МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Тульский государственный университет»

Институт *высокоточных систем им. В.П. Грязева*
Кафедра «Приборы управления»

Утверждено на заседании кафедры
«Приборы управления»
«22» января 2024 г., протокол №1

Заведующий кафедрой

_____ В.В. Матвеев

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
по проведению практических (семинарских) занятий по дисциплине (модулю) «Микропроцессоры в оптотехнике»

основной профессиональной образовательной программы высшего образования – программы бакалавриата

по направлению подготовки
12.03.03 Фотоника и оптоинформатика

с направленностью (профилем)
Интеллектуальные фотонные системы
Форма обучения: очная

Идентификационный номер образовательной программы: 120303-01-24

Тула 2024 год

Разработчик методических указаний:

Алалуев В.В., доц. каф., к.т.н.,_____ 

Практическая работа №1. Изучение языка ассемблера. Команды передачи данных.....4
Практическая работа №2 Изучение языка асссемблела. Команды логические 8
Практическая работа №3. Изучение языка асссемблела. Команды арифметические. 10
Практическая работа№4. Изучение языка асссемблела. Команды передачи управления и обработки подпрограмм 14
Практическое занятие 5. Язык AlteraHDL составление программ. Логические выражения...17
Практическое занятие 6. Язык AlteraHDL составление программ. Комбинационная логика.23
Практическое занятие 7. Язык AlteraHDL составление программ. Последовательностная логика 28
Практическое занятие №8. Изучение системы проектирования QUARTUS8.1 31
Практическое занятие №9. Создание простейшего проекта с использованием графического редактора системы QUARTUS8.1 40
Практическое занятие №10. Создание простейшего проекта с использованием языка AHDL и текстового редактора системы QUARTUS8.1 48
Практическая работа №11. Создание проекта с использованием макрофункций на языке AHDL.....62

Практическая работа №1. Изучение языка ассемблера. Команды передачи данных

Цель работы: изучение арифметических команд. Использование арифменических команд на правтике.

Эту группу команд условно можно разделить на 3 подгруппы:
- команды засылки константы;
- команды пересылки;
- команды чтения-записи.

Команды передачи данных обеспечивают пересылку данных между регистрами, между памятью и регистрами.

Рассмотрим формат передачи данных ***КОП DST, SRC***.

КОП — код операции (число от 0 до 255).

DST — приемник информации.

SRC — источник информации.

Команды передачи данных при своем выполнении не изменяют значения флагового регистра.

Команды засылки константы

В результате выполнения команды засылки константы в регистр или регистровую пару загружается константа, содержащаяся во втором или во втором и третьем байтах команды. Рассмотри формат команд засылки констант:

Команда засылки 8 разрядной константы имеет формат

MVI DST,D8

где ***DST***-любой из ***РОН*** (***A, B, C, D, E, H, L***) или ячейка памяти (***M***) адрес которой берется из регистровой пары ***HL***; ***MVI*** — код операции; ***D8*** - 8-разрядная константа, содержащаяся во втором байте команды.

Задание 1: требуется загрузить в ячейку памяти с адресом 0860_{16} число 3***F***.

Введите программу на стенде и запустите. Объясните результат.

Адрес	Команда	Содержание ячейки памяти	Комментарий
0800	MVI H		В регистр H засылается константа 10_{16} H<-10
0801	08	08	
0802	MVI L		В регистр L запишем младший байт адреса 05 (HL=1005_{16})
0803	60	60	
0804	MVI M		В ячейку памяти адресуемую с помощью регистровой пары HL записываем число $3F_{16}$
0805	3F	3F	

Команда засылки 16 разрядной константы имеет формат

LXI RP,D16 где ***RP*** - регистровая пара ***B, D, H*** или указатель стека ***SP***;

D16-16-разрядная величина константы, причем младший байт константы находится во втором байте команды, а старший в третьем.

Задание 2: Требуется загрузить в ячейку памяти с адресом 1005 число 3***F***. (Введите программу на стенде и запустите. Объясните результат)

Адрес	Команда	Содержание ячейки памяти	Комментарий
0800	LXI H		В регистровую пару HL засылается константа 0510_{16} причем в регистр H засылается 10_{16}, а в регистр L - 05_{16}
0801	60	60	
0802	08	08	
0803	MVI M		В ячейку памяти, адресуемую с помощью регистровой пары HL записываем число $3F_{16}$
0804	3F	3F	

Команда пересылки данных

При выполнении команд пересылки содержимое источника ***SRC*** пересылается в приемник ***DST***, при этом содержимое источника не изменяется. В качестве источника и приемника операндов может быть использован любой из регистров общего назначения или ячейка памяти, адресуемая через регистровую пару (***H, L***). Пересылка данных типа "ПАМЯТЬ-ПАМЯТЬ" запрещена.

Формат команд пересылки ***MOV DST, SRC***,

где ***DST***, ***SRC***-любой из ***РОН*** (***A, B, C, D, E, H, L***) или ячейка памяти (***M***), адресуемая через регистровую пару HL.

Задание 3: переслать данные из ячейки с адресом 0860 в ячейку с адресом 0861. Введите программу на стенде и запустите. Объясните результат.

Адрес	Команда	Содержание ячейки памяти	Комментарий
0800	LXI H		В регистровую пару HL засылается константа 0860_{16} причем в регистр H засылается 08_{16}, а в регистр L - 60_{16}
0801	60	60	
0802	08	08	
0803	MOV A,M		Пересылка данных из памяти в аккумулятор
0804	MVI L		Засылка в HL адреса 0861
0805	61	61	
0806	MOV M,A		Передача данных из A в ячейку памяти с адресом 0861.

К командам пересылки можно также отнести команду ***XCHG***, в результате выполнения которой регистровые пары (***H, L***) и (***D, E***) обмениваются содержимым следующим образом:

H с ***D***

L с ***E***

Команды чтения-записи.

LDAX RP- чтение из памяти в аккумулятор содержимого ячейки памяти, адресуемой через регистровую пару ***BC*** или ***DE***;

STAX RP- запись содержимого аккумулятора в ячейку памяти адресуемую через регистровую пару ***BC*** или ***DE***;

Задание 4: необходимо записать в память по адресу 0860_{16} число $3F_{16}$ с помощью команды ***STAX*** (Введите программу на стенде и запустите. Объясните результат)

Адрес	Команда	Содержание ячейки памяти	Комментарий
0800	LXI B		В регистровую пару HL засылается константа 0510_{16} причем в регистр H засылается 10_{16}, а в регистр L - 05_{16}
0801	60	60	
0802	08	08	
0803	MVI A		Засылка в А числа 3F
0804	3F	3F	
0805	STAX B		Пересылка данных из аккумулятора А в память. Адрес ячейки памяти определяется содержимым регистровой пары BC.

LDA Adr-чтение содежимого ячейки памяти в аккумулятор, адрес ячейки памяти определяется 16-разрядным адресом ***Adr***;

Задание: Введите команду на стенде и запустите. Объясните результат

Адрес	Команда	Содержание ячейки памяти	Комментарий
0800	LDA		A <-M_{0801}<-0801
0801	01	01	Значок <- означает направление передачи.
0802	08	08	

LHLD Adr - запись в регистровую пару ***HL*** содержимого двух последовательных ячеек памяти с адресами ***Adr*** и ***Adr+1***, причем в регистр ***H*** загружается содержимое ячейки по адресу ***Adr+1***, а в регистр ***L*** - по адресу ***Adr***;

Задание: Введите команду на стенде и запустите. Объясните результат

Адрес	Команда	Содержание ячейки памяти	Комментарий
0800	LHLD	2A	HL <-002A
0801	00	00	
0802	08	08	
0803			

SHLD Adr - запись содержимого регистровой пары ***HL*** в две последовательные ячейки памяти с адресами ***Adr*** и ***Adr+1***, причем содержимое регистра ***H*** записывается в ячейку с адресом ***Adr+1***, а регистра ***L*** - в ячейку с адресом ***Adr***.

Практическая работа №2 Изучение языка асссемблела. Команды логические

При выполнении логических команд в аккумулятор заносится результат логической операции над операндом-источником и аккумулятором. Операндом-источником может быть регистр, ячейка памяти или константа, которая в этом случае записывается в следующем за кодом команды байте.

К логическим командам относятся:

ANA SRC - "Логическое И" содержимого аккумулятора и операнда-источника. При выполнении логических команд операции выполняются побитно, т.е. 7-й бит результата определяется 7-м битом обоих операндов, 6-й бит результата — 6-м битом обоих операндов и т.д.

Задание : Выполнить операцию логического И над двумя числами, находящимися в памяти по адресам 0861 и 0860. Результат надо оставить в регистре ***A*** (аккумуляторе). Введите программу на стенде и запустите. Объясните результат.

Адрес	Команда	Содержимое ячеек памяти	Комментарии
0800	LXI B	01	BC<-0860
0801	60	60	
0802	08	08	
0803	LDAX B	0A	A<-M(BC)<-3E
0804	LXI H	21	
0805	61	61	HL<-0861
0806	08	08	
0807	MOV D, M	36	D <- M(HL)<- A8
0808	ANA D	A2	A<- A&D<- 28
...			
0860	3E	3E	
0861	A8	A8	

Операцию побитового **И** поясняет таблица:

Таблица

0860	A=3E	0	0	1	1	1	1	1	0
0861	D=A8	1	0	1	0	1	0	0	0
A	A=28	0	0	1	0	1	0	0	0

ORA SRC - "Логическое ИЛИ" содержимого аккумулятора и операнда-источника;

XRA SRC - "Исключающее ИЛИ" (сложение по модулю 2) содержимого аккумулятора и операнда-источника;

CMP SRC- сравнение содержимого аккумулятора и операнда-источника. При выполнении этой операции операнд вычитается из содержимого аккумулятора без изменения участвующих в операции операндов. Состояние битов флагового регистра устанавливается по результату вычитания, т.е. результатом является установка флагов.

ANI D8 - "Логическое **И**" содержимого аккумулятора и константы D8, следующей во втором байте команды;

ORI D8 - "Логическое **ИЛИ"** содержимого аккумулятора и константы;

XRI D8 - " Исключающее **ИЛИ"** (сложение по модулю 2) содержимого аккумулятора и константы;

CPI D8 - сравнение содержимого аккумулятора с константой. При выполнении этой операции константа вычитается из содержимого аккумулятора без изменения участвующих в операции операндов. Состояние битов условий устанавливается.

Задание 2: Выполнить операцию логического ИЛИ, Исключающее ИЛИ над двумя числами находящимися в памяти по адресам 0871 и 0870. Необходимо составить программу аналогичную заданию 1 ввести и выполнить ее на стенде объяснить результат.

Практическая работа №3. Изучение языка асссемблела. Команды арифметические.

Цель работы: изучение арифметических команд. Использование арифменических команд на правтике.

Арифметические команды обеспечивают выполнение операций сложения и вычитания, а также изменение операнда на единицу.

Арифметические операции можно разделить на следующие группы:

1. операции с одним операндом, регистром или регистровой парой;
2. операции с двумя операндами, причем в качестве первого операнда всегда используется аккумулятор, а в качестве второго операнда может быть регистр, ячейка памяти или константа, которая в этом случае записывается в следующем за кодом команды байте.

Арифметические команды изменяют состояние битов условий флагового регистра.

Команды сложения

При выполнении команды сложения в аккумулятор заносится результат сложения аккумулятора и операнда-источника. Операндом-источником может быть регистр, ячейка памяти или констант.

К этим командам относятся:

ADD SRC - сложение содержимого аккумулятора с регистром или ячейкой памяти, например ***ADD*** B означает, что в аккумулятор помещается результат сложения аккумулятора и регистра ***B*** (***A<-A+B***)

ADC SRC - сложение содержимого аккумулятора, операнда-источника и бита ***CY*** признакового регистра ***F*** (***A<- A+SRC+CY(F***)).

Задание 1: сложить 2 двухбайтных числа (35A0 и 67B2) результат поместить в регистровую пару BC. Ввести программу на стенде выполнить и пояснить результаты.

35A0
+67B2
9D52

Адрес	Команда	Содержание ячейки памяти	Комментарий
0800	LXI B		BC<-35A0
0801	A0	A0	
0802	35	35	

0803	LXI D		DE<-67B2
0804	B2		
0805	67		
0806	MOV A, C		A <- A0
0807	ADD E		A<- A+E
0808	MOV C, A		C< -A
0809	MOV A, B		A<- B
080A	ADC D		A<- A+D+CY
080B	MOV B, A		B<- A

ADI D8 **-** сложение содержимого аккумулятора с константой, при этом константа содержится во втором байте команды (***A< A+D8***);

ACI D8 **-** сложение содержимого аккумулятора с константой и битом ***CY*** признакового регистра ***F.***

Задание 2: составить программу, рассмотренную в предыдущем примере, но считая второе слагаемое константой. Ввести программу на стенде выполнить и пояснить результаты.

Адрес	Команда	Содержание ячейки памяти	Комментарий
0800	LXI B		BC<-35A0
0801	A0	A0	
0802	35	35	
0803	MOV A, C		A<-C
0804	ADI		A <- A+B2
0805	B2	B2	
0806	MOV C, A		C<- A
0807	MOV A, B		
0808	ACI		A<- A+67+CY
0809	67	67	
080A	MOV B, A		B<- A

DAD RP **-** сложение содержимого регистровой пары ***RP*** с регистровой парой ***HL***. Результат сложения записывается в пару ***HL***, эта команда изменяет только состояние бита ***CУ*** в признаковом регистре (например: ***DAD B: HL<- HL+BC***).

Команды вычитания.

При выполнении команд вычитания в аккумулятор заносится результат вычитания операнда-источника из аккумулятора. Операндом-источником может быть регистр, ячейка памяти или константа, которая записывается в следующем за кодом команды байте.

К этим командам относятся:

SUB SRC-вычитание из содержимого аккумулятора содержимого регистра или ячейки памяти, адрес которой определяется содержимым регистровой пары ***HL*** (***A< A-SRC***).

SBB SRC-вычитание из содержимого аккумулятора операнда-источника и бита ***CY*** (***A< A-SRC-CY***(***F***)).

SUI D8-вычитание константы из содержимого аккумулятора;

SBI D8-вычитание из содержимого аккумулятора значения бита ***CY*** и константы.

Эти команды изменяют все биты условий.

Задание1: вычесть 2 двухбайтных числа (34F3 и 12A3) результат поместить в регистровую пару BC. Ввести программу на стенде выполнить и пояснить результат.

Задание2: вычесть 2 двухбайтных числа считая уменьшаемое константой (34F3 и 12A3) результат поместить в регистровую пару BC. Ввести программу на стенде выполнить и пояснить результат.

Команды изменения операнда на 1.

Команды изменения операнда на единицу предназначены для увеличения или уменьшения операнда на единицу. Операндом может являться содержимое регистра, ячейки памяти или регистровой пары.

Эти команды часто применяются для организации счетчиков или изменения адресов, используемых при косвенной адресации (обработка массивов, матриц, строк и т.п.).

INR SRC - увеличение на 1 содержимого регистра или ячейки памяти, адресуемой содержимым регистровой пары ***HL***;

DCR SRC - уменьшение на единицу содержимого регистра или ячейки памяти.

Команды ***INR*** и ***DCR*** изменяют состояние всех флагов, кроме бита **CY.**

INX RP-увеличение на единицу содержимого регистровой пары. В этом случае число в регистровой паре рассматривается как 16-разрядный операнд;

DCX RP-уменьшение на единицу содержимого регистровой пары.

Команды ***INX*** **и** ***DCX*** не изменяют состояние битов условий.

DAA – команда десятичной коррекции. Применяется для работы с двоично-десятичными числами.

При выполнении команды ***DAA*** 8-битное число в аккумуляторе рассматривается как две 4-битные десятичные двоично-кодированные цифры. Коррекция содержимого аккумулятора производится по следующим правилам:

- если значение младшей тетрады аккумулятора больше 9 или флаг вспомогательного переноса ***AC*** равен 1, то к содержимому аккумулятора добавляется число 6;

- если значение старшей тетрады аккумулятора больше 9 или если признак переноса ***CY*** равен 1, то к содержимому аккумулятора добавляется число 96 или к старшей тетраде прибавляется 6.

Практическая работа№4. Изучение языка асссемблела. Команды передачи управления и обработки подпрограмм

К командам передачи управления относятся команды безусловного и условного переходов, команды безусловного и условного вызова и выхода из подпрограммы, а также команды перезапуска.

Команды этой группы не изменяют состояния битов условий флагового регистра, а используют их при своем выполнении, осуществляя переход по указанному адресу, если один из четырех битов условий находится в состоянии «1» или «0».

Рассмотрим условия переходов:

Условие	Условия перехода	
NZ	Результат операции не равен 0	(*Z*=0)
Z	Результат операции равен 0	(*Z*=1)
NC	Переноса не было	(*CY*=0)
C	Был перенос	(*CY*=1)
PE	Число единиц в аккумуляторе четно	(*P*=1)
PO	Число единиц в аккумуляторе нечетно	(*P*=0)
P	Результат операции положителен	(*S*=0)
M	Результат операции отрицателен	(*S*=1)

Команды условного перехода имеют следующий формат:

Jcond Adr

где ***cond*** - одно из условий перехода перечисленных в таблице;

Adr - адрес перехода, расположенный в следующих двух байтах команды.

Выполнение команд условных переходов происходит следующим образом: если условие перехода истинно, то управление передается по указанному в команде адресу перехода, в противоположном случае выполняется следующая команда. В соответствии с приведенными в таблице условиями перехода в систему команд МП входят следующие команды условного перехода:

JNZ JZ JNC JC JPE JPO JP JM

Задание 1: пусть требуется числу, находящемуся в памяти по адресу 0860 пять раз прибавить число, находящееся в памяти по адресу 0861. Введите программу на стенде, выполните и поясните результаты

Адрес	Команда	Содержание ячейки памяти	Комментарий
0800	LDA		A<-M_{0860}
0801	60	60	
0802	08	08	
0803	LXI H		HL<-0861
0804	61	61	
0805	08	08	
0806	MOV B, M		B<-M_{0861}

0807	MVI C		C<- 5
0808	5	5	
0809	ADD B		A <- A+B
080A	DCR C		C<- C-1 — установка флагов (Z)
080B	JNZ		Если в результате последней операции получился не ноль, то перейти по адресу 0809
080C	09	09	
080D	08	08	
080E	HLT		

Команда безусловного перехода, имеющая формат ***JMP Adr*** осуществляет безусловный переход по указанному адресу во втором и третьем байтах адреса перехода.

Команды условного вызова подпрограмм имеют следующий формат:

Ccond Adr

где ***cond*** - одно из условий перехода, перечисленных в таблице;

Adr - адрес перехода, расположенный в последующих двух байтах команды.

Выполнение команд условных вызовов подпрограмм происходит следующим образом: если условие истинно, то управление передается по указанному в команде адресу перехода, а содержимое программного счетчика (регистр PC) загружается в стек (в стеке сохраняется адрес возврата).

В соответствии с приведенными в таблице условиями перехода в систему команд МП входят следующие команды условного вызова подпрограмм:

CNZ CZ CNC CC CPE CPO CP CM

Команда безусловного вызова, подпрограммы, имеющая формат ***CALL Adr*** осуществляет безусловный вызов подпрограммы, расположенной по указанному адресу.

Команды условного возврата из подпрограммы имеют следующий формат:

Rcond.

Команда возврата из под программы выполняется следующим образом: если условие возврата истинно, то в программный счетчик заносится содержимое вершины стека - адрес возврата, в противоположном случае выполняется следующая команда подпрограммы.

В систему команд МП входят следующие команды условного возврата из подпрограммы:

RNZ RZ RNC RC RPE RPO RP RM

Задание 2: пусть требуется к массиву чисел, расположенному по адресу от 0860 до 0865, 5 раз прибавить число, находящееся по адресу 0866. Необходимо использовать подпрограмму. Введите программу на стенде, выполните и поясните результаты

Адрес	Команда	Содержимое ячеек памяти	Комментарии
0800	LXI H		

0801	60	60	HL<0860
0802	08	08	
0803	MOV A,M		A<M_{0860}
0804	CALL		вызов подпрограммы по адресу 0870
0805	70	70	
0806	08	08	
0807	LXI H		
0808	60		
0809	08		
080A	MOV M,A		M_{0860}<A
080B	INX H		HL< HL+1
080C	MOV A,M		A<M_{0861}
080D	CALL		вызов подпрограммы по адресу 0870
080E	70	70	
080F	08	08	
0810	LXI H		
0811	61		
0812	08		
0813	MOV M,A		
...	INX	Далее программа повторяется еще 3 раза	
Подпрограмма			
0870	LXI H		HL<-0866
0871	66	66	
0872	08	08	
0873	MOV B,M		B<- M_{0860}
0874	MVI C		C<- 5 счетчик цикла
0875	05	05	
0876	ADD B		A< A+B
0877	DCR C		C< C-1
0878	YNZ		если C≠0 то переходим на 0876
0879	76		
087A	08		
087B	RET		Безусловный возврат из подпрограммы

Практическое занятие 5. Язык AlteraHDL составление программ. Логические выражения

Цель работы: Изучение логических команд создание простейших проектов с использованием логических команд

В языке AHDL определены два типа выражений - арифметические и логические. Арифметические выражения - это средство формирования конструкций языка AHDL, эти выражения вычисляются компилятором на этапе синтаксического анализа, для их реализации ресурсы PLD не используются.

Логические выражения определяют собственно функционирование проектируемого устройства, правила их формирования и интерпретации связаны с особенностями элементов цифровых устройств.

Арифметические выражения используются для указания:

- выражений в разделе Define Statement;
- значений констант в разделе Constante Statement;
- границ диапазонов изменения индексов групп;
- границ диапазона переменной в операторе FOR GENERATE;
- выражения в операторах IF GENERATE и ASSERT.

Результат арифметического выражения должен быть целым положительным числом, если это не так, то он округляется до большего целого. Существуют две функции для явного задания правил округления: CETL - округление до большего целого; FLOOR - округление до меньшего целого. Арифметические и логические операции, операции сравнения (компараторы) имеют следующие приоритеты при вычислении арифметических выражений:

Таблица 3.7.

Операция/ Компаратор	Пример	Описание	Приоритет
+ (унарный)	+1	плюс	1
- (унарный)	-1	минус	1
!NOT	!a	отрицание	1
^	a^2	степень	1
MOD	4 MOD 2	модуль	2
DIV	4 DIV 2	деление	2
*	a*2	умножение	2
LOG2	LOG2 (4-3)	логарифм по основанию 2	2
+	1+1	сложение	3
-	1-1	вычитание	3
= = (numeric)	5= =5	равенство чисел	4

= = (string)	"a"= ="b"	равенство строк	4
!=	5! =4	не равно	4
>	5>4	больше	4
>=	5>=5	больше или равно	4
<	a<b+2	меньше	4
Операция/ Компаратор	Пример	Описание	Приоритет
<=	a<=b+2	меньше или равно	4
&	a & b	AND	5
AND	a AND b		
!&	1 !& 0	NAND	5
NAND	1 NAND 0		
$	1 $ 1	XOR	6
XOR	1 XOR 1		
!$	1 !$1	XNOR	6
XNOR	1 XNOR 1		
#	a # b	OR	7
OR	a OR b		
!#	a !# b	NOR	7
NOR	a NOR b		
?	(5<4) ? 3:4	условная операция	8

Приоритеты операций могут изменяться при помощи скобок.

Булевы выражения и уравнения. Булевы выражения - это операнды (числа, цепи, группы), разделенные знаками арифметических и логических операций, компараторами (операторами сравнения), сгруппированные с помощью скобок. Эти выражения используются в булевых уравнениях и операторах CASE и IF_THEN.

Булево выражение может иметь один из следующих видов:

- операнд

Пример: a, b[5..1], 7, VCC

- ссылка на логическую функцию

Пример: out [15..0]=16dmux(q[3..0]);

- Префиксный унарный оператор (! или -), примененный к булеву выражению

Пример: !c

- Два булевых выражения, разделенных бинарным оператором

Пример: d1 $ d3

- Булево выражение, заключенное в скобки

Пример: (!foo & bar)

Результат булева выражения имеет ту же ширину, что и операнды.

Булево уравнение устанавливает цепь, шину, порт и т.п. в состояние, определяемое булевым выражением. Символ (=) в булевом уравнении указывает, что результат булева выражения справа является источником для цепи или шины слева. В булевом уравнении слева может находиться идентификатор, имя порта или группы. Перед ним можно использовать оператор (!) инверсии. Справа в уравнении находится булево выражение, вычисляемое по правилам приоритетов (операции одного приоритета выполняются слева направо). Скобки могут изменять порядок вычислений.

В булевых выражениях могут использоваться следующие логические операции:

Таблица 3.8.

Операция	Пример	Описание
! NOT	!music NOT music	инверсия
& AND	a & b a AND b	И
!& NAND	a [3..1] !& b [5..3] a [3..1] NAND b [5..3]	И-НЕ
Операция	Пример	Описание
!$ XNOR	x2 !$ x4 x2 XNOR x4	инверсия исключающего ИЛИ
# OR	tris # tran tris OR tran	ИЛИ
!# NOR	c [8..5] !# d [5..4] c [8..5] NOR d [5..4]	ИЛИ-НЕ

Унарная операция NOT может применяться к одноразрядной переменной, группе переменных и к числу. При одноразрядной переменной результатом является инвертированное значение. В случае группы инвертируется каждый член группы. В случае числа инвертируется каждый разряд его двоичного представления.

Бинарные операции AND, NAND, OR, NOR, XOR, NXOR допускают следующие комбинации операндов:

- оба операнда одноразрядные (переменные, порты, VCC, GND);
- оба операнда являются группами - операция применяется поразрядно, поэтому операнды должны быть одинаковой разрядности;
- один оператор одноразрядный, а другой является группой - одноразрядный операнд тиражируется до группы, затем к двум группам поразрядно применяется операция;
- оба операнда числа - они представляются в двоичном формате и к группам двоичных разрядов применяется операция;

• один операнд число, а другой является одноразрядным, либо группой - число представляется группой двоичных разрядов и операция применяется к двум группам двоичных разрядов. Одноразрядный операнд тиражируется в группу, разрядность которой соответствует разрядности двоичного представления числа.

В булевых выражениях могут применяться следующие компараторы (операции сравнения):

Таблица 3.9.

Компаратор	Пример	Описание
= = (логическое)	Bus [15..0]= =H "B800"	равно
!= (логическое)	a1 !=a3	не равно
> (арифметическое)	c [ ] > d [ ]	больше
>= (арифметическое)	fiu [ ] >= fiu [ ]	больше или равно
< (арифметическое)	c < d+2	меньше
<= (арифметическое)	e <= f-2	меньше или равно

Результатом операции сравнения является логический ноль (GND), если условие не выполнено, и логическая единица (VCC), если условие выполнено.

Из таблицы видно, что компараторы делятся на логические и арифметические. При логическом сравнении осуществляется поразрядное сравнение операндов, а при арифметическом сравнении группа разрядов интерпретируется как двоичное число без знака.

В булевых выражениях могут применяться следующие арифметические операции:

Таблица 3.10.

Операция	Пример	Описание
+ (унарный)	+ 2	Плюс
• (унарный)	-f [ ]	Минус
+	dd [2..0] + ca[2..0]	Сложение
-	gamma [ ] – sigma [ ]	Вычитание

Арифметические операции могут выполняться над группами и числами. Если оба операнда являются группами, то они должны иметь одинаковую разрядность. При операциях над числами они представляются в виде групп двоичных разрядов и автоматически выравниваются.

Приоритет операций в булевых выражения следующий:

Таблица 3.11.

Приоритет	Операция/	Описание

	Компаратор:	
1	-	минус, дополнение до 2 (negative)
1	!	инверсия, логическое НЕ (NOT)
2	+	сложение (addition)
2	-	вычитание (subtraction)
3	= =	Равно (equal to)
3	!=	не равно (not equal to)
3	<	Меньше (less than)
3	<=	меньше или равно (less than or equal to)
3	>	Больше (greater than)
3	>=	больше или равно (greater than or equal to)
4	&	И (AND)
4	!&	И-НЕ (NAND)
5	$	ИСКЛЮЧАЮЩЕЕ ИЛИ (XOR)
5	!$	ИНВЕРСИЯ ИСКЛЮЧАЮЩЕГО ИЛИ (XNOR)
6	#	ИЛИ (OR)
6	!#	ИЛИ-НЕ (NOR)

Пример сложного булева уравнения:

a [ ] = ((c [ ] &-B "001101")+e [6..1]) # (p, q, r, s, t, v);

Это выражение вычисляется в следующем порядке:

Двоичное число B"001101" дополняется до 2 и становится B"001101". Унарный (-) имеет высший приоритет.

2. Выполняется операция AND над B"001101" и группой c [ ]. Эта операция имеет второй приоритет из-за скобок.
3. Результат складывается с группой e[6..1].
4. Выполняется операция OR над результатом и группой (p, q, r, s, t, v).

Общий результат присваивается группе a [ ]. Для правильности уравнения разрядность группы слева должна быть кратна разрядности группы справа.

Задание 1. Создать проект в среде QUARTUS II, в проекте создать текстовый файл boole1.TDF. Ввести в него текст. Откомпилировать проект и запустить симуляцию проекта. Пояснить результаты симуляции.

```
SUBDESIGN  boole1
(
  a0, a1, b: INPUT;
  out1, out2: OUTPUT;
BEGIN
  out1=a1 & !a0;
  out2=out1 # b;
```

```
END;
```

Здесь выходам out1 и out2 присваиваются значения, определяемые булевыми выражениями, в которых участвуют имена входных и выходных портов. Используются операции И, ИЛИ.

Задание 2. Создать проект в среде QUARTUS II, в проекте создать текстовый файл boole2.TDF. Ввести в него текст. Откомпилировать проект и запустить симуляцию проекта. Пояснить результаты симуляции.

```
SUBDESIGN boole2
(
  a0, al, b: INPUT;
  out1, out2: OUTPUT;
)
VARIABLE
  a_equals_2 : NODE;
BEGIN
  a_equals_2 = a1 & !a0;
  out1 =   a_equals_2
  out2 = a_equate_2 # b;
END;
```

Практическое занятие 6. Язык AlteraHDL составление программ. Комбинационная логика

Цель работы: Изучение команд создание простейших проектов с использованием логических команд

Комбинационная логика - это конечные автоматы без памяти. Эта логика описывается в AHDL с помощью булевых выражений и уравнений, таблиц истинности, мега и макрофункций.

Реализация табличной логики. Таблицы истинности в языке AHDL задаются с помощью оператора *Truth Table.* Оператор имеет следующие правила синтаксиса и семантики

- Первая строка оператора до символа (;) является заголовком таблицы. Вначале идет ключевое слово TABLE, за которым следует список входов (имена, разделенные запятыми). Далее следует символ => и список выходов.
- Последующие строки таблицы содержат списки значений входов и выходов. Списки разделены символом =>. Значения могут быть числами, константами VCC и GND, именами констант, группами чисел или констант. Входные значения могут быть безразличными состояниями (X). Входные и выходные значения соответствуют входам и выходам, указанным в заголовке таблицы.
- Описание таблицы завершается ключевыми словами END TABLE и (;).

При использовании этого оператора необходимо соблюдать следующие правила.

- Имена в заголовке таблицы должны представлять собой либо одиночные цепи, либо группы.
- В таблице в качестве входных значений можно употреблять символ безразличного состояния (X) для задания диапазона кодов.
- Размеры списков значений входов и выходов должны точно соответствовать размеру списков имен входов и выходов в заголовке. В противном случае в качестве выходных используются значения по умолчанию
- При использовании символа (X) необходимо следить, чтобы диапазоны задаваемых значений не перекрывались в пределах одной таблицы.

Задание 1. Создать проект в среде QUARTUS II, в проекте создать текстовый файл decode3.TDF. Ввести в него текст. Откомпилировать проект и запустить симуляцию проекта. Пояснить результаты симуляции.

```
SUBDESIGN decode3
(
  addr [15..0], m/io        : INPUT;
```

```
  rom, ram, print, sp [2..1] : OUTPUT;
)
BEGIN
 TABLE
   m/io, addr [15..0]              =>          rom, ram, print, sp [ ]
  1,     B "00XXXXXXXXXXXXXX" => 1,      0,    0,    B "00";
  1,     B "100XXXXXXXXXXXX"  => 0,      1,    0,    B "00";
  0,     B "0000001010101110"          => 0,      0,    1,    B "00";
  0,     B "0000001011011110"          => 0,      0,    0,    B "01";
  0,     B "0000001101110000"          => 0,      0,    0,    B "10";
 END TABLE;
END;
```

А это дешифратор адреса для внешней памяти микропроцессорной системы с шестнадцатиразрядным адресом. Использование безразличных состояний позволяет проще запрограммировать устройство и использовать меньше ресурсов ПЛИС.

Задание 2. Создать проект в среде QUARTUS II, в проекте создать текстовый файл 7segment.TDF. Ввести в него текст. Откомпилировать проект и запустить симуляцию проекта. Пояснить результаты симуляции.

Дешифратор для семисегментного индикатора

```
SUBDESIGN  7segment
(
  i [3..0]              :INPUT;
  a, b, c, d, e, f, g  : OUTPUT;
)
BEGIN
TABLE
  i [3..0] => a, b, c, d, e, f, g;
  H "0"   => 1, 1, 1, 1, 1, 1, 0;
  H "1"   => 0, 1, 1, 0, 0, 0, 0;
  H "2"   => 1, 1, 0, 1, 1, 0, 1;
  H "3"   => 1, 1, 1, 1, 0, 0, 1;
  H "4"   => 0, 1, 1, 0, 0, 1, 1;
  H "5"   => 1, 0, 1, 1, 0, 1, 1;
  H "6"   => 1, 0, 1, 1, 1, 1, 1;
  H "7"   => 1, 1, 1, 0, 0, 0, 0;
  H "8"   => 1, 1, 1, 1, 1, 1, 1;
  H "9"   => 1, 1, 1, 1, 0, 1, 1;
  H "A"   => 1, 1, 1, 0, 1, 1, 1;
  H "B"   => 0, 0, 1, 1, 1, 1, 1;
  H "C"   => 1, 0, 0, 1, 1, 1, 0;
  H "D"   => 0, 1, 1, 1, 1, 0, 1;
  H "E"   => 1, 0, 0, 1, 1, 1, 1;
```

```
 H "F"   => 1, 0, 0, 0, 1, 1, 1;
 END TABLE;
END;
```

Использование для переданных значений по умолчанию на языке *AHDL*.

Можно определить значение по умолчанию для узла или группы, которые будут автоматически использоваться для них, если в файле их значения не будут заданы. *Я*зык ***AHDL*** позволяет присваивать значения узлу или группе неоднократно и, если произойдет конфликт, система автоматически будет использовать значение по умолчанию. Если значения по умолчанию не были заданы, то узлам и группам автоматически присваивается значение ***GND*** или все 0. Значение по умолчанию можно использовать и в условных операторах ***IF*** и ***CASE***.

Рассмотрим пример:

Пусть на вход системы подается шестнадцатеричный одноразрядный код. На выходе системы должен появиться соответствующий ***ASCII*** код. А если одновременно поданы сигналы на несколько входов, то устройство должно выдавать «?».

Задание 3. Создать проект в среде QUARTUS II, в проекте создать текстовый файл Default1.TDF. Ввести в него текст. Откомпилировать проект и запустить симуляцию проекта. Пояснить результаты симуляции.

```
SUBDESIGN Default1
{ i [3..0]: INPUT;
   ASCII_CODE OUTPUT;}
BEGIN
  DEFAULTS
    ASCII_CODE [] = b “01111111”; % “?”
END DEFAULTS
TABLE
     I [3..0] => ASCII_CODE;
    B “1000” => B “01100001”; % “a”
    B “0100” => B “01100010”; % “b”
    B “0010” => B “01100011”; % “c”
    B “0001” => B “01100100”; % “d”
END TABLE;
END.
```

Реализация условной логики. Условная логика реализуется с помощью операторов *IF Then* и *Case.*

Оператор *Case* определяет список альтернативных вариантов, один из которых выполняется, если значение селектора (переменной, группы или выражения), стоящего за ключевым словом CASE, соответствует значению, стоящему за ключевым словом WHEN этого варианта. Оператор имеет следующие правила синтаксиса и семантики.

- Вначале идет ключевое слово CASE, да которым следует селектор, далее ключевое слово IS.
- Оператор завершается ключевым словосочетанием END CASE, за которым стоит символ (;).
- Тело оператора представляет собой список альтернативных вариантов, каждый из которых начинается ключевым словом ШНЕМ. Каждый вариант представляет собой набор констант, разделенных запятыми, за которым следует символ =>. За этим символом следует список операторов, разделенных символом (;). Последний вариант может начинаться ключевым словосочетанием WHEN OTHERS.
- Если значение селектора равно одному из значений констант варианта, выполняются все операторы этого варианта. Если значение селектора не равно ни одной из констант, выполняются операторы за словосочетанием WHEN OTHERS (если оно есть).
- Если оператор CASE используется для описания конечного автомата, словосочетание WHEN OTHERS не может использоваться для выхода из неразрешенных состояний.

Задание 4. Создать проект в среде QUARTUS II, в проекте создать текстовый файл decoder.TDF. Ввести в него текст. Откомпилировать проект и запустить симуляцию проекта. Пояснить результаты симуляции.

```
SUBDESIGN decoder
(
  code [1..0]                :INPUT;
  out [3..0]                 :OUTPUT;
)
BEGIN
   CASE code [ ] is
         WHEN 0 => out [ ] = B "0001";
         WHEN 1 => out [ ] = B "0010";
         WHEN 2 => out [ ] = B "0100";
         WHEN 3 => out [ ] = B "1000";
   END CASE;
END;
```

Это описание дешифратора 2→4, который преобразует двухразрядный двоичный код в код «one hot» (четыре значения содержат каждое по одной

единице). В зависимости от кода на входе активизируется одна из ветвей оператора CASE.

Оператор If Then определяет списки операторов, выполняемых в зависимости от значения булева выражения. Оператор имеет следующие правила синтаксиса и семантики.

- Вначале идет ключевое слово IF, за которым следует булево выражение, далее ключевое слово THEM и список операторов, разделенных символом (;).
- Далее между ключевыми словами ELSEIF и THEM может следовать дополнительное булево выражение, а за THEM - список операторов, выполняемых или нет в зависимости от значения этого выражения. Эта необязательная конструкция может повторяться многократно. Операторы за словом THEM выполняются, если булево выражение истинно, при этом последующая конструкция ELSEIF THEM игнорируется.

За конструкцией ELSEIF THEM может следовать ключевое слово ELSE, а за ним список операторов, которые выполняются, если ни одно из булевых выражений ни приняло истинного значения. Значения булевых выражений за ключевыми словами IF и ELSEIF вычисляются последовательно. Оператор завершается словосочетанием END IF и символом (;).

Задание 4. Создать проект в среде QUARTUS II, в проекте создать текстовый файл priority.TDF. Ввести в него текст. Откомпилировать проект и запустить симуляцию проекта. Пояснить результаты симуляции.

```
SUBDESIGN priority
(
        low, middle, high     :INPUT;
        highest_level [1..0] :OUTPUT;
)
BEGIN
  IF higtTHEN
        highest_ level [ ] =3;
  ELSIF middle THEN
        highest_ level [ ] =2;
  ELSIF low THEN
        highest_ level [ ] =1;
  ELSE
        highest_ level [ ] =0;
  END IF;
END;
```

Это шифратор приоритетов. На выходах устанавливается код, соответствующий приоритету входа, на котором имеется значение VCC.

Практическое занятие 7. Язык AlteraHDL составление программ. Последовательностная логика

Цель работы: изучение схем с последовательностной логикой регистров и счетчиков.

Последовательностная логика (конечные автоматы с памятью) описывается в языке AHDL с помощью триггеров, регистров, машин состояний и библиотеки параметрических модулей (LMP).

Регистры. Регистр создается объявлением его в разделе VARIABLE или ссылкой в строке в разделе LOGIC. После определения регистра его можно подключить к другой логике при помощи его портов. Обращение к портам регистра производится по имени регистра и имени порта:

<имя регистра>.<имя порта>

В следующем примере сформирован восьмиразрядный регистр из триггеров D-типа, который защелкивает данные с входов d по положительному фронту синхросигнала при положительном активном значении на входе разрешения.

Задание 1. Создать проект в среде QUARTUS II, в проекте создать текстовый файл bur_reg.TDF. Ввести в него текст. Откомпилировать проект и запустить симуляцию проекта. Пояснить результаты симуляции.

```
SUBDESIGN bur_reg
(
  clk, load, d [7..0]  :INPUT;
  q [7..0]           :OUTPUT;

)
VARIABLE
  ff [7..0]          :DFFE;

BEGIN
  ff [ ] .clk = clk;
  ff [ ] .ena = load;
  ff [ ] .d = d [ ];
  q [ ] =ff [ ] .q;
END;
```

Первое уравнение логической секции соединяет вход тактового сигнала проекта с тактовыми входами триггеров. Во втором уравнении подключается разрешающий сигнал, в третьем уравнении подключаются входы, а в четвертом выходы. Altera рекомендует использовать в качестве тактирующего глобальный синхросигнал.

Можно объявить выходы проекта как D-триггеры в разделе VARIABLE. Это дает экономию за счет использования триггеров на выводах микросхемы.

Задание 2. Создать проект в среде QUARTUS II, в проекте создать текстовый файл reg_out.TDF. Ввести в него текст. Откомпилировать проект и запустить симуляцию проекта. Пояснить результаты симуляции.

```
SUBDESIGN reg_out
(
  clk, load, d [7..0]  :INPUT;
  q [7..0]:OUTPUT;
)
VARIABLE
  q [7..0]:DFFE; % выходы объявлены как регистры %

BEGIN
  q [ ] .clk = clk;
  q [ ] .ena = load;
  ff [ ] .d = d [ ];
  q [ ] =d [ ];
END;
```

Счетчики. Счетчики обычно реализуются на D-триггерах с использованием оператора IF. В следующем задании реализован 16-разрядный загружаемый счетчик со сбросом.

Задание 3. Создать проект в среде QUARTUS II, в проекте создать текстовый файл ahdlcnt.TDF. Ввести в него текст. Откомпилировать проект и запустить симуляцию проекта. Пояснить результаты симуляции.

```
SUBDESIGN ahdlcnt
(
  clk, load, ena, clr, d[15..0] : INPUT;
  q [15..0]                     :OUTPUT;
)
VARIABLE
  count[15..0]                  : DFF;
BEGIN
  count [ ] .clk = clk;
  count [ ] .clm = !clr;
IF load THEN
  count [ ] .d = count [ ] .q+1;
ELSE
  count [ ] .d = count [ ] .q;
  count Q .d = count)].q;
END IF;
  q [ ] = count [ ];
END;
```

Практическое занятие №8. Изучение системы проектирования QUARTUS8.1

1 Цель работы: Целью работы является изучение способов создания цифровых устройств (основных логических элементов) на основе ПЛИС в среде Quartus II.

2 Краткие сведения о системе автоматизированного проектирования цифровых устройств – Quartus II

2.1 ПЛИС

Программируемые логические интегральные схемы – ПЛИС являются одними из самых перспективных элементов цифровой схемотехники. ПЛИС представляет собой кристалл, на котором расположено большое количество простых логических элементов. Изначально эти элементы не соединены между собой. Соединение элементов (превращение разрозненных элементов в электрическую схему) осуществляется с помощью электронных ключей, расположенных в этом же кристалле. Электронные ключи управляются специальной памятью, в ячейки которой заносится код конфигурации цифровой схемы. Таким образом, записав в память ПЛИС определенные коды, можно собрать цифровое устройство любой степени сложности (это зависит от количества элементов на кристалле и параметров ПЛИС). В отличие от микропроцессоров, в ПЛИС можно организовать алгоритмы цифровой обработки на аппаратном (схемном) уровне. При этом быстродействие цифровой обработки резко возрастает. Достоинствами технологии проектирования устройств на основе ПЛИС являются:

- минимальное время разработки схемы (нужно лишь занести в память ПЛИС конфигурационный код);
- в отличие от обычных элементов цифровой схемотехники здесь отпадает необходимость в разработке и изготовлении сложных печатных плат;
- быстрое преобразование одной конфигурации цифровой схемы в другую (замена кода конфигурации схемы в памяти);
- для создания устройств на основе ПЛИС не требуется сложное технологическое производство. ПЛИС конфигурируется с помощью персонального компьютера на столе разработчика. Потому иногда эту технологию называют «фабрикой на столе».

2.2 Система автоматизированного проектирования Quartus II

Одним из мировых лидеров по производству ПЛИС является фирма Altera. Для создания цифровых устройств на основе своих изделий Altera разработала специальную программную среду Quartus II. Эта среда позволяет:

- с помощью графического редактора ввести в память персонального компьютера электрическую схему;
- проверить и исправить ошибки;
- определить параметры и характеристики разработанного устройства;
- сформировать файл конфигурации для конкретной ПЛИС;
- загрузить этот файл в память интегральной схемы.

2.2.1 Создание проекта

Работа в среде Quartus II начинается с действий, которые называют созданием проекта. Прежде всего, необходимо создать папку для хранения файлов проекта. Имя папки желательно вводить латинскими буквами. Затем следует запустить программу Quartus II. Открыв пакет, Quartus II выбираем из меню File пункт ***New Project Wizard*** - мастер создания новых проектов. В открывшемся окне нажимаем кнопку ***Next*** и попадаем в окно для задания текущей директории проекта. Заполняем три строки как, показано на рисунке 2.1 .

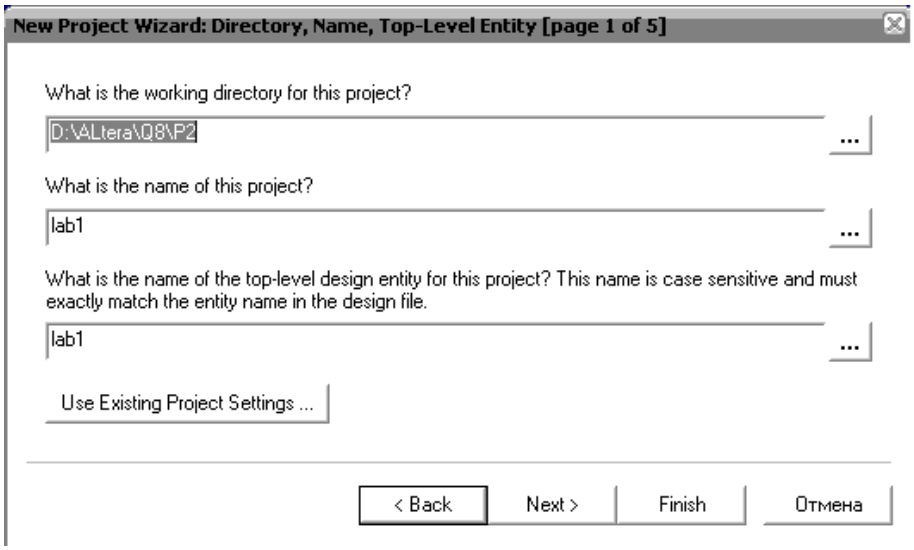


Рисунок 2.1 - Меню задания текущей директории проекта

В данном случае текущий проект будет назван LAB1. Проект будет создан в папке D\altera\Q8\P2 на рабочем столе. Далее необходимо выбрать тип микросхемы (рисунок 2.2) используемой в проекте, например, EPM7032STc44. После нажатия кнопки ***Finish*** подтверждается создание проекта.

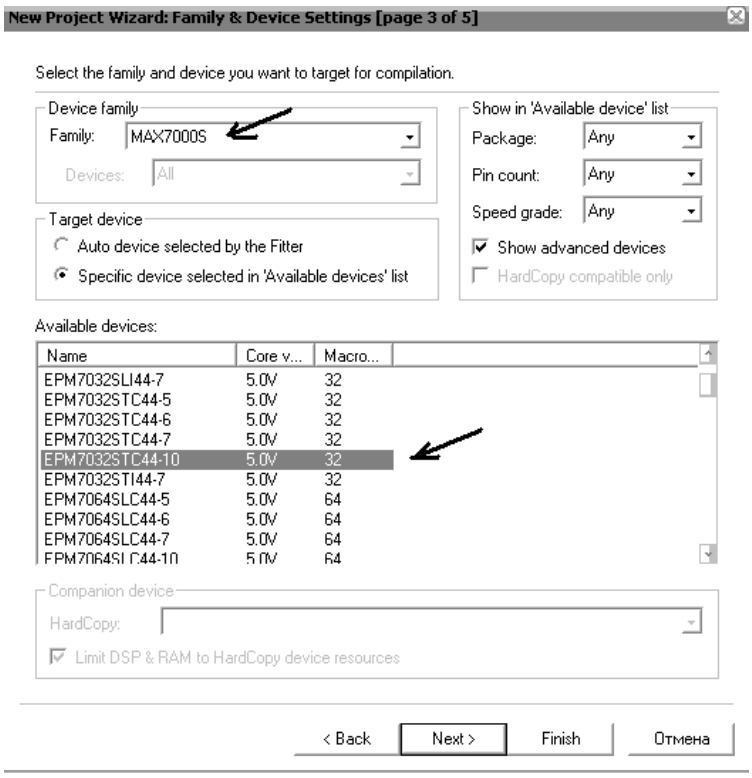


Рисунок 2.2. Выбор типа микросхемы

В созданный таким образом проект необходимо добавить файл. Для этого выберем File – New – Block Diagram/ Schematic File, как показано на рисунке 2.3.

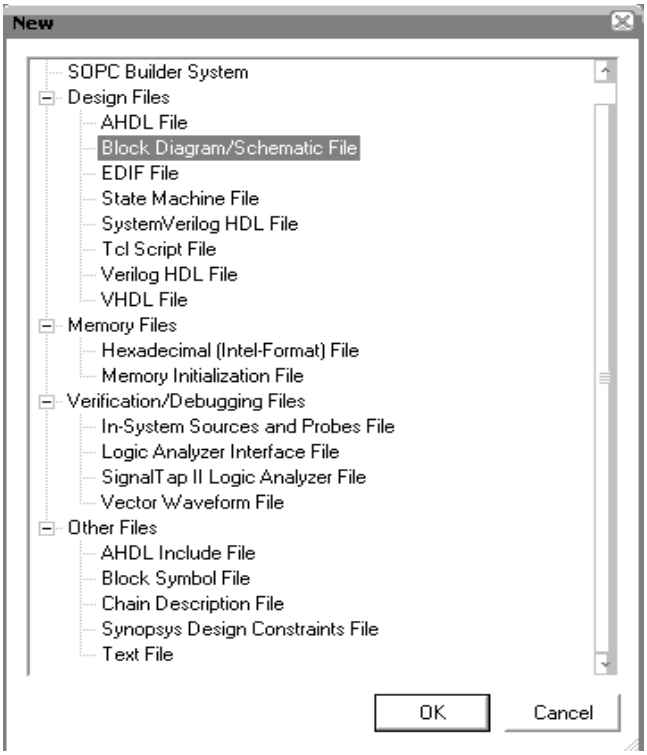


Рисунок 2.3. Выбор типа файла

2.2.2 Работа в графическом редакторе

Графический редактор предназначен для ввода принципиальной схемы устройства. Для создания файла, который будет содержать принципиальную схему устройства (после создания проекта) следует выполнить команду ***New*** меню File. В появившемся окне выберем Block Diagram/Schematic File

Далее выберем тип используемой микросхемы, например EPM7032STC44-10.

В результате откроется окно графического редактора с файлом *Block1.bdf*, в котором создается схема (Рисунок 2.3).

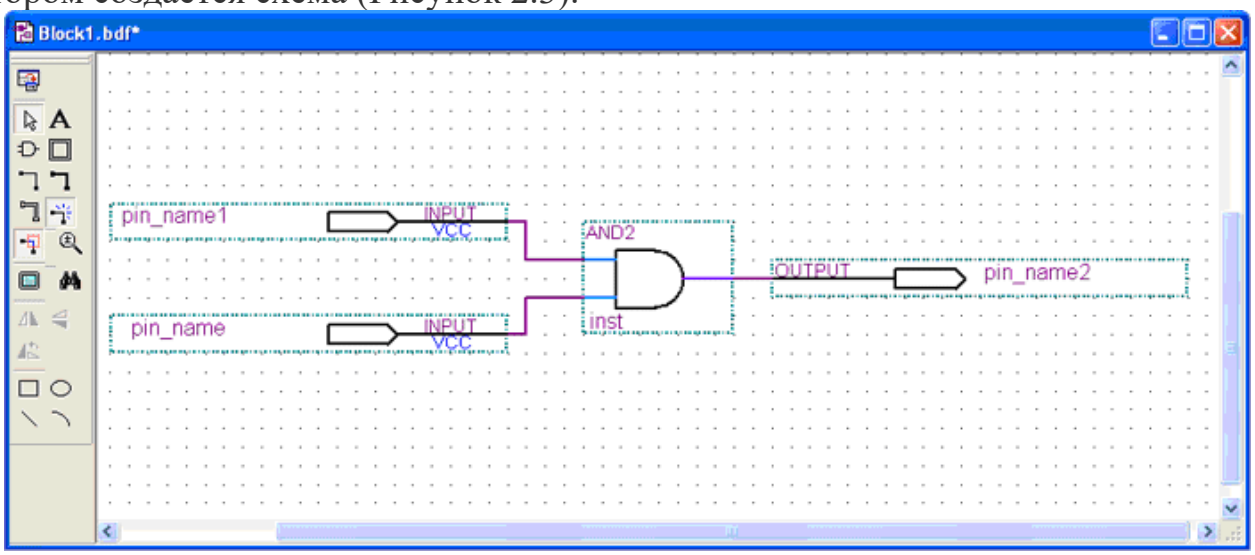


Рисунок 2.4 - Окно графического редактора.

После создания файла проекта становится активной панель инструментов, расположенная слева от рабочей области окна. Для ввода элемента схемы следует «щелкнуть» по ***Symbol Tool*** (Рисунок 2.4). В результате откроется окно с библиотеками элементов. Следует обратить внимание, что графические изображения элементов схемы в системе Quartus II отличаются от обозначений, принятых в России. В таблице 1 приведены российские изображения и соответствующие им обозначения элементов в системе Quartus II.

Например, для ввода логического элемента «И» следует выбрать библиотеку ***primitives/logic***. После размещения компонентов на схеме следует разместить входные (input) и выходные (output) контакты, которые находятся в папке ***primitives/pin***. Соединение компонентов производится следующим образом: переместить курсор в одну из двух точек схемы, которые нужно соединить, нажать левую кнопку мыши и, не отпуская ее, перемешать курсор ко второй из соединяемых точек. Далее следует переименовать входные и выходные контакты. Для этого дважды щелкаем левой кнопкой мыши по имени контакта и редактируем его. В результате получим схему, изображенную на рисунке 2.5.

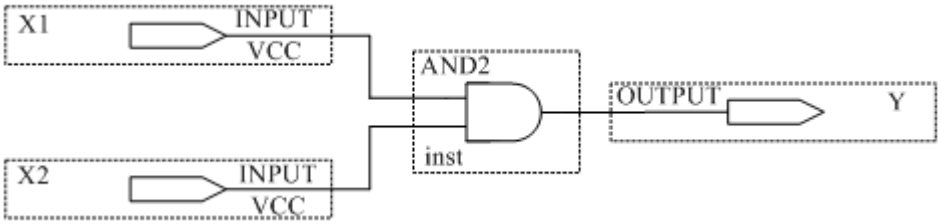
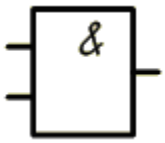
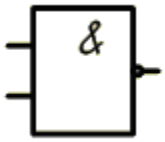
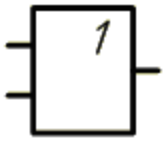
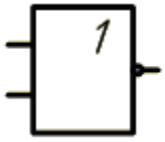
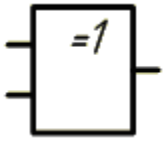
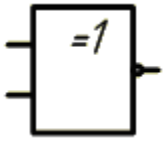
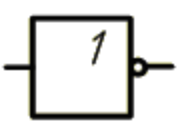


Рисунок 2.5 - Схема после переименования входов и выходов.

После этого сохраняем получившийся файл File - Save – Lab1

Таблица 2.1 – Соответствия российских обозначений элементов и обозначений в Quartus II.

Название элемента	**Российское обозначение**	**Название элемента в Quartus**	**Обозначение в Quartus**
«И»		and	
«И-НЕ»		nand	
«ИЛИ»		or	
«ИЛИ-НЕ»		nor	
«ИСКЛЮЧАЮЩЕЕ ИЛИ»		xor	
«ИСКЛЮЧАЮЩЕЕ ИЛИ-НЕ»		xnor	
«НЕ»		not	

2.2.3 Компиляция проекта

Для запуска процесса компиляции выберем пункт ***Start Compilation*** из меню ***Processing***. Подтвердим сохранение текущего файла и ожидаем окончания процесса компиляции. По окончании компиляции появляется окно с сообщением о результатах компиляции и количестве ошибок и предупреждений.

2.2.4 Подключение схемы к внешним выводам ПЛИС

После того, как произведена компиляция проекта следует подключить входные и выходные контакты к внешним выводам ПЛИС. Для этого следует выбрать ***Assignments/Pins***. В результате появится окно (Рисунок 2.5)

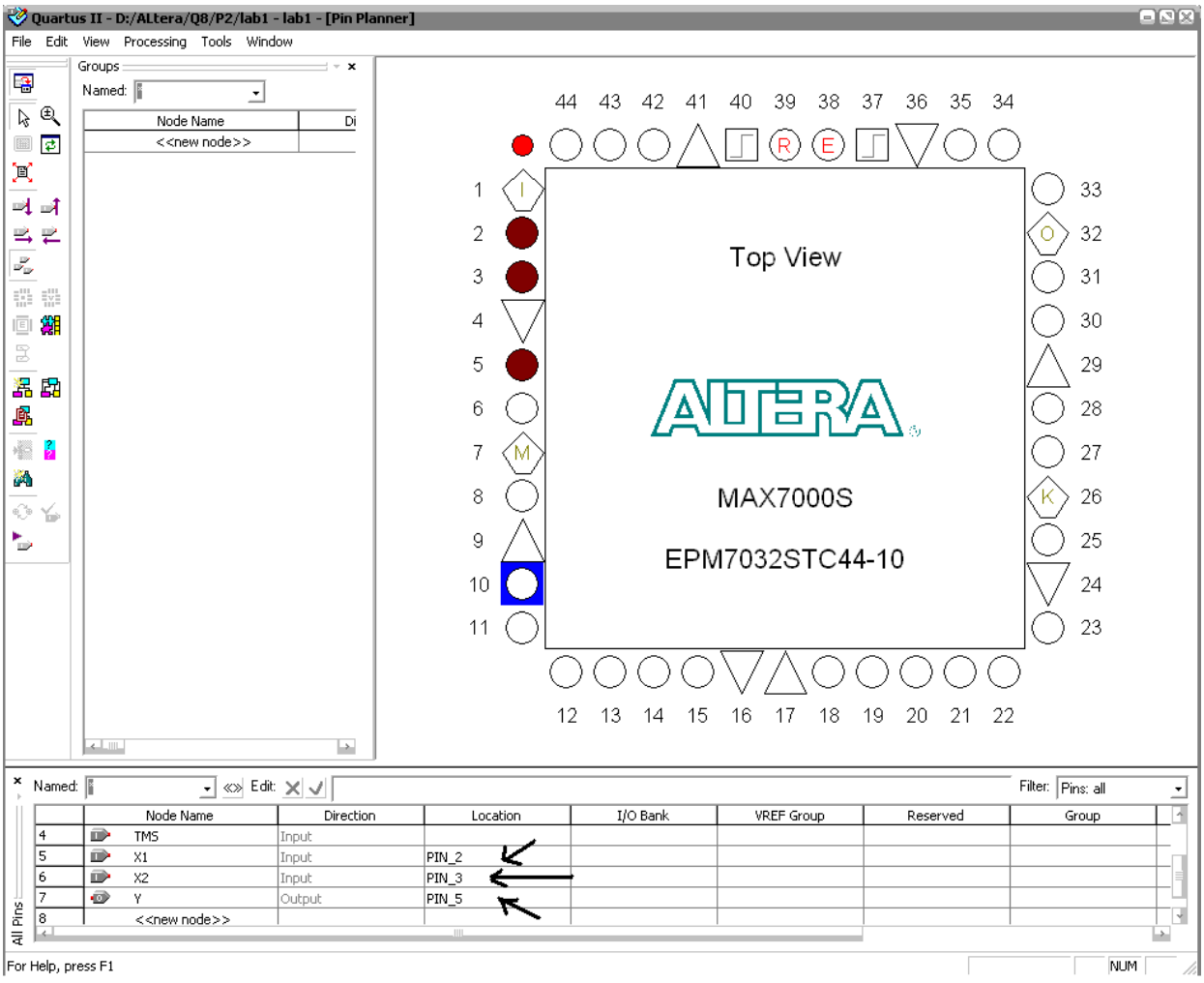


Рисунок 2.5 – Окно подключения схемы к внешним выводам

В колонке ***Node Name*** располагаются имена входов и выходов схемы. Для их подключения к выводам ПЛИС следует дважды «щелкнуть» по соответствующему элементу в колонке ***Location*** и выбрать вывод, к которому нужно подключить вход (выход) электрической схемы. Можно также просто переместить соответствующее имя (например, X1) на вывод, изображенный на рисунке ПЛИС. После подключения всех выводов следует еще раз

скомпилировать проект. В результате требуемая схема примет вид, изображенный на рисунке 2.6.

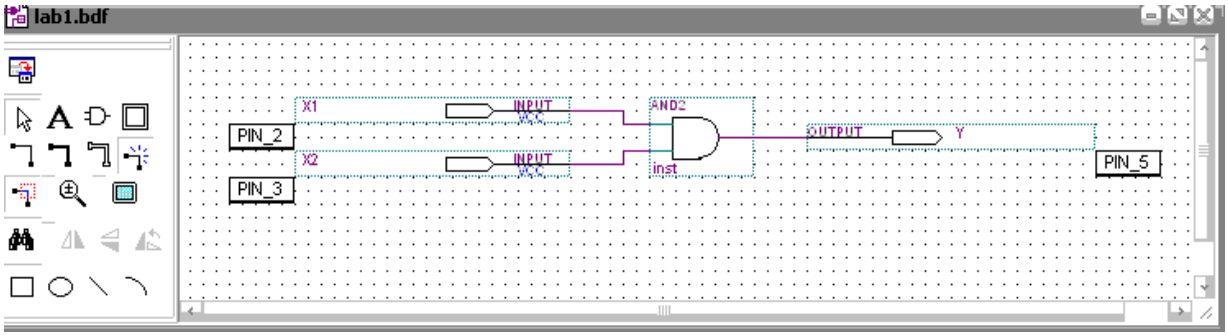


Рисунок 2.6 – Схема после подключения внешних выводов.

2.2.4. Проведение моделирования работы схемы.

Для начала моделирования необходимо создать файл временных диаграмм File – New – Vector Waweform File. Далее, необходимо добавить сигналы в проект, как показано на рисунке 2.7. в колонке Name правой кнопкой мыши выбираем Insert – Insert Node or Bus

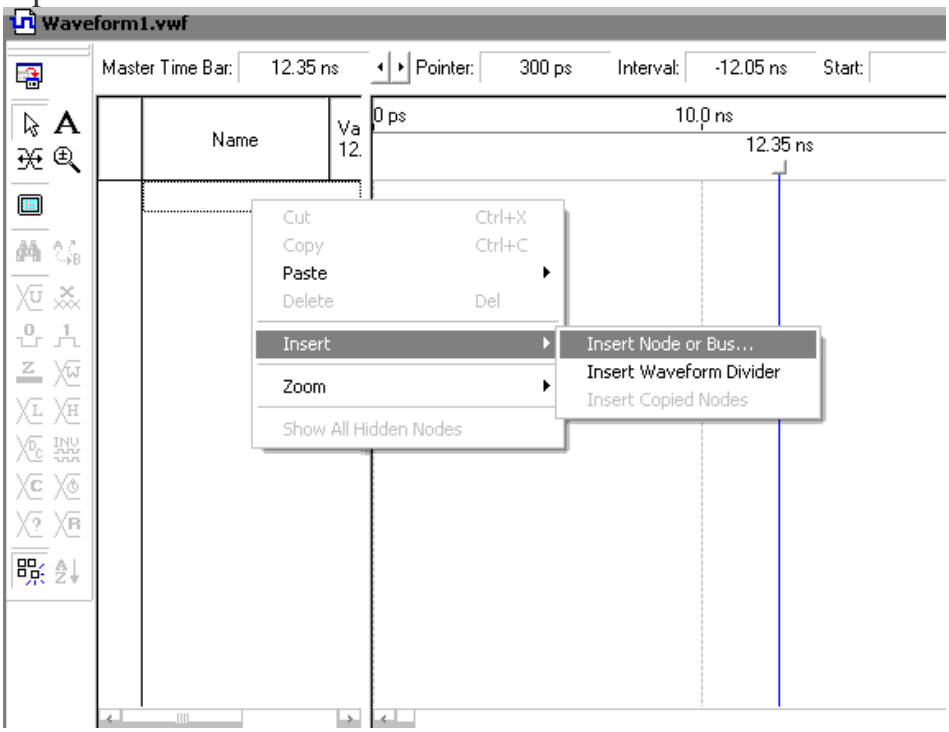


Рисунок 2.7. Вызов окна поиска цепей

Далее, как показано на рисунке 2.8. последовательно выбираем Node Finder - List - >> - Ок – Ok.

Как видим наши сигналы появились во временной диаграмме. щелкнув правой кнопкой на сигнале выберем Value – Clock.. и в появившемся окне рисунок 2.9. введем период импульсов 100нс для сигнала X1 и 200 нс для сигнала X2. Далее выберем инструмент Zoom Tool и уменьшим масштаб сигнала.

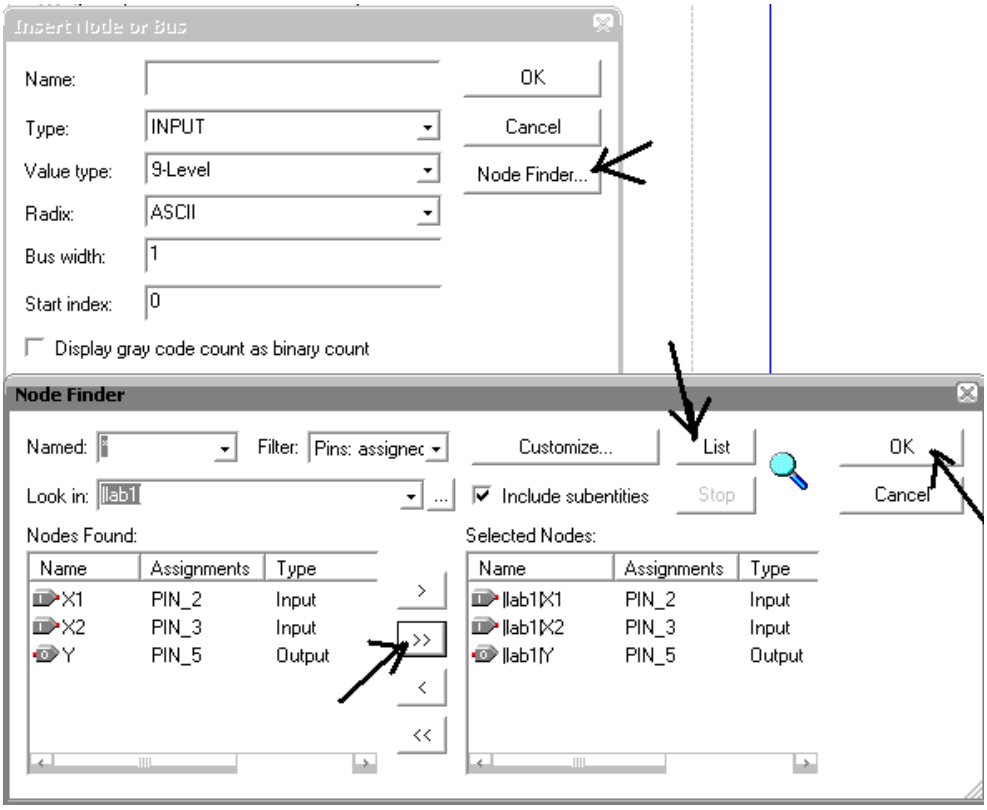


Рисунок 2.8. Добавление цепей во временную диаграмму.

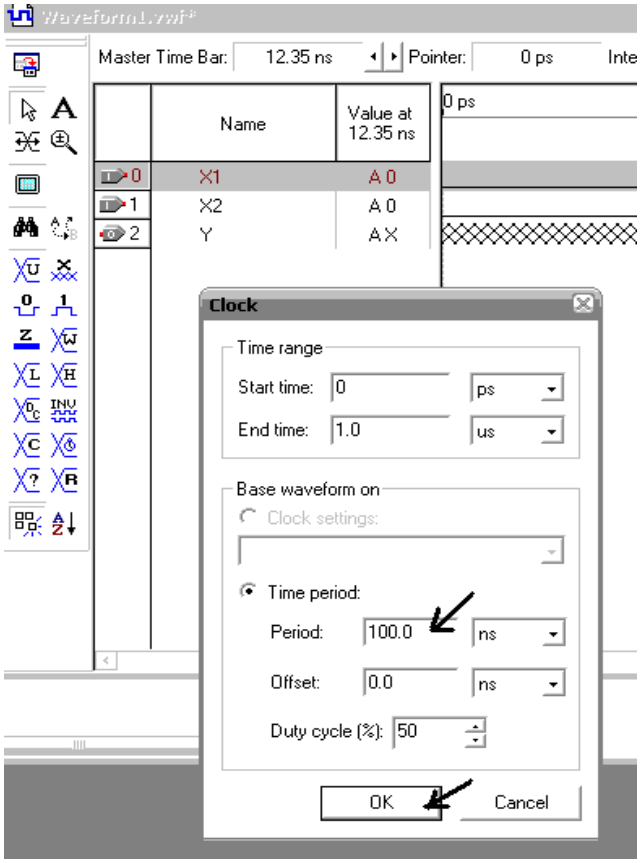


Рисунок 2.9. Задание частоты входных сигналов.

Сохраним полученный результат под именем LAB 1. В результате получим рис 2.10.

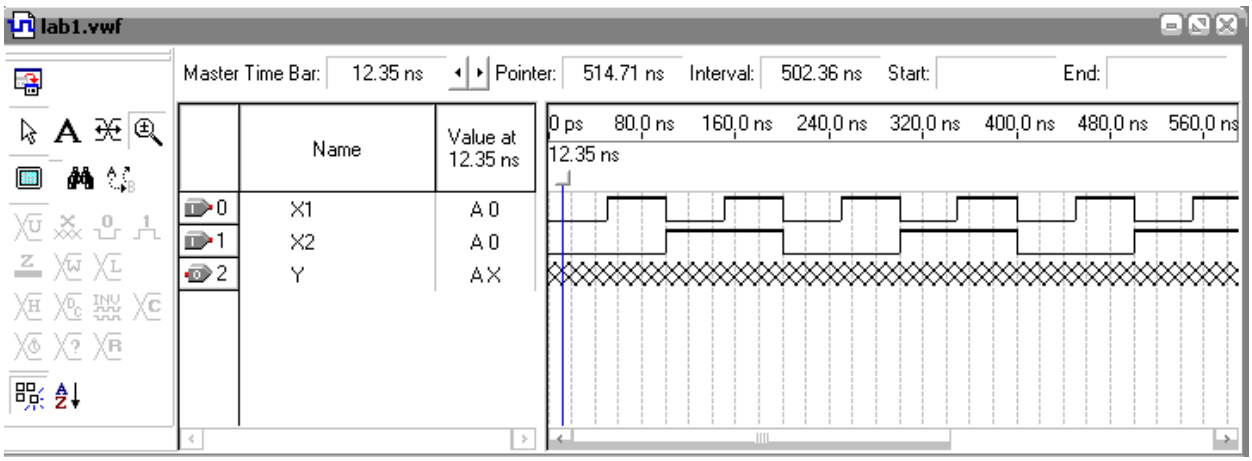


Рис2.10 Окно временной диаграммы до моделирования.

Для начала моделирования выберем Processing – Start Simulation. Результаты моделирования выдаются в специальном окне Simulation report представленном на рисунке 2.11.. как видно элемент И работает в соответствии со своей таблицей истинности.

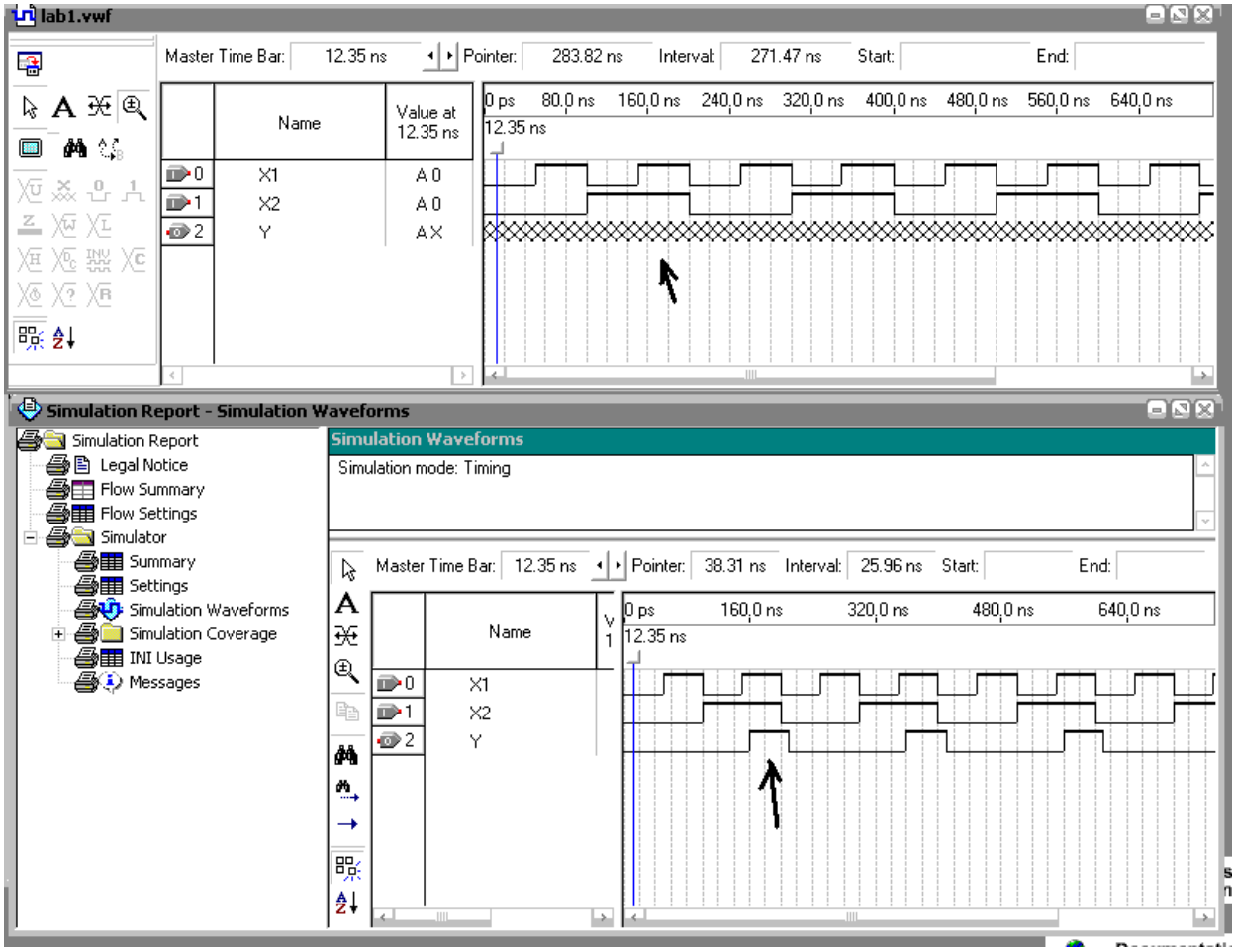


Рисунок 2.11. Результаты моделирования.

По окончании создания проекта его можно запрограммировать с использованием окна Programmer на панели инструментов.

3. Задания для работы

Сформировать проект и провести моделирования для всех базовых логических элементов.

4. Контрольные вопросы

1. Назначение системы автоматического проектирования QUARTUS 8.1;
2. Охарактеризуйте основные функции системы QUARTUS 8.1;
3. Порядок создания проекта в системе QUARTUS 8.1;
4. Что такое программируемая логическая интегральная схема.

5. Литература

1. Проектирование арифметических устройств цифровой обработки сигналов с использованием сапр quartus ii и ее библиотеки мегафункций/ НГТУ; Сост.: А.Д.Плужников, Н.Н.Потапов, А.А.Цветков. Н.Новгород, 2005. 22 с.

Практическое занятие №9. Создание простейшего проекта с использованием графического редактора системы QUARTUS8.1

1 Цель работы: Изучение графического редактора системы QUARTUS 8.1

2. Порядок работы:

2.1. Создайте проект lab2 с главным файлом lab2.bdf на базе микросхемы EPM240T100. Окно проекта представлено на рисунке 1.

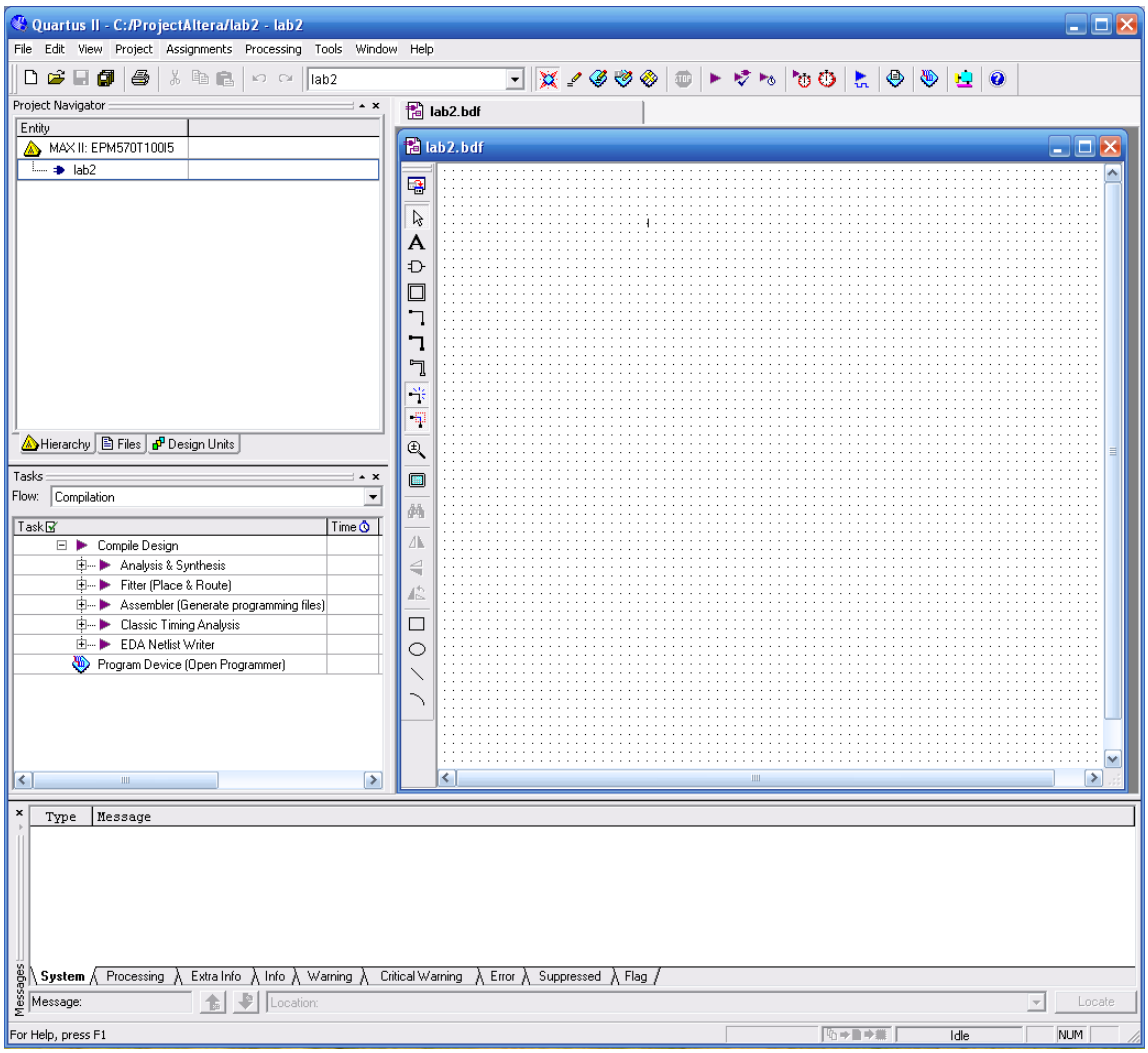


Рис.1. Окно проекта.

2.2. Элементы графического проекта

2.2.1. Выводы

Для того чтобы использовать вывод микросхемы в проекте, нужно с помощью меню "*Symbol tool*" установить примитива - "Bidir", "Tri","Input","Output" в проект. Для запуска меню "*Symbol*" можно нажать два раза левой кнопкой мыши на свободном поле или соответствующую кнопку в меню редактора.

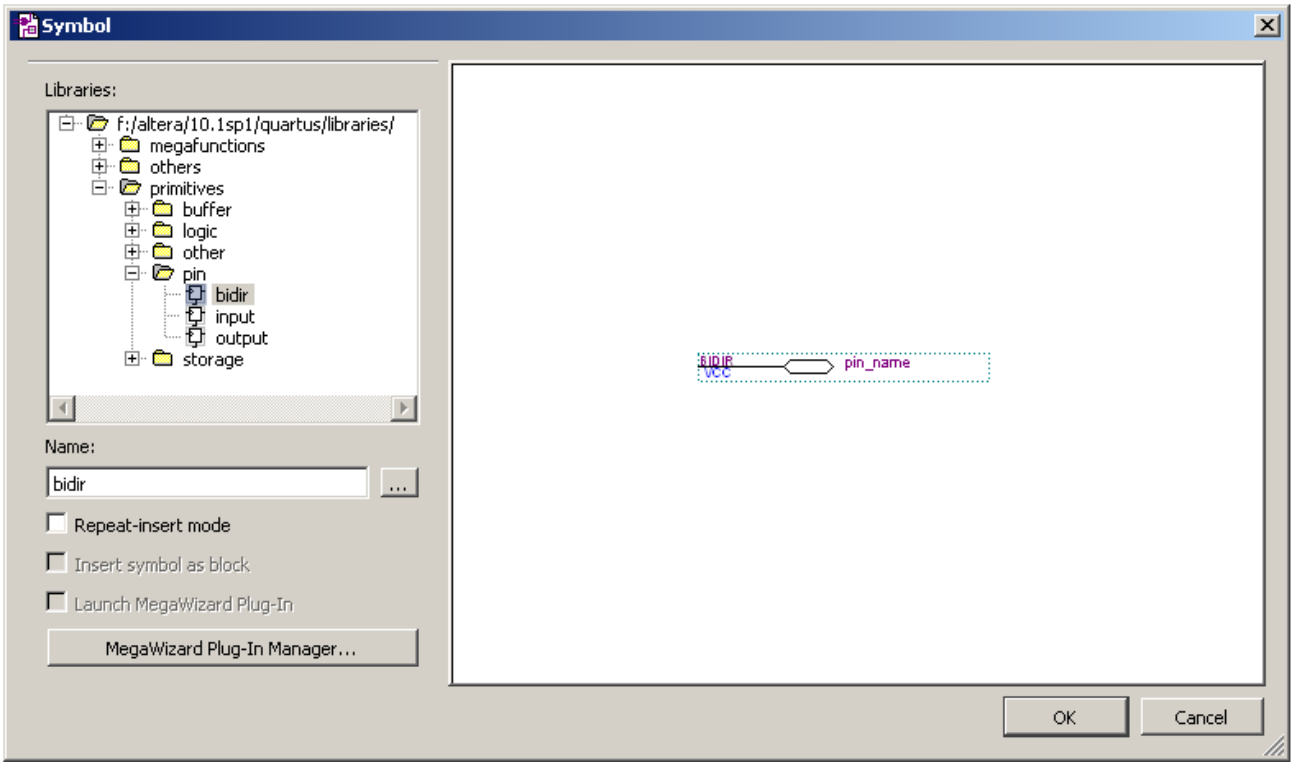
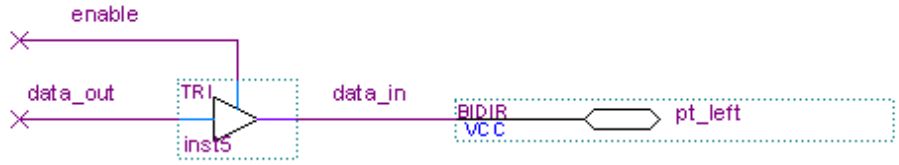


Рис. 2 Выбор элемента в меню «Symbol tool»

Нужный нам примитив можно выбрать в библиотеке или ввести его название с строке "Name:". Сформируем примитивы по типу:



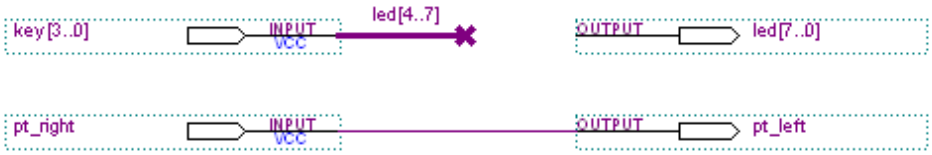
"*Bidir*" – описывает двунаправленный вывод.

"*Opndrn*" – описывает вывод с открытым коллектором

.

"*Input*" - когда выходной буфер всегда находится в "Z"- состоянии и используется только вход.

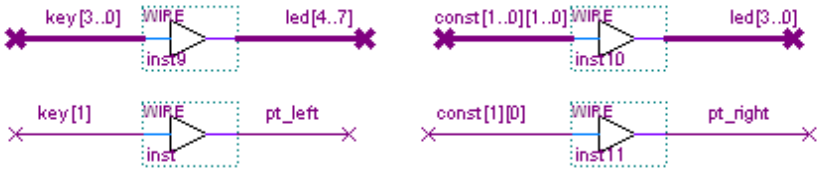
"*Output*" - когда он всегда разрешён, и данные сразу поступают на выход (вход использовать нельзя).



Примитивы "*Input*" ,"*Output*" и "*Bidir*" могут описывать сразу несколько выводов и могут быть поименованы соответственно (key[3..0] , pt_left и т.д.). Тогда они описываются как массив выводов.

2.2.2. Проводники

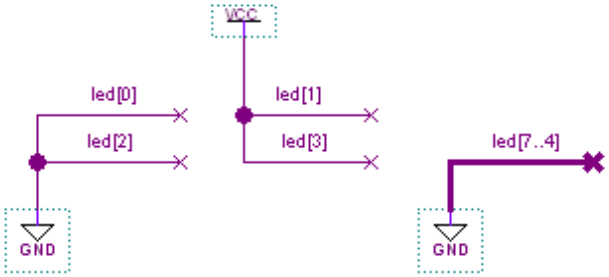
Проводники могут быть одиночные ("*Node Line*") или шины ("*Bus Line*") и тоже могут иметь свои имена. Если имена проводников совпадают с именами пинов значит они соединены. В некоторых случаях проводники нужно переименовывать. Для этого можно использовать примитив "*Wire*". Сформируем примитивы как показано ниже:



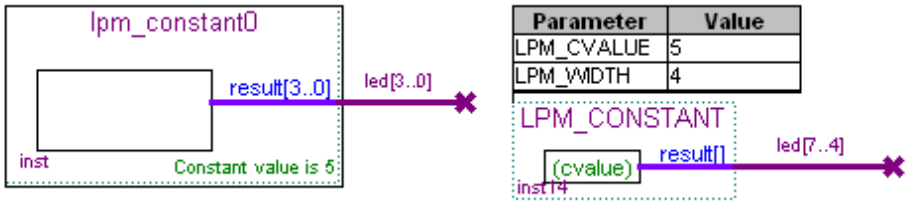
Имена шин могут иметь две размерности (в некоторых случаях это удобно) и менять порядок бит (на выходе одного элемента, например, *bus[3..0],* а на входе другого *bus[0..3]*). Размерность шины должна точно совпадать с размерностью входа или выхода, к которому она подключена.

2.2.3. Константы

Иногда, на вход какого-нибудь компонента или на вывод, нужно подать какое-то, не меняющееся во времени, значение (константу). Для этого можно использовать примитивы "*GND*" и "*VCC*". Сформируем примитивы как показано ниже:



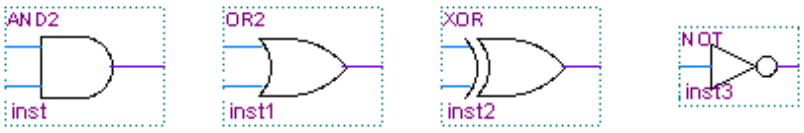
В случае, если константа нужна большая, чтобы не рисовать много "*GND*" и "*VCC*" можно использовать параметризованную мегафункцию "*LPM_CONSTANT*" (здесь и далее "мегафункция"- некоторое количество связанных между собой примитивов, реализующее определенную функциональность. Обычно называется LPM_XXX). Сформируем примитивы как показано ниже:



Установить константу можно , как и примитивы, с помощью меню "*Symbol*", но двумя способами:
С помощью помошника. И, если в меню "*Symbol*" убрать галочку "*Launch MegaWizard Plug-In*", то можно установить так-называемую "чистую" мегафункцию. В этом случае все параметры придётся устанавливать "в ручную", в меню "*Properties*". Эти параметры видны в табличке возле элемента.

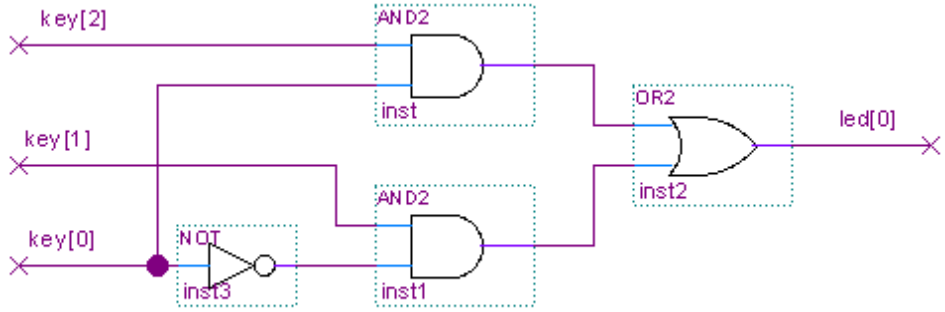
2.2.4. Логические элементы

Для описания логических функций в проекте можно использовать базовые примитивы "AND", "OR", "XOR" и "NOT":



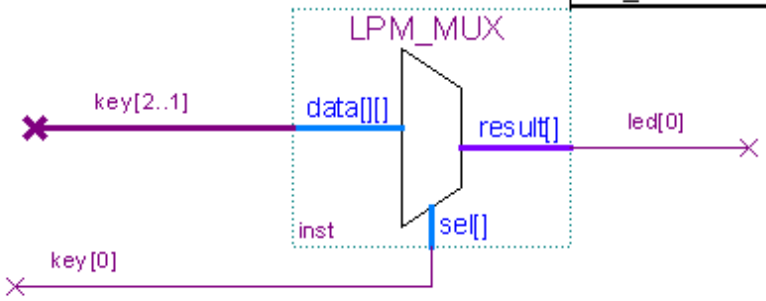
2.2.4. Комбинационная логика
2.2.4.1. Мультиплексор

Сформируем примитивы как показано ниже:

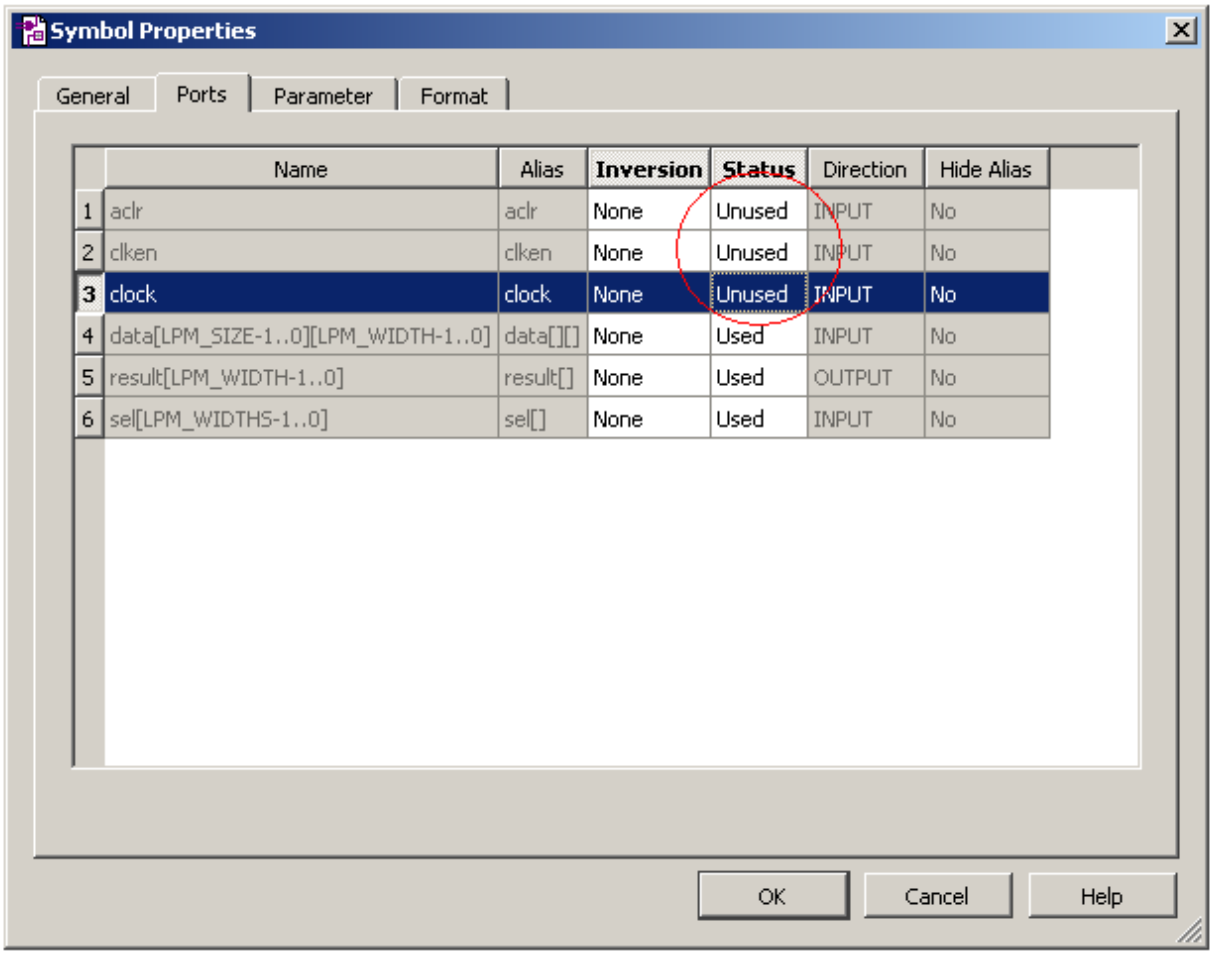


Данная схема – мультиплексор на 2 входных канала. В зависимости от состояния адресного входа *key[0]*, на выходе будет или сигнал *key[1]*, или *key[2]*. Также для создания мультиплексора можно использоватьмегафункциюLPM_MUX:

Parameter	Value
LPM_PIPELINE	
LPM_SIZE	2
LPM_WIDTH	1
LPM_WIDTHS	CEIL(LOG2(LPM_SIZE))



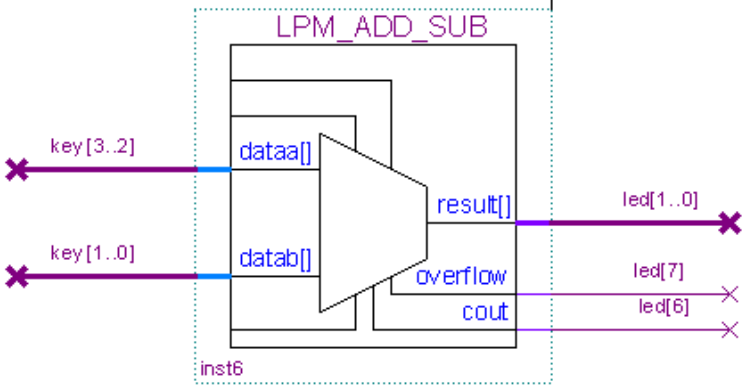
В отличии от использования LPM_CONSTANT , в меню "*Properties*", нужно не только задать параметры (в данном случае LPM_WIDTH и LPM_SIZE), но и отключить входы регистровой логики (aclr,clken и clock). Это нужно для того, чтобы компонент содержал только комбинаторную логику:



2.2.4.2. Сумматор (вычитатель)

Сумматор или вычитатель можно построить на базе мегафункции LPM_ADD_SUB:

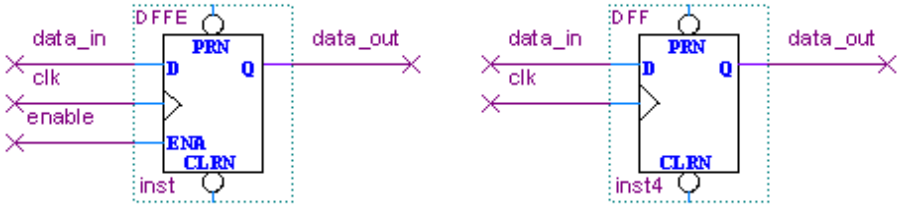
Parameter	Value
LPM_DIRECTION	
LPM_PIPELINE	
LPM_REPRESENTATION	
LPM_WIDTH	2
MAXIMIZE_SPEED	
ONE_INPUT_IS_CONSTANT	



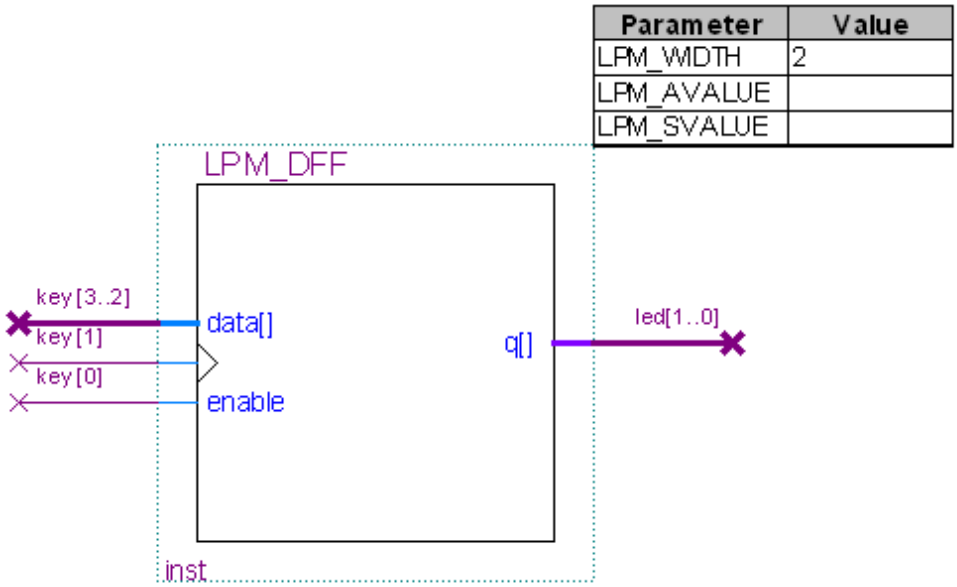
На *led[1..0]* будет арифметическая сумма сигналов *key[1..0]* и *key[3..2]*.

2.2.5. Последовательностная логика

Реализация D триггера. Для реализации в схемотехническом дизайне в среде Altera Quartus II можно использовать примитив "*DFFE*", или его частный случай (когда запись разрешена всегда) "*DFF*":

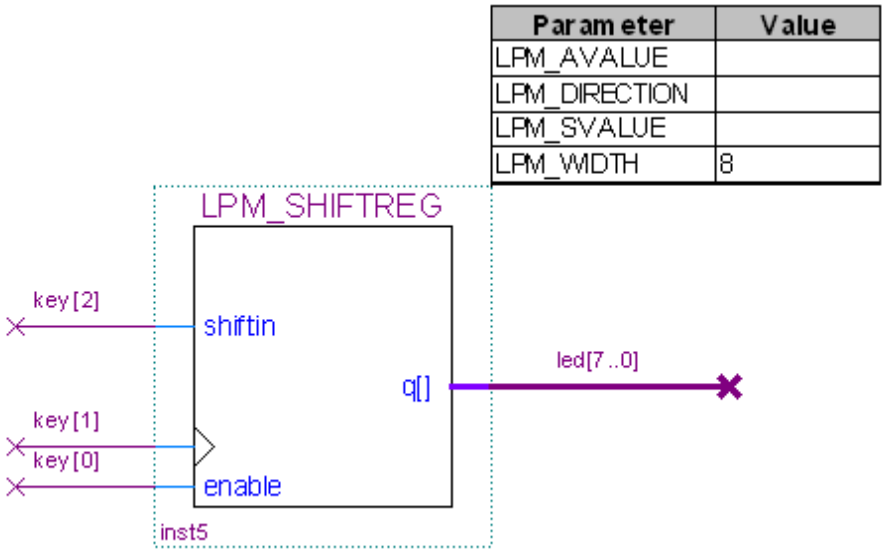


Для создания регистров можно использовать мегафункцию. Мегафункция LPM_DFF - параллельный регистр:



Здесь "*data[]*" - входы всех триггеров, "*q[]*" их выходы, а сигналы клока и разрешения записи для всех триггеров общие.

Мегафункция LPM_SHIFTREG - сдвиговый (последовательный) регистр:



В отличие от параллельного регистра , здесь "*shiftin*" является входом для самого младшего триггера, а выход этого триггера является входом для следующего и т.д. Сигналы клока и разрешения записи, так-же как и в параллельном регистре, для всех триггеров общие.

3. Задания для работы

Сформировать графический проект и провести моделирование работы устройства в соответствии с вариантом.

№	Содержание	Примечание
1	Мультиплексор на 4 канала	
2	Демультиплексор на 4 канала	
3	Трехразрядный асинхронный счетчик	
4	Трехразрядный синхронный счетчик	
5	4 Разрядный регистр сдвига	
6	4 разрядный параллельный сумматор	
7	4 разрядный параллельный вычитатель	
8	Трехразрядный асинхронный счетчик с модулем счета 5	

4. Контрольные вопросы

1. Порядок работы в графическом редакторе QUARTUS II
2. Реализация выводов в графическом редакторе QUARTUS II
3. Реализация проводников в графическом редакторе QUARTUS II
4. Реализация базовых логических в графическом редакторе QUARTUS II
5. Реализация последовательностных схем в графическом редакторе QUARTUS II
6. Реализация комбинационных схем в графическом редакторе QUARTUS II

5. Литература

1. Проектирование арифметических устройств цифровой обработки сигналов с использованием сапр quartus ii и ее библиотеки мегафункций/ НГТУ; Сост.: А.Д.Плужников, Н.Н.Потапов, А.А.Цветков. Н.Новгород, 2005. 22 с.

Практическое занятие №10. Создание простейшего проекта с использованием языка AHDL и текстового редактора системы QUARTUS8.1

1 Цель работы: Изучение графического редактора системы QUARTUS 8.1

2. Порядок работы:

2.1.1 Создайте проект lab3 с главным файлом lab3.bdf на базе микросхемы EPM240T100. Окно проекта представлено на рисунке 1.

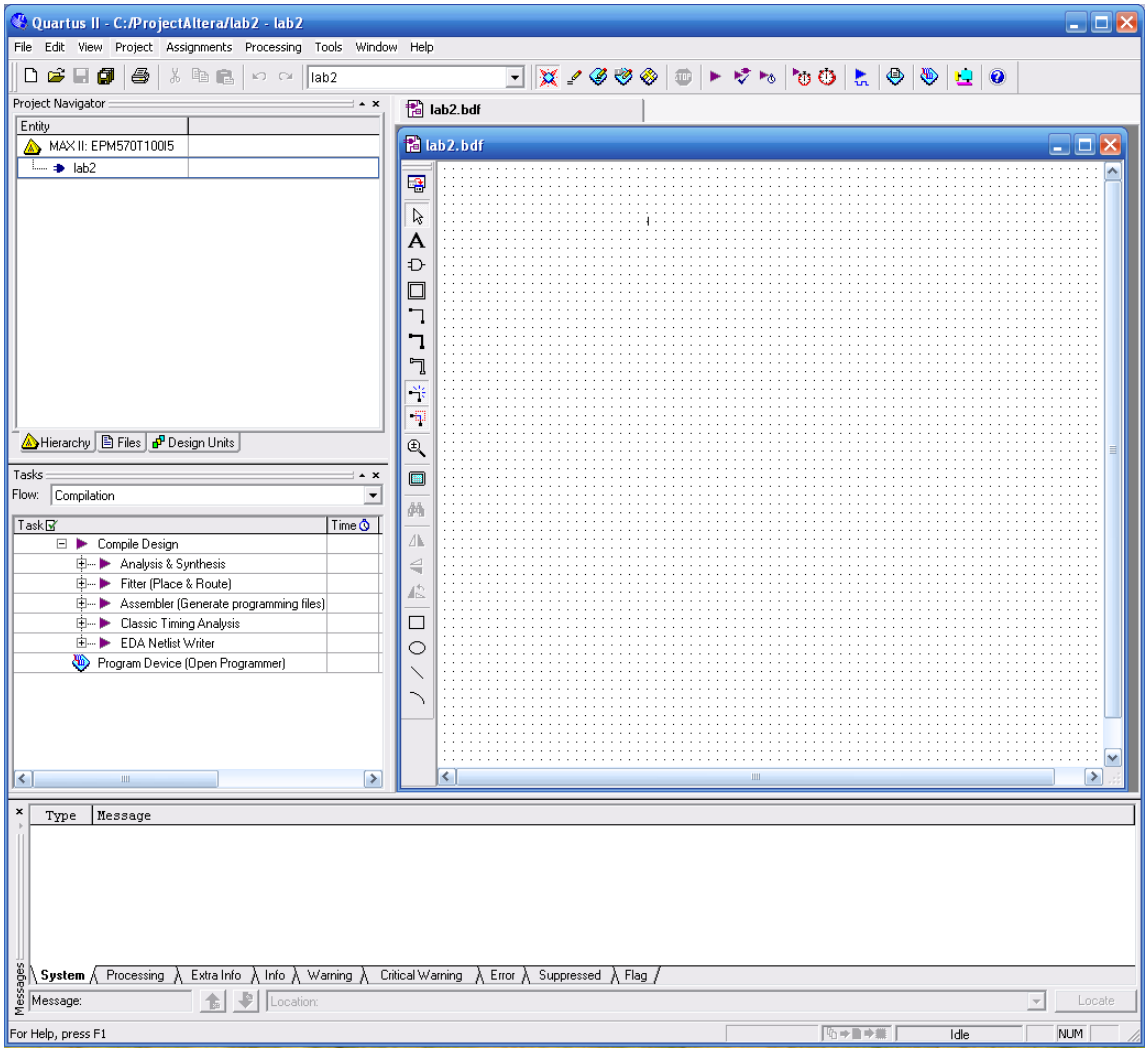


Рис.1. Окно проекта.

2.1.2 Создайте новый AHDL файл

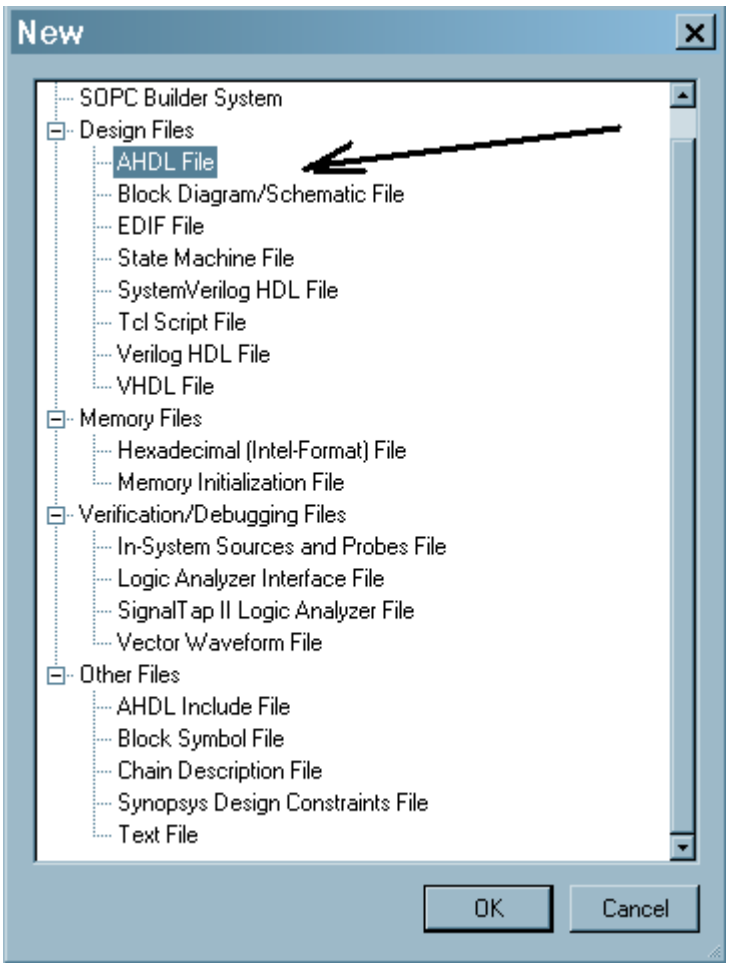


Рисунок 2 Создание нового файла.

2.1.3 Добавьте в файл текст программы из пункта 2.2.1.

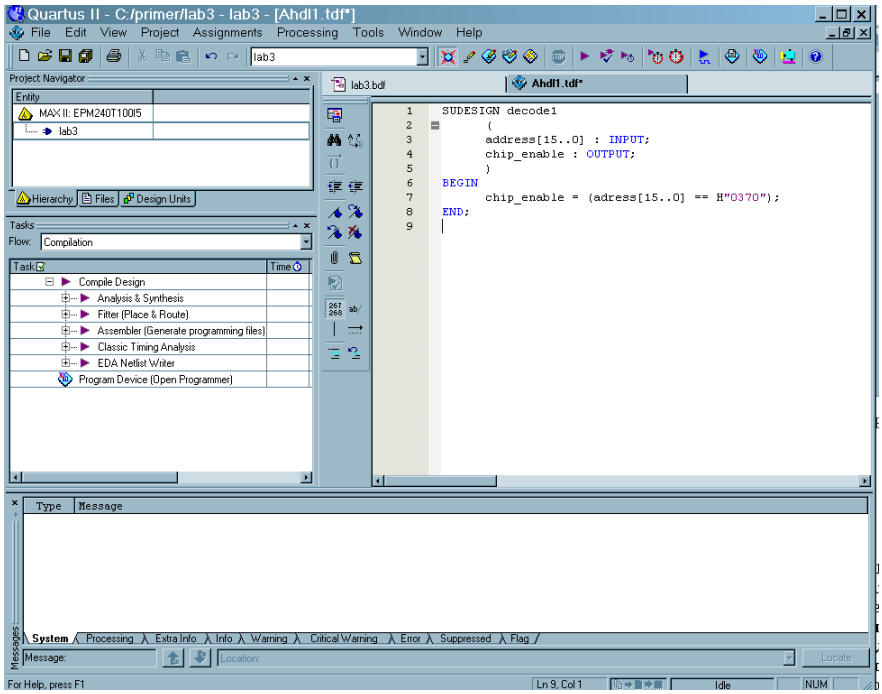


Рисунок 3 Добавление текста программы.

2.1.4 Сохраните файл под именем *decode1.tdf*

2.1.5 Выберите File – Greate/Update –Create symbol file for current file.

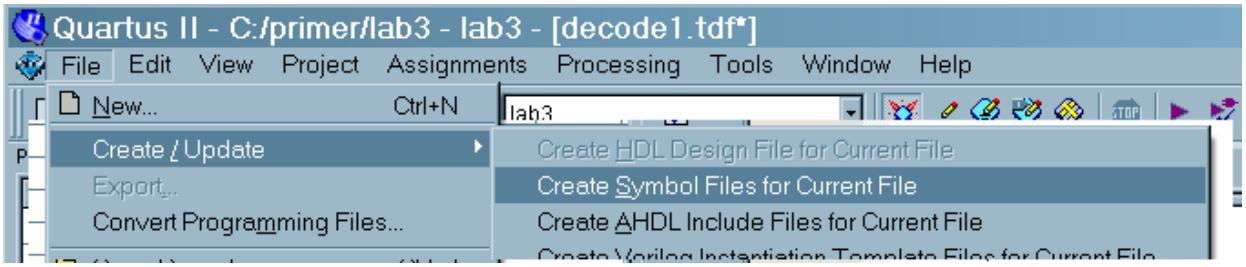


Рисунок 4. Создание примитива.

2.1.6 После успешной компиляции созданный компонент появляется в графическом редакторе. Выберите Symbol toll , в раскрывающемся списке выберите Project-decode1-OK

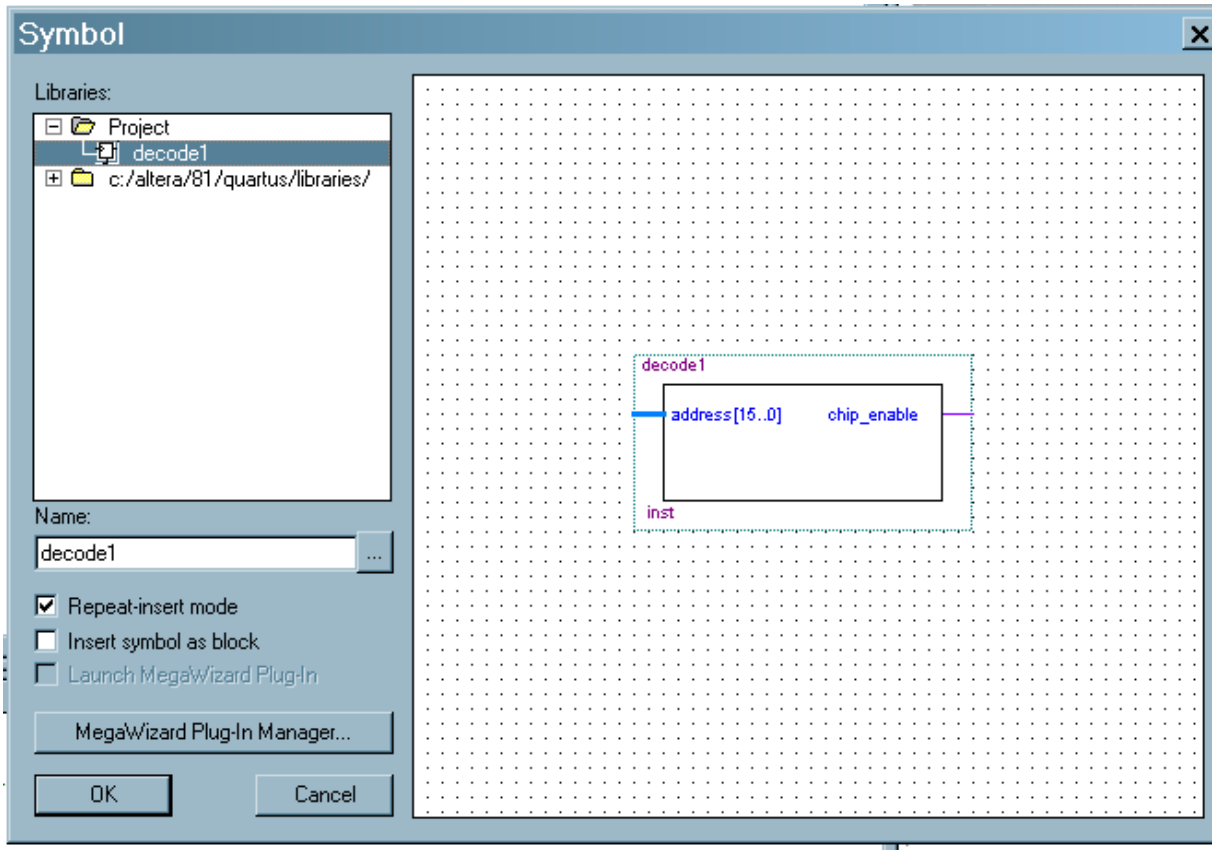


Рисунок 5. Вставка компонента написанного на языке AHDL.

2.1.7 Добавьте в проект примитивы input на 16 разрядов и output как показано на рис.6.

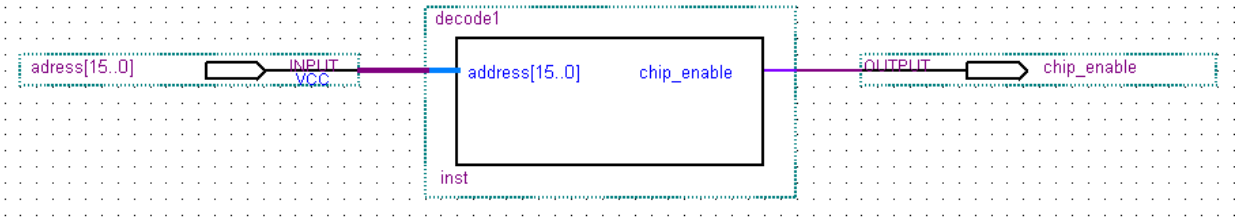


Рисунок 6. Схема для проведения моделирования.

2.1.8 Создайте файл временной диаграммы vector_waweform_file и проведите моделирование работы схемы (кнопка start simulation см. практическое занятие 1) при подачи различных кодов на вход дешифратора.

2.2. Примеры программ на языке AHDL

2.2.1. Использование чисел и констант в языке AHDL

Числа используются для представления констант в булевых выражениях и уравнениях. Язык AHDL поддерживает все комбинации десятичных, двоичных, восьмеричных и шестнадцатеричных чисел.
Создадим текстовый файл *decode1.tdf*, который представляет собой дешифратор адреса, генерирующий высокий активный уровень сигнала разрешения доступа к шине, если адрес равен шестнадцатеричному числу 370h.

```
SUBDESIGN decode1
    (
    address[15..0] : INPUT;
    chip_enable : OUTPUT;
    )
BEGIN
    chip_enable = (adress[15..0] == H"0370");
END;
```

В этом примере десятичные числа использованы для указания размерности массива бит, которым записывается адрес шины. Шестнадцатеричным числом H”0370” записано значение адреса, при котором обеспечивается высокий уровень сигнала.

2.2.2. Комбинационная логика

Комбинационная логика в языке AHDL реализована булевыми выражениями и уравнениями, таблицами истинности и большим количеством макрофункций. В число примеров комбинаторных логических функций входят дешифраторы, мультиплексоры и сумматоры. Составим файл boole1.tdf, в котором даны два простых булевых выражения, представляющих два логических элемента.

```
SUBDESIGN boole1
```

```
    (
    a0, a1, b : INPUT;
    out1, out2 : OUTPUT;
    )
BEGIN
    out1 = a1 & !a0;
    out2 = out1 # b;
END;
```

Здесь выход out1 получается в результате логической операции И, применённой ко входу a1 и инвертированному входу a0, а выход out2 получается в результате применения логической операции ИЛИ к выходу out1 и входу b. Поскольку эти уравнения обрабатываются одновременно, последовательность их следования в файле не важна.

Узел, который объявляется в секции переменных VARIABLE в объявлении NODE, можно использовать для хранения промежуточных выражений.

Это полезно делать, если булево выражение повторяется несколько раз и его целесообразно заменить именем узла. Файл boole1.tdf можно переписать по-другому).

```
SUBDESIGN boole2
    (
    a0, a1, b : INPUT;
    out : OUTPUT;
    )
VARIABLE
    a_equals_2 : NODE;
BEGIN
    a_equals_2 = a1 & !a0;
    out = a_equals_2 # b;
END;
```

Здесь объявляется узел a_equals_2, и ему присваивается значение выражения ”a1 & !a0”. Если узел используется в нескольких выражениях, то его объявление помогает экономить ресурсы устройств.

Важным понятием AHDL является группа. Она может включать в себя до 256 элементов (бит), рассматривается как совокупность узлов и участвует в различных действиях как единое целое. В булевых уравнениях группа может быть приравнена булевому выражению, другой группе, одному узлу, VCC, GND, 1 или 0. В каждом случае значения группы разные.

Если группа определена, для краткого указания всего диапазона ставят две квадратные скобки [ ]. Например, группу a[4..1] можно кратко записать как a[ ].

2.2.3. Условная логика

Условная логика делает выбор между режимами в зависимости от логических входов. Для реализации условной логики используются операторы IF или CASE:

- В операторе IF оценивается одно или несколько булевых выражений и затем описываются режимы для разных значений этих выражений.
- В операторе CASE даётся список альтернатив, которые имеются для каждого возможного значения некоторого выражения. Оператор оценивает значение выражения и по нему выбирает режим в соответствии со списком.

2.2.3.1 Логика оператора IF

Создадим файл файл *priority.tdf* , в котором описан кодировщик приоритета, который преобразует уровень самого приоритетного активного входа в значение. Он генерирует двухразрядный код, показывающий вход с наивысшим приоритетом, запускаемый VCC.

```
SUBDESIGN priority
    (
    low, middle, high : INPUT;
    highest_level[1..0] : OUTPUT;
    )
BEGIN
    IF high THEN
      highest_level[] = 3;
    ELSIF middle THEN
      highest_level[] = 2;
    ELSIF low THEN
      highest_level[] = 1;
    ELSE
      highest_level[] = 0;
    END IF;
END;
```

Здесь оцениваются входы low, middle и high, чтобы определить, запущены ли они VCC. На выходе получится код, соответствующий приоритету того входа, который был запущен VCC. Если ни один вход не запущен, значение кода будет равно 0.

2.2.3.2 Логика оператора CASE

В качестве примера рассмотрим файл decoder.tdf , реализующий функции дешифратора, преобразующего код из двухразрядного в четырёхразрядный. В результате его работы два двухразрядных двоичных входа преобразуются в один “горячий код”, называемый так потому, что четыре его допустимых значения содержат по одной единице: 0001, 0010, 0100, 1000.

```
SUBDESIGN decoder
        (
            code[1..0] : INPUT;
            out[3..0] : OUTPUT;
        )
    BEGIN
        CASE  code[] IS
WHEN 0 => out[] = B"0001";
WHEN 1 => out[] = B"0010";
WHEN 2 => out[] = B"0100";
WHEN 3 => out[] = B"1000";
          END CASE;
     END;
```

Здесь группа входа code [1..2] может принимать значения 0, 1, 2, 3. В зависимости от реального кода активизируется соответствующая ветвь оператора, и только она одна, в данный момент времени. Например, если вход code [] равен 1, на выходе out устанавливается значение B”0010”.

2.2.3.3 Логика оператора TABLE

Рассмотрим описание типовых комбинационных схем на AHDL. Дешифратор содержит комбинаторную логику, которая преобразует входные схемы в выходные значения или задаёт выходные значения для входных схем. Для создания дешифратора в языке AHDL используется объявление таблицы истинности TABLE.

Создадим файл *7segment.tdf*, представляющий собой дешифратор, который задаёт логику схемы управления светодиодами семисегментного индикатора. Светодиоды отображают на семисегментном дисплее шестнадцатеричные цифры (от 0 до 9 и буквы от A до F).

```
SUBDESIGN 7segment
(
 i[3..0] : INPUT;
 a, b, c, d, e, f, g : OUTPUT;
```

```
)
BEGIN
 TABLE
 i[3..0] => a, b, c, d, e, f, g;

 H"0" => 1, 1, 1, 1, 1, 1, 0;
 H"1" => 0, 1, 0, 0, 0, 0, 0;
 H"2" => 1, 1, 0, 1, 1, 0, 0;
 H"3" => 1, 1, 1, 1, 0, 0, 1;
 H"4" => 0, 1, 1, 0, 0, 1, 1;
 H"5" => 1, 0, 1, 1, 0, 1, 1;
 H"6" => 1, 0, 1, 1, 1, 1, 1;
 H"7" => 1, 1, 1, 0, 0, 0, 0;
 H"8" => 1, 1, 1, 1, 1, 1, 1;
 H"9" => 1, 1, 1, 1, 0, 1, 1;
 H"A" => 1, 1, 1, 0, 1, 1, 1;
 H"B" => 0, 0, 1, 1, 1, 1, 1;
 H"C" => 1, 0, 0, 1, 1, 1, 0;
 H"D" => 0, 1, 1, 1, 1, 0, 1;
 H"E" => 1, 0, 0, 1, 1, 1, 1;
 H"F" => 1, 0, 0, 0, 1, 1, 1;
 END TABLE;

END;
```

В этом примере показаны все возможные шестнадцатиричные цифры (i[3..0]) и соответствующие состояния светодиодов (a, b, c, d, e, f, g), которые обеспечивают “начертание” цифры на дисплее. Изображение светодиодов на дисплее дано в виде комментария перед файлом.

2.2.4 Объявление регистров

Регистры используются для хранения значений данных и промежуточных результатов счётчика, тактирование осуществляется синхросигналом. Регистр создаётся его объявлением в секции VARIABLE.

Для подсоединения примера примитива, макрофункции или цифрового автомата к другой логике в TDF-файле можно использовать порты. Порт примера описывается в следующем формате: “<имя примера>.<имя порта>”.

Имя порта — это вход или выход примитива, макрофункции или цифрового автомата, что является синонимом имени вывода в файлах проектов (GDF-файл), “*.WDF” и других.

Составим файл bur_reg.tdf, содержащий байтовый регистр, который фиксирует значения на входах d по выходам q на фронте синхроимпульса, когда уровень загрузки высокий.

```
SUBDESIGN bur_reg
        (
              clk, load, d[7..0]  :  INPUT;
              q[7..0]  : OUTPUT;
         )
    VARIABLE
        ff[7..0]  :  DFFE;
    BEGIN
        ff[].clk = clk;
        ff[].ena = load;
        ff[].d = d[];
        q[] = ff[].q
    END;
```

Как видно из файла, регистр объявлен в секции VARIABLE как D-триггер с разрешением (DFFE). В первом булевом уравнении в логической секции происходит соединение входа тактового сигнала подпроекта к портам тактового сигнала триггеров ff[7..0]. Во втором уравнении отпирающий тактовый сигнал соединяется с загрузкой. В третьем уравнении входы данных подпроекта соединяются с портами данных триггеров ff[7..0]. В четвёртом уравнении выходы подпроекта соединяются с выходами триггеров. Все четыре уравнения оцениваются одновременно.

Можно также в секции VARIABLE объявить T-, JK- и SR-триггеры и затем использовать их в логической секции. При использовании T-триггеров придётся в третьем уравнении изменить порт d на t. При использовании JK- и SR-триггеров вместо третьего уравнения придётся записать два уравнения, в которых происходит соединение портов j и k или s и r с сигналами. При загрузке регистра по заданному фронту глобального тактового сигнала фирма Altera рекомендует использовать вход разрешения одного из регистров: DFFE, TFFE, JKFFE или SRFFE, чтобы контролировать загрузку регистра.

Можно объявить регистровые выходы, если объявить выходы подпроекта как D-триггеры в секции VARIABLE.

Приведённый на врезке 2 файл reg_out.tdf обеспечивает те же функции, что и предыдущий (bur_reg.tdf), но имеет регистровые выходы.

```
SUBDESIGN reg_out
        (
              clk, load, d[7..0]  :  INPUT;
```

```
        q[7..0]  : OUTPUT;
        )
    VARIABLE
            q[7..0]  :  DFFE;
    BEGIN
            q[].clk = clk;
            q[].ena = load;
            q[] = d[];
    END;
```

При присвоении значения регистровому выходу в логической секции это значение формирует входные d-сигналы регистров. Выход регистра не изменяется до тех пор, пока не придёт фронт синхросигнала. Для определения тактирующего сигнала регистра нужно использовать описание в формате “<имя выходного вывода>.clk” для входа тактирующего сигнала регистра в логической секции. Глобальный синхросигнал можно реализовать примитивом GLOBAL или выбором в диалоговом окне компилятора Logic Synthesis (логический синтез) опции Automatic Global Clock.

Каждый отпирающий D-тригер, объявленный в секции VARIABLE, возбуждает выход с таким же именем, поэтому можно обращаться к q выходам объявленных триггеров без использования q-порта этих триггеров.

2.2.5 Создание счётчиков

Счётчиками называются последо-вательностные логические схемы для счёта тактовых импульсов. В некоторых счётчиках реализован счёт вперед и назад (реверсивные счётчики), в некоторые счётчики можно загружать данные, а также обнулять их. Счётчики обычно определяют как D-триггеры (DFF и DFFE) и используют операторы IF.

Составим файл ahdlcnt.tdf, который реализует 16-бит загружаемый счётчик со сбросом.

```
SUBDESIGN ahdlcnt
        (
            clk, load, ena, clr, d[15..0]  :  INPUT;
    q[15..0]  : OUTPUT;
        )
```

```
    VARIABLE
            count[15..0]  :  DFFE;
    BEGIN
            count[].clk = clk;
            count[].clrn = !clr;
            IF load THEN
                    count[].d = d[];
            ELSIF ena THEN
count[].d = count[].q + 1;
            ELSE
count[].d = count.q;
            END IF;
            q[] = count[];
    END;
```

В данном файле в секции VARIABLE объявлены 16 D-триггеров, и им присвоены имена от count0 до count15. В операторе IF определяется значение, загружаемое в триггеры по фронту синхросигнала (например, если загрузка запускается VCC, то триггерам присваивается значение d[ ]).

2.2.6. Двунаправленные выводы

Двунаправленный вывод задаётся как порт BIDIR, который подсоединяется к выходу примитива TRI. Сигнал между этим выводом и буфером с тремя состояниями является двунаправленным, и его можно использовать в проекте для запуска других логических схем.

Составте файл *bus_reg2.tdf* реализует регистр, который делает выборку значения, найденного на шине с тремя состояниями, а также может передать обратно на шину хранимое значение.

```
SUBDESIGN bus_reg2
        (
            clk : INPUT;
            oe : INPUT;
            io : BIDIR;
        )
   BEGIN
        io = TRI(DFF(io, clk,, ), oe);

   END;
```

Двунаправленный сигнал io, запускаемый примитивом TRI, используется в качестве входа d для D-триггера (DFF). Запятые в конце списка параметров

отделяют места для сигналов триггера clrn и prn. Эти сигналы по умолчанию установлены в неактивное состояние.

Двунаправленный вывод можно также использовать для подсоединения TDF-файла более низкого уровня к выводу с высоким уровнем. Прототип функции для TDF-файла более низкого уровня должен содержать двунаправленный вывод в предложении RETURNS. В приведённом на врезке 14 файле bidir1.tdf даны четыре примера использования макрофункции bus_reg2.

```
FUNCTION bus_reg2 (clk, oe) RETURNS (io);
    SUBDESIGN bidir1
        (
            clk, oe : INPUT;
            io[3..0] : BIDIR;
        )
    BEGIN
        io0 = bus_reg2 )clk, oe);
        io1 = bus_reg2 )clk, oe);
        io2 = bus_reg2 )clk, oe);
        io3 = bus_reg2 )clk, oe);

    END;
```

3. Задания для работы

Сформировать проект на языке AHDL и провести моделирование работы устройства в соответствии с вариантом.

№	Содержание	Примечание
1	Мультиплексор на 4 канала	
2	Демультиплексор на 4 канала	
3	4 разрядный счетчик	
4	4 разрядный вычитающий счетчик	
5	Дешифратор двоичного в сигнал семисегментного индикатора	
6	4 разрядный параллельный сумматор	
7	4 разрядный параллельный вычитатель	
8	Трехразрядный счетчик с модулем счета 5	

4. Контрольные вопросы

1. Порядок работы в QUARTUS II при создании проекта на AHDL;
2. Реализация последовательностных схем на AHDL;
3. Реализация комбинационных схем на AHDL.

5. Литература

1. Проектирование арифметических устройств цифровой обработки сигналов с использованием сапр quartus ii и ее библиотеки мегафункций/ НГТУ; Сост.: А.Д.Плужников, Н.Н.Потапов, А.А.Цветков. Н.Новгород, 2005. 22 с.

2. В. Стешенко, А. Самохин Школа разработки аппаратуры цифровой обработки сигналов на ПЛИС. Интернет ресурс http://www.chipnews.ru/html.cgi/arhiv/00_03/stat-11.htm

Практическая работа №11. Создание проекта с использованием макрофункций на языке AHDL

1 Цель работы: Целью работы является изучение способов создания цифровых устройств (сумматоров, вычитателей, умножителей, делителей, счетчиков, регистров, мультиплексоров) на основе ПЛИС в среде Quartus II.

2 Основные сведения.

Как известно все цифовые схемы строятся из похожих блоков, например мультиплексоры, сумматоры, регистры, счетчики, умножители и т.д.. Для создания этих блоков в состав системы QUARTUS II входит библиотека мегафункций. Мегафункция реализует законченное цифровое устройство. Инженер может использовать как стандартные мегафункции, так и проектировать свои мегафункции. При создании мегафункции автоматически создается соответствующий файл на языке AHDL, VHDL или Verilog.

Мегафункция - сложный или высокоуровневый блок, который можно использовать совместно с примитивами вентилей и триггеров и/или с макрофункциями старого типа в файлах проекта.

Altera поставляет библиотеку мегафункций, включая функции из библиотеки параметризуемых модулей (LPM). Параметризуемая функция - логическая функция, использующая параметры для достижения масштабируемости, адаптируемости и эффективной реализации в кремнии.

2.1 Создание блоков с помощью помощника MegaWizard.

Создайте проект LAB2 на основе микросхем серии FLEX10K например EPF10K10TC144. Создайте файл схемы «Block diagram/schematic file» и сделайте его корневым файлом проекта с именем LAB2.BDF.

Выберите «Tools-MegaWizard Plugin Manager». В появившемся окне выберите «Create a new custom megafunction variation» и нажмите «Next»

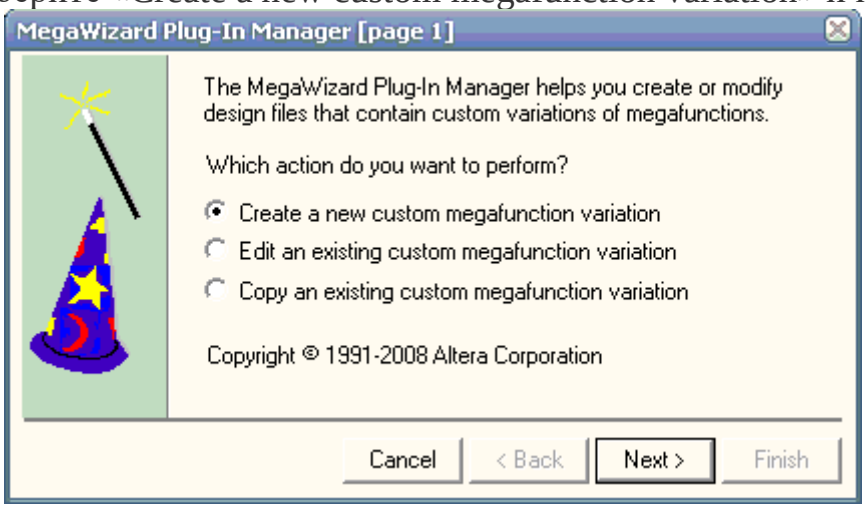


Рис. 1

Перед Вами раскроется страница 2а рис.__. с обширным набором прототипов мегафункций. Рассмотрим создание двоичного счетчика. Укажите

имя файла например «D:\ALtera\LAB2\Counter». Выберите прототип мегафункции LPM_COUNTER как показано на рисунке _2__. и нажмите «Next».

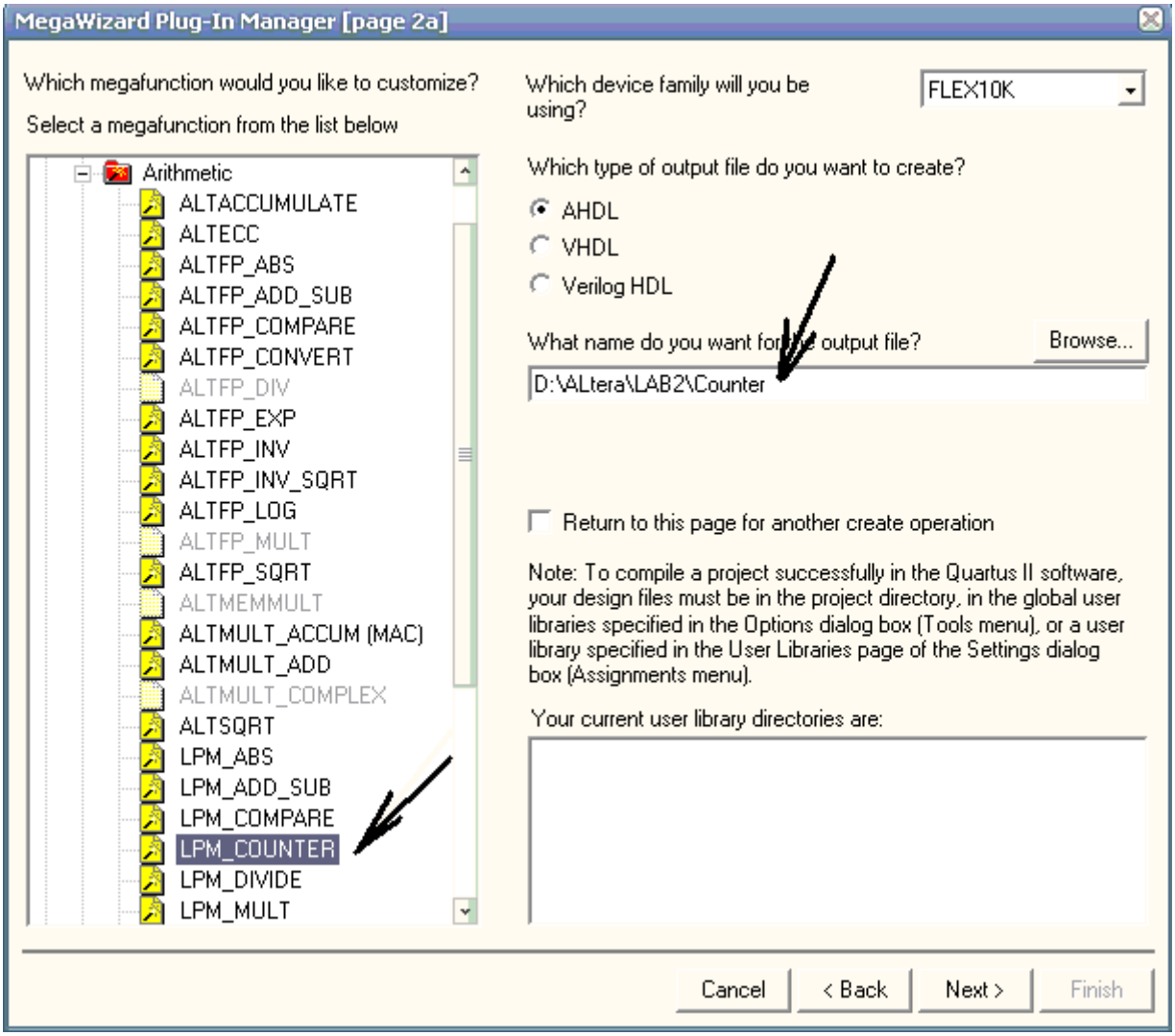


Рис.2 Создание счетчика.

В появившемся окне на вкладке Parameter Settings

в поле General1 укажите разрядность счетчика, и тип (суммирующий, вычитающий или реверсивный) Рис.__.

на поле General2 укажите другие параметры счетчика

Тип счетчика: со сквозным переносом (plain) или с переносом по модулю (Modulus). Выберем счетчик с переносом по модулю 100.

Наличие входа разрешения работы Clock Enable (ставим галочку)

Наличие входа разрешения подачи импульсов Count Enable (ставим галочку)

Вход переноса Carry In (не ставим галочку)

Выход переноса Carry Out (ставим галочку)

Для просмотра временных диаграмм работы схемы можно воспользоваться кнопкой Documentation-Generate Sample Waveforms.

На поле Optional Inputs можно указать наличие синхронных и асинхронных входов сброса (Clear), параллельной загрузки начального значения (Load), установки (Sset) произвольного заранее выбранного значения. Работу каждого

входа можно наглядно посмотреть на временной диаграмме Documentation-Generate Sample Waveforms. Будем считать, что нам эти входы не нужны, то есть, галочки не ставим.

Далее, нажимая, Next-Next-Finish закончим создание компонента

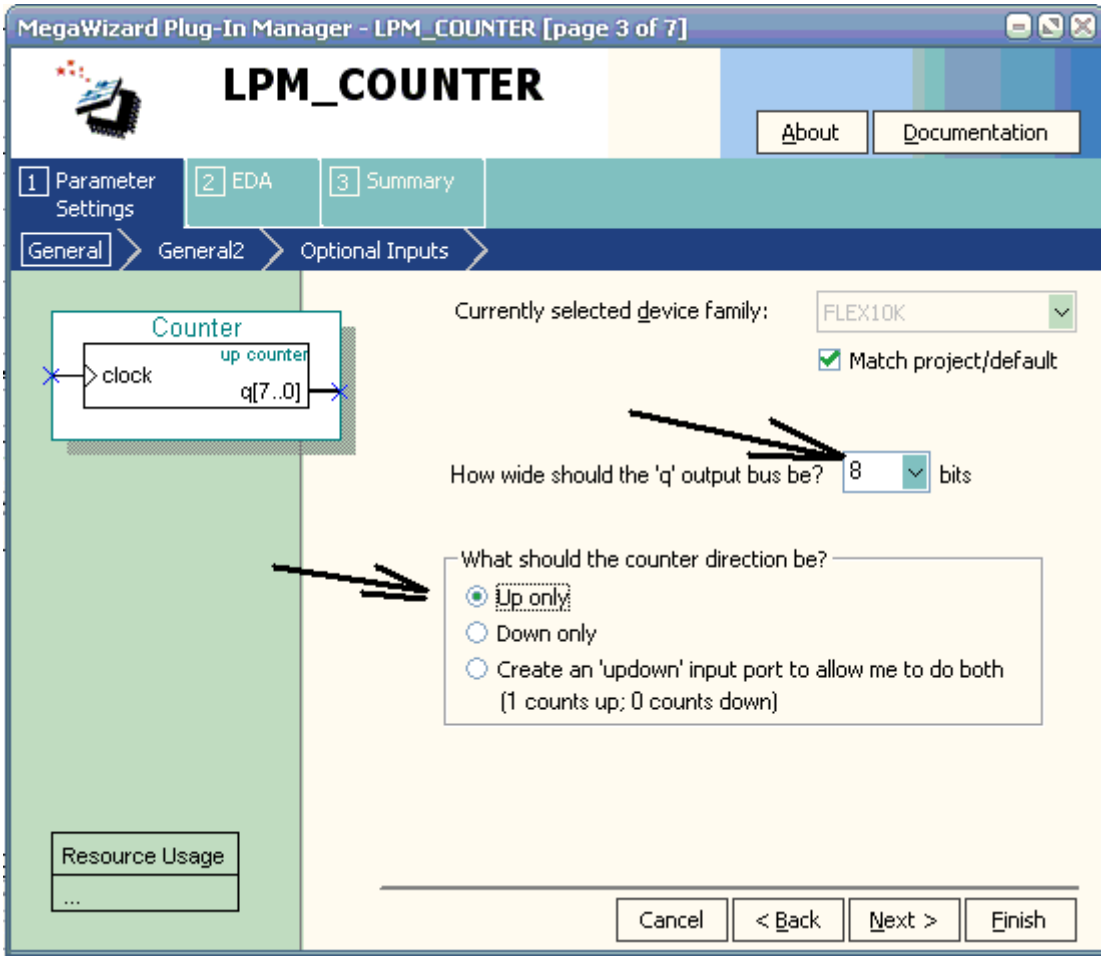


Рис._3__.

Для вставки созданного компонента в ваш проект выберем «Symbol toll»
В появившемся окне выберем компонент «Counter» как показано на рис. __

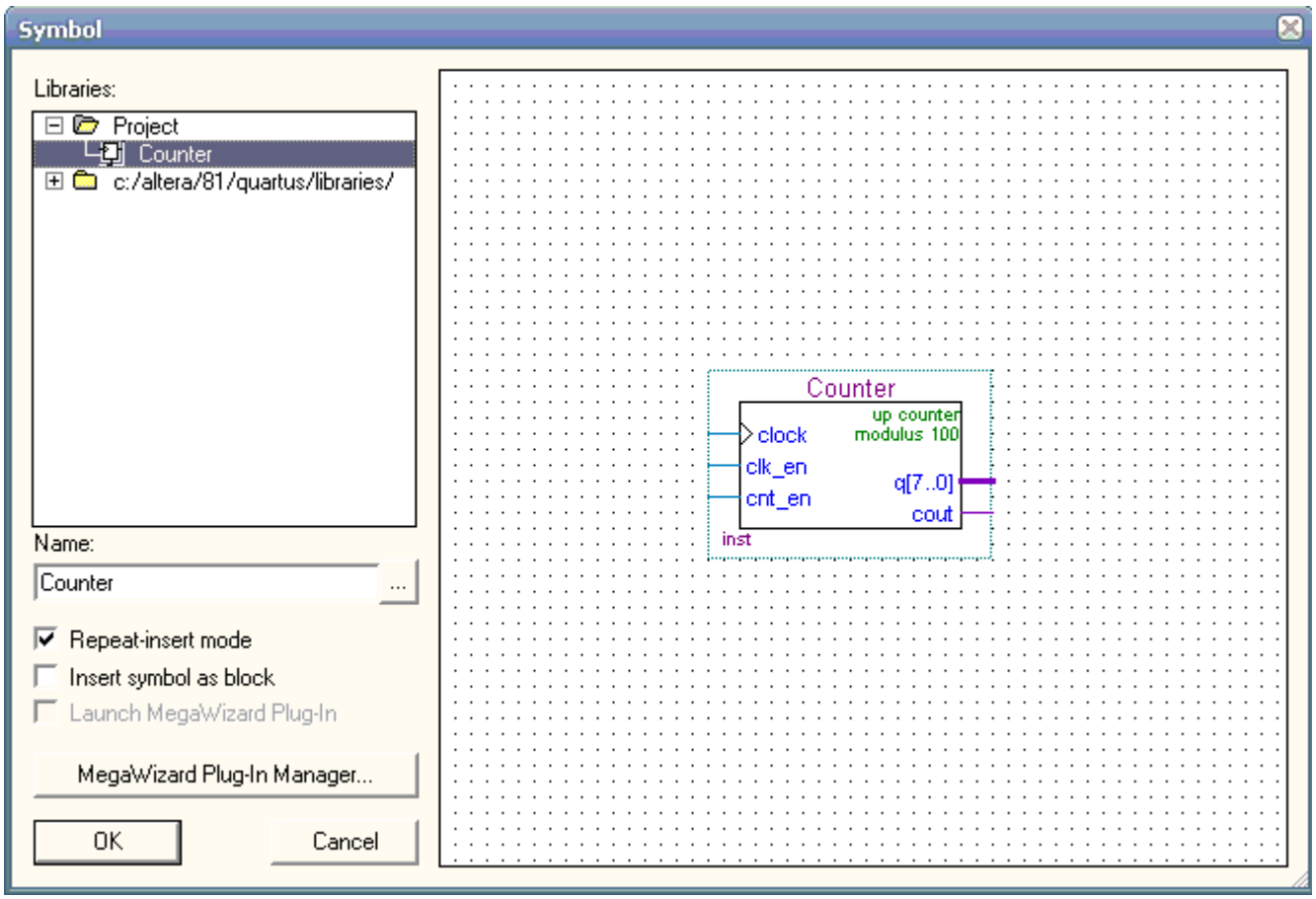


Рис. _4_

2.2 Краткое описание некоторых стандартных мегафункций.

1. Арифметические мегафункции (аккумуляторы, сумматоры, умножители, и параметрические арифметические функции)

Название	**Мегафункция**	**Комментарий**
ALTACCUMULATE	altaccumulate	Параметрическая мегафункция аккумулятора
ALTECC	altecc_decoder	Декодер ECC кода. ECC (англ. error-correcting code, код коррекции ошибок) — данные, присоединяемые к каждому передаваемому сигналу, позволяющие принимающей стороне определить факт сбоя и (в некоторых случаях) исправить несущественную ошибку
	altecc_encoder	Кодер ECC кода.
ALTFP_ABS	altfp_abs	Модуль числа с плавающей точкой.
ALTFP_ADD_SUB	altfp_add_sub	Сумматор, вычитатель чисел с плавающей точкой.
ALTFP_COMPARE	altfp_compare	Компаратор чисел с плавающей точкой
ALTFP_CONVERT	altfp_convert	Параметрический преобразователь чисел с плавающей точкой
ALTFP_DIV	altfp_div	Делитель чисел с плавающей точкой
ALTFP_EXP	altfp_exp	Нахождение экспоненты числа с плавающей точкой

ALTFP_INV	altfp_inv	Нахождение обратной величины для числа с плавающей точкой.
ALTFP_INV_SQRT	altfp_inv_sqrt	Корень квадратный обратной величины
ALTFP_LOG	altfp_log	Квадратный корень логарифма для чисел с плавающей точкой.
ALTFP_MULT	altfp_mult	Параметрический умножитель чисел с плавающей точкой.
ALTFP_SQRT	altfp_sqrt	Параметрическая функция квадратного корня для чисел с плавающей точкой.
ALTMEMMULT	altmemmult	Умножитель на базе блоков ОЗУ
ALTMULT_ACCUM (MAC)	altmult_accum	Параметрический умножитель аккумулятор
ALTMULT_ADD	altmult_add	Параметрический умножитель/сумматор
ALTMULT_COMPLEX	altmult_complex	Умножитель комплексных чисел
ALTSQRT	altsqrt	Корень квадратный
LPM_ABS	lpm_abs	Модуль числа
LPM_ADD_SUB	lpm_add_sub	Сумматор вычитатель
LPM_COMPARE	lpm_compare	Многоразрядный компаратор
LPM_COUNTER	lpm_counter	Счетчик
LPM_DIVIDE	lpm_divide	Делитель
	divide*	Делитель
LPM_MULT	lpm_mult	Параметрический умножитель
	altsquare	Возведение в квадрат
PARALLEL_ADD	parallel_add	Параллельный сумматор

Базовые мегафункции

Название	**Мегафункция**	**Комментарии**
LPM_AND	lpm_and	Логическое И
LPM_BUSTRI	lpm_bustri	Тристабильный буфер
LPM_CLSHIFT	lpm_clshift	Параметрический регистр сдвига
LPM_CONSTANT	lpm_constant	Генаратор константы
LPM_DECODE	lpm_decode	Дешифратор
LPM_INV	lpm_inv	Инвертер
LPM_MUX	lpm_mux	Мультиплексор
LPM_OR	lpm_or	Логическое ИЛИ
LPM_XOR	lpm_xor	Логическое исключающее ИЛИ

Мегафункции ввода вывода

Название	**Мегафункция**	**Комментарии**
ALT2GXB	alt2gxb	GXB. Гигабитный передатчик.
ALT2GXB_RECONFIG	alt2gxb_reconfig	GXB. Динамически реконфигурируемый гигабитный передатчик.
ALTASMI_PARALLEL	altasmi_parallel	Active serial memory interface parallel megafunction.
ALTCLKCTRL	altclkctrl	Clock control block megafunction.
ALTCLKLOCK	altclklock	Parameterized PLL megafunction.
ALTDDIO_BIDIR	altddio_bidir	DDR двунаправленный порт
ALTDDIO_IN	altddio_in	DDR вход.
ALTDDIO_OUT	altddio_out	DDR выход.

ALTDLL	altdll	Delay locked loop (DDL) megafunction.
ALTDQ	altdq	Строб данных.
ATLDQS	altdqs	Параметрическая функция строба данных
	atldq_dqs	Параметрическая функция строба данных
ALTGX	alt4gxb	High-Speed Serial Interface (HSSI) GXB megafunction.
ALTGXB	altgxb	GXB megafunction.
ALTIOBUF	altiobuf_bidir	Двунаправленный буфер ввода вывода
	altiobuf_in	Входной буфер ввода вывода
	altiobuf_out	Выходной буфер ввода вывода.
ALTLVDS	altlvds_rx	Примник Low voltage differential signalling (LVDS) сигнала.
	altlvds_tx	Передатчик Low voltage differential signalling (LVDS) сигнала.
ALTMEMPHY	ALTMEMPHY	Интерфейс для внешней DDR памяти. External DDR Memory PHY interface megafunction.
ALTOCT	alt_oct	On-chip termination (OCT) megafunction.
ALTPLL	altpll	Parameterized PLL megafunction.
ALTPLL_RECONFIG	altpll_reconfig	Parameterized PLL reconfiguration megafunction.
ALTREMOTE_UPDATE	altremote_update	Parameterized remote update megafunction.
	altstratixii_oct	Parameterized OCT megafunction.
MAX II oscillator	altufm_osc	Встроенный генератор

Устройства хранения

Название	**Мегафункция**	**Комментарии**
ALTCAM	altcam	Content-addressable memory (CAM) Память с адресацией по содержимому
ALTQPRAM	altqpram*	Двухпортовое ОЗУ
LPM_FF	lpm_dff*	D триггер и регистр сдвига
	lpm_ff	Триггер.
	lpm_tff*	T - триггер.
LPM_LATCH	lpm_latch	Parameterized latch megafunction.
LPM_SHIFTREG	lpm_shiftreg	Регистр сдвига.

3. Задания для работы

3.1. Арифметическое устройство реализующее выполнение арифметических операций представленных в таблице.

Задания:

№ п/п	Уравнение	Разрядность параметров (бит)		
		X(n)	a	b
1	$y(n) = a \cdot x(n) + b$	6	4	7
2	$y(n) = a \cdot x(n) - b$	4	5	8

3	$y(n)=(a+x(n))\,\&\,(b+x(n-1))$	7	7	7
4	$y(n)=(a-x(n))\,\&\,(b-x(n-1))$	7	7	7
5	$y(n)=a/x(n)+b$	6	4	7
6	$y(n)=a/x(n)-b$	4	5	8

Пояснения:
$+$ - Арифметическое сложение многоразрядных двоичных чисел.
$-$ - Арифметическое вычитание многоразрядных двоичных чисел.
$\cdot$ и / - Арифметическое умножение и деление многоразрядных двоичных чисел.
$\oplus$ - Логическая операция исключающее ИЛИ над многоразрядными двоичными числами.
$\&$ - Логическая операция исключающее И над многоразрядными двоичными числами.
X(n) – входная цифровая последовательность (разрядность см. в таблице).
Y(n) – выходная последовательность.
a,b, - двоичные константы.
n – номер такта

3.2. Устройство измерения длительности импульса или интервала между импульсами

3.2.1. Пусть на вход системы поступают два одиночных импульса рис.1.

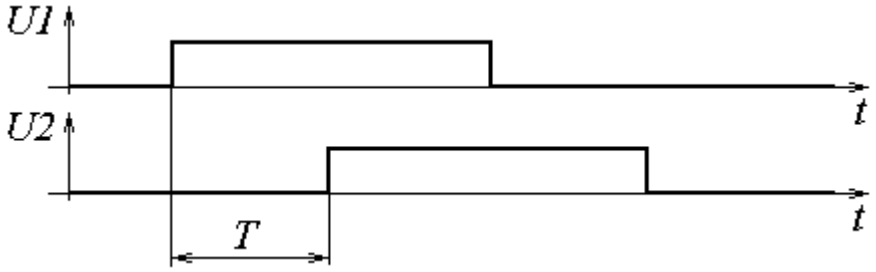


Рис. 1 Временная диаграмма на входах ***U1 U2*** ПЛИС

Необходимо составить схему, реализующую измерение задержки T импульса U2 относительно импульса U1. Если импульс U2 придет раньше чем U1 то время задержки необходимо представить в виде отрицательного числа в дополнительном коде. Выходной сигнал представить в параллельном коде.

№ п/п	Заданная временная диаграмма	Тактовая частота внешнего генератора МГц	Диапазон изменения измеряетого периода ***T,*** мс

1		5	±10
2		10	±10
3		15	±10

3.2.2. Пусть на вход схемы поступает произвольная импульсная последовательность

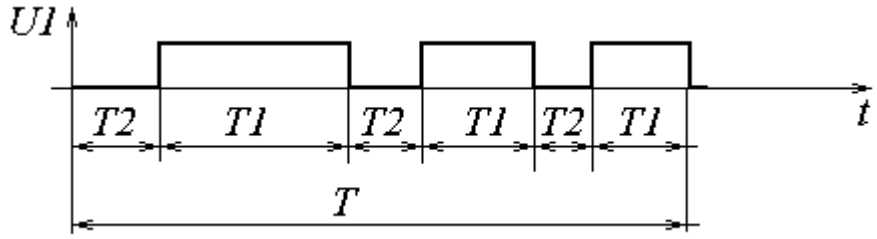


Рис.2.

Составить схему вычисляющую длительность суммарных отрезков $A = \sum T1$ $B = \sum T2$, Выходной сигнал схемы рассчитывается по формуле приведенной в таблице. Выходной сигнал необходимо представить в параллельном коде.

№ п/п	Формула расчета выходного сигнала	Тактовая частота внешнего генератора МГц	Период ***T,*** мс
4	$Y = A - B$	5	1
5	$Y = A / B$	10	2
6	$Y = A \cdot B$	15	3

4. Контрольные вопросы

1. Расскажите о использованных в проекте мегафункциях
2. Порядок использования мегафункций.
3. Охарактеризуйте параметры выбранной мегафункции.

5. Литература

1. Проектирование арифметических устройств цифровой обработки сигналов с использованием сапр quartus ii и ее библиотеки мегафункций/ НГТУ; Сост.: А.Д.Плужников, Н.Н.Потапов, А.А.Цветков. Н.Новгород, 2005. 22 с.

2. Новиков, Ю.В. Основы микропроцессорной техники : Курс лекций для вузов / Ю.В.Новиков,П.К.Скоробогатов;Интернет ун-т информ.технологий .— М., 2003 .— 432с. : ил. — (Основы информационных технологий) .— Библиогр.в конце кн. — ISBN 5-9556-0004-3 /в пер./ : 165.00. 28экз.
3. Токарев, В.Л. Аппаратные средства вычислительной техники : учеб.пособие для вузов / В.Л.Токарев .— Тула : Изд-во ТулГУ, 2005 .— 470с. — (75-летию ТулГУ посвящается) .— Библиогр.в конце кн. — ISBN 5-7679-0762-5 /в пер./ : 230.00. 65 экз.
4. Нарышкин, А.К. Цифровые устройства и микропроцессоры : учеб.пособие для вузов / А.К.Нарышкин 2-е изд., стер..— М. : Академия, 2008 .— 319с.: ил. 7 экз.

5. Корякин О.Г., Евстигнеев Е.Т. Микропроцессоры в системах стабилизации и управления: Учебн.пособие/ Корякин О.Г., Евстигнеев Е.Т. Тульский политехнический институт-Тула, 1992.83с.ил.-15экз.
6. Г.И.Волович. Схемотехника аналоговых и аналого-цифровых электронных устройств. М.: Издательский дом «Додэка-XXI», 2005.-528с.-3экз.
7. Стешенко Б.В. ПЛИС фирмы ALTERA: проектирование устройств обработки сигнала.-М.: ДОДЕКА, 2000.-128С.2экз.
8. Бродин Б.В. Системы на микроконтроллерах и БИС программируемой логики/В.Б.Бродин, А.В.Калинин.-М.: ЭКОМ, 2002.-399с.: ил- (Современная микропроцессорная техника)- 3экз.