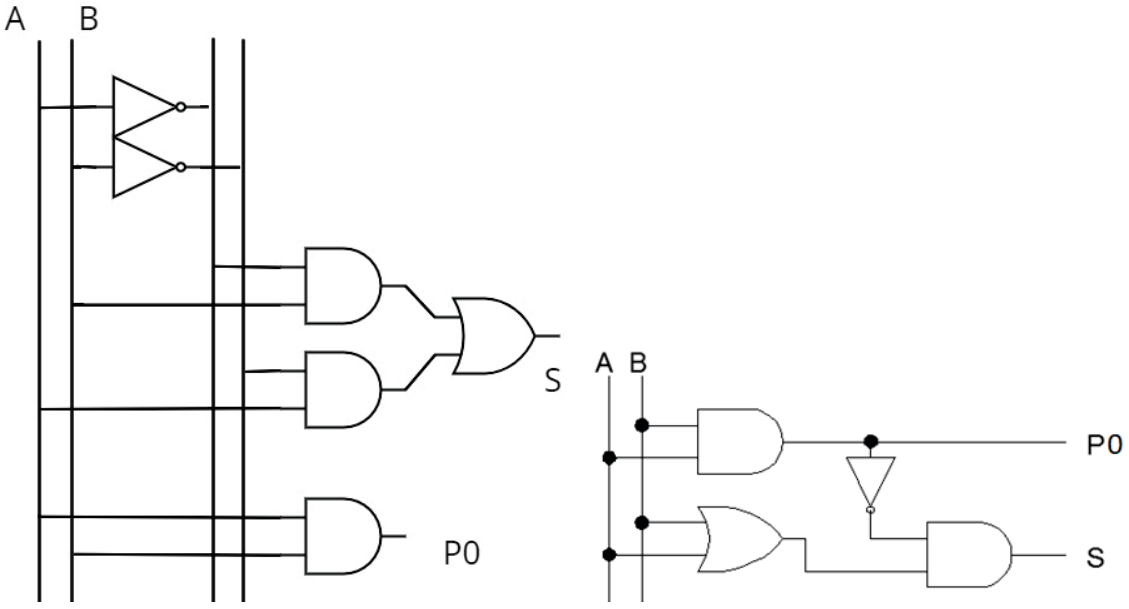
така че от схемите се изисква да генерират пренасянето в следващия двоичен бит. Таблицата на истинността на такава верига, наречена полусуматор, е дадена в таблица L2-6.

Таблица L2-6 – Таблица на истинността на полусуматора

A	**B**	**S**	**Po**
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Където **A** и **B** – Двата входни двоични бита, които трябва да бъдат събрани.;
S – Резултатът от събирането на входовете A и B, без да се взема предвид пренасянето.;
Po – Индикатор за пренос, който показва дали е имало пренос към следващия бит.

По лог.1 уравнението ще бъде S = $\bar{A} \cdot B + A \cdot \bar{B}$, и съответно Po = A · B. По лог.0 записа ще има вида S = (A + B) . ($\bar{A} + \bar{B}$) . След преобразуване по Де Морган израза $\overline{(\bar{A} + \bar{B})}$ се получава $\overline{A.B}$ което е $\overline{Po}$, т.е S = (A + B). $\overline{Po}$



Фиг. U2.9. *Схеми на полусуматор*

Пълен суматор

Схемата на полусуматор генерира пренасяне в MSB, но не може да отчете пренасянето от LSB. При събиране на многоцифрени двоични числа е необходимо във всяка цифра да се добавят три цифри - 2 събираеми и една за пренасяне от предходната цифра PI.

Таблица L2-7 – Пълна таблица на истинността на суматора

PI	A	B	S	Po
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

PI – Вход за пренос от предходен разряд

Po – Изход за пренос старши разряд.

На основание на таблицата на истинността на пълния суматор можем да запишем системата от логически функции, които описват изходите на схемата. В пълния суматор имаме два основни изхода: сумата (S) и преносът (Po).

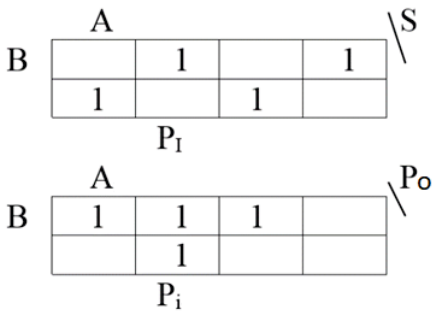
$$S = \bar{A} \cdot B \cdot \overline{PI} + A \cdot \bar{B} \cdot \overline{PI} + \bar{A} \cdot \bar{B} \cdot PI + A \cdot B \cdot PI$$

$$S = A \oplus B \oplus PI$$

$$Po = (A \cdot B \cdot \overline{PI}) + (\bar{A} \cdot B \cdot PI) + (A \cdot \bar{B} \cdot PI) + (A \cdot B \cdot PI)$$

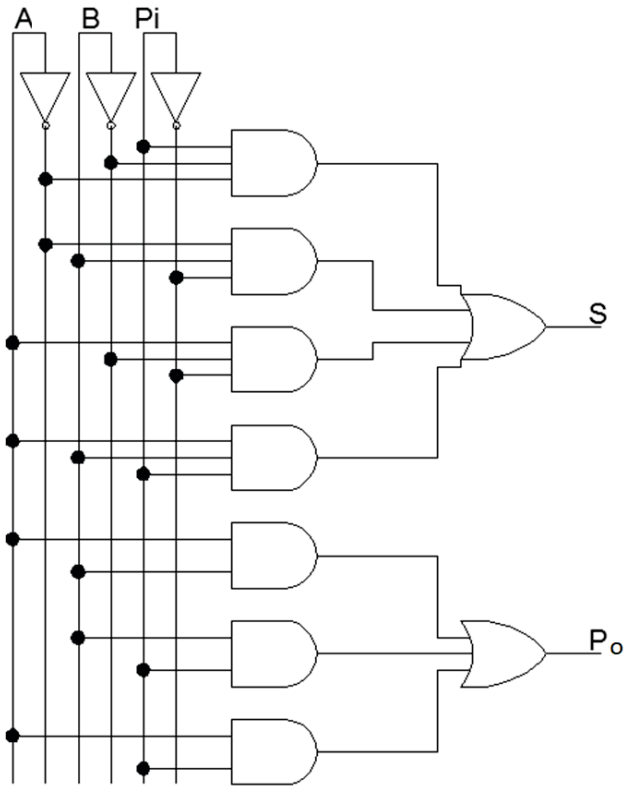
$$Po = A.B + (PI.(A \oplus B))$$

Функцията на изхода за пренос (Po) може да се минимизира с карта на Вейч, при което се получава следното уравнение:



$$Po = A.B+B.PI+A.PI$$

В резултат получаваме схема на пълен суматор (фиг. U2.10).

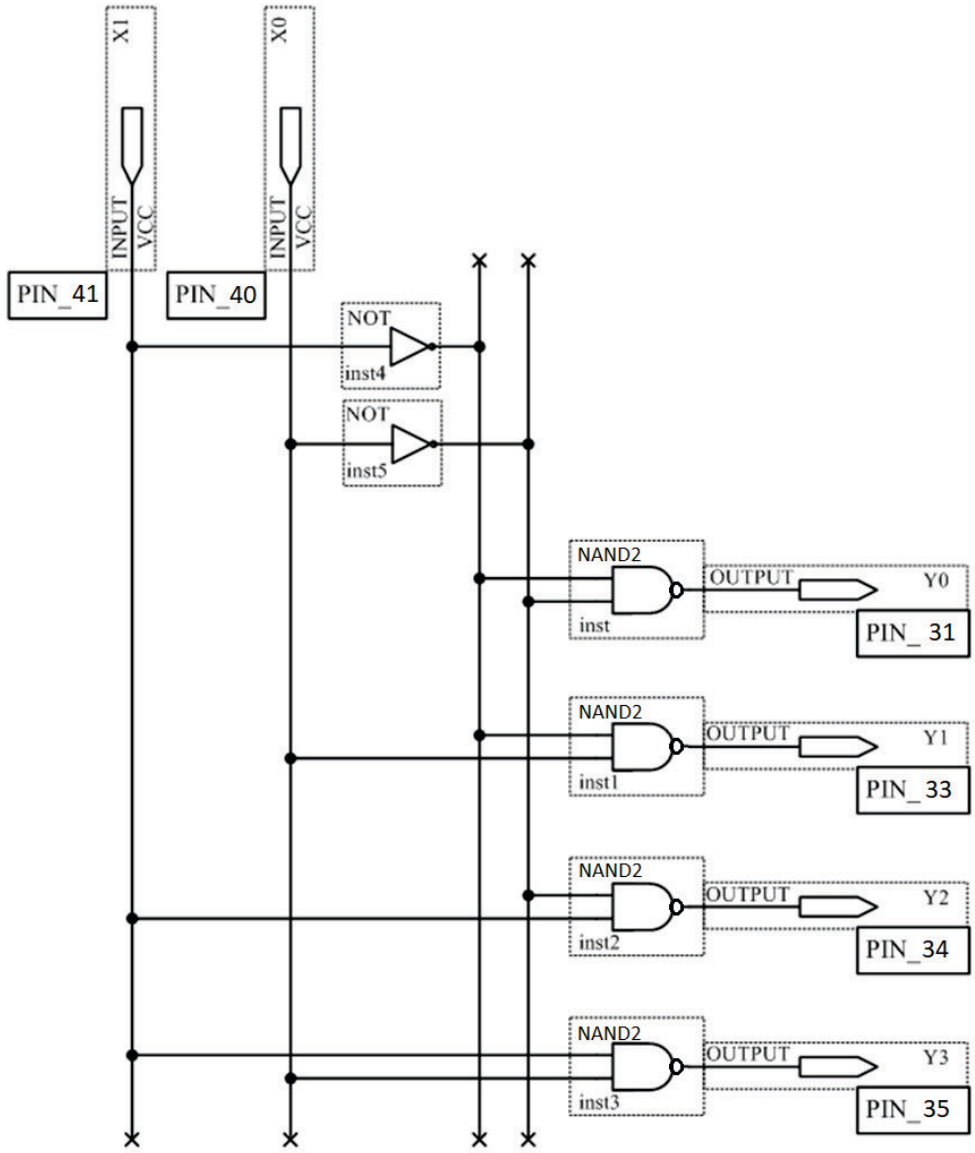


***Фиг. U2.10** – пълен двоичен еднобитов суматор*

3. Задачи за изпълнение

3.1 Изследвайте принципа на работа на декодера 2x4. Конфигурирайте CPLD според фиг.U2.11. Свържете входовете X0 и X1 към превключвателите SWDIP(1) и SWDIP(2), а изходите Y0, Y1, Y2, Y3 към светодиодните индикатори LED4, LED5, LED6, LED7. За тази цел се присвояват номерата на входовете и изходите на декодера към съответните пинове на CPLD. Да

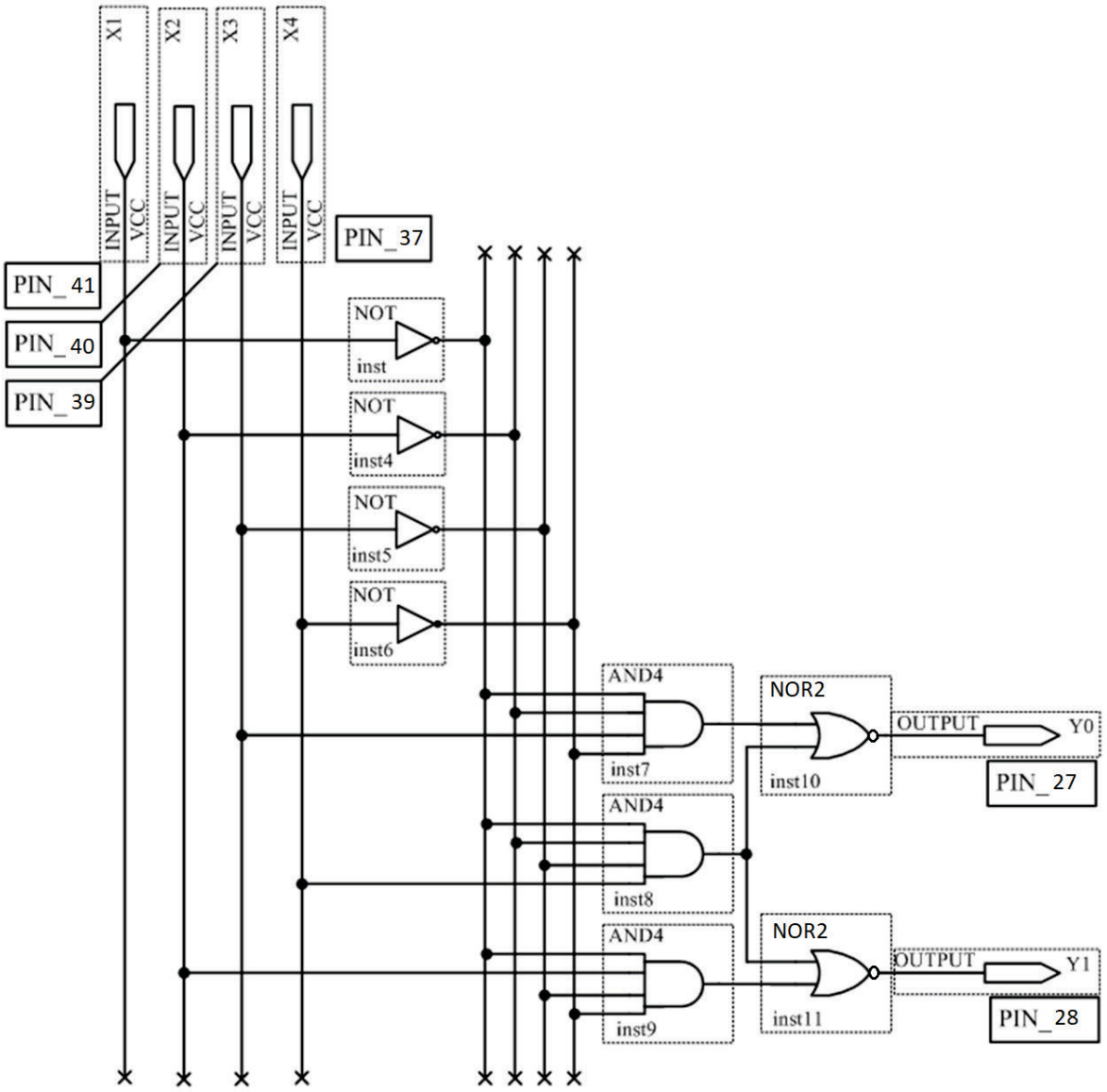
не забравяме, че светодиодите се управляват по лог.0. Това ще наложи изходите да бъдат инвертирани.



Фиг. U2.11 *– Схема на дешифратор*

Подават се комбинационните логически набори на входовете X1 и X0 и се наблюдават светодиодните индикатори. Да се попълни таблицата на истинност на дешифратора.

3.2 Изследване принципа на действие на шифратор 4x2. Да се конфигурира CPLD със схемата на фиг. U 2.12.



Фиг. U2.12 *– Схема на шифратор 4x2*

Свържете към входовете X1, X2, X3, X4 превключвателите SWDIP4(4), SWDIP4(3), SWDIP4(2), SWDIP4(1), а към изходите Y0, Y1 светодиодните индикатори LED0, LED1. За целта свържете входовете и изходите на дешифратора към съответните пинове на CPLD. Подавате всички възможни комбинации от логически нива на входовете X1, X2, X3, X4 с помощта на ключове и наблюдавате състоянието на светодиодните индикатори LED6, LED7, попълнете таблицата на истинност на шифратора.

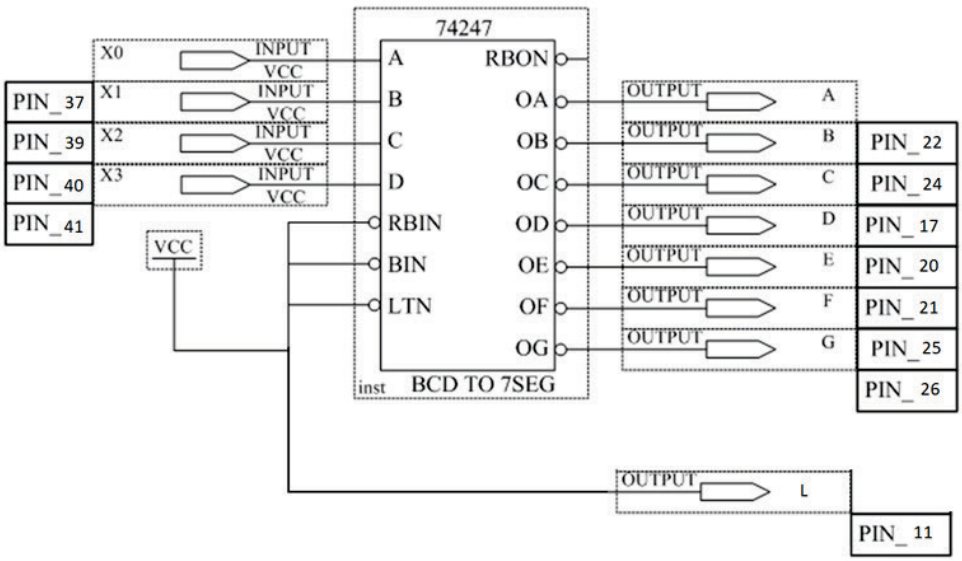
3.3 Изследване на работата на преобразователя на код за седемсегментен индикатор. Съставяне на таблица на истинност на преобразователя (Таблица L2-8). Програмиране на схемата, показана на фиг.U2.13. Софтуера разполага с налични в библиотеката интегрални схеми от серията 74xxx. Тук

ще се използва готов модул 74247 с инверсни изходи, което позволява директно управление на индикатора. Да се разучи действието на входовете RBIN, BIN, LTN.

Таблица L2-8 – Таблица на истинност на преобразователя

x3	x2	x1	x0	A	B	C	D	E	F	G
0	0	0	0							
0	0	0	1							
0	0	1	0							
0	0	1	1							
0	1	0	0							
0	1	0	1							
0	1	1	0							
0	1	1	1							
1	0	0	0							
1	0	0	1							

Подавате с помощта на ключове SWDIP4(4), SWDIP4(3), SWDIP4(2), SWDIP4(1) различни кодови комбинации на входовете X0, X1, X2, X3, за да се определят цифрите, които се показват на индикатора.



Фиг. U2.13 *– Схема на преобразовател на код за 7seg индикатор*

Въз основа на резултатите от експеримента попълнете таблица L2-9. Макета CSE1 разполага с 4-ри седемсегментни индикатора, което налага да изберем едни и да го направил активен. Използваме 7-Seg M1 управляван от извод номер 11 на CPLD.

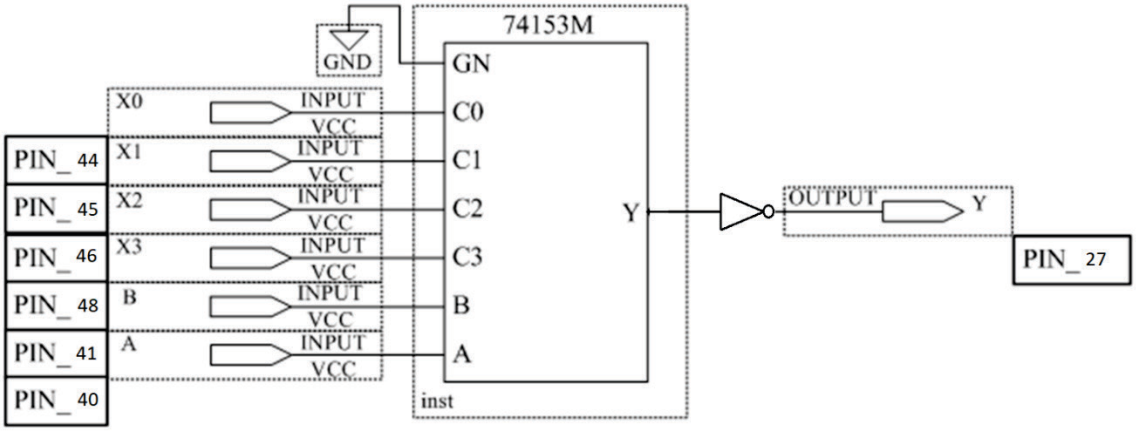
Таблица L2-9 – Таблица, описваща работата на преобразователя на код за седемсегментен индикатор.

x3	x2	x1	x0	**Показание на индикатора**
0	0	0	0	
0	0	0	1	
0	0	1	0	
0	0	1	1	
0	1	0	0	
0	1	0	1	
0	1	1	0	
0	1	1	1	
1	0	0	0	
1	0	0	1	

3.4 Изследване работата на мултиплексор 4x1

Програмиране на CPLD в съответствие със схема на фиг. U2.14.

Поетапно задавайки всички възможни кодови комбинации на адресните входове A и B, определете номера на превключваните канали. Номерът на превключвания канал се определя чрез последователно подаване на логическо ниво единица/нула към информационните входове X0, X1, X2, X3 и наблюдение на изхода Y. Попълнете таблица L3.5.



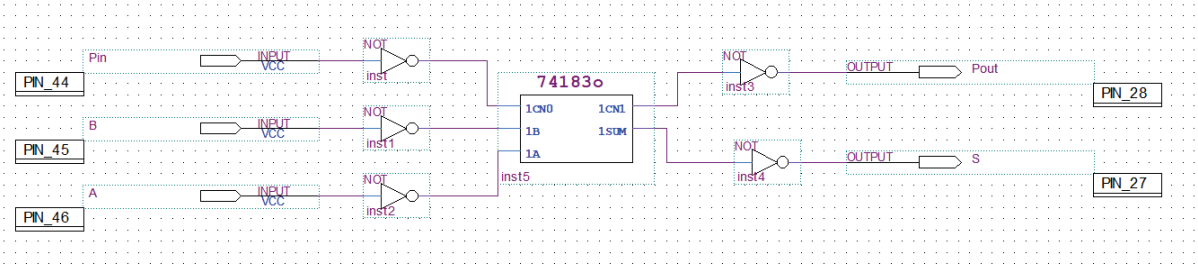
Фиг. U2.14 *– Схема на мултиплексор 4x1*

Таблица L2-10 – Таблица, описваща работата на мултиплексор.

B	A	Номер канал
0	0	
0	1	
1	0	
1	1	

3.5 Изследване схема на суматор

Програмиране на CPLD в съответствие със схема на фиг.U2.15. Тук **Pin**, **Pout** са съответно вход и изход за пренос, **A** и **B** – събираеми, **S** – сума.



Фиг. U2.15 *– Схема на суматор*

Да се попълни таблицата на истинност на суматора (таблица L2-11).

Таблица L2-11 – Таблица на истинност на пълен суматор

Pin	B	A	Pout
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

4 Съдържание на протокола

1. Цел на работата.
2. Схеми за изследване на дешифратор, шифратор, преобразовател на код за седемсегментен индикатор, мултиплексор и суматор.
3. Таблици на истинност за всяка схема.
4. Изводи по всяко задание.

5 Контролни въпроси

1. Принцип на работа на дешифратора?
2. Как да се синтезира дешифратор с произволна разреденост?
3. Как работи шифратор?
4. Изобразете таблица на истинност на шифратора.
5. Как работи преобразователят на код за седемсегментен индикатор?
6. Как е устроен седемсегментен индикатор?
7. Как работи мултиплексорът?
8. Как беше проведено изследването на мултиплексора в лабораторната работа?
9. Как работи суматорът?
10. Изобразете таблица на истинност на шифратора.
11. Какво е единица за пренос?

III. Изследване на тригери

1. Цел на упражнението

Целта на работата е запознаване с принципа на действие и експериментално изследване на различни типове тригери.

2. Кратки теоретични сведения

Тригерите са предназначени за запаметяване на двоична информация. Използването на тригери позволява реализирането на устройства за оперативна памет (тоест памет, в която информацията се съхранява само по време на изчисленията). Въпреки това, тригерите могат да се използват и за създаването на някои цифрови устройства с памет, като броячи, преобразуватели на последователен код в паралелен или цифрови линии за забавяне.

2.1 RS-тригер

Основният тригер, върху който се базират всички останали тригери, е RS-тригерът.

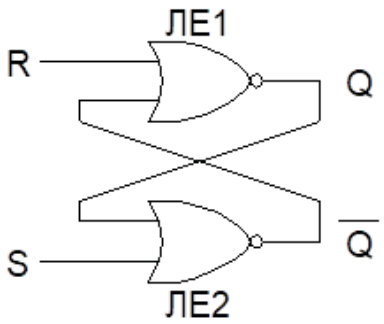
RS-тригерът има два логически входа:

- **R** - задаване на 0 (от думата *reset*);
- **S** - задаване на 1 (от думата *set*).

RS-тригерът има два изхода:

- **Q** - директен;
- $\overline{\mathbf{Q}}$- обратен (инверсен).

Състоянието на тригера се определя от състоянието на директния изход. Най-простият RS-тригер се състои от два логически елемента, обхванати от кръстосана положителна обратна връзка (фигура U3.1).



Фиг. U3.1. *схема на RS тригер*

Нека R=0, S=1 и да приемем, че изхода Q=1 при първоначално включване на схемата към захранване. Долният логически елемент изпълнява логическата функция ИЛИ-НЕ, т.е. 1 на всеки от неговите входове води до това, че на неговия изход ще има логическа нула $\overline{Q}$=0. На правия изход ще има Q=1, тъй като на двата входа на горния елемент се подават нули (една

нула от входа R, друга - от инверсния изход $\bar{Q}$) Тригерът се намира в установено състояние. Ако сега премахнем сигнала за задаване (R=0, S=0), на изхода ситуацията няма да се промени, тъй като въпреки че на долния вход на долния логически елемент ще постъпва 0, на горния му вход постъпва 1 от изхода Q на горния логически елемент. Тригерът ще остане в установено състояние, докато на входа R не постъпи сигнал за нулиране.

Нека промени R=1, S=0. Тогава изходите ще превключат Q=0, а $\bar{Q}$=1. Тригерът се е превключил в "0". Ако след това премахнем сигнала за нулиране (R=0, S=0), тригерът отново няма да промени своето състояние.
За описание на работата на тригера се използва таблица на състоянията (преходите).

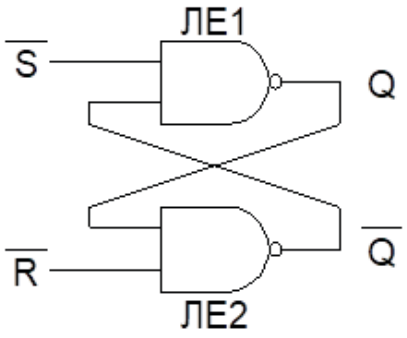
Нека обозначим:

- Q(t) - състояние на тригера преди подаването на управляващите сигнали (промени на входовете R и S);
- Q(t+1) - състояние на тригера след промяната на входовете R и S.

Таблица L3-1 - Таблица на преходите на RS тригер в ИЛИ-НЕ

R	**S**	**Q(t)**	**Q(t+1)**	**Пояснения**
0	0	0	0	Режим на запазване на информации R=S=0
0	0	1	1	
0	1	0	1	Режим на установяване S=1
0	1	1	1	
1	0	0	0	Режим на нулиране R=1
1	0	1	0	
1	1	0	*	R=S=1 забранени комбинации
1	1	1	*	

RS тригер може да бъде съставен и с елементи И-НЕ. В литературата може да се срещне като $\bar{R}\bar{S}$ тригер. Активното ниво на входовете е лог.0 - фигура U3.2.



Фиг. U3.2. *схема на $\bar{R}\bar{S}$ тригер*

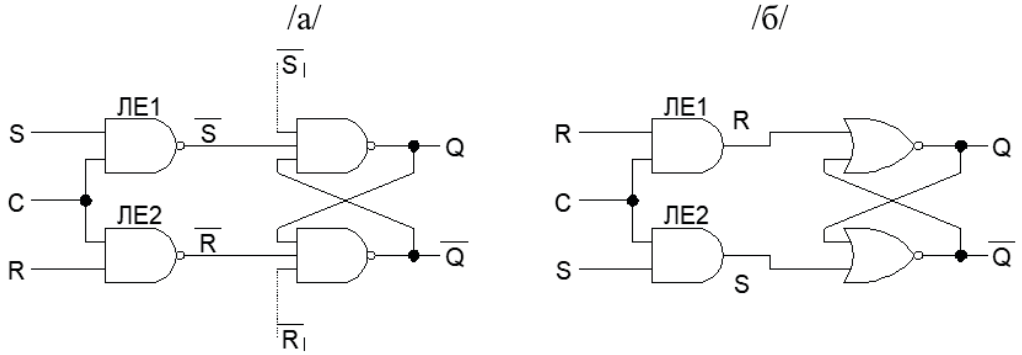
Входовете R и S са инверсни (активно ниво „0“). Преходът (превключването) на този тригер от едно състояние в друго се осъществява при задаване на „0“ на един от входовете. Комбинацията R=S=0 е забранена.

Таблица L3-2 - Таблица на преходите на $\overline{R}\overline{S}$ тригер в И-НЕ

$\overline{R}$	$\overline{S}$	Q(t)	Q(t+1)	Пояснения
0	0	0	*	R=S=0 забранени комбинации
0	0	1	*	
0	1	0	0	Режим на нулиране R=0
0	1	1	0	
1	0	0	1	Режим на установяване S=0
1	0	1	1	
1	1	0	0	Режим на запазване на информации R=S=1
1	1	1	1	

2.2 Синхронен по ниво RS-тригер

Схемата на RS-тригера позволява запаметяването на състоянието на логическата схема, но тъй като при промяна на входните сигнали може да възникне преходен процес (в цифровите схеми този процес се нарича „опасни гонки“), запаметяването на състоянията на логическата схема трябва да се извършва само в определени моменти от времето, когато всички преходни процеси са приключили и сигналът на изхода на комбинационната схема съответства на изпълняваната от нея функция. Това означава, че повечето цифрови схеми изискват сигнал за синхронизация (тактов сигнал). Всички преходни процеси в комбинационната логическа схема трябва да приключат в рамките на периода на синхронизиращия сигнал, подаван на входовете на тригерите. Тригерите, които запомнят входните сигнали само в момент от времето, определен от синхронизиращия сигнал, се наричат синхронни. Те се разделят на синхронни по ниво и синхронни по фронт. Принципната схема на синхронния по ниво RS-тригер е показана на фигура U3.3а и U3.3б.



Фиг. U3.3. *Синхронен по ниво RS тригер*

2.3 Тригер тип D

Тригерите от този тип имат един информационен вход D, като изходът Q повтаря състоянието на този вход след задействането на схемата. Това определя таблицата на истинност и логическата зависимост, описващи действието на тригера – таблица L3-3. От това описание следва, че D тригерът може да се разглежда като закъснителна линия със закъснение един такт.

Таблица. L3-3 Логическо описание на тригер тип D

D	Q(t)	Q(t+1)
0	0	0
0	1	0
1	0	1
1	1	1

	D	Q(t+1)
Q(t)	1	
	1	

Q (t+1) = D

Практическо приложение намират главно синхронните D тригери, тъй като асинхронната реализация на логическата зависимост се осъществява от повторител или два последователно свързани инвертора. Схемата на синхронен D тригер може да се получи от RS тригер фиг. U3.4. Преобразуване от RS в D тригер е показано в таблица L3-4

Таблица. L3-4 Преобразуване от RS в D тригер.

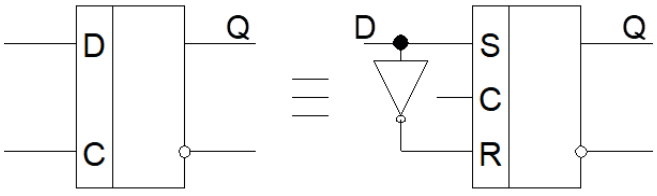
D	Q(t)	Q(t+1)	R	S
0	0	0	*	0
0	1	0	1	0
1	0	1	0	1
1	1	1	0	*

	D	R
Q(t)		1
		*

$R = \overline{D}$

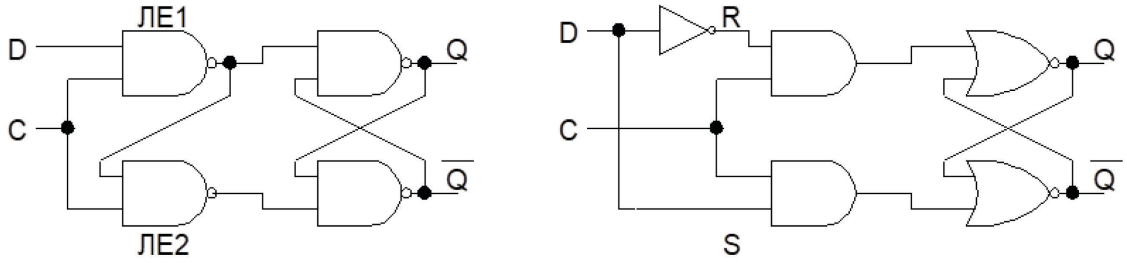
	D	S
Q(t)	*	
	1	

S = D



Фиг. U3.4. *Блокова схема на D- тригер*

От направеното преобразуване следва, че за получаване на D тригер могат да се използват известните структури на RS тригер, като входният сигнал D се свърже към вход S, а към вход R – чрез инвертор фиг. U3.5.



Фиг. U3.5. *Синхронен по ниво D-тригер*

2.4 Тригери тип T

При тези тригери има един информационен вход T, като сигнал T=0 не води до промяна на вътрешното състояние [Q(t+1) = Q(t)], а сигнал T=1 го превключва в противоположното. Това определя главното приложение на схемата – за броене на входни сигнали. Въпреки широкото приложение в цифровата техника този тригер не се произвежда в интегрално изпълнение, тъй като лесно се получава от останалите типове тригери. От принципа на действие следва таблицата на истинност и логическата зависимост – таблица L3-5. Преобразуване от D в T тригер, както и получената схема е показано на фиг. U3.6.

Таблица L3-5 Тригер тип T таблица на състоянията

T Q(t)	Q(t+1)
0 0	0
0 1	1
1 0	1
1 1	0

Q(t) \ T		1
1		

Q(t+1)

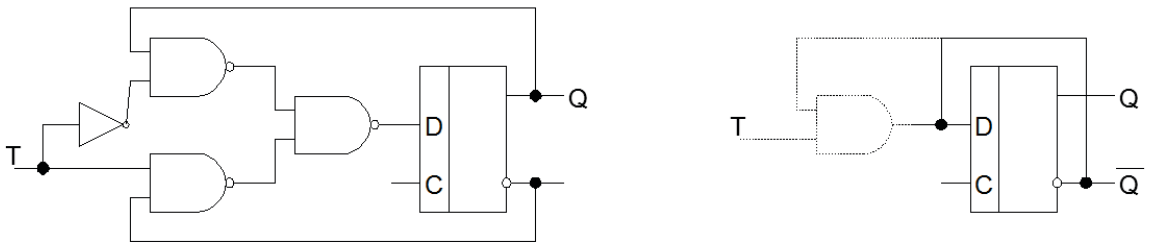
$Q(t+1)=\overline{T}\,Q(t)\ v\ T\overline{Q}(t)$

$Q(t+1)=T \oplus Q(t)$

T Q(t)	Q(t+1)	D
0 0	0	0
0 1	1	1
1 0	1	1
1 1	0	0

$D = \overline{T}\,Q(t)\ \ v\ \ T\overline{Q}(t)$

При $T = 1$ $\quad D = \overline{Q}(t)$



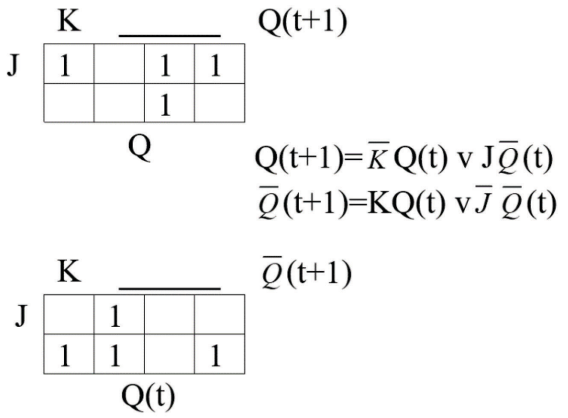
Фиг. U3.6. *Схема на T-тригер на базата на тригер тип D.*

2.5 Тригерите тип JK

Тригерите тип JK имат два информационни входа, от които J е еквивалентен на входа S от RS тригера /единичен вход/, а входът K - на R входа /нулиращ вход/. Разликата между двата тригера е, че при J = K = 1 се извършва превключване в противоположно състояние /Q(t+1) = $\overline{Q}$(t)/, докато R = S = 1 води до неопределеност /забранена комбинация/. Тогава таблицата на истинност и логическата зависимост, определящи схемното решение, се представят по следния начин – таблица L3-6:

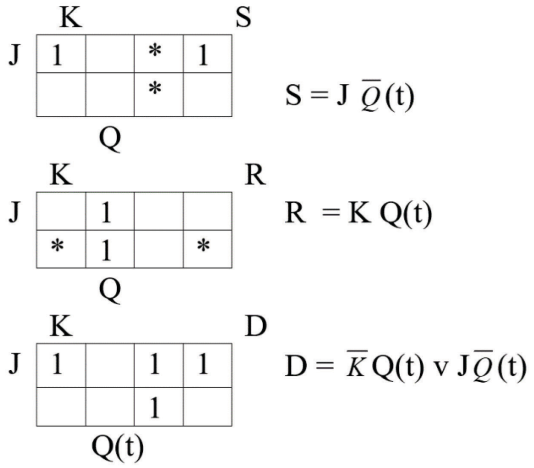
Таблица L3-6 Представяне на JK тригер

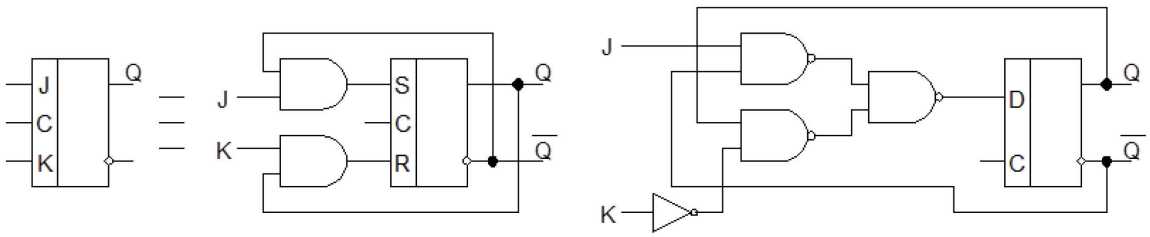
K	J	Q(t)	$\overline{Q}$(t)	Q(t+1)	$\overline{Q}$(t+1)
0	0	0	1	0	1
0	0	1	0	1	0
0	1	0	1	1	0
0	1	1	0	1	0
1	0	0	1	0	1
1	0	1	0	0	1
1	**1**	**0**	**1**	**1**	**0**
1	**1**	**1**	**0**	**0**	**1**



Ако се приложи алгоритъмът за преобразуване на един тип тригер в друг, от RS тригер и D тригер се получават следните схемни решения на JK тригер – фиг. U3.7.

K	J	Q(t)	Q(t+1)	R	S	D
0	0	0	0	*	0	0
0	0	1	1	0	*	1
0	1	0	1	0	1	1
0	1	1	1	0	*	1
1	0	0	0	*	0	0
1	0	1	0	1	0	0
1	1	0	1	0	1	1
1	1	1	0	1	0	0



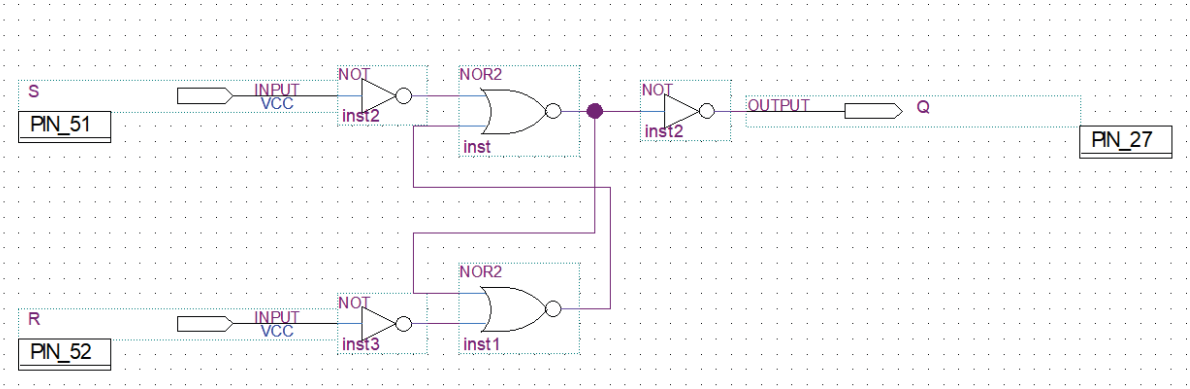


Фиг. U3.7. *преобразуване на RS и D тригер в JK*

3. Задачи за изпълнение

3.1 Изследване на асинхронен RS-тригер

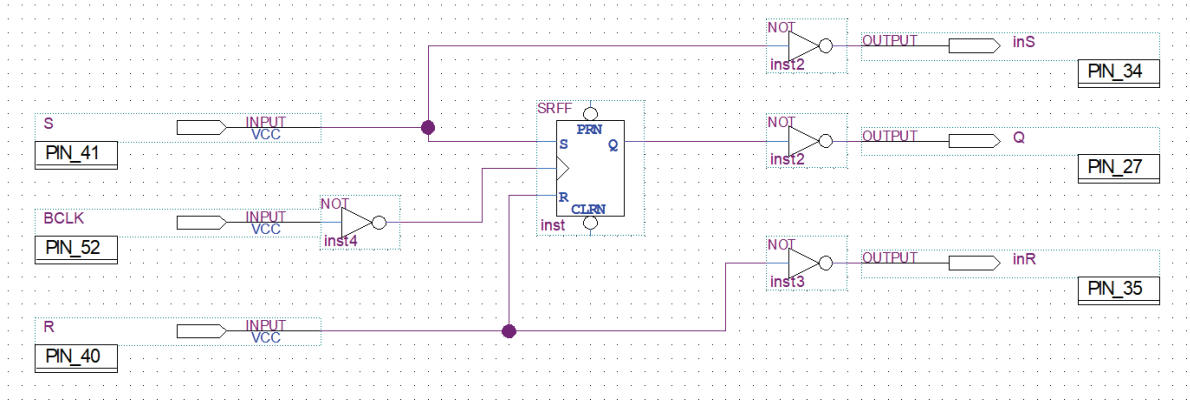
Конфигурирайте програмируемата логическа матрица в съответствие с фигура U3.8. Всички използвани елементи се намират в **Symbol tools -> Primitives**. Като задавате различни комбинации на логическите нива с помощта на бутоните SW6 и SW7 и наблюдавате светодиода LED-L0, попълнете експериментално, таблица на истинност.



Фиг. U3.8. *Изследване на асинхронен RS тригер*

3.2 Изследване на синхронен по фронт RS тригер

Конфигурирайте програмируемата логическа матрица в съответствие с фигура U3.9. Тригера се взима от библиотеката **Symbol tools -> Primitives**. Може да напишете името SRFF в прозореца за търсене и ще се появи автоматично.

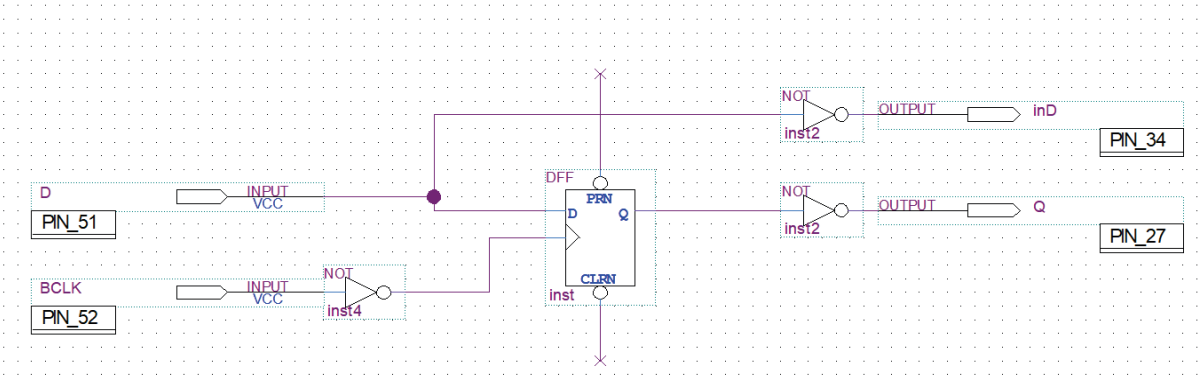


Фиг. U3.9. *Изследване на синхронен RS тригер*

Като задавате различни комбинации на логическите нива с помощта на плъзгащите бутони SWDIP4(1) и SWDIP4(2) и последствие натискате бутона SW7 (BCLK), наблюдавате светодиода LED-L0 (изхода Q). Обърнете внимание на преден или заден фронт се получава записа. Входните променливи могат да се наблюдават на LED-L6 и L7. Да се направи експеримент при постоянна лог.1 на входа BCLK и промяна на входовете RS ще има ли промяна на изхода Q.

3.3 Изследване на синхронен D-тригер

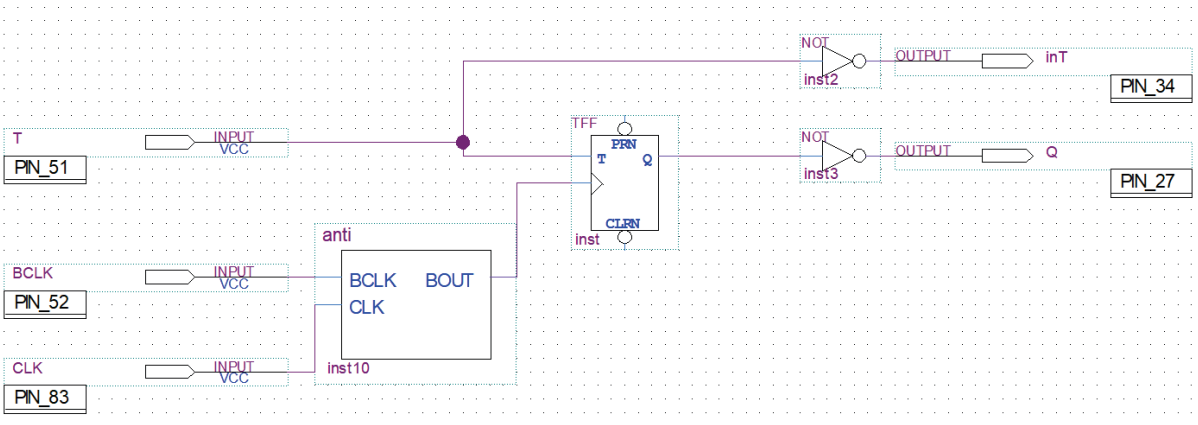
Конфигурирайте програмируемата логическа матрица в съответствие с фигура U3.10. Задавайте с помощта на бутона SW6 различни нива на входа, който се виждат на LED-L6. При натискане на бутона BCLK (SW7) ще се получи промяна на изхода Qна тригера. Променете схемата и добавете бутони към асинхронните входове PRN и CLRN, примерно SW0 и SW1. Изследвайте работата функциите на тези входове. Зависими ли са от тактовия сигнал BCLK?



Фиг. U310

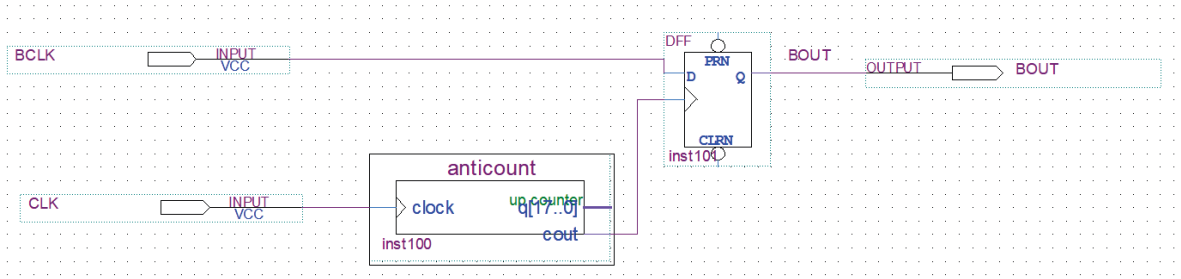
3.4. Изследване на Т-тригер

Конфигурирайте програмируемата логическа матрица в съответствие с фигура U3.11.



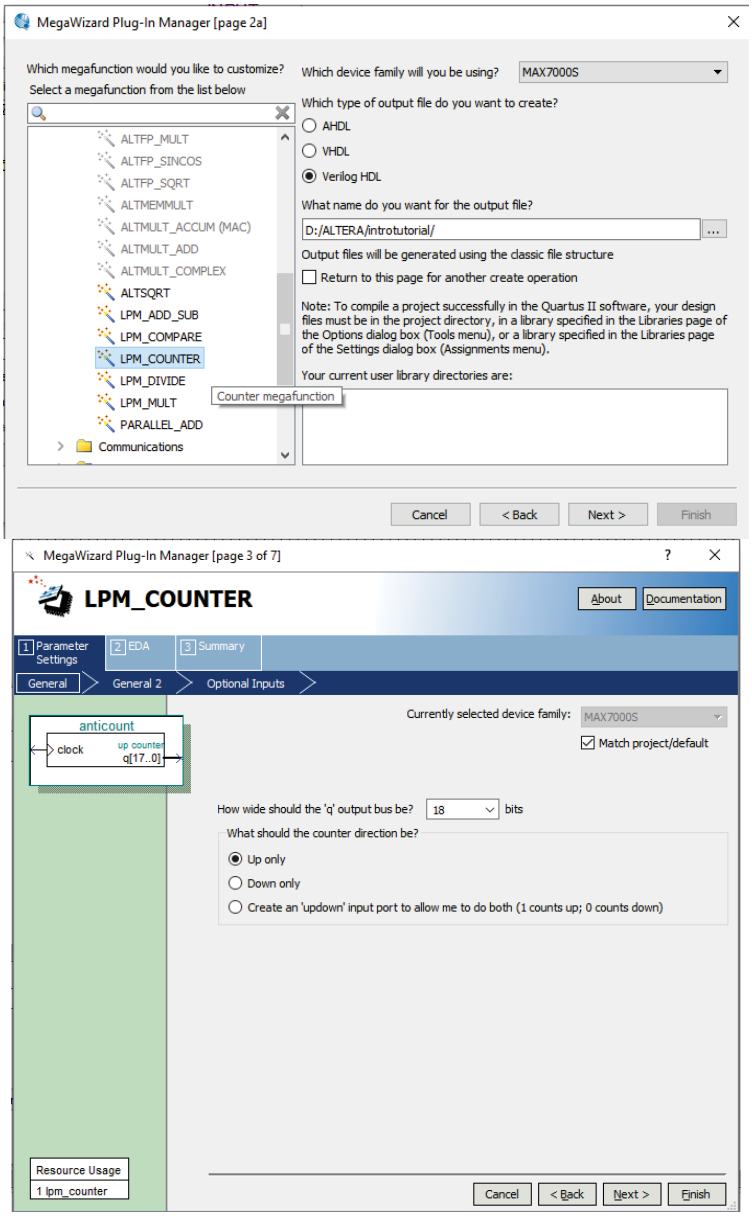
Фиг. U3.11

Тук новото е добавянето на схема за намаляване на паразитните импулси, получавани при натискане на бутон – блок anti. Вътрешната структура представлява намаляване на получения „шум“ на бутона, чрез редуциране на времето за превключване на синхронен D -тригер. Развойната платка разполага с 50MHz генератор, чиято честота разделяме на 2^{18} със създаден допълнително брояч - фиг. U3.12 . Важно “instance name” за двата елемента да са “inst100“ и “inst101“. Така за следващи проекти няма да съвпадат имената с други елементи.



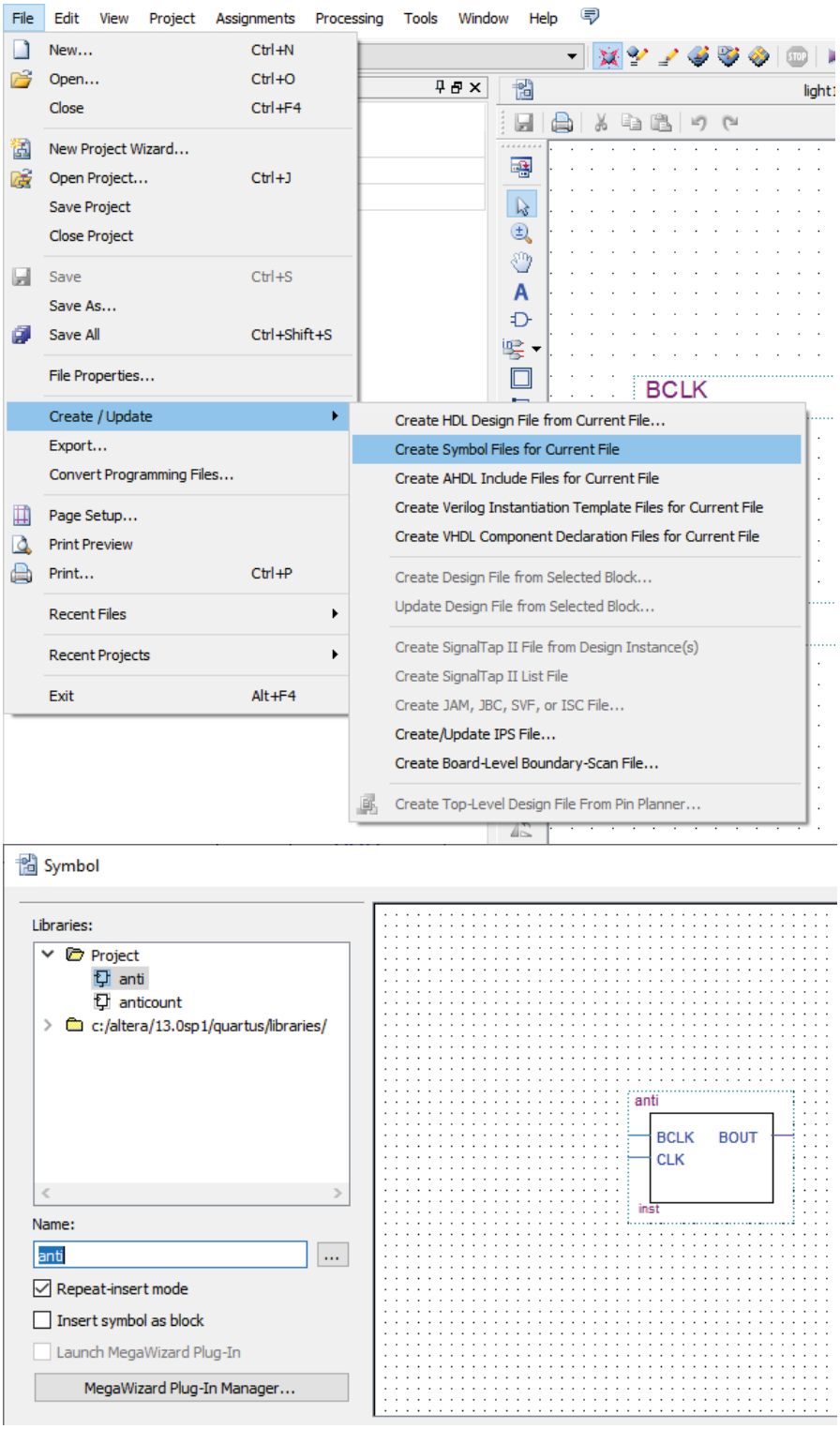
Фиг. U3.12.

За целта създаваме нов схематичен файл, записваме го със името „anti“. Използваме функцията “MegaWizard” от бибиотеката и от категория “Aritmetic” избираме lpm_counter фиг. U3.13. Необходимо е да бъде 18 бита за да се получи честота от изхода за пренос 200Hz. Изходът за пренос Cout се избира на следващия екран.



Фиг. U3.13

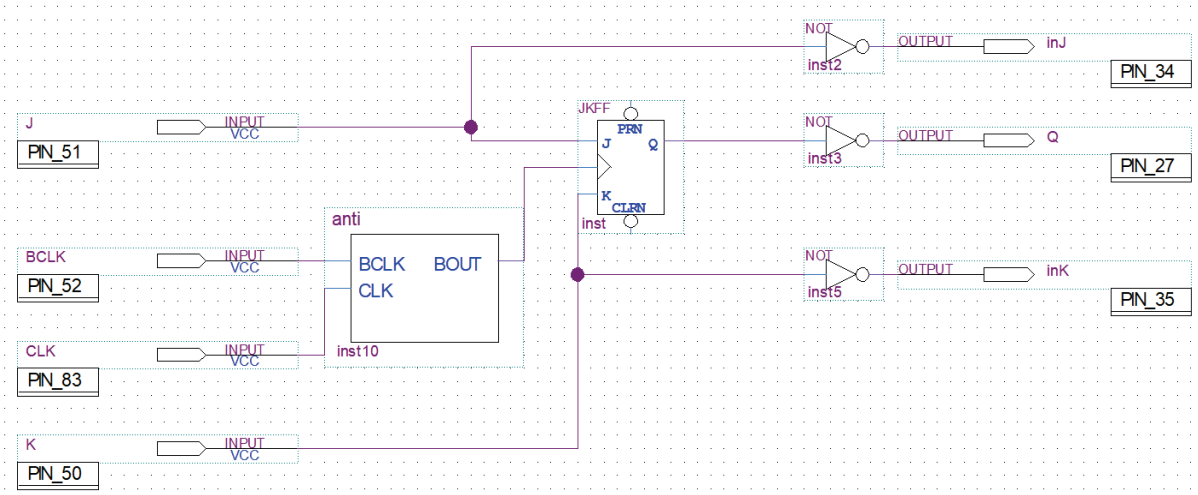
След създаване на схемата правим запис “Create Symbol file for current file” фиг. U3.14. Задаваме име, което после ще извикаме във проекта за изследване на T тригера. В случая сме задали отново име “anti”. Създадения обект (елемент) се извиква от библиотеката папка “Project”.



Фиг. U3.14

3.5 Изследване на синхронен JK-тригер

Конфигурирайте програмируемата логическа схема в съответствие с фигура U3.15.



Фиг. U3.15.

Свържете блока **Anti** по същия начин, както беше направено в предишното задание. Задавайте различни логически нива на входовете J и K с помощта на бутоните SW5 (K) и SW6(J), след това натиснете бутона **SW7 – тактовия сигнал**, за да направите превключване на изхода Q. Експерименталните резултати попълните в таблица на преходите.

4. Съдържание на протокола

1. Цел на работата.
2. Схеми за изследване на тригерите.
3. Представяне на условните графични обозначения на изследваните тригери и времедиаграми.
4. Таблица на преходите на изследваните тригери, логически зависимости.
5. Изводи.

5. Контролни въпроси

1. От какво се определя бързодействието на тригера?
2. Начертайте схема на RS-тригер с логически елементи „ИЛИ-НЕ“ и обяснете принципа на неговата работа.
3. Защо JK-тригерът се нарича универсален?
4. Обяснете работата на D-тригера по таблицата на преходите.
5. Каква характерна особеност притежава периодичната последователност от импулси на входа на T-тригера?
6. Методи за описание на последователни цифрови устройства.
7. Какво предимство има двустепенният master-slave тригер?

IV. Изследване на регистри

1. Цел на упражнението

Целта на работата е да се изучи принципът на действие на схеми с тригерни регистри и да се придобият практически умения в изпълнението на микрооперации върху регистри в статичен и динамичен режим.

2. Кратки теоретични сведения

Регистрите са предназначени за съхранение и преобразуване на многоразрядни двоични числа. За запаметяване на отделните разреди на числото могат да се използват тригери от различни типове. Единичният тригер може да се разглежда като едноразреден регистър. Въвеждането на информация в регистъра се нарича операция на запис. Операцията за извеждане на информация от регистъра – четене

Преди записа на информация в регистъра, той трябва да бъде нулиран.

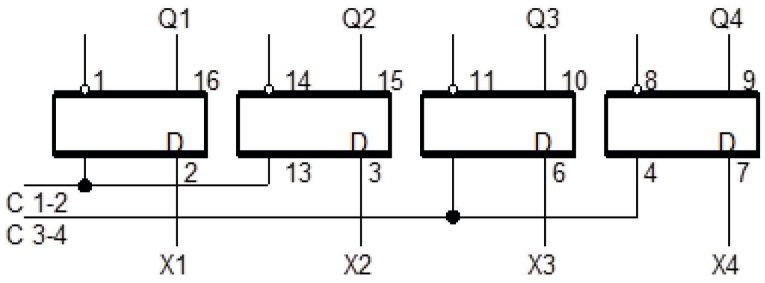
Класификация на регистрите:

1. **Според начина на въвеждане/извеждане на информация:**
   - **Паралелни (регистри за съхранение):** Информацията се въвежда и извежда едновременно по всички разреди.
   - **Последователни (регистри с отместване):** Информацията се предава бит по бит през регистъра и се извежда също последователно.
   - **Комбинирани:** Паралелен вход и последователен изход (и обратно).
2. **Според начина на представяне на информацията:**
   - **Еднофазни:** Информацията се представя в едно от двете, пряк или обратен (инверсен) вид.
   - **Парафазни:** Информацията се представя както в пряк, така и в обратен вид.

2.1Паралелен регистър

Паралелните регистри осъществяват приемане и извеждане на информация в паралелен код, което означава, че за предаването на всеки разред се използва отделна линия.

Голямо приложение намират регистрите, реализирани чрез D тригери. Такъв е примерно 7475 /фиг. U4.1/, записът в който се извършва при разрешаващи сигнали C1-2 = C3-4 = 1 – синхронизация по ниво. Тогава вътрешното състояние следва /повтаря/ подадените входни сигнали X. След това на изходите Q1-Q4 ще се появи записаната дума. От схемата са изведени правите и инверсни изходи на тригерите. Обикновено двата разрешаващи сигнала се свързват накъсо. Съществува голямо разнообразие от паралелни регистри от този вид – примерно 74373, имащ осем D тригера с общ разрешаващ сигнал и възможност за установяване в трето високоимпедансно състояние на изходите.



Фиг. U4.1. *Четириразряден паралелен регистър 7475*

При някои разработки съществува възможност за индивидуален достъп до отделните разряди /ИС 74259/, като изборът на определен разряд се осъществява посредством дешифратор. В този случай има само един D вход, общ за всички разряди.

Голяма група паралелни регистри се управляват по фронт на тактов сигнал, по който се извършва запис. Такива са ИС74175 /четири D тригера с общо асинхронно нулиране/, ИС74273 – осем D тригера с общо асинхронно нулиране и запис по фронт на тактов сигнал 0-1, и др.

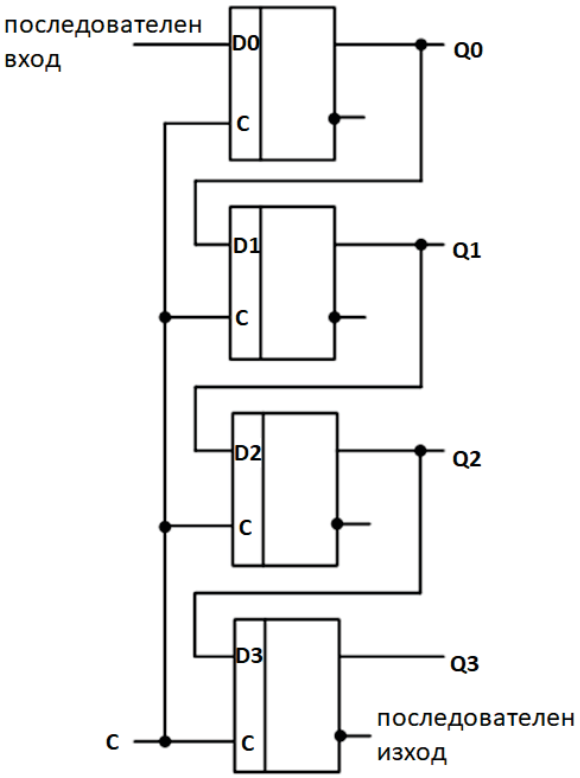
Приложението на паралелните регистри обикновено е свързано с реализиране на запомнящи устройства с произволен достъп, предназначени за запис и четене на големи информационни масиви, с използване като буферна памет на входа и изхода на редица цифрови устройства и схеми за преобразуване на информацията.

2.1 Последователни регистри

Освен паралелното свързване на тригерите за изграждане на регистри се използва и последователното свързване на тези елементи.

Последователният регистър обикновено служи за преобразуване на последователен код в паралелен и обратно. Тези регистри са синхронни схеми и за реализацията им се използват предимно D или JK тригери. Използването на последователен код е свързано с необходимостта от предаване на голямо количество двоична информация по ограничен брой свързващи линии. При паралелно предаване на разредите е необходимо голямо количество свързващи проводници. Ако двоичните разреди се предават последователно, бит по бит, по един проводник, може значително да се намалят размерите на свързващите линии на платката (и размерите на корпусите на микросхемите). Съвременната компютърна техника използва последователно предаване на данни. Примерната схема на четириразряден преместващ регистър е показана на фиг. U4.2.

Нека разгледаме работата на този регистър. Може да се предположи, че в началото всички тригери на регистъра са в състояние на логическа нула, т.е. Q0=0, Q1=0, Q2=0, Q3=0. Ако на входа на D0-тригера има логическа 0, тогава подаването на синхронен импулс на входа "C" на тригерите не променя тяхното състояние.



Фиг. U4.2.

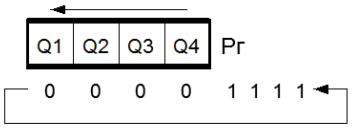
При подаване на "1" на вход "D0" на първия тригер, с идването на първия синхронен импулс в този тригер ще се запише "1", а в останалите тригери – "0", тъй като към момента на постъпване на фронта на импулса на входовете D1, D2 и D3 има „0“.

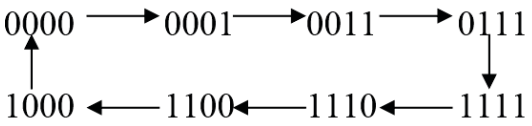
При постъпването на втория импулс на входа „C“ логическата "1" от изхода на първия тригер ще се запише във втория тригер и в резултат се извършва изместване на първоначално записаната "1" от тригера D0 към тригера D1 и т.н. Така се осъществява последователно изместване на постъпващата на входа на регистъра информация (в последователен код) с един разред надясно при всеки такт на импулсите.

След постъпването на четири синхронни импулса регистърът се оказва напълно запълнен. В течение на следващите четири тактови импулса се извършва последователно поразрядно извеждане от регистъра на записаното число, след което регистърът се оказва напълно изчистен (регистърът ще бъде напълно изчистен само при условие, че на неговия вход се подава ниво "0" в режим на извеждане на записаното число).

Чрез преместващи регистри могат да се реализират броячни схеми с различен брой вътрешни състояния. Примерно три разряден брояч до пет или 4-ри разряден кръгов брояч. Подходът при синтез на четириразряден брояч в код на Джонсън е показан на фиг. U4.3. В този случай се получават осем работни и осем неработни състояние.

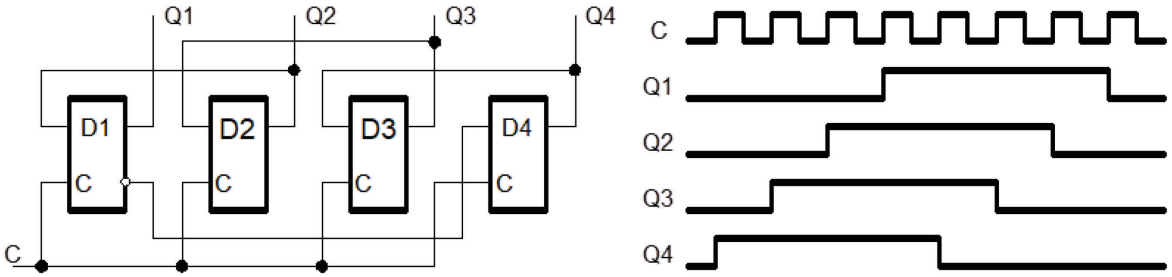
Таблица L4.1





$Q_1(t)$	$Q_2(t)$	$Q_3(t)$	$Q_4(t)$	$Q_1(t+1)$	$Q_2(t+1)$	$Q_3(t+1)$	$Q_4(t+1)$	D_1	D_2	D_3	D_4
0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	1	0	0	1	1	0	0	1	1
0	0	1	1	0	1	1	1	0	1	1	1
0	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	0	1	1	1	0
1	1	1	0	1	1	0	0	1	1	0	0
1	1	0	0	1	0	0	0	1	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0

$$D_1 = Q_2;\ D_2 = Q_3;\ D_3 = Q_4;\ D_4 = \overline{Q}_1.$$



Фиг. U4.3

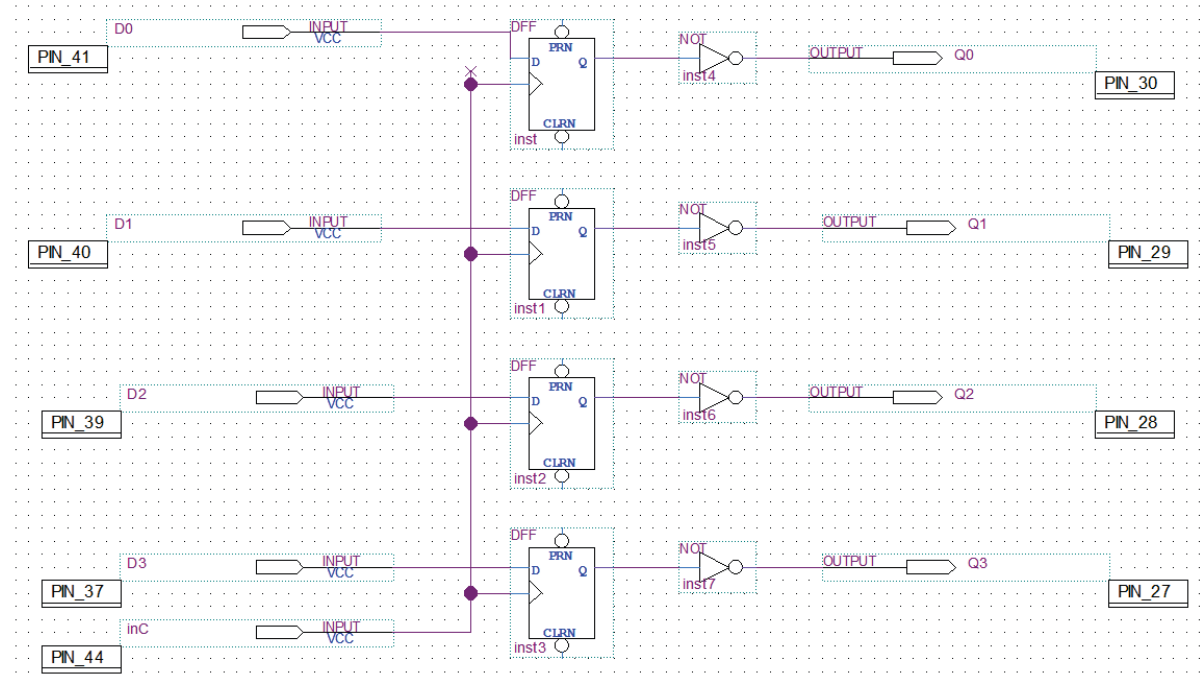
От анализът на схемното решение се вижда, че при случайно попадане в неработно /забранено/ състояние се получава граф на преходите, съдържащ само такива състояния. Примерно от неработното състояние 0010 се преминава към 0101 /също неработно/, после следва 1011 и т.н.
В този случай е необходимо или принудително установяване на схемата в едно от работните състояния /примерно нулевото/, или промяна на възбудителните функции така, че да се реализира автоматичен преход към разрешено състояние.

3. Задачи за изпълнение

3.1 Изследване на паралелен регистър.

Конфигурирайте програмируемата логическа матрица в съответствие с фигура U4.4. Всички използвани елементи се намират в **Symbol tools -> Primitives**. Като задавате различни комбинации (четни числа от 0-15) с помощта на плъзгащите бутони SWDIP4(1) до SWDIP4(4) и правите запис с

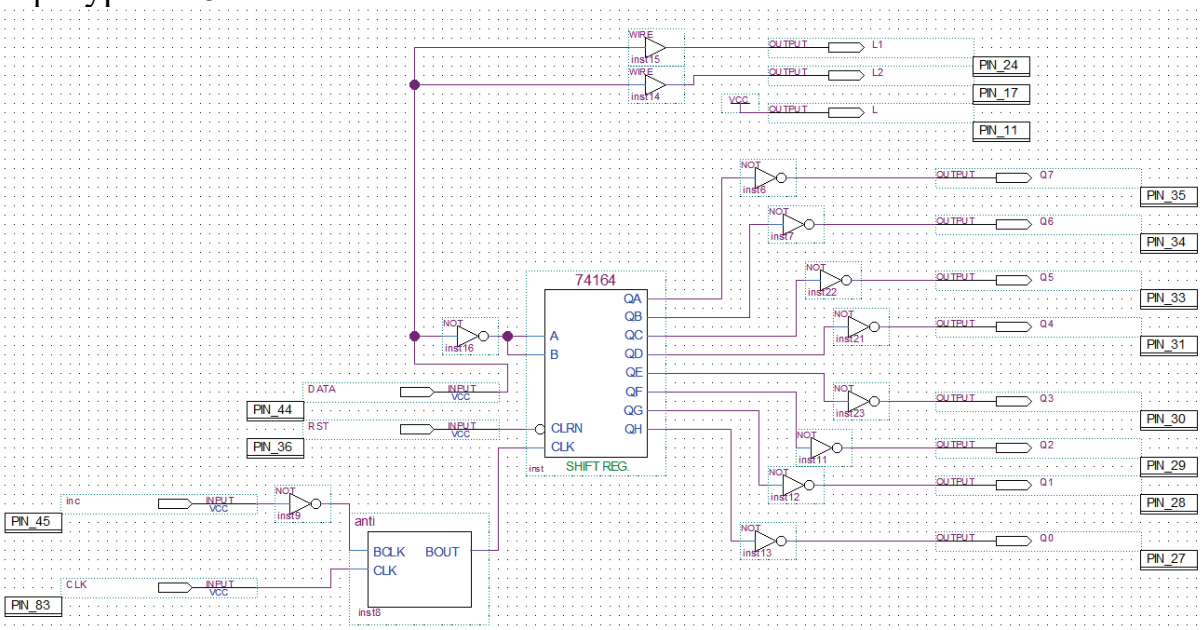
бутон SW0, наблюдавате светодиодите LED-L0 до LED-L3. Попълнете експериментално таблица на всяко входно десетично число кое двоично съответства.



U4.4

3.2 Изследване на последователно-паралелен регистър

Конфигурирайте програмируемата логическа матрица в съответствие с фигура U4.5.



Фиг. U4.5

Запишете три нечетни числа в интервала от 42 до 76 в последователен код, като поразрядно го премествате чрез натискане на бутона inC. Запишете

резултата. Състоянието на паралелния вход DATA се показва на седемсегметен индикатор. Когато бутона е натиснат се изписва числото 1. Модула за борба с нежеланото превключване на бутона inC е разгледан в предходно упражнение. Изпълнява се по същия начин. Да се разгледа вътрешната структура на 74164. Да се обърне внимание на първоначалното състояние на регистъра след програмиране.

3.3 Синтез и изследване на кръгов брояч

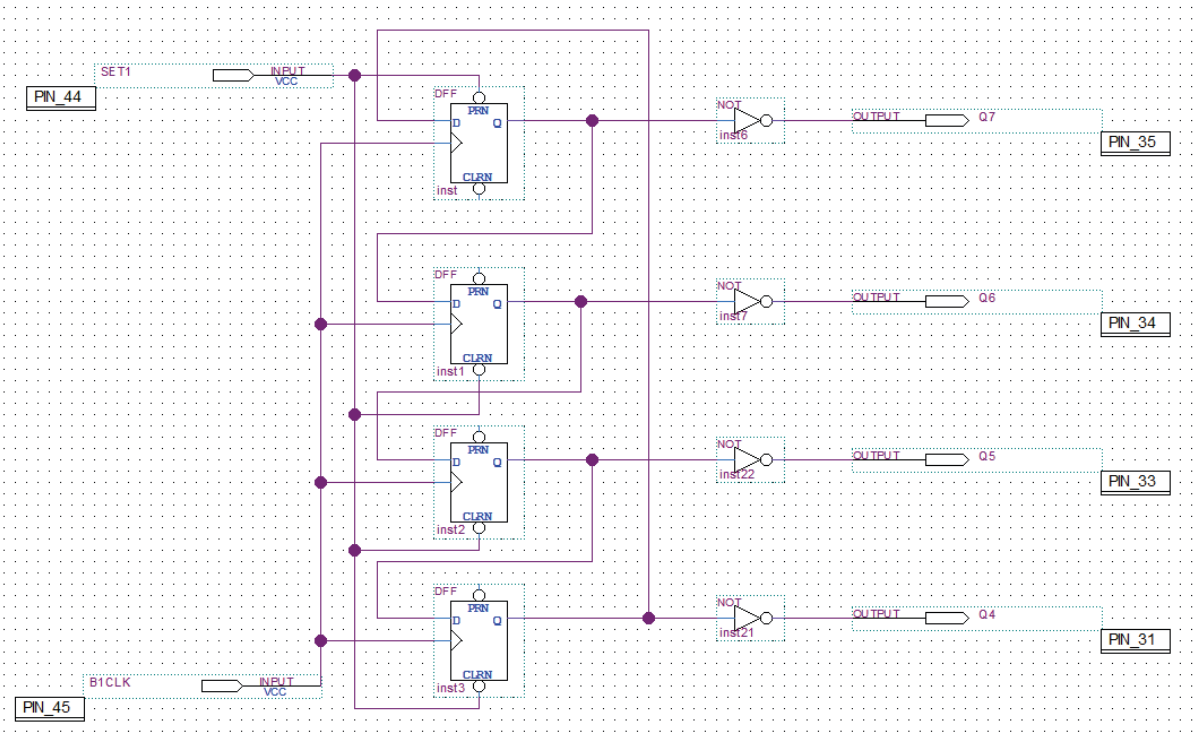
Конфигурирайте програмируемата логическа матрица в съответствие с фигура U4.6. Наименованието на схемата идва от „въртенето“ на една единица наляво или надясно. Това е непълен брояч, което предполага за нежеланата възможност да попадне в броене на грешни числа (грешни комбинации). Схемата може да бъде реализирана с D-тригери или с някой от последователно-паралелните регистри от серията 74xx. Логическите уравнения се получават от таблицата на истинност - таблица L4-2. Да се попълни таблицата и изведат логическите зависимости. Да се синтезира схема за самоконтрол, като се използва карта на Вейч. Да се реализира два 4-ри разрядни изхода, които да преместват изходните числа противоположно.

Таблица L4-2

Q1	Q2	Q3	Q4	Q1(t+1)	Q2(t+1)	Q3(t+1)	Q4(t+1)	D1	D2	D3	D4
0	**0**	**0**	**1**								
0	**0**	**1**	**0**								
0	**1**	**0**	**0**								
1	**0**	**0**	**0**								

D1=Qn, D2=…, D2=…, D4=…,

Важно е да се отбележи че първоначалното състояние на тригерите е или 0 или са установени в 1 (след програмиране на CPLD се установяват в 1). Това налага задаване и установяване в 1 на един от тригерите. В реализираната схема, синхроните входове PRN и CLRN на D-тригерите са свързани към бутон SW0 за да се зададе числото 0001. Чрез бутон SW1 се подават тактовите импулси. Проблема с нежеланото генериране на импулси от бутона при натискане се решава по предложения начин по-горе.



Фиг. U4.6.

4. Съдържание на протокола

1. Цел на работата.
2. Схема на последователно-паралелен регистър с резултати от изследванията.
3. Схема на изследване на 4-битов кръгов брояч с D-тригери. Схема за самоконтрол.
4. Схема на изследване на 3-битов брояч до 5 с D-тригери.
5. Изводи за всяко задание.

5. Контролни въпроси

1. Предназначение на регистрите.
2. По какви признаци се класифицират регистрите?
3. От какво се определя разрядността на регистрите?
4. Предназначение и принципа на работа на паралелния регистър.
5. Обяснете принципа на работа на последователния регистър.
6. Обяснете принципа на работа на последователно-паралелния регистър.
7. Обяснете принципа на работа на паралелно-последователния регистър 74166.

V. Изследване на броячни схеми

1. Цел на упражнението

Целта на работата е изучаване на асинхронни и синхрони двоични броячи и придобиване на умения в изграждането и експерименталното изследване на броячи.

2. Кратки теоретични сведения

При ПС от броячен тип вътрешните състояния зависят от броя на постъпилите на входа сигнали. Резултатът от броенето се фиксира в определен двоичен код. Броят на възможните вътрешни състояния се определя като **коефициент** или **модул на броене К**. Той зависи от броя на ЕА, от които е изграден броячът, и от начина им на свързване. Ако броят на ЕА /разрядите/ е n, то **пълният брояч** има $К = 2^n$. При $К < 2^n$ броячът се определя като **непълен /частичен**/.Най-често използваните непълни броячи са с К = 10 – **десетични** броячи.

Към основните характеристики на броячните схеми се отнасят:

-капацитет – броят на вътрешните състояния К;

-разрешаваща способност – минималният период на следване на входните сигнали, при който броячът работи надеждно;

-минимална продължителност на тактовия сигнал;

-време за регистрация – времето от постъпване на поредния входен сигнал до завършване на най-дългия възможен преходен процес;

-апаратни разходи и др.

Класификация:

По посока на броене:

- сумиращи;
- изваждащи;
- реверсивни;

По начин на изграждане на веригата за пренос:

- с последователен пренос;
- с паралелен пренос;
- с комбиниран пренос;

По начин на превключване на тригера:

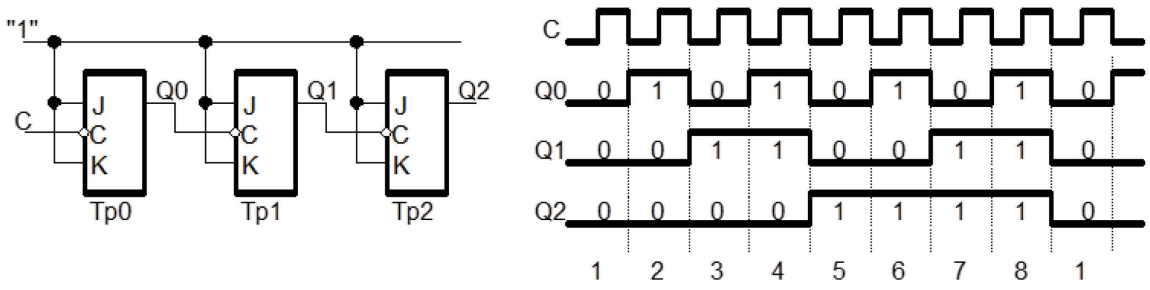
- синхронни;
- асинхронни.

2.1 Асинхронни броячи. Това са броячите с най-проста структура, в която всеки следващ тригер е свързан с правия /или инверсен/ изход на предния и се превключва по неговия преход 1-0. Очевидно в този случай тригерите няма да се превключват едновременно, а последователно – организиран е последователен пренос – фиг. U5.1. Ако началното състояние е $Q_2\ Q_1\ Q_0 = 000$ и JK входовете са свързани към единица, от времедиаграмата се

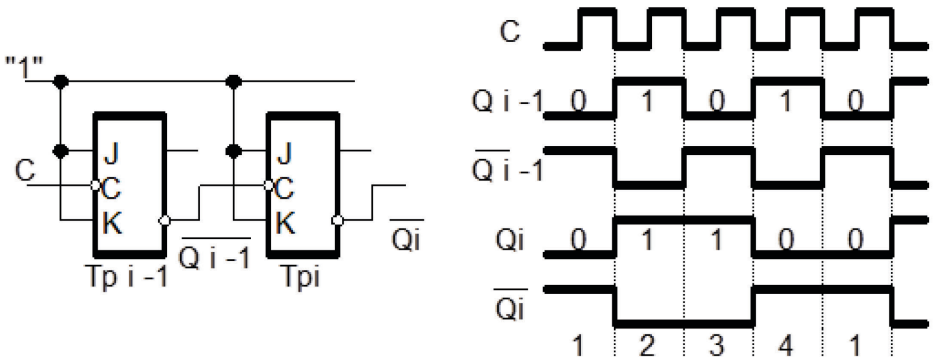
вижда, че схемата ще работи като пълен сумиращ брояч, имащ осем вътрешни състояния. Разгледаното схемно решение може да се разшири до n разряда.

Свързването на тактовия вход C към инверсния изход на предния тригер води до схемно решение на изваждащ брояч – фиг. U5.2.

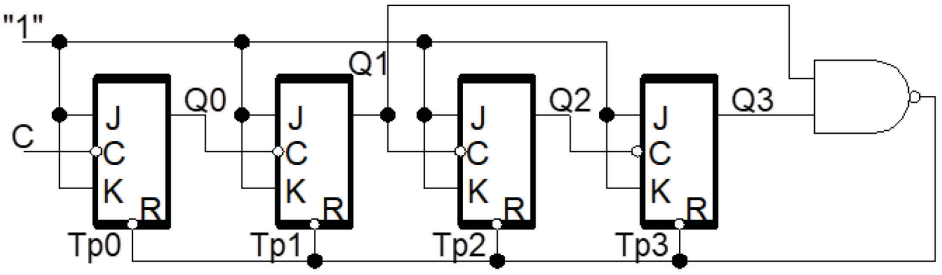
За получаване на непълни асинхронни броячи се използват нулиращите входове на тригерите, към които се свързват дешифриращите сигнали на определени вътрешни състояние, които определят коефициента на броене. На фиг. U5.3 е показана схемата на брояч до 10, при която дешифрираното вътрешно състояние, от което се взема обратна връзка към нулиращия вход на тригерите, е 1010. При това състояние на изхода на ЛЕ И-НЕ се получава нула, нулират се тригерите и започва нов цикъл на броене. При пренебрегване на състоянието 1010 и свързаното с него време за нулиране, броячът има десет състояния – от 0000 до 1001. По аналогичен начин се получават и други коефициенти на броене.

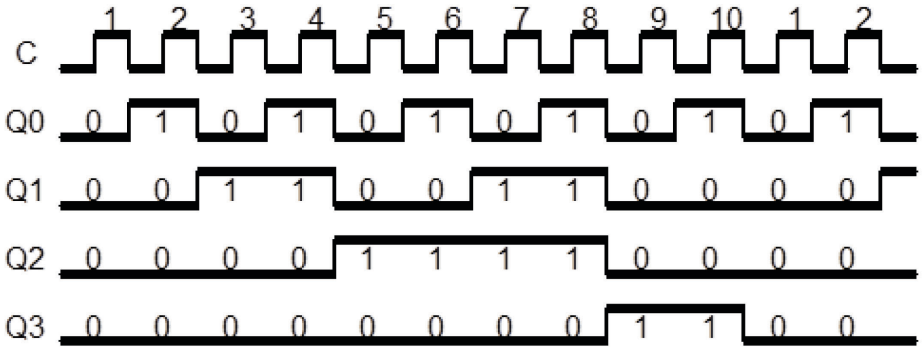


Фиг. U5.1. *Триразряден асинхронен пълен сумиращ брояч*



Фиг. U5.2. *Представяне на два разряда от асинхронен изваждащ брояч*





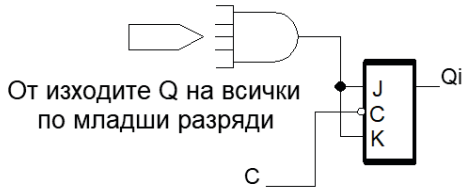
N	0	1	2	3	4	5	6	7	8	9	10
Q_3	0	0	0	0	0	0	0	0	1	1	1
Q_2	0	0	0	0	1	1	1	1	0	0	0
Q_1	0	0	1	1	0	0	1	1	0	0	1
Q_0	0	1	0	1	0	1	0	1	0	1	0

Фиг. U5.3. *Асинхронен сумиращ брояч до десет*

2.2 Синхронни броячи. При тези схеми всички тригери се превключват по фронт на синхронизиращ сигнал, което значително намалява времето за установяване. Синтезът им не се различава принципно от структурния синтез на ПС. Към кодираната таблица на преходите, отразяваща програмата на работа на синтезираната схема, се добавят колони, съответстващи на входовете на използваните тригери. В резултат на синтеза се получава схема с **паралелен /едновременен/ пренос**, имаща най-голямо бързодействие. При нея възбудителната функция на входа на всеки тригер е логическо произведение на сигналите от правите изходи на всички по-младши тригери – фиг. U5.4.

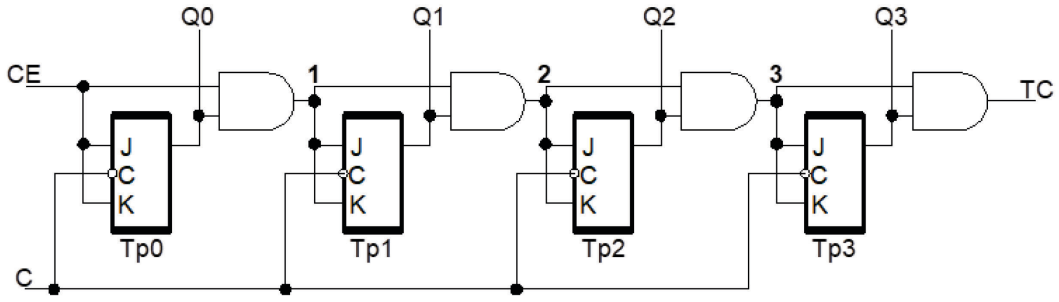
$Q_3(t)$ $Q_2(t)$ $Q_1(t)$ $Q_0(t)$	$Q_3(t+1)$ $Q_2(t+1)$ $Q_1(t+1)$ $Q_0(t+1)$	K_3J_3	K_2J_2	K_1J_1	K_0J_0
0 0 0 0	0 0 0 1	* 0	* 0	* 0	* 1
0 0 0 1	0 0 1 0	* 0	* 0	* 1	1 *
----------	----------	----	----	----	----
1 1 1 1	0 0 0 0	1 *	1 *	1 *	1 *

$$Ki = Ji = \overset{i-1}{\underset{m=0}{\wedge}} Qm \ , (i = 1,2, \dots , n)$$



Фиг. U5.4. *Синтез на пълен четириразряден сумиращ синхронен брояч*

Компромисна организация между бързодействие и апаратни разходи се получава при броячи с **ускорен пренос**. В този случай използваните във веригата за пренос ЛЕ са само двувходови, като се реализира логическата зависимост Ki = Ji = Qi-1. (K/Ji-1) – фиг. U5.5.



Фиг.U5.5. *Синхронен сумиращ брояч с ускорен пренос*

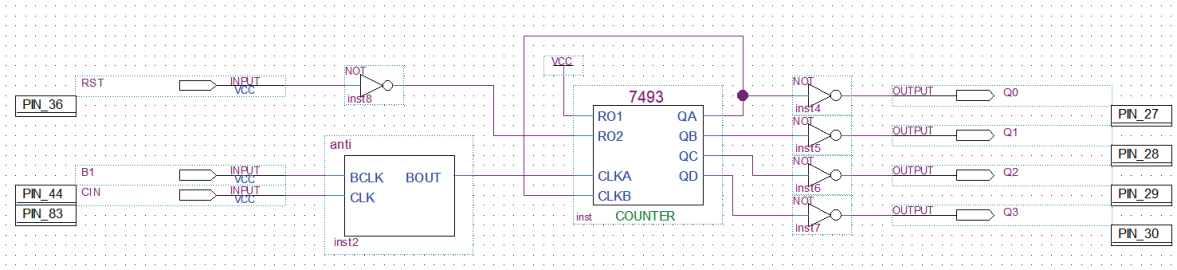
Характерни особености на схемата са:

- броячният вход е тактовия вход C;
- входът CE /Count Enable/ дава общото разрешение за броене – CE = 1. Ако CE = 0 на всички JK входове има нула;
- изходът TC /Terminal Count/ се използва за връзка със следващи броячни схеми.

3. Задачи за изпълнение.

3.1Изследване на асинхронен сумиращ брояч.

Конфигурирайте програмируемата логическа матрица в съответствие с фигура U5.6. Запознайте се със справочните данни за работата на брояча 7493. Защо има два тактови входа? Как може да бъде нулиран?



Фиг. U5.6

Като подавате импулси към входа на брояча с помощта на бутона SW0 и наблюдавате изходите Q, попълнете таблица L5-1.

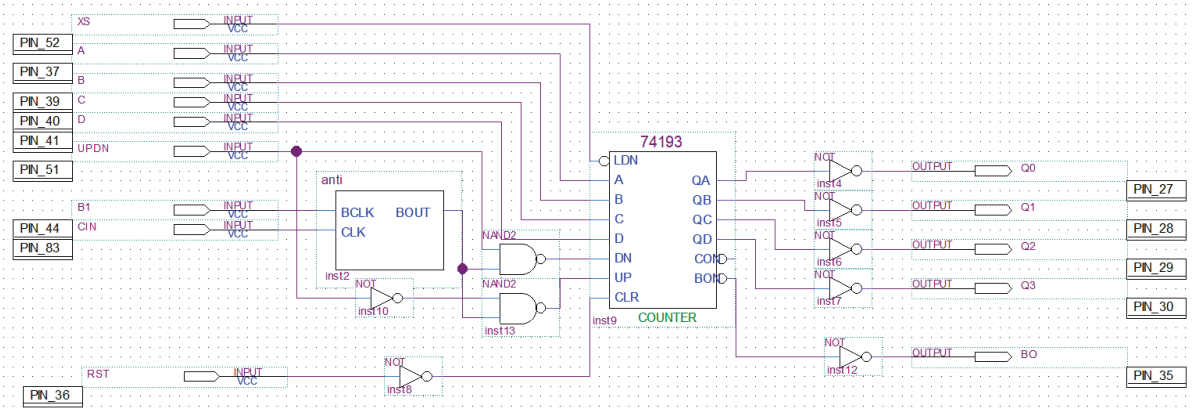
Таблица L5-1.

Номер на вхо-ден импулс	Q3	Q2	Q1	Q0
0				
1				
.				
15				

Променете схемата, като добавите принудително нулиране. Получава се брояч със произволен модул на броене. Експериментирайте примерно с получаване на брояч до 7 или до 10. Добавете ИС74247 и свържете изходите на седем-сегментен индикатор, както е показано в упражнеие-2.

3.2 Изследване на синхронен реверсивен брояч.

Конфигурирайте програмируемата логическа матрица в съответствие с фигура U5.7. Запознайте се със справочните данни за работата на брояча 74193. При какви нива на тактовите входове става управлението на посо-ката? Изследвайте кога изход “BO” ще превключи в лог.0, какво е неговото предназначение? Задайте число със превключватели SWDIP(1-4), направете запис със SW0. Какво приложение може да намери в този случай броячът?



Фиг. U5.7

4 Съдържание на протокола

Цел на работата.
Схема за изследване на сумиращ брояч с таблица на състоянията.
Времедиаграма на входните и изходните импулси на сумиращия брояч.
Схема за изследване на реверсивен брояч с таблица на състоянията.
Времедиаграма на входните и изходните импулси при запис.
Схема за изследване на брояч с произволен модул на броене.
Изводи по всяко задание.

5 Контролни въпроси

1. Обяснете принципа на работа на сумиращия брояч.
2. Изобразете времедиаграма на работата на сумиращия брояч.
3. Обяснете принципа на работа на изваждащия брояч.
4. Изобразете временните диаграми на работата на изваждащия брояч.
5. Обяснете принципа на работа на брояча с произволен модул на броене.
6. Разкажете за класификацията на броячите.
7. Къде се използват броячи?

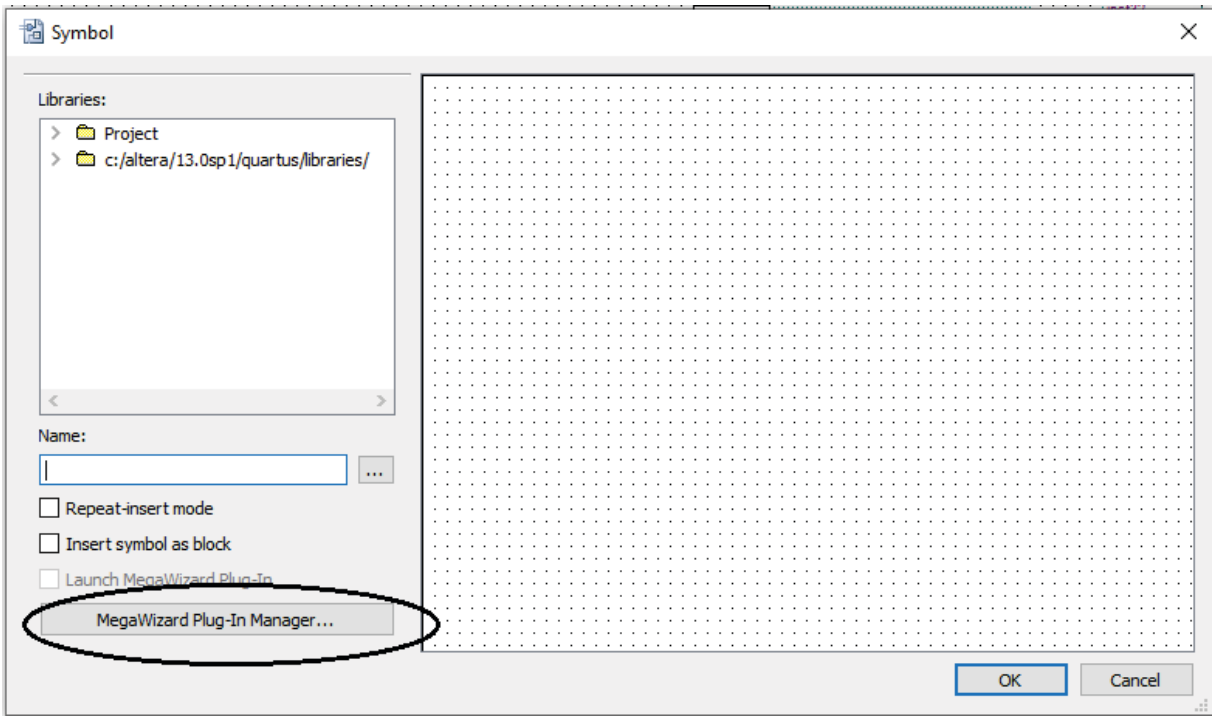
4. Примерни проекти

4.1. Проектиране на цифров часовник с динамична индикация

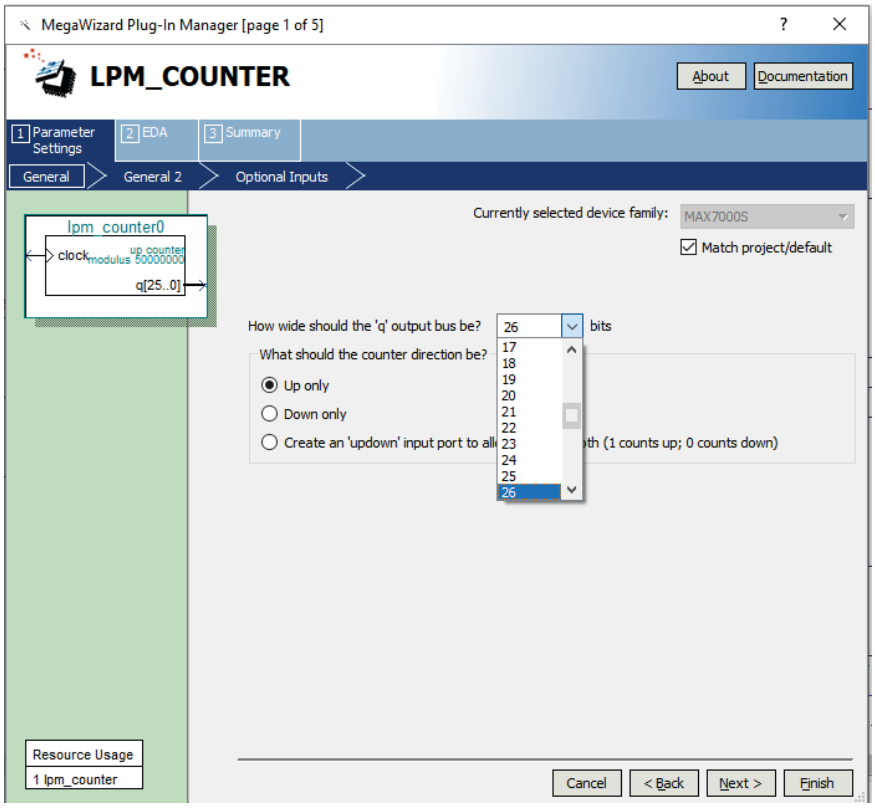
4.1.1 Делител на честота

Софтуера позволява логическата схема да бъде въведена чрез графичен редактор, което улеснява проектирането. Една от функциите е създаване на специфични и индивидуални цифрови блокове необходими според нуждите на цифровия дизайн. Както всеки подобен софтуер, тук има библиотека с основни логически елементи и макроси, отговарящи на интегрални схеми от серията 7400. Това също способства за по-добро проектиране, чрез възможността да се използват справочните данни за принцип на действие на всички логически елементи от тази серия.

За полочване на 1Hz честота необходима за часовника се използва създаването на делител (брояч) на 50 000 000. Това е така, защото имаме генератор с 50MHz тактова честота. Стартираме прозореца на библиотеката от бутона, приличащ на логически елемент И или чрез два пъти щракване на празно поле – фиг. 4.1. Избираме опцията megafunction>aritmetic и оттам lpm_counter. Този макрос има много възможности, като асинхронно установяване в лог.1 или нулиране, избиране на посока на броене, запис на входно число и т.н.



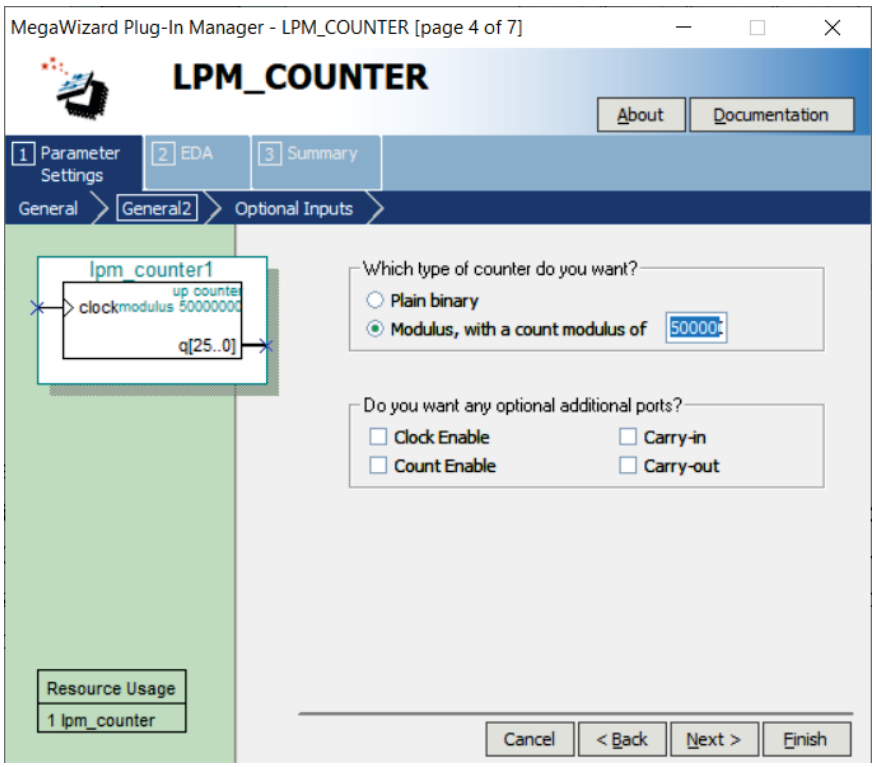
Фиг. 4.1. *Делител на честота*



Фиг. 4.2. *Избор на брой разряди.*

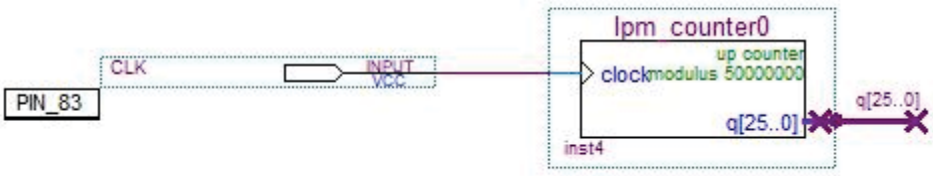
Важна особеност е задаването на коефизиента на делене. Броячът е 26 бита, (избира се от меню показано на фиг. 4.2) което позволява бит с

номер 25 да дели на 2^{26} , което е 67 108 864. Това число ще направи изходна честота по-малка от 1Hz, а именно 0.74505 Hz. За това се задава коефициент на делене, показан на фиг. 4.3. Така на изход q[25] ще получим честота 1Hz.



Фиг. 4.3. *Задаване на коефициент 50 000 000*

След натискане на бутон „финиш“ се получава макроса показан на фиг.4.4. Към него е присвоен вход с номер съответстващ на входа на основната тактова честота.

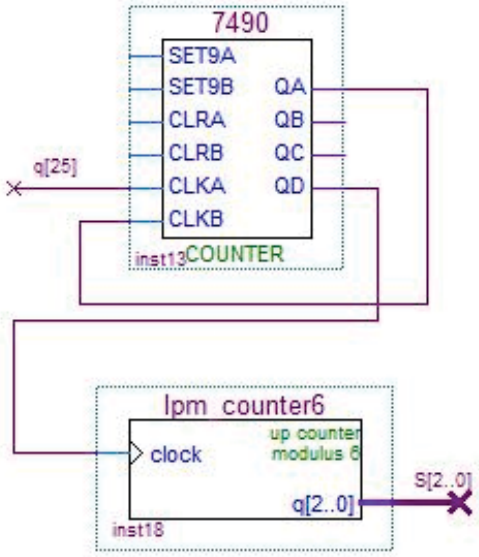


Фиг. 4.4. *Брояч с нулиране при 50 000 000*

4.1.2 Създаване на схема за отчитане на 60 секунди.

На развойната система има запоени четири седем-сегментни дисплеи и можем да показваме часове и минути във формат 23:40. От библиотеката отново е използвана функцията lpm_counter за създаване на делител на 6 и макрос 7490, което е брояч до 10. Общия коефициент на делене е 60 , което

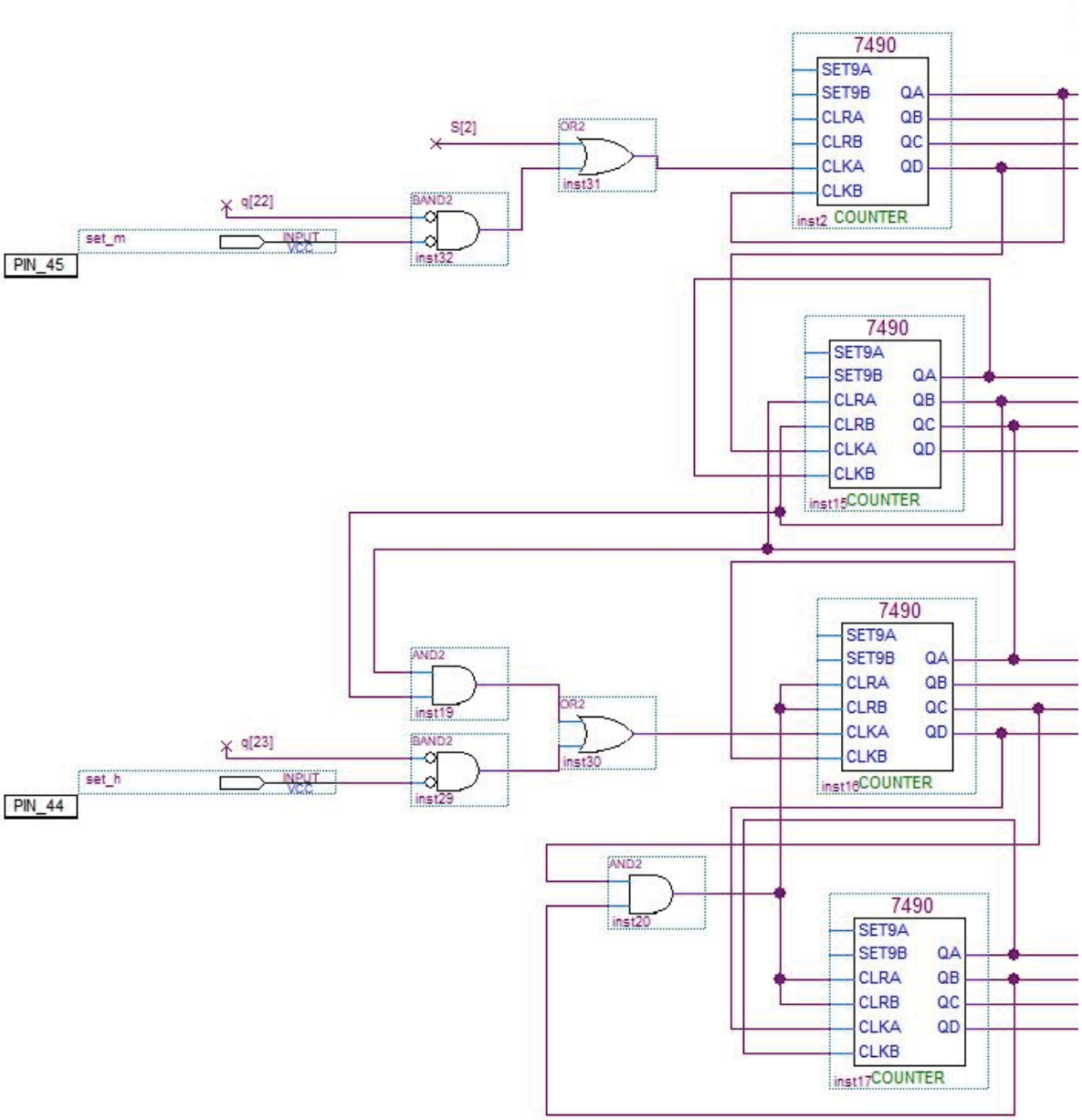
дава отброяването на 60 секунди (фиг. 4.5). Така че изходът на последния старши бит на lpm_counter6 ще сменя състоянието си на всяка минута. Полученият сигнал s[2] трябва да управлява десетичен брояч за „минути-единици“. При преход от 9-0 на единиците ще се увелича „минути – десетици“ (следващия брояч). Имаме да отброим 60 min, което означава броячът за десетици се нулира при преход от 5-6 и увеличава брояч за часовете.



Фиг. 4.5

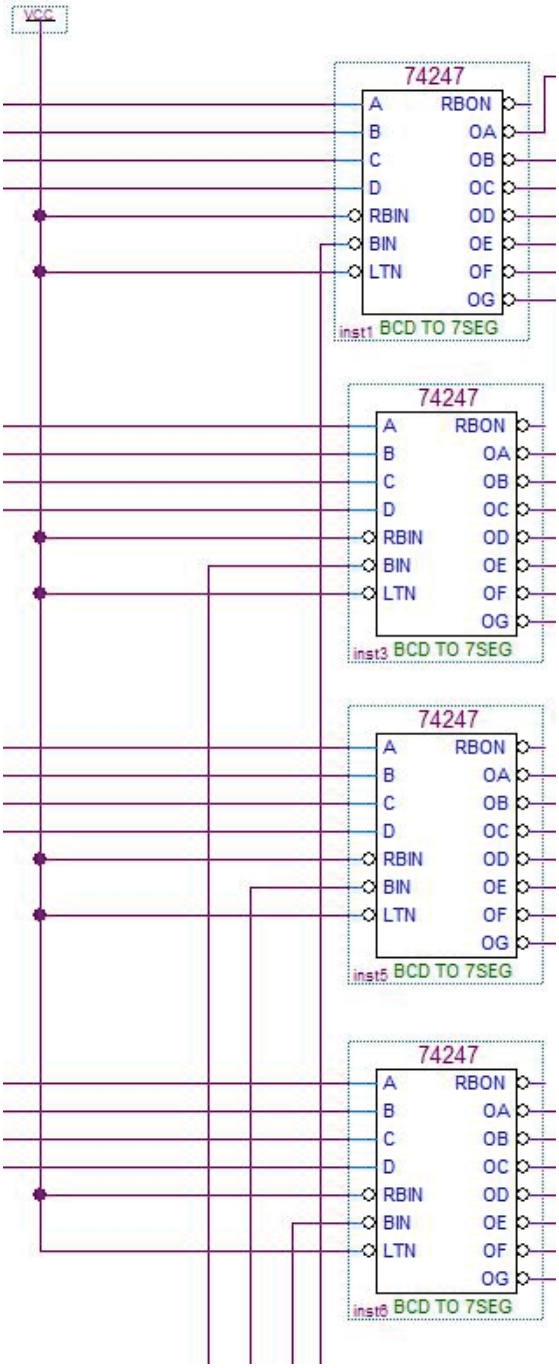
4.1.3 Отброяване на минути и часове

Отброяване на минути съответно 60 min за един час е постигнато отново с 7490, като за минути единици брояча е с коефициент на броене 10, а за минути десетици коефициента е 6. Коефициент на делене 6 се постига, като се вземе сигнал за нулиране на брояча и съответно управление на следващия брояч отчитащ часовете в единици от изходи Qb и Qc (фиг. 4.6) което е числото 0110. Решението за нулиране на броячите, отчитащи часовете, е направено с общо нулиране при достигане на числото 24, съответно сигнал се взема от Qc на брояча за единици и Qb на брояча за десетици. Предвидена е допълнителна логика за сверяване на минути и часове. Използвана е по-висока честота от основния делите, а именно изходи q[22] и q[23]. Честотата е 5,96Hz.



Фиг. 4.6. *Броячи за отброяване на минути и часове в формат 24:00*

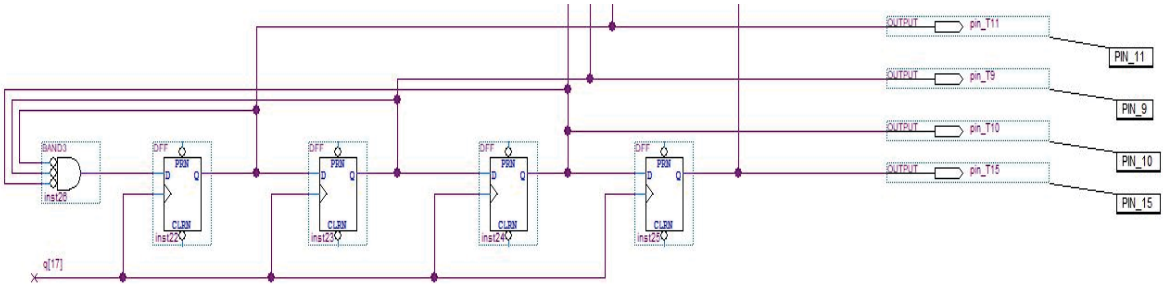
Получените двоични числа от броячите се преобразуват от BCD-7SEG преобразувател на код, показан на фиг. 4.7. Характерна особеност, е че изходите на тези преобразуватели са инверсни, което спомага за управлението на индикатора.



Фиг. 4.7.

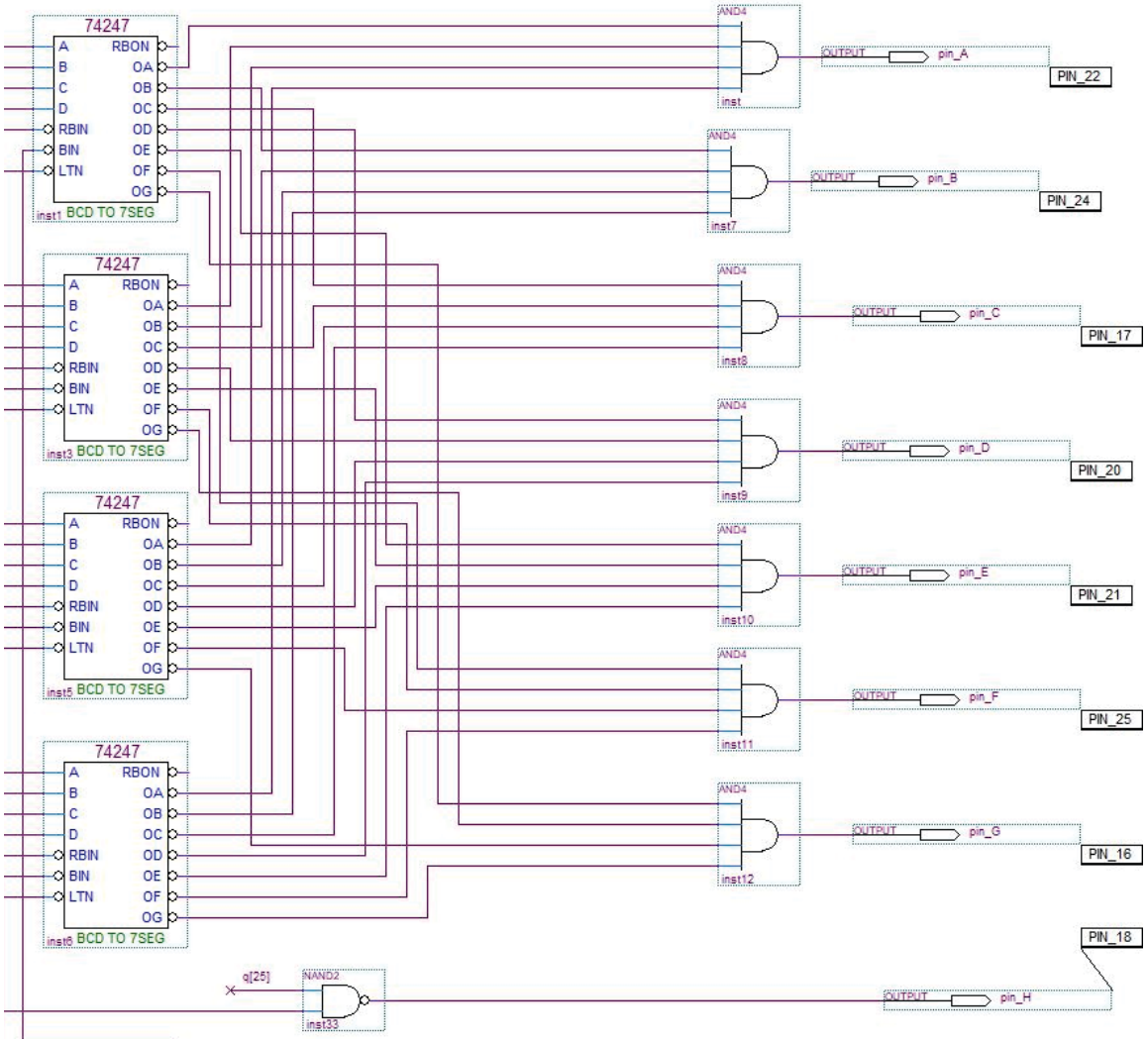
Седемсегментните индикатори са със общ анод (плюс) и се управляват по логическата нула. Друга особеност е входът за тест на диодите LTN и вход за празен изход BIN. Входът BIN (blanking) е ползван за създаване на динамичната индикация. През 47,68Hz само един от четирите кодови преобразувателя ще имат активни изходи, т.е. BIN=1. Завъртането е реализирано на базата на четири D-тригера (кръгов брояч), чиито изходи управляват входовете BIN (фиг. 4.8).

Същите тези изходи се използват за включване на съответния седемсегментен индикатор на изводи 11, 9, 10 и 15. Общата тактова честота на кръговия брояч е разделена от основната на 2^{18}, което прави 190,734Hz.



Фиг. 4.8. *Кръгов брояч*

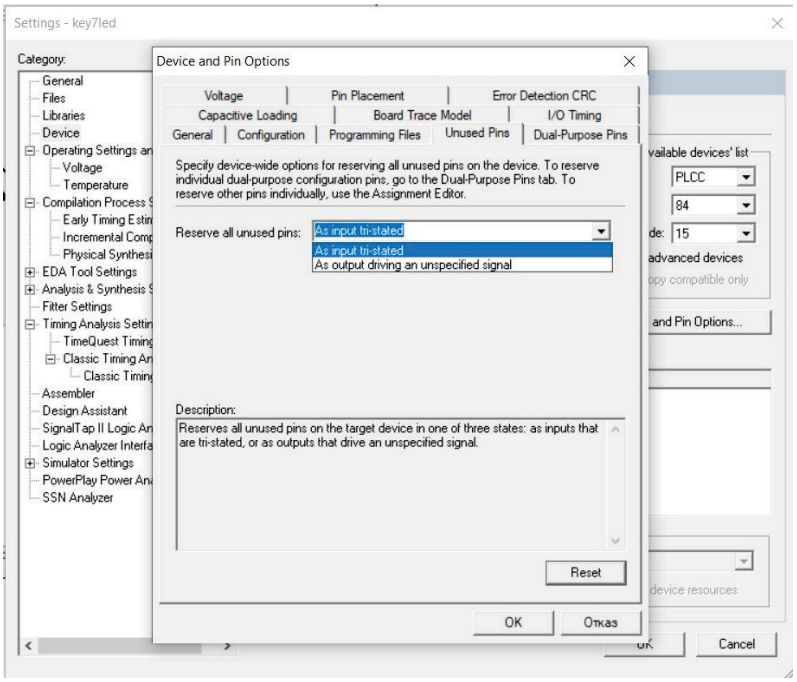
Изходите на преобразувателите на код от BCD в седемсегментен са обединени чрез логически елементи И. Така, който преобразувател е в режим blanking, неговите изходи ще са в логическа 1 и няма да влияят на работещия, чиито изходи ще са в логическа 0 спрямо входното двоично число (фиг. 4.9).



Фиг. 4.9. *Обединяване на изходи на 74247*

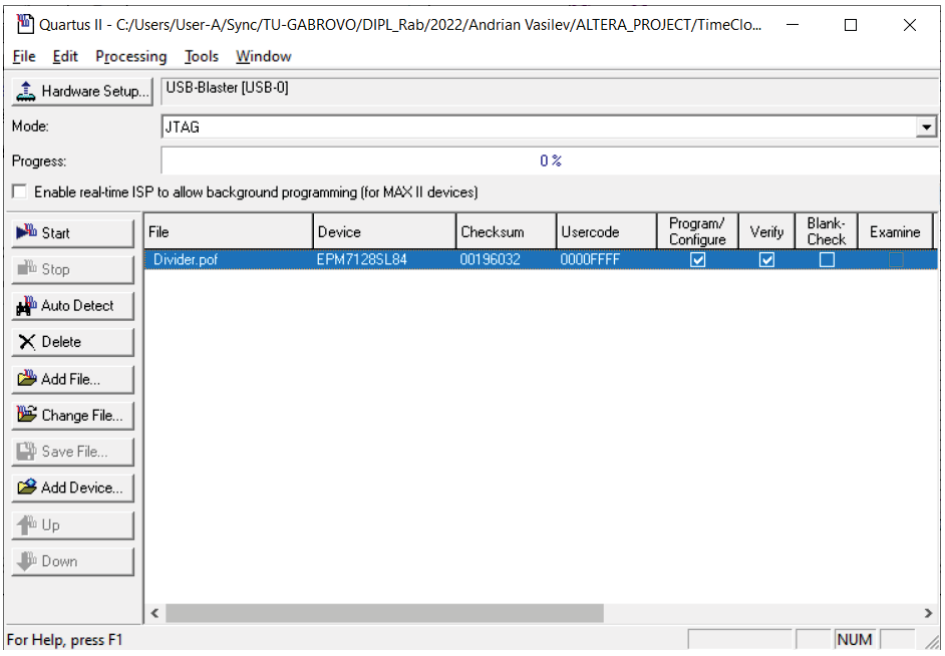
4.1.4. Програмиране на CPLD

Преди за пристъпим към програмиране, е важно да укажем значението на неизползваните изводи. За да няма управление на другите елементи, запоени на платката, задаваме неизползваните изводи като входове във високо импедансно ниво (фиг. 4.10).



Фиг. 4.10

Следваща стъпка е програмиране на разработена логическа схема, като използваме USB-JTAG програматор. Зареждаме файла Divide.pof чекваме Program/Configure и натискаме старт.

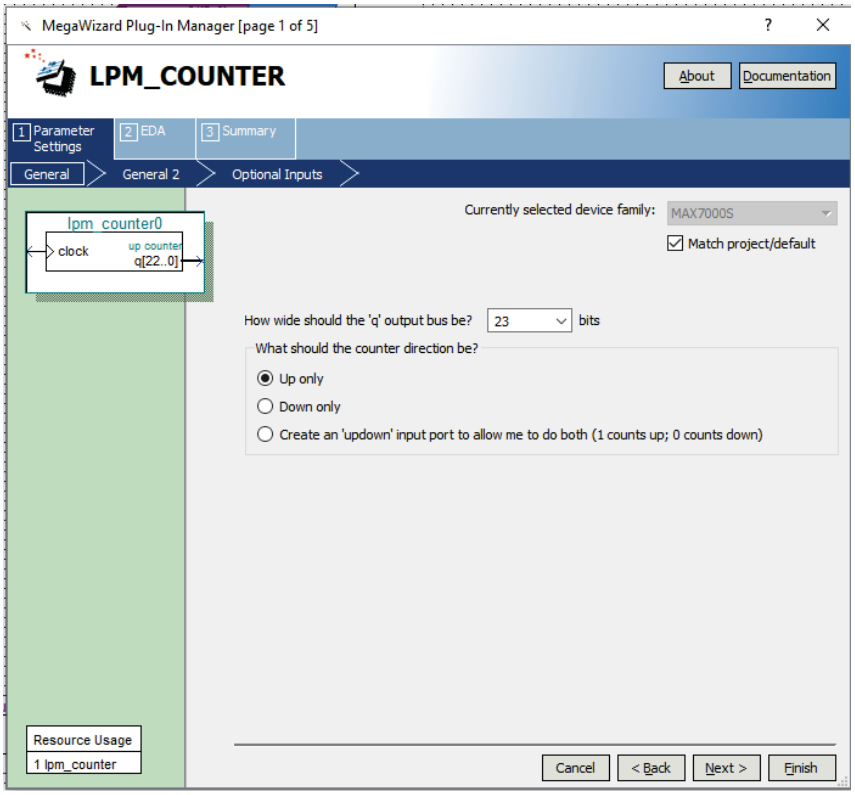


Фиг. 4.11

4.2. Работа с константи. Проектиране на светлини ефекти.

4.2.1 Делител на честота

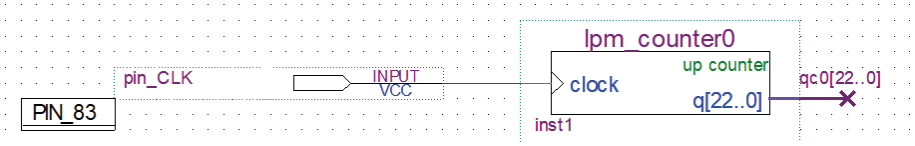
За получване на 6 Hz честота, необходима за визуализация на светлинните ефекти, се използва създаването на делител (брояч) на 2^{23}. Отново това се налага, защото използваме вградения генератор с 50MHz тактова честота. Стартираме прозореца на библиотеката от бутона, приличащ на логически елемент И, или чрез два пъти щракване на празно поле – фиг. 4.12. Избираме опцията megafunction>aritmetic и от там lpm_counter. Този макрос има много възможности, като асинхронно установяване в лог.1 или нулиране, избиране на посока на броене, запис на входно число и др.



Фиг. 4.12. *Избор на брой разряди.*

Важна особеност е задаването на коефициента на делене. Броячът е 23 бита, (избира се от меню показано на фиг. 4.12) което позволява бит с номер 22 да дели на 2^{23} , което е 8 388 608. Това число ще направи изходна честота от 6Hz.

След натискане на бутон финиш се получава макроса, показан на фиг. 4.13. Към него е присвоен вход с номер, съответстващ на входа на основната тактова честота.



Фиг. 4.13. *Брояч като делител на честота*

Основната идея на този проект е да се запишат определен брой 8-битови двоични числа (константи), които да „излизат“ в определен ред на 8-битова светодиодна индикация. Ще използваме 32 броя константи, чиято подредба на единиците и нулите в тези 8 бита на всяко число ще доведе до светлинно шоу. За да бъде ефекта видим с просто око, трябва честотата за смяна на числата да е в порядъка 2-6Hz. Това налага създаването на делителя, показан на фиг.4.13.

Избраните константи, с които ще се работи, в реда на излизане са следните:

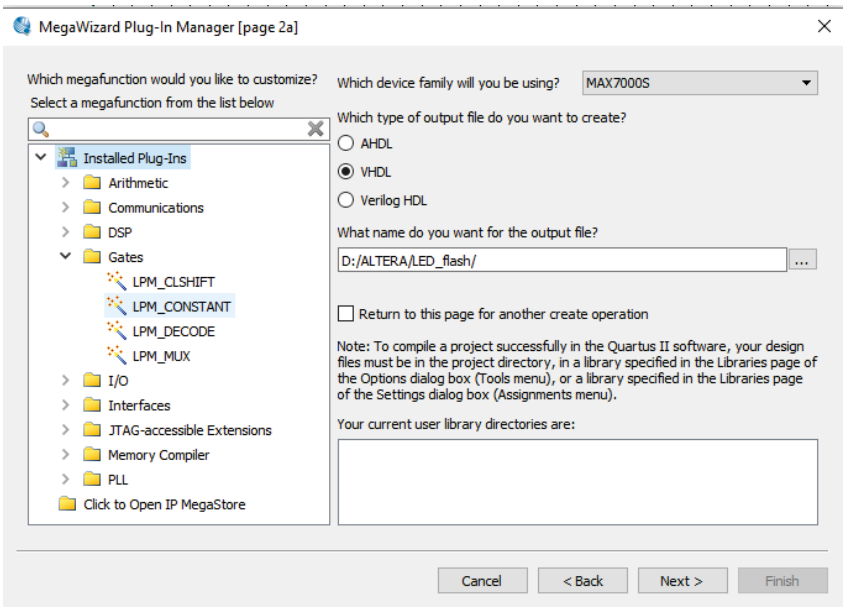
```
5'b00000: led=8'b11111111; (255) (Подаваме число 0)
5'b00001: led=8'b00000000; (0) (Подаваме число 255)
5'b00010: led=8'b01010101; (85) (Подаваме число 170)
5'b00011: led=8'b00000000; (0)
5'b00100: led=8'b10101010; (170) (Подаваме число 85)
5'b00101: led=8'b00000000; (0)
5'b00110: led=8'b00000001; (1) (Подаваме число 254)
5'b00111: led=8'b00000010; (2) (Подаваме число 253)
5'b01000: led=8'b00000100; (4) (Подаваме число 251)
5'b01001: led=8'b00001000; (8) (Подаваме число 247)
5'b01010: led=8'b00010000; (16) (Подаваме число 239)
5'b01011: led=8'b00100000; (32) (Подаваме число 223)
5'b01100: led=8'b01000000; (64) (Подаваме число 190)
5'b01101: led=8'b10000000; (128) (Подаваме число 127)
5'b01110: led=8'b01000000; (64)
5'b01111: led=8'b00100000; (32)
5'b10000: led=8'b00010000; (16)
5'b10001: led=8'b00001000; (8)
5'b10010: led=8'b00000100; (4)
5'b10011: led=8'b00000010; (2)
5'b10100: led=8'b00000001; (1)
5'b10101: led=8'b00000000; (0)
5'b10110: led=8'b11111111; (255) (Подаваме число 0)
5'b10111: led=8'b01111110; (126) (Подаваме число 129)
5'b11000: led=8'b00111100; (60) (Подаваме число 195)
5'b11001: led=8'b00011000; (24) (Подаваме число 231)
5'b11010: led=8'b00000000; (0)
5'b11011: led=8'b00011000; (24)
5'b11100: led=8'b00111100; (60)
5'b11101: led=8'b01111110; (126)
5'b11110: led=8'b11111111; (255)
5'b11111: led=8'b00000000; (0)
```

Посочените двоични константи трябва да бъдат инвертирани, защото светодиодите на развойната платка се управляват по логическа нула. Когато светодиодът свети, ние тълкуваме това състояние като лог.1, но за да стане това на съответния извод трябва да има лог.0. В зависимост от търсения ефект стойностите на константите могат да бъдат и други. Тъй като стойностите които ще извикваме се повтарят, е достатъчно да се запиша по един

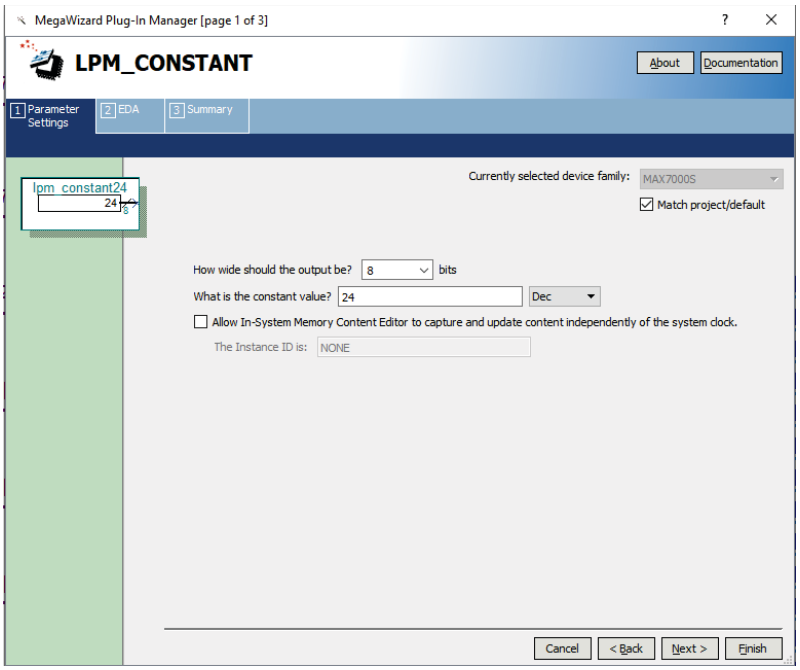
вид от константите. Примерно числото нула (0) се повтаря шест пъти, достатъчно е да го има записано веднъж.

4.2.2 Задаване на константи

Стартираме прозореца на библиотеката от бутона, приличащ на логически елемент И или чрез два пъти щракване на празно поле – фиг.4.14. Избираме опцията megafunction>gates и от там lpm_constant. Задаваме име, в нашия случай lpm_constant24, и продължаваме с бутона Next. На следващия екран задаваме колко битова да е константата и каква да е нейната стойност - фиг.4.15, съответно в десетична бройна система, за да е по разбираемо.

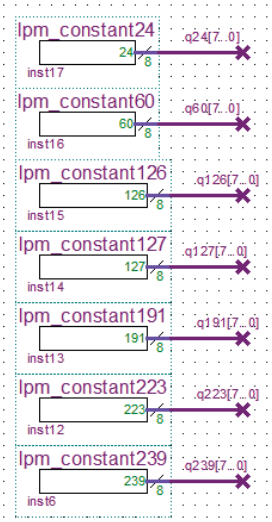


Фиг. 4.14



Фиг. 4.15

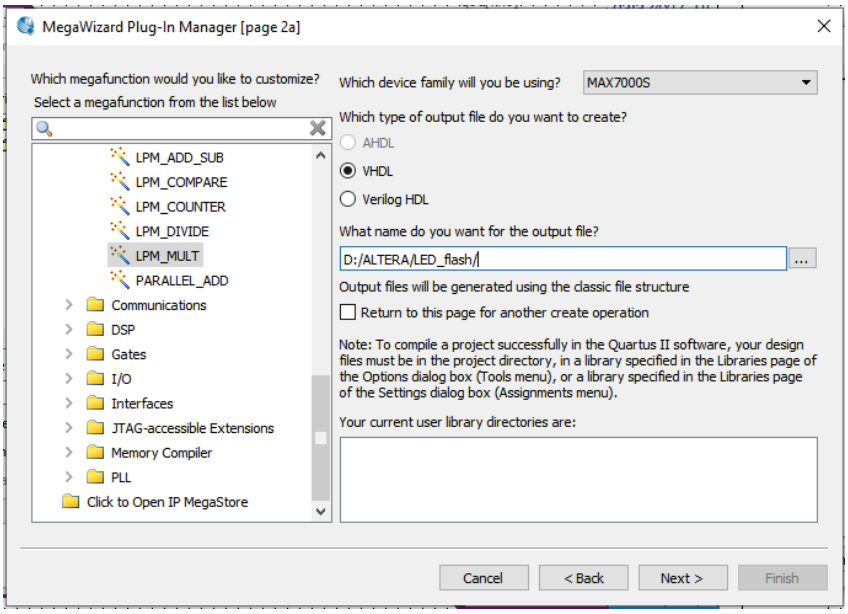
В този проект някои константи са инвертирани, други не. В участъка, където се получава кръгов брояч (1-2-4-8-16 и т.н), за да се запази ефектът на „бягащата единица“, числата са инвертирани, останалите не. На изхода на всяка създадена константа, тъй като е 8-битова, се добавя шина със съответното име (примерно q127). Това ще опрости значително последващото свързване към 32 входовия 8-битов мултиплексор – фиг.4.16.



Фиг. 4.16

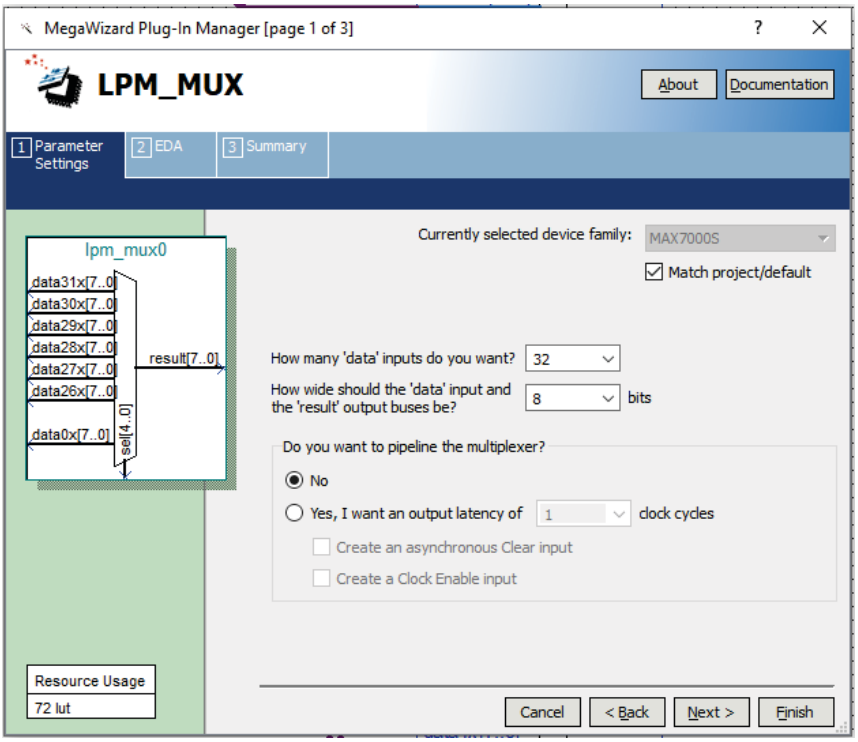
4.2.3 Създаване на мултиплексор

Всички тези числа ще се визуализират на 8 светодиода. Те ще бъдат подадени на 32-входов 8-битов мултиплексор, които ще има 5-битов управляващ вход (адресни входове). Това не е стандартен мултиплексор и не може да се използва библиотеката от серията 74xx. Отново чрез опцията megafunction>aritmetic и от там lpm_MULT и задаваме име на мултиплексора – фиг. 4.17.



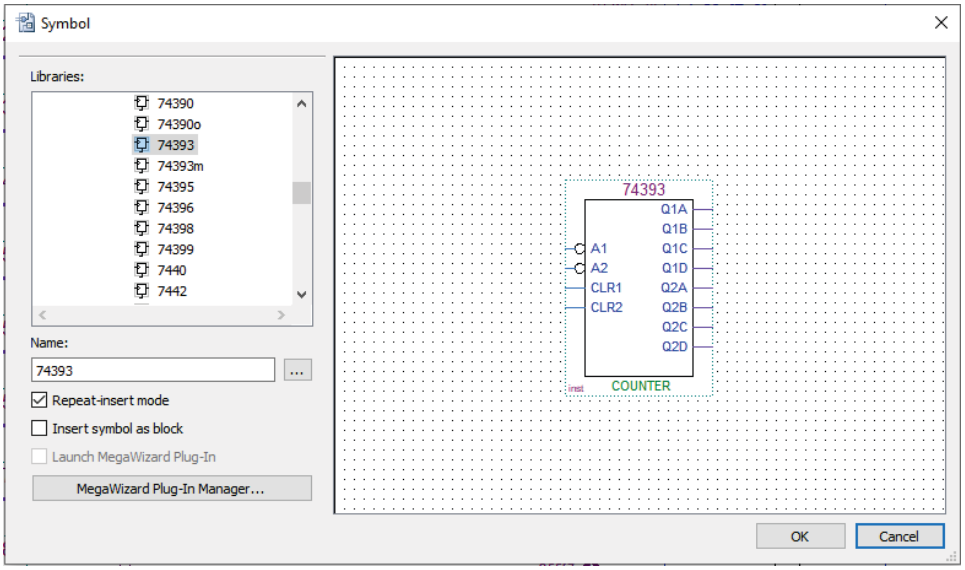
Фиг. 4.17

Задаваме да бъде със 32 входа 8 бита – фиг.4.18

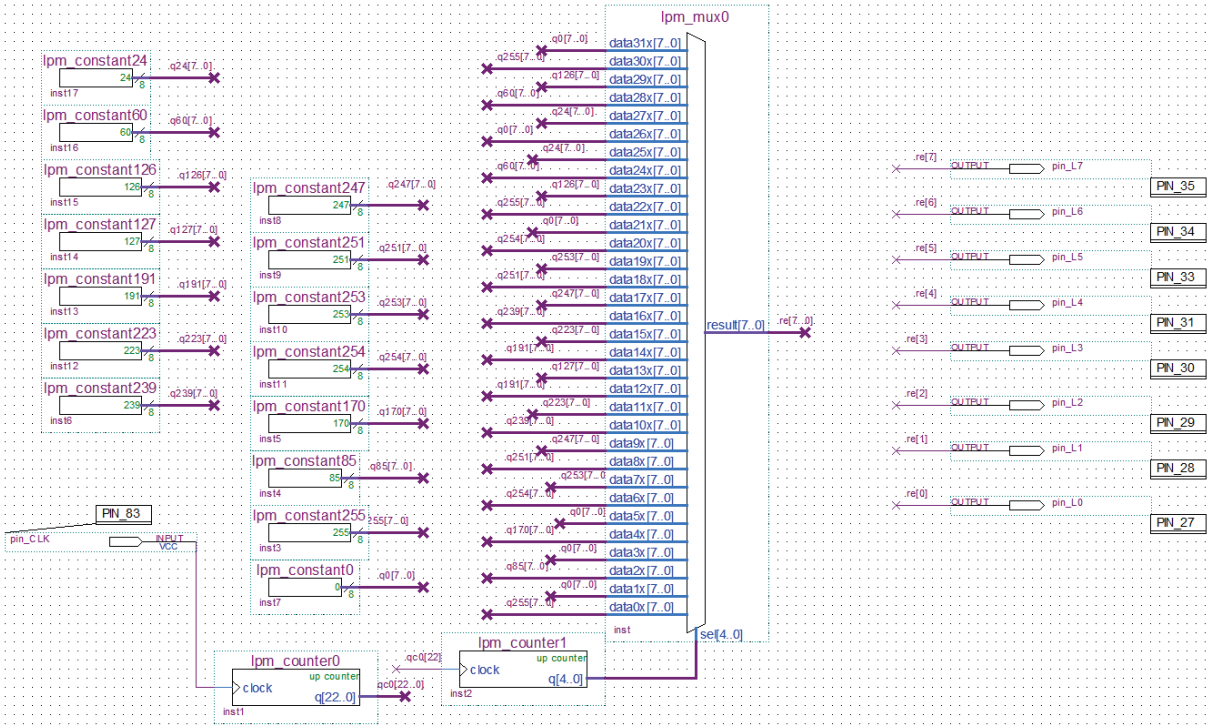


Фиг. 4.18

Входовете и изхода на получения мултиплексор – фиг. 4.20 също се свързват със шини, като за входовете се задават имена на шината в реда, по които искаме да излизат константите. Името на входната шина на мултиплексора трябва да съответства на името на изходната шина на съответната константа. Превключването на входовете на мултиплексора се реализира от 5-битов брояч. Може да бъде реализиран от два 4-битови брояча, примерно 74393 (фиг. 4.19) в един корпус и отново индивидуално създаден чрез опцията megafunction>aritmetic.



Фиг. 4.19


lpm_mux0
lpm_constant24
lpm_constant60
lpm_constant126
lpm_constant127
lpm_constant191
lpm_constant223
lpm_constant239
lpm_constant247
lpm_constant251
lpm_constant253
lpm_constant254
lpm_constant170
lpm_constant85
lpm_constant255
lpm_constant0
PIN_83
pin_CLK
INPUT
VCC
lpm_counter0
up counter
clock
q[22..0]
lpm_counter1
q[4..0]
data31x[7..0]
data30x[7..0]
data29x[7..0]
data28x[7..0]
data27x[7..0]
data26x[7..0]
data25x[7..0]
data24x[7..0]
data23x[7..0]
data22x[7..0]
data21x[7..0]
data20x[7..0]
data19x[7..0]
data18x[7..0]
data17x[7..0]
data16x[7..0]
data15x[7..0]
data14x[7..0]
data13x[7..0]
data12x[7..0]
data11x[7..0]
data10x[7..0]
data9x[7..0]
data8x[7..0]
data7x[7..0]
data6x[7..0]
data5x[7..0]
data4x[7..0]
data3x[7..0]
data2x[7..0]
data1x[7..0]
data0x[7..0]
result[7..0]
sel[4..0]
OUTPUT
pin_L7
pin_L6
pin_L5
pin_L4
pin_L3
pin_L2
pin_L1
pin_L0
PIN_35
PIN_34
PIN_33
PIN_31
PIN_30
PIN_29
PIN_28
PIN_27

Фиг. 4.20

ЛИТЕРАТУРА

1. Георги К. Петров, Дизайн на цифрови електронни устройства с VHDL и Quartus II, Част II - Основи на VHDL в примери и задачи, УЧЕБНО ПОМАГАЛО София 2010
2. Xilinx, Programmable Logic Design, Quick Start Handbook 2006
3. Акчурин А.Д., Юсупов К.М., Колчев А.А., ОСНОВЫ РАБОТЫ В СРЕДЕ QUARTUS II, Казань – 2017, УДК 621.396.075
4. Н.В. Ефремов, Введение в систему автоматизированного проектирования Quartus II, Издательство Московского государственного университета леса 2011, УДК 004.896(075)
5. Design Automation Standards Committee, IEEE Standard Verilog® Hardware Description Language, Published by The Institute of Electrical and Electronics Engineers, Inc. 3 Park Avenue, New York, 2001, Print ISBN 0-7381-2826-0
6. Горан Горанов, Цифрова схемотехника, Васил Априлов ТУ-Габрово 2016, ISBN 978 – 954 – 683 – 554 – 3
7. Горан Горанов, Цифрова схемотехника, Васил Априлов ТУ-Габрово 2016, ISBN 978 – 954 – 683 – 554 – 38.
8. М. Г. Царёв, Проектирование цифровых устройств на ПЛИС, Ульяновск УлГТУ 2017, УДК 004.3+371.693 (076)
9. Вервейко Александр Иванович, Система автоматизированного проектирования QUARTUS II, Чернигов ЧГТУ 2013
10. Altera Corporation - University Program, QUARTUS II introduction using vhdl designs For Quartus II 13.0, 2013
11. Altera Corporation, Quartus II Handbook Version 13.0 Volume 1: Design and Synthesis, 2013
12. Thomas L. Floyd, Digital Fundamentals 11Th Edition, Pearson 2015, ISBN 10: 1-292-07598-8

СЪДЪРЖАНИЕ