Министерство науки и высшего образования Российской Федерации
Мытищинский филиал
Федерального государственного автономного образовательного учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МФ МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ *Космический* .

КАФЕДРА *Прикладной математики, информатики и вычислительной техники*

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

Проектирование арифметического устройства в кристалле ПЛИС с использованием языка VHDL в САПР QUARTUS II в соответствии с вариантом № 10 технического задания

Студент *К3-76Б* _____ *Несмеянов С.А.*
(Группа) (Подпись, дата) (Фамилия И. О.)

Руководитель курсового проекта _____ *Ефремов Н.В.*
(Подпись, дата) (Фамилия И. О.)

2026г

Министерство науки и высшего образования Российской Федерации
Мытищинский филиал
Федеральное государственное автономное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МФ МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ
Заведующий кафедрой *К3 МФ*
(Индекс)
_____ _____
(И. О. Фамилия)
«__» _____ 20 __ г.

З А Д А Н И Е
на выполнение курсовой работы

по дисциплине *Организация ЭВМ и систем*

Студент группы *К3-76Б* *Несмеянов Степан Алексеевич*
(Фамилия, имя, отчество)

Тема курсовой работы *Проектирование арифметического устройства в кристалле ПЛИС с использованием языка VHDL в САПР QUARTUS II в соответствии с вариантом № 10 технического задания*

Направленность КР (учебная, исследовательская, практическая, производственная, др.)
_____

Источник тематики (кафедра, предприятие, НИР) _____

Оформление курсовой работы:

Расчетно-пояснительная записка на 42 листах формата А4.

Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)
_____
_____
_____

Дата выдачи задания « __ » _____ 2026 г.

Руководитель курсовой работы _____ *Н.В. Ефремов*
(Подпись, дата) (И. О. Фамилия)

Студент _____ *С.А. Несмеянов*
(Подпись, дата) (И. О. Фамилия)

Министерство науки и высшего образования Российской Федерации
Мытищинский филиал
Федеральное государственное автономное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МФ МГТУ им. Н.Э. Баумана)

К а л е н д а р н ы й п л а н
на выполнение курсовой работы

по дисциплине *Организация ЭВМ и систем*

Студент группы *К3-76Б* *Несмеянов Степан Алексеевич*
(Фамилия, имя, отчество)

№ п/п	Наименование этапов курсовой работы	Сроки выполнения этапов		Отметка о выполнении
		план	факт	Руководитель
1.	Задание на выполнение курсовой работы			
2.	Выполнение задания курсовой работы			
3.	Оформление отчета			
4.	Защита курсовой работы			

Руководитель курсовой работы _____ *Н.В. Ефремов*
(Подпись, дата) (И. О. Фамилия)

Студент _____ *С. А. Несмеянов*
(Подпись, дата) (И. О. Фамилия)

Содержание

Техническое задание

Дробные числа с фиксированной запятой представлены в прямом коде.

Формат команд

Команды двухадресные и двух форматов: "регистр-регистр" (РР) и "память-память" (ПП). В формате РР записываются команды короткой операции, в формате ПП - команды длинной операции и перехода.

В формате РР указываются прямые адреса 1-го и 2-го операндов, расположенных в РП. В формате ПП 1-й операнд указывается с помощью прямого адреса, 2-й операнд - косвенного регистрового адреса (оба операнда хранятся в ОП).

Результат операций записывается по адресу 1-го операнда.

Операции

а) УМНОЖЕНИЕ. Операция выполняется по алгоритму умножения чисел в прямом коде со старших разрядов множителя и сдвигом множимого вправо [1-4,6,8]. При нулевом значении анализируемого разряда множителя такт суммирования пропускается. Первый операнд множимое, второй - множитель.

б) ОТРИЦАТЕЛЬНОЕ ПРИРАЩЕНИЕ КОДА. Из первого операнда (двоичного кода) вычитается приращение, которое записано в поле адреса второго операнда кода команды. Результат помещается по адресу первого операнда. Устанавливается признак результата: 0-разность равна нулю, перенос отсутствует; 1-разность не равна нулю, перенос отсутствует; 2-разность равна нулю, есть перенос; 3-разность не равна нулю, есть перенос. Имеется в виду перенос из старшего (левого) разряда.

в) ПЕРЕХОД ПО ОСТУТСТВИЮ ПЕРЕНОСА. Адрес в счетчике команд (СК) замещается адресом перехода, если значение признака результата равно 0 или 1. В противном случае (СК)=(СК)+2. В качестве адреса перехода используется прямой адрес. Команда перехода занимает 2 ячейки ОП.

Введение

В современном мире цифровые технологии играют ключевую роль в различных сферах жизни, от бытовых устройств до сложных промышленных систем. Проектирование цифровых систем, таких как микропроцессоры и управляющие устройства, становится все более важным направлением в инженерии и информатике. Эффективное взаимодействие между компонентами системы, такими как память, арифметические устройства и управляющие блоки, требует глубоких знаний в области архитектуры вычислительных машин и принципов работы цифровых схем.

Целью данной курсовой работы является разработка блока операций, местного устройства управления и центрального устройства управления, что включает в себя проектирование, верификацию и тестирование различных модулей. В процессе работы будет рассмотрен функционал ключевых элементов системы, таких как счетчик команд, регистр команд и регистр адреса, а также их взаимодействие через управляющие сигналы. Использование современных инструментов автоматизированного проектирования, таких как QUARTUS II, позволит не только реализовать проект на уровне схемы, но и проверить его работоспособность с помощью моделирования.

Таким образом, данная курсовая работа представляет собой важный шаг на пути к пониманию принципов проектирования и реализации цифровых систем, а также формированию необходимых компетенций для успешной карьеры в области высоких технологий.

1.Схемотехническое проектирование блока операций

Алгоритм выполнения заданной операции в текстовой форме можно представить так:

1. Обнуляем сумму частичных произведений S, счетчик i выставляем в 0, сдвигаем множимое вправо на 1 разряд
2. Анализируем старший разряд множителя:
   - Если равен 1, то $S = S + |X| * 2^i$
   - Если 0, то пропускаем сложение
3. Сдвигаем множимое вправо на 1 разряд и i = i + 1:
   - Если i == n-1, то идем дальше
   - Иначе повторяем пункт 2 и 3
4. Знак произведения вычисляем по формуле:

$$X_{зн} \oplus Y_{зн}$$

Ниже рассмотрены примеры выполнения заданной операции для n=4.

Умножение

Пример 1

$[X]_п$ = 0,100 (4/8)

$[Y]_п$ = 0,011 (3/8)

$Y_{-1}=0$	0,0000000	S_0
	0,0100000	$\|X\|*2^{-1}$
$Y_{-2}=1$ +	0,0000000	S_1
	0,0010000	$\|X\|*2^{-2}$
$Y_{-3}=1$ +	0,0010000	S_2
	0,0001000	$\|X\|*2^{-3}$
	0,0011000	$[Z]_п$

Знак вычисляем по формуле $X_{зн} \oplus Y_{зн} = 0 \oplus 0 = 0$

$[Z]_п$ = 0,0011000 = (3 * 4)/64 = 12/64

Результат верный

Пример 2

$[X]_п$ = 0,100 (4/8)

$[Y]_п$ = 1,011 (-3/8)

$Y_{-1}=0$	0,0000000	S_0
	0,0100000	$\|X\|*2^{-1}$
$Y_{-2}=1$ +	0,0000000	S_1
	0,0010000	$\|X\|*2^{-2}$
$Y_{-3}=1$ +	0,0010000	S_2
	0,0001000	$\|X\|*2^{-3}$
	0,0011000	$[Z]_п$

Знак вычисляем по формуле $X_{зн} \oplus Y_{зн} = 0 \oplus 1 = 1$

$[Z]_п$ = 1,0011000 = - (3 * 4)/64= - 12/64

Результат верный

Пример 3

$[X]_п$ = 1,100 (-4/8)

$[Y]_п$ = 0,011 (3/8)

$Y_{-1}=0$	0,0000000	S_0
	0,0100000	$\|X\|*2^{-1}$
$Y_{-2}=1$ +	0,0000000	S_1
	0,0010000	$\|X\|*2^{-2}$
$Y_{-3}=1$ +	0,0010000	S_2
	0,0001000	$\|X\|*2^{-3}$
	0,0011000	$[Z]_п$

Знак вычисляем по формуле $X_{зн} \oplus Y_{зн} = 1 \oplus 0 = 1$

$[Z]_п$ = 1,0011000 = - (3 * 4) /64 = - 12/64

Результат верный

Пример 4

$[X]_п$ = 1,100 (-4/8)

$[Y]_п$ = 1,011 (-3/8)

$Y_{-1}=0$	0,0000000	S_0
	0,0100000	$\|X\|*2^{-1}$
$Y_{-2}=1$ +	0,0000000	S_1
	0,0010000	$\|X\|*2^{-2}$
$Y_{-3}=1$ +	0,0010000	S_2
	0,0001000	$\|X\|*2^{-3}$
	0,001100	$[Z]_п$

Знак вычисляем по формуле $X_{зн} \oplus Y_{зн} = 1 \oplus 1 = 0$

$[Z]_п$ = 0,0011000 = (3 * 4) /64 = 12/64

Результат верный

Отрицательное приращение кода

1)

$A_п$=0111

$B_п$=0111

Z = 00010110

ПР = 11

2)
$A_п$=0011
$B_п$=1011

Z = 00000110
ПР = 01

3)
$A_п$=0000
$B_п$=1000

Z = 00000000
ПР = 00

4)
$A_п$=0100
$B_п$=0100

Z = 00010000
ПР = 10

Определим слова, которые необходимы для выполнения умножения по описанному алгоритму. Операнды – множимое и множитель, представляем словами **а**и **в**. Их формат задан в техническом задании. Значения этим словам присваиваются вне алгоритма до начала операции, поэтому тип этих слов I (входные). Чтобы преобразовывать значения исходных операндов внутри алгоритма, будем использовать слова **ra** и **rb**, причем слово **ra** определим 2n разрядным. Присвоим им тип L (внутренние). Значение искомого произведения будем представлять словом **rc** (2n-1:0). Это слово должно иметь тип LO (внутреннее и выходное), поскольку оно формируется в микропрограмме и по окончании операции представляет значение результата, используемое вне микропрограммы. Поскольку все разряды множителя обрабатываются одинаково, организуем цикл. Для этого применяем счетчик *i*. Так как он используется только внутри микропрограммы, присваиваем соответствующему слову тип L.

1.1. Описание слов в микропрограмме умножения

Тип	Наименование	Пояснения
I	a(n-1:0)	1 операнд (множимое)
I	b(n-1:0)	2 операнд (множитель)
O	c(2n-1:0)	результат (произведение)
L	*i*	счетчик анализируемого разряда
L	ra(2n-1:0)	регистр множимого
L	rb(n-1:0)	регистр множителя
LO	rc(2n-1:0)	регистр результата

Табл.1

В первом проекте использован традиционный подход, схемотехническое проектирование. После создания функциональной схемы блока операций на этапе логического проектирования из библиотеки доступных элементов выбираются подходящие узлы и настраиваются соответствующим образом для работы в общем проекте, выполняется проектирование комбинационных схем

Регистр ra выполняет одну единственную микрооперацию – прием первого операнда. Извлекаем из библиотеки параметризированных модулей QUARTUS I, из раздела *storage* модуль lpm_dff, и настраиваем его на выполнения этой единственной микрооперации. Для управления им используем сигнал **y1**. Если **y1** не активен, то регистр ra просто хранит своё содержимое, то есть первый операнд.

Регистр rb помимо приема второго операнда в начале выполнения операции умножения выполняет его сдвиг влево, так как по техническому заданию анализ множителя проводится, начиная с его старших разрядов. Извлекаем из раздела *storage* библиотеки параметризированных модулей QUARTUS II модуль lpm_shiftreg, и настраиваем его соответствующим образом. Сдвиг выполняется так, чтобы обеспечить сохранность знакового разряда. Поэтому знаковый разряд дублируем на самый старший разряд регистра rb. Освобождающийся при сдвиге влево старший разряд регистра rb будет заполняться знаковым значением B, которое будет подаваться на его вход переноса, *shiftin*.

Для реализации регистром rb двух микроопераций используем два управляющих сигнала **y2** и **y3**. Первый из них определяет режим работы регистра rb. При активном сигнале **y2** выполняется прием второго операнда, иначе выполняется его сдвиг влево. Второй управляющий сигнал **y3** разрешает регистру rb реагировать на тактовый сигнал, то есть либо принимать второй операнд, либо его сдвигать по положительному фронту тактового сигнала.

Комбинационная схема КС1 подает на первый вход сумматора sum либо +ra, либо ноль. На выходе этой схемы формируется 2n-разрядное значение путем заполнения младших разрядов знаковым. Для управления КС1 используется один управляющий сигнал **y4**. При активном значении **y4** на выход КС1 подается прямое значение первого операнда. Во всех остальных случаях на выходе КС1 формируется нулевое значение.

Комбинационная схема КС2 реализует функцию подключения ко второму входу сумматора либо регистра результата rr, при выполнении умножения, либо регистра rb, при выполнении отрицательного приращения, причем старшие разряды второго операнда дополняются до восьми разрядов знаковым. Для управления КС2 используем управляющий сигнал **y9**. Он активен при выполнении инвертирования. И также **y9** подключен к КС1, чтобы заполнение значения a было с младших разрядов для второй операции

Комбинационные схемы КС3 и КС4 выполняют анализ n+1 разрядного результата операции сложения и на основании этого анализа формируют двухразрядный признак результата в соответствии с техническим заданием и признак отрицательного нуля, соответственно.

Проектирование комбинационных схем выполняется традиционным путем. Вначале составляем таблицу истинности, затем заполняем карты Карно и выполняем минимизацию логических функций, то есть находим их тупиковые формы ТДНФ и ТКНФ. Затем реализуем одну из них в основном базисе или любом другом. Каждую комбинационную схему проектируем отдельно, после чего для неё создаем символьное обозначение и добавляем его в схемный проект блока операций.

Комбинационная схема КС5 реализует функцию заполнения знака в старший разряд результата. Для управления комбинационной схемой используется разрешающий сигнал **y5.**

Регистр ra_sd выполняет две микрооперации – прием первого операнда и его сдвиг вправо. Извлекаем из библиотеки параметризированных модулей QUARTUS I, из раздела *storage* модуль lpm_shiftreg, и настраиваем его на выполнения микроопераций приема и сдвига.

Для реализации регистра сдвига используем два управляющих сигнала **y0** и **y11**. Первый из них определяет режим работы регистра ra. При активном сигнале **y11** выполняется прием второго операнда, иначе выполняется его сдвиг вправо. Второй управляющий сигнал **y12** разрешает регистру rb реагировать на тактовый сигнал, то есть либо принимать второй операнд, либо его сдвигать по положительному фронту тактового сигнала.

Для реализации **сумматора sum** берем из библиотеки модуль lpm_add_sub и с помощью mega_wizard настраиваем его на выполнение суммирования восьмиразрядных операндов в дополнительном коде.

Регистр признака результата rpr является просто регистром хранения. Он запоминает сформированный КС3 признак результата по положительному фронту синхросигнала, при наличии разрешающего сигнала **y10**.

Регистр rr используется для запоминания суммы частичных произведений, что требуется по техническому заданию и обнуления в начале выполнения операции умножения. По этой причине из библиотеки параметризированных модулей QUARTUS II выбираем регистр сдвига lpm_shiftreg и при настройке этого модуля в mega_wizard добавляем вход синхронной очистки его содержимого. Для управления регистром rr используем три управляющих сигнала: **y6** определяет режим работы rr (при активном его значение выполняется запись суммы в rr, иначе выполняется сдвиг влево его содержимого), **y7**

разрешает регистру rr реагировать на тактовые сигналы, а **y8** обнуляет его содержимое по положительному фронту тактового сигнала.

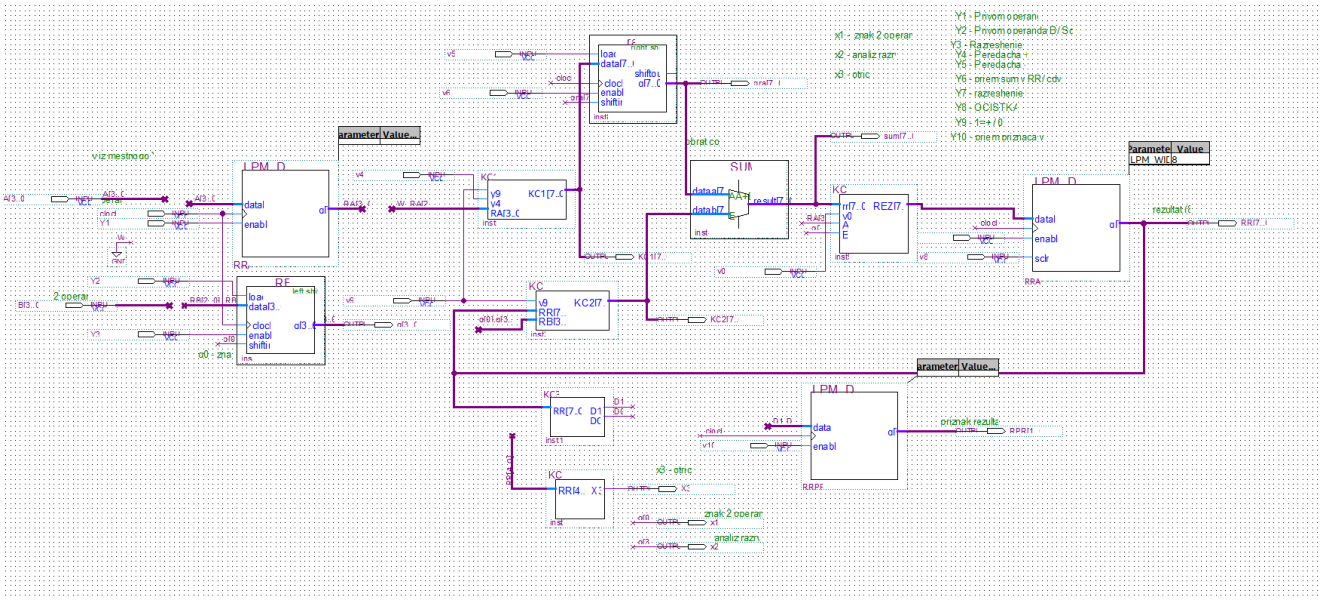


Рис.1. Полная схема блока операций

По данной схеме были выполнены примеры на умножение и отрицательного приращения. На рис.2 и рис.3 представлено моделирование примеров.

1.2. Верификация схемотехнического проектирования блока операций

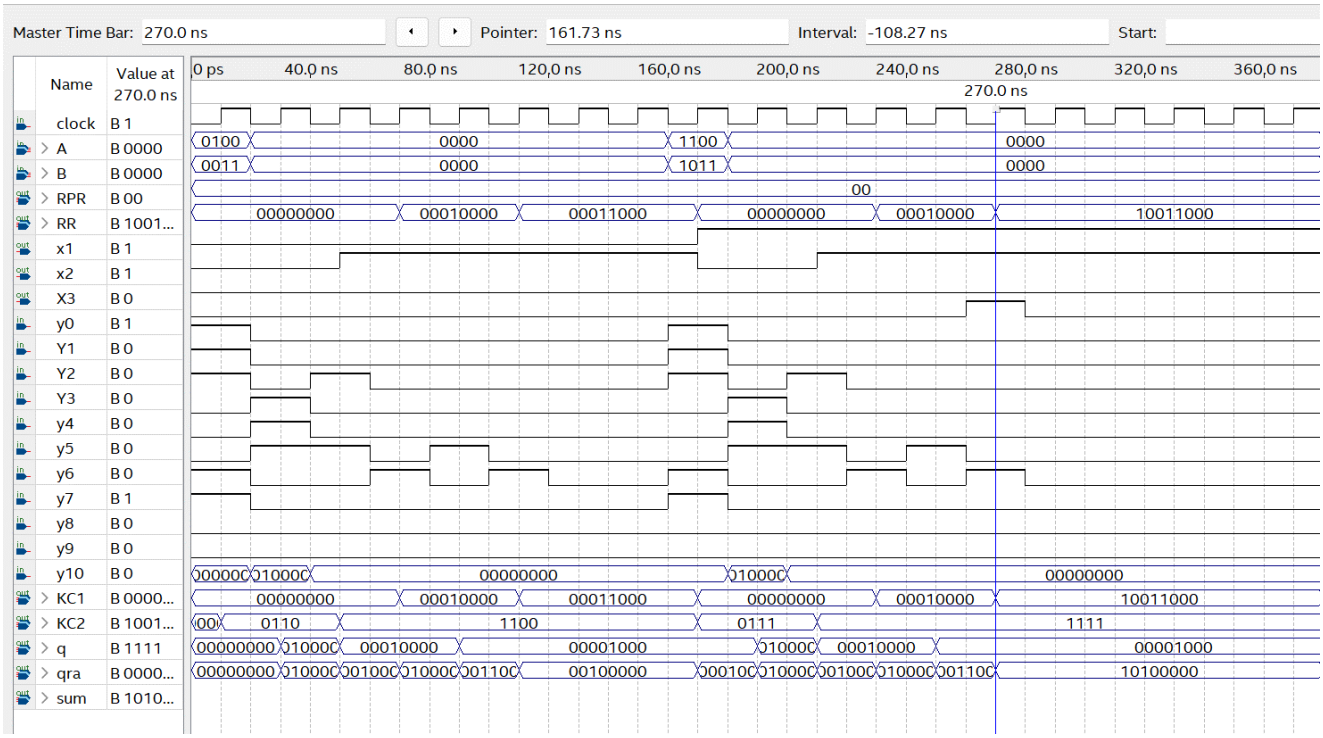


Рис.2. Моделирование схемы блока операций. Примеры на умножение.

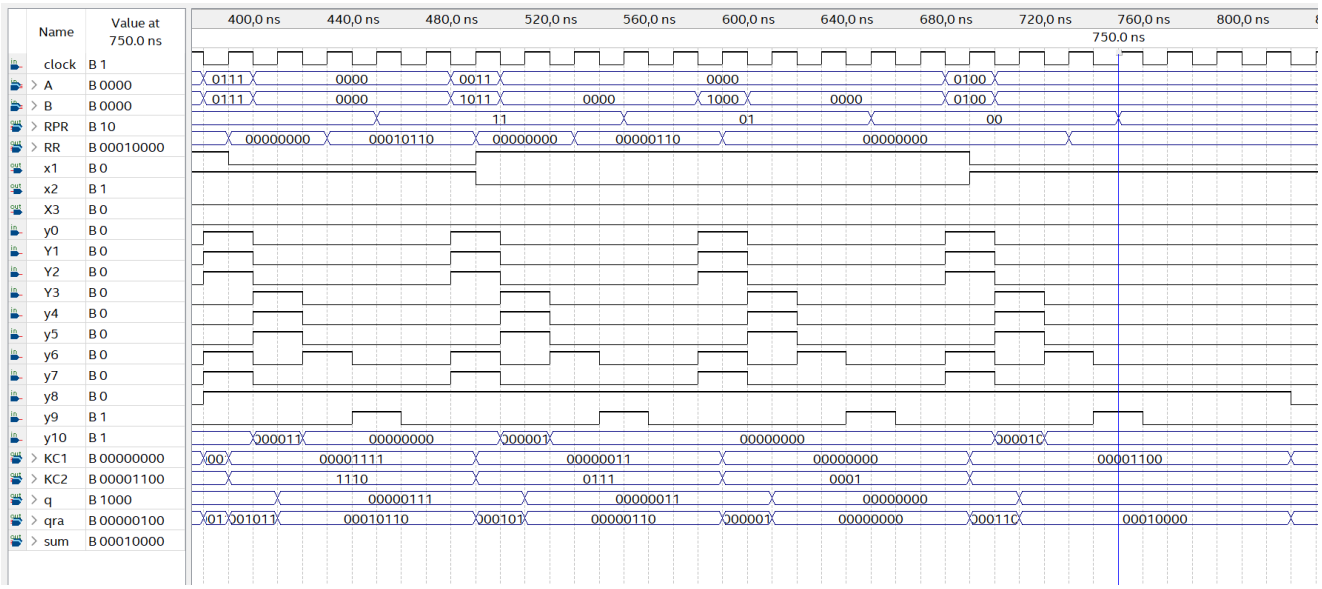


Рис.3. Моделирование схемы блока операций. Примеры второй операции.

2. Проектирование местного устройства управления

Проект представлен в виде двух модулей, интегрированных в единое арифметическое устройство с помощью схемы. ВО созданный полностью с использованием языка VHDL. Он выполняет две операции, умножение и отрицательное приращения кода. Модуль МУУ создан для реализации двух операции умножения и отрицательное приращения кода. Модуль также создан на языке vhdl и его описание содержится в одноименном файле. В entity этого модуля используется параметр n, задающий разрядность операндов. Название этого модуля подчеркивает его реализацию в виде автомата МИЛИ.

Проектирование модуля МУУ выполнено в соответствии с методическим пособием [5]. Вначале выполняется разметка закодированного графа микропрограмм операций умножения и отрицательное приращения кода. Затем составляется список переходов управляющего автомата, после чего в архитектурном теле МУУ описываются два процесса, определяющих следующее состояние и текущее состояние управляющего автомата, реализованного в виде автомата МИЛИ.

В архитектурном теле декларируется перечислимый тип с именем state_type для описания шести состояний управляющего автомата s0, s1, s2, s3, s4, достаточных для выполнения операции умножения и отрицательное приращения кода. Дальше декларируются два сигнала state и next_state, соответствующие текущему и следующему состояниям управляющего автомата. Процесс с именем TS определяет текущее состояние управляющего автомата. По внешнему сигналу set автомат принудительно устанавливается в исходное состояние s0, а по положительному фронту синхросигнала он переключается в следующее состояние, определяемое процессом с именем NS. В этом процессе описана логика формирования следующего состояния автомата и его выходных сигналов, для управления блоком операций.

В состав местного устройства управления добавлен счетчик обработанных разрядов множителя. Соответствующий сигнал **i**. Для описания поведения счетчика используется процесс count_i. В списке чувствительности этого процесса содержатся сигналы **sno** и **clk**. По сигналу **sno** счетчик асинхронно

устанавливается в исходное состояние, соответствующее единице. По положительному фронту **clk** счетчик переключается в следующее состояние при наличии разрешающего сигнала **incr_i**.

Логика формирования сигнала **sko**, оповещающего центральное устройство управления (ЦУУ) о завершении операции умножения или отрицательное приращения кода. Используется оператор условного назначения сигнала.

С целью отладки в entity устройства управления добавлены выходные сигналы **incr_i**, **s_out**, **next_state_out**. Это позволит наблюдать правильность формирования сигналов разрешения переключения для счетчика разрядов множителя, текущего и следующего состояний МУУ.

В проекте содержатся временные диаграммы для моделирования арифметического устройства по части выполнения им операции умножения и отрицательное приращения кода. Используются те же самые примеры, которые использовались для моделирования блока операций в предыдущем проекте. В отличии от временной диаграммы из файла test_for_bo теперь достаточно задавать операнды A и B и формировать сигнал начала операции **sno**. Управляющие сигналы **y[1-12]** теперь формируются местным устройством управления. Для контроля правильности их формирования они также выведены на временную диаграмму. Возможность наблюдения сигналов логических условий **x[3..1]** в процессе отладки арифметического устройства также способствует её облегчению.

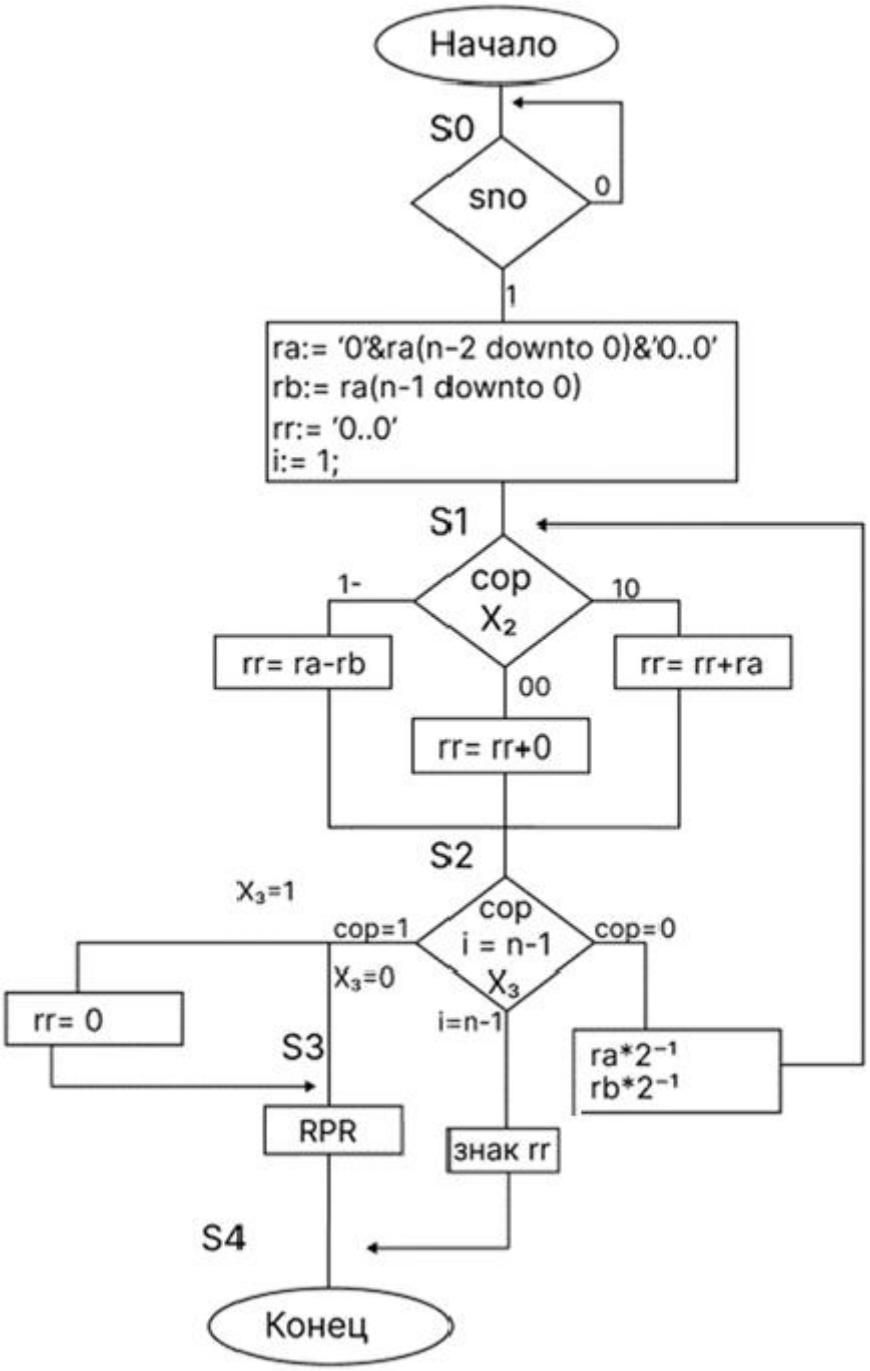


Рис. 4 Содержательный граф микропрограммы

Блок операций

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity BO is
generic (n:integer:=4);     -- n параметр, задает разрядность операндов
    Port ( a : in  STD_LOGIC_VECTOR (n-1 downto 0);-- первый операнд
                        b : in  STD_LOGIC_VECTOR (n-1 downto 0);-- второй операнд
                        y : in  STD_LOGIC_VECTOR (12 downto 1);-- управляющие сигналы
           rr : buffer  STD_LOGIC_VECTOR (2*n-1 downto 0);-- результат
           priznak : out  STD_LOGIC_VECTOR (1 downto 0); -- признак результата
                        x : out STD_LOGIC_VECTOR (3 downto 1);-- признак отрицательного нуля, анализируемый
разряд множителя, знак множителя
                        --sum_s : out  STD_LOGIC_VECTOR (2*n downto 0);-- сумматор
                        clk : in  STD_LOGIC);-- синхросигнал
end BO;
```

```
architecture Behavioral of BO is

signal RB :STD_LOGIC_VECTOR (n-1 downto 0);-- для запоминания в
signal RA : STD_LOGIC_VECTOR (2*n-1 downto 0);-- регистр сдвига a
signal d                :STD_LOGIC_VECTOR (2*n-1 downto 0);-- выход КС1
signal q                :STD_LOGIC_VECTOR (2*n-1 downto 0);-- выход КС2
signal s                :STD_LOGIC_VECTOR (2*n-1 downto 0);-- выход сумматора
signal pr               :STD_LOGIC_VECTOR (1 downto 0);-- выход КС3
signal sym     :STD_LOGIC_VECTOR (2*n downto 0); -- для вычисления суммы
signal rrr              :STD_LOGIC_VECTOR (2*n-1 downto 0);-- выход сумматора

begin


pr_RA: process (clk) -- этот процесс описывает логику работы регистра RA сдвиговый
        begin
      if clk'event and clk='1' then -- по положительному фронту
                 if y(12)='1' and  y(11)='1' and y(9) = '0' then -- если есть разрешение тактирования
                 RA(2*n-1 downto n) <= a;
                                      RA(n-1 downto 0) <=(others => '0');

                elsif y(11)='0' and y(12)='1' then  RA <= RA(2*n-1) & RA(2*n-1 downto 1); -- циклический сдвиг

                end if;
                        if y(12)='1' and  y(11)='1'  and y(9)='1' then -- если есть разрешение тактирования
                 RA(2*n-1 downto n) <= (others => '0') ;
                                      RA(n-1 downto 0) <= a;
                                    end if;
                             end if;
        end process pr_RA;

pr_RB: process(clk) -- этот процесс описывает логику работы регистра RB
    begin
        if clk'event and clk='1' then  -- по положительному фронту
            if y(3) = '1' then -- если есть разрешение тактирования
                if y(2) = '1' then
                    RB <= b; -- если разрешена загрузка, то прием второго операнда
                else
                    RB <=RB(n-1) & RB(n-3 downto 0)&'0'; -- иначе сдвиг с сохранением знака
                end if;
            end if;
        end if;
    end process pr_RB;

-- КС1
with y(4) select
        d(2*n-1 downto 0)<= '0' & RA(2*n-2 downto 0)   when '1',-- передаем +A(в старшие)если y4=1
        (others=>'0') when '0'; -- ноль в остальных случаях

-- КС2
q(2*n-1 downto n)<=RR(2*n-1 downto n) when y(9)='0' else -- RR когда умножение
                        (others=>'0'); -- старшая часть 0

q(n-1 downto 0)<=RR(n-1 downto 0) when y(9)='0' else -- когда умножение
                        (not RB(n-1)) & RB(n-2 downto 0); -- когда приращение

SM: process(d,q)
variable sym:STD_LOGIC_VECTOR (2*n downto 0); -- для вычисления суммы
begin

sym:=('0'&d)+('0'&q);-- сложение
        if (sym(2*n)='1')then sym(2*n) :='0'; sym :=sym;
   end if;

s <=sym(2*n-1 downto 0);
end process SM;

-- КС5 - знак результата
ZN: process(RA, RB, y, s)
    variable z: STD_LOGIC;
begin
z := RA(2*n-1) xor RB(n-1); -- автоматически считает знак
    if y(1) = '1' then
rrr  <= z & s(2*n-2 downto 0);
else
rrr  <= s(2*n-1 downto 0);

    end if;
end process ZN;

--znak<=z;
```

```
  pr_RR: process (clk) -- этот процесс описывает работу регистра результата

  begin
          if clk'event and clk='1' then   -- по положительному фронту синхросигнала
                  if y(8)='1' then rr<=(others=>'0');   --очистка rr
                  elsif (y(7)='1') then -- если есть разрешение тактирования
                          if y(6)='1' then rr<=rrr;--загрузка rr


                          end if;
                  end if;
          end if;
  end process pr_RR;

  --ниже приводится описание КС3, которая формирует признак результата
          pr<="00" when rr(n downto 0 ) = 0  else -- результат равен нулю, нет переноса
                  "01" when (rr (n) = '0') and rr(n-2 downto 0 ) > 0 else -- не 0 и нет переполнения
                  "10" when (rr (n) = '1') and rr(n-1 downto 0 ) = 0  else -- переполнение
                  "11"  ; -- результат меньше 0

  pr_RPR: process(clk) --этот процесс описывает работу регистра признака
  begin
          if clk'event and clk='1' then -- по положительному фронту
                  if y(10)='1' then priznak<=pr; -- запоминаем признак результата
                  end if;
          end if;
  end process pr_RPR;
  -- ниже приводится описание логических условий
  x(1)<= RB(n-1);   --знак множителя
  x(2)<= RB(n-2); --       анализируемый разряд множителя
  x(3)<= '0' when RR (n downto 0)=(2**(n+1))-1 else -- признак отрицательного нуля
                          '0'; -- иначе ноль


  end Behavioral;
```

Местное устройство управления

```
-- этот файл содержит описание МУУ в виде автомата МИЛИ, предназначенного для выполнения умножения и сложения
-- sko формируется после получения произведения

library ieee;
use ieee.std_logic_1164.all;
entity MYY is
generic (n:integer:=4);     -- n параметр, задает разрядность операндов
  port
  (
          clk              : in   std_logic; -- тактовый сигнал
          set              : in   std_logic; --  сигнал начальной установки
          cop              : in   std_logic; --  код операции 1-умножение,0 - сложение
          x                       : in   std_logic_vector(3 downto 1);-- логические условия,x3,x2,x1
          sno              : in   std_logic; -- сигнал начала операции
          sko              : out  std_logic; -- сигнал конца операции
          y                       : out  std_logic_vector(12 downto 1); -- управляющие сигналы для блока
  операций
          -- следующие сигналы добавлены для отладки
          incr_i   : buffer std_logic; -- разрешение инкремента i
          s_out            : out   integer range 0 to 4; --  отладочный выход для наблюдения состояний
          next_state_out : out   integer range 0 to 4 --  отладочный выход для наблюдения состояний
  );

end entity;

architecture rtl of MYY is

  type state_type is (s0, s1, s2, s3, s4); -- определяем состояния МУУ

  signal next_state, state : state_type; -- следующее состояние, текущее состояние
  signal i : integer range 1 to 3 ; -- счетчик анализируемых разрядов множителя

begin

TS: process (clk,set) -- этот процесс определяет текущее состояние МУУ
   begin
          if set = '1' then
                  state <= s0;
          elsif (rising_edge(clk)) then -- по положительному фронту переключаются состояния
                  state <= next_state;
          end if;
```

```
   end process;

NS: process (state,sno,cop,x,i) -- этот процесс определяет следующее состояние МУУ, управляющие сигналы для БО
   begin

                  case state is
                          when s0=> -- переходы из s0
                           if  (sno = '1' and cop='1') then       -- если вторая операция
                                   next_state <= s1; y<="110101100110";
                                elsif (sno = '1' and cop='0') then      --
                                   next_state <= s4; y<="110011100110";
                                   elsif (sno = '0' ) then
                                   next_state <= s0; y<="000000000000";
                                end if;


                          when s1=>
                          next_state <= s2;

                          if cop='0' and x(2) = '1'  then
                                   y<="000001101000"; -- rr=rr +RA

                                elsif cop='0' and x(2) = '0'  then
                                   y<="000000000000"; -- rr=rr+0 - пропускаем и делаем сдвиг
                                elsif  cop='1' then     -- если приращение
                                   y<="000101101000";

                                end if;
                                --end if;

                          when s2=>
                                if i = n-1 then
                                   next_state <= s3; y<="000001100001"; -- формируем сигнал конца операции
                                elsif cop='0' then                                                    -- если
умножение
                                   next_state <= s1; y<="100000000100";  -- иначе сдвиг ra, сдвиг RB
                                elsif cop='1' then                       -- если 2оп
                                   next_state <= s3; y<="001100000000";  -- иначе запись признака в RPR
                                end if;

                          when s3 =>
                                   next_state <= s0; y<="000000000000";  -- иначе запись признака в RPR

                          when s4 =>
                                   next_state <= s1; y<="100000000000";  -- формируем сигнал конца
операции

                  end case;
  end process;

  sko<='1' when (state=s3 and (i=n-1))  or state =s3 else -- формирование sko
                  '0';
  incr_i<='1' when state=s2 and cop='0' and i/=n-1 else --инкремент i, когда умножение и не последний разряд
  множителя
                  '0';
count_i:   process (sno, clk) -- этот процесс определяет поведение счетчика i

  begin
          if (sno='1') then i<=1; --устанавливаем в начальное состояние
          elsif clk'event and clk='1' then
            if (incr_i='1') then i<=i+1; -- инкремент счетчика
            end if;
           end if;
  end process;
  s_out<=0 when state=s0 else
                   1 when state=s1 else
                   2 when state=s2 else
                   3 when state=s3 else
                   4;
  next_state_out<=0 when next_state=s0 else
                   1 when next_state=s1 else
                   2 when next_state=s2 else
                   3 when next_state=s3 else
                   4;


end rtl;
```

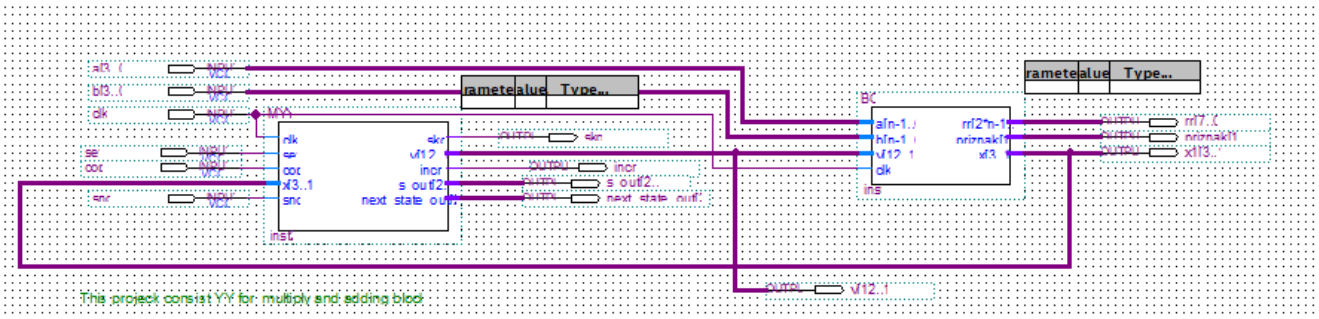


Рис.5.Схема местного устройства управления

1.2 Верификация местного устройства управления

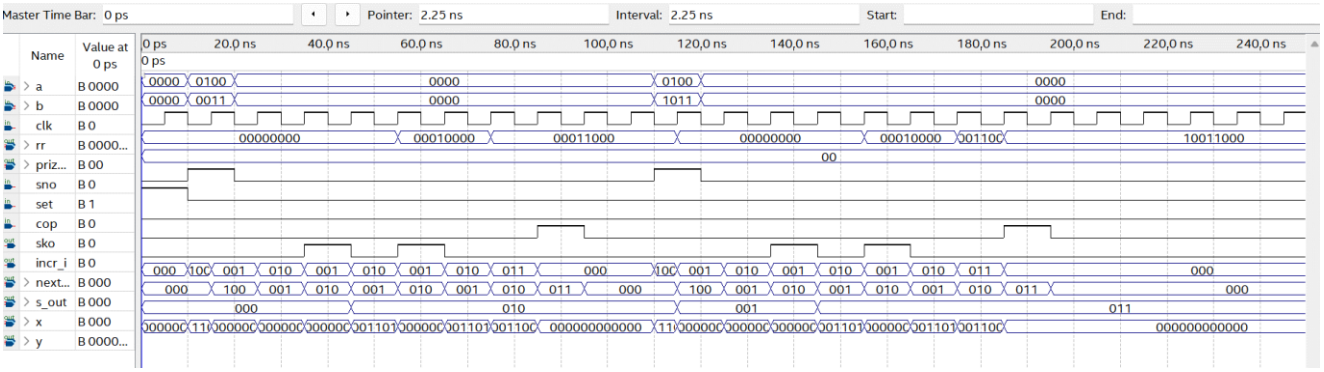


Рис.6. моделирование местного устройства управления. Операция умножения

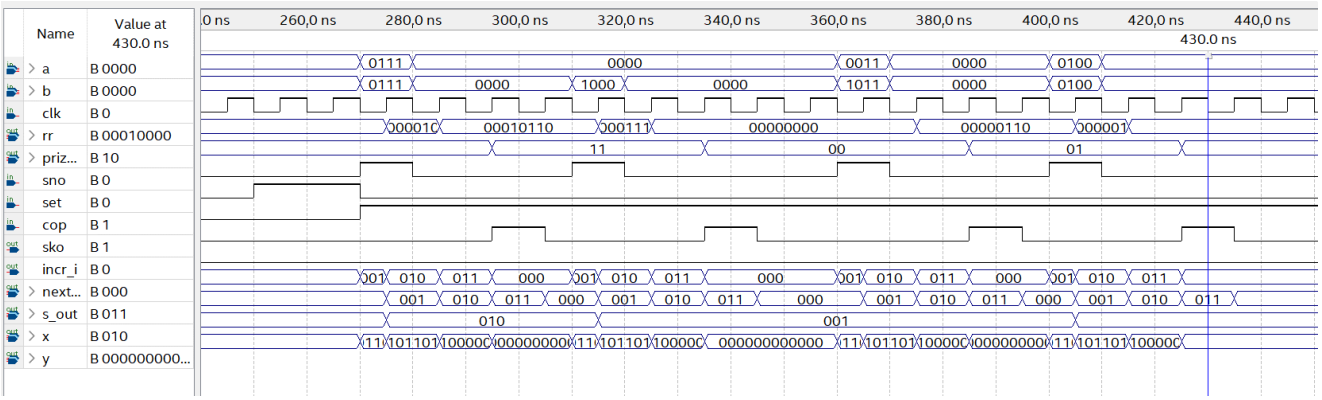


Рис.6. моделирование местного устройства управления. Вторая операция

3. Проектирование центрального устройства управления (ЦУУ)

3.1. Разработка формата команд

Каждая команда должна кодироваться целым количеством слов, чтобы её можно было разместить в оперативной памяти процессорной системы. В соответсвии с ТЗ разрядность слова составляет восемь разрядов, то есть один байт. Поэтому длина команды должна быть кратна байту.

Обязательным полем в каждой команде является поле кода операции **cop**, задающее выполняемую операцию. В рассматриваемом примере требуется закодировать три операции. С целью упрощения отладки реализованного в кристалле ПЛИС учебного процессора будет полезным добавить к системе команд еще одну команду - останов. Следовательно, для задания **cop** достаточно будет двух разрядов в коде команды.

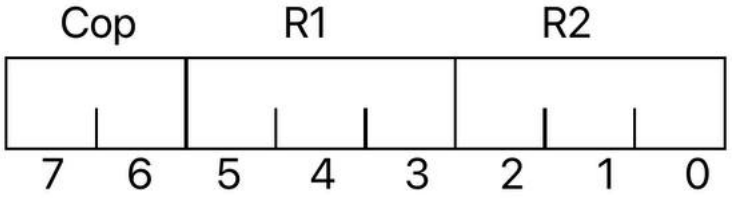


Рис. 9. Формат команд регистр-регистр

Разрядность адреса операнда в ОП составляет 8 разрядов, разрядность адреса операнда в регистровой памяти составляет три разряда.

Для короткой операции используется формат «регистр – регистр» используется для короткой операции и содержит в себе только одно слово. В поле R1 хранится адрес регистровой памяти для первого операнда, а в поле R2 – адрес для второго операнда.

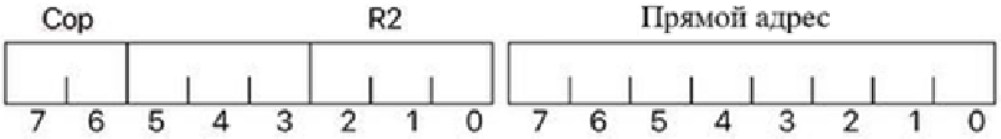
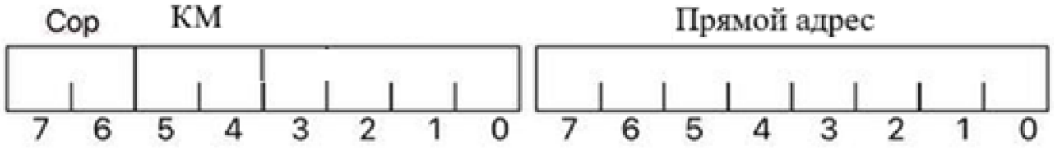


Рис. 10. Формат команд память – память

Для длинной операции используется формат «непосредственный операнд - память» и используется два слова для кодирования данной команды. Первое поле будет содержать в себе только поле cop – код операции и косвенный адрес

регистровой памяти. (Далее в регистровой памяти будет храниться адрес в оперативную память, где и будет находится второй операнд), во втором слове – прямой адрес первого операнда.



Для условного перехода используется формат «непосредственный операнд - память», он будет располагаться в двух ячейках памяти. КМ будет отвечать за код маски, а во втором слове – прямой адрес для перехода, при выполнении условия.

3.2. Составление содержательных графов алгоритмов выполнения команд

Содержательный граф алгоритма выполнения операции умножения

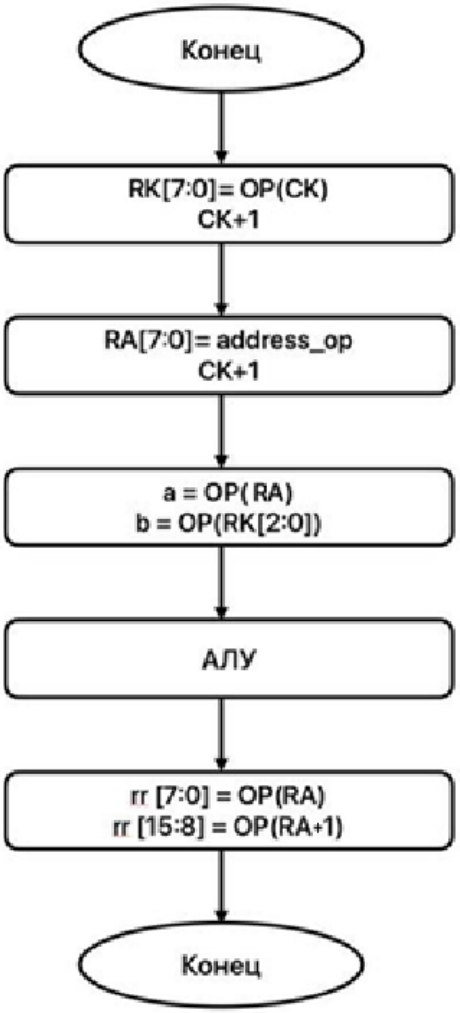


Рис 11. Содержательный граф формата память- память

Содержательный граф алгоритма выполнения операции сложения кодов

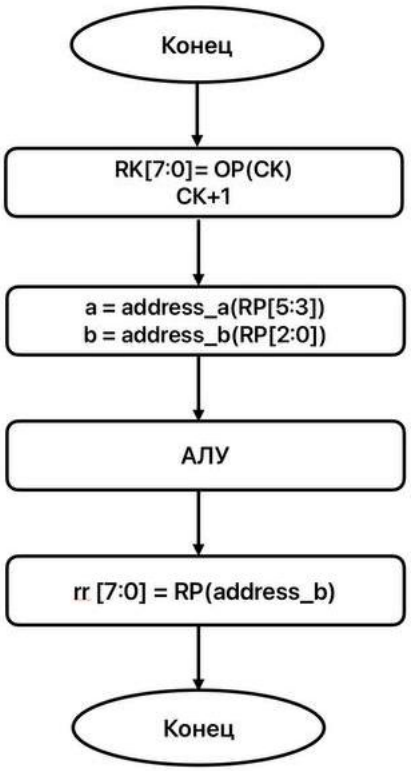


Рис 12. Содержательный граф для второй операции, формата регистр-регистр

Содержательный граф алгоритма выполнения операции перехода

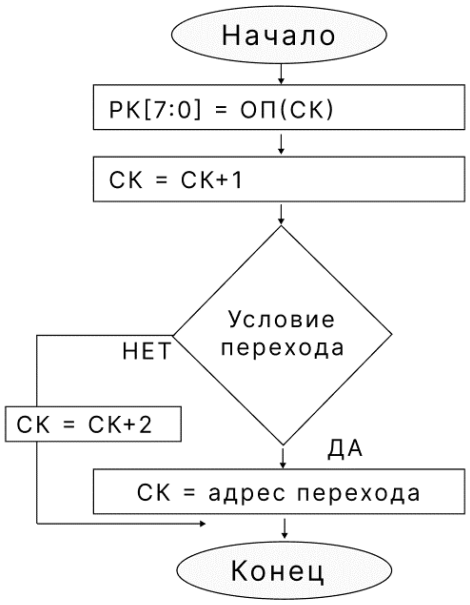


Рис 13. Содержательный граф для перехода

3.3. Функциональные узлы, входящие в состав ЦУУ. Их назначение и связь с другими узлами и модулями в процессорной системе

Проектирование блока, предназначенного для выполнения заданных в ТЗ

операций, на предыдущем этапе проектирования мы начинали с разработки алгоритмов их выполнения. Затем была создана функциональная схема блока операций (БО), в которой были объединены в единую структуру все функциональные узлы, необходимые для выполнения требуемых действий над словами информации с целью формирования нужного результата. Потом мы обсудили выполняемые каждым узлом действия над словами информации, то есть микрооперации и ввели соответствующие управляющие сигналы.

На следующем этапе проектирования мы разработали местное устройство управления, формирующее эти управляющие сигналы. Следует напомнить, что управляющие сигналы зависят от логических условий формируемых блоком операций и от выполняемой операции.

Поступим аналогичным образом при проектировании ЦУУ. Вначале обсудим, какие функциональные узлы надо включить в состав ЦУУ, что они должны делать, то есть какие выполнять действия и над какими словами информации, какие сигналы потребуются для управления ими, и каким образом они должны взаимодействовать с другими узлами.

Счетчик команд (СК)

Предназначен для адресации в ОП текущей выполняемой команды. Возможные варианты его изменения перечислены ниже.

Загрузка начального адреса $A_{нач}$ для выполнения программы с начала. Для упрощения проекта будем считать, что выполнение программы всегда начинается с нулевого адреса. Для установки счетчика СК в это состояние будем использовать внешний сигнал начальной установки **set**.

Инкремент счетчика для адресации в ОП следующего слова текущей команды либо следующей команды. Адрес следующей команды будем называть продвинутым адресом.

Загрузка в СК нового значения для выполнения команд перехода. Если используется прямая адресация, то прямой адрес извлекается из команды. Если используется косвенная адресация, то в команде указывается адрес на адрес значения в памяти.

Для увеличения счетчика команд СК введем разрешающий сигнал **incr_CK.** Для выполнения перехода необходимо предусмотреть возможность прибавления к его текущему значению знакового смещения.

Таким образом, управляющими сигналами для СК в нашем проекте будут:

Сигнал **set**, для сброса счетчика;

Сигнал **incr_CK**, для его увеличения и сигнал **sum_CK**, для записи в счетчик адреса перехода. Помимо управляющих сигналов на счетчик команд необходимо подать ещё и вычисленный адрес перехода.

В счетчике команд СК асинхронно по единичному сигналу **set** устанавливается нулевое значение. При наличии разрешающего сигнала **incr_CK** он по положительному фронту синхросигнала переключается в следующее состояние, а при наличии разрешающего сигнала **sum_CK** он по положительному фронту синхросигнала загружает новое значение.

Регистр команд (RK)

Регистр команд предназначен для хранения команды или её части в процессе её выполнения процессором. В нашем случае в качестве RK используем восьмиразрядный регистр, который умеет только принимать информацию со своего входа. Так как команда всегда извлекается из ОП вход RK должен быть подключен к выходу данных ОП. Для управления RK используем разрешающий сигнал **load_RK**.

Регистр признака результата (RPR)

Регистр признака результата ранее был включен в блок операций, поэтому соответствующий выход БО подключаем к ЦУУ для выполнения команд условного перехода, в которых он должен быть проанализирован.

Модуль оперативной памяти (ОП)

Следует заметить, что оперативная память является самостоятельной частью процессорной системы и в состав ЦУУ не входит. Однако в проекте ЦУУ она должна присутствовать в виде отдельного модуля, с которым ЦУУ непосредственно взаимодействует.

ОП используется для хранения программы и данных. В нашем случае на

адресный вход ОП необходимо подать либо СК, для выборки команды, либо RA для выборки из ОП первого операнда, либо RI – для второго операнда. Произведение также помещается в ОП и записывается в две смежные ячейки памяти по адресу первого операнда.

На вход данных ОП необходимо подать две половинки произведения, для записи его в две смежные ячейки ОП, для чего потребуется комбинационная схема, выполняющая функцию коммутатора.

Выход данных оперативной памяти подключаем к регистру команд RK, для сохранения в нём выбранной команды, к входу сумматора, вычисляющего исполнительный адрес операнда в ОП для передачи в него смещения и к шине **b**, используемой для передачи первого операнда из ОП в блок операций.

Модуль регистровой памяти (РП)

Регистровая память РП также является самостоятельной частью процессора и в состав ЦУУ не входит. В проект ЦУУ РП входит в виде отдельного модуля.

Для второй операции – сложения кодов, первый и второй операнды находятся в РП и адресуются с помощью прямого адреса, причем её результат, то есть сумма записывается обратно в РП и помещается по адресу первого операнда.

Выход данных РП первого порта подключаем ко входу сумматора, для передачи в него базового адреса с целью вычисления исполнительного адреса. Вход данных первого порта соединяем с младшей половиной регистра результата БО, для записи результата в регистр процессора.

3.4. Подход, используемый при проектировании учебного процессора

ЦУУ, входящее в состав центрального процессора, должно взаимодействовать с компонентами памяти, то есть ОП и РП и с реализованным ранее арифметическим устройством, для выполнения заданных в ТЗ операций. САПР QUARTUS II предлагает для использования в проектах большое количество мегафункций - универсальных параметризованных модулей, одним из которых является модуль синхронной памяти altsyncram. Немаловажным параметром, настраиваемым пользователем, является возможность использования файла

инициализации модуля памяти. Это позволит в нашем проекте выполнить загрузку в ОП исходной программы и данных, что существенно облегчит процесс проектирования.

В состав ЦУУ входят перечисленные в предыдущем разделе функциональные узлы - счетчик команд СК, регистр команд RK, регистр адреса RA и несколько комбинационных схем. Будем описывать их поведение на языке VHDL с использованием параллельного оператора process.

3.5. Реализация проекта центрального устройства управления

ctrl_un_BO. В этом файле содержится описание блока операций и местного устройства управления

```
-- этот файл содержит описание операционного устройства для выполнения умножения и сложения
-- он представляет собой vhdl описание схемного проекта contr_unit_BO, представленного на верхнем уровне как
МУУ(файл control unit) + БО (файл BO)
-- В описании представлено entity и архитектурное тело операционного устройства
-- Операнды n разрядные

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity ctrl_un_BO is
generic (n:integer);     -- n параметр, задает разрядность операндов
        port
        (
                a : in  STD_LOGIC_VECTOR (n-1 downto 0);-- первый операнд
                b : in  STD_LOGIC_VECTOR (n-1 downto 0);-- второй операнд
                clk              : in   std_logic; -- тактовый сигнал
                set              : in   std_logic; --  сигнал начальной установки
                cop              : in   std_logic; --  код операции 1-умножение,0 - сложение

                sno              : in   std_logic; -- сигнал начала операции
                rr               : buffer  STD_LOGIC_VECTOR (2*n-1 downto 0);-- результат
      priznak    : out  STD_LOGIC_VECTOR (1 downto 0); -- признак результата
                sko              : out  std_logic -- сигнал конца операции

        );

end entity;

architecture arch of ctrl_un_BO is

        type state_type is (s0, s1, s2, s3, s4); -- определяем состояния МУУ

        signal next_state, state : state_type; -- следующее состояние, текущее состояние
        signal i : integer range 1 to n-1 ;    -- счетчик анализируемых разрядов множителя
        signal incr_i    :std_logic;            -- разрешение инкремента i
        signal RA       :STD_LOGIC_VECTOR (2*n-1 downto 0);-- для запоминания a и в
        signal RB       :STD_LOGIC_VECTOR (n-1 downto 0);-- для запоминания a и в

signal d                :STD_LOGIC_VECTOR (2*n-1 downto 0);-- выход КС1
signal q                :STD_LOGIC_VECTOR (2*n-1 downto 0);-- выход КС2
signal s                :STD_LOGIC_VECTOR (2*n-1 downto 0);-- выход сумматора
signal pr               :STD_LOGIC_VECTOR (1 downto 0);-- выход КС3
signal sym     :STD_LOGIC_VECTOR (2*n downto 0); -- для вычисления суммы
signal rrr              :STD_LOGIC_VECTOR (2*n-1 downto 0);-- выход сумматора
signal x                :std_logic_vector (3 downto 1);-- логические условия
signal y                :std_logic_vector(12 downto 1); -- управляющие сигналы для блока операций

begin

TS: process (clk,set) -- этот процесс определяет текущее состояние МУУ
```

```
         begin
                if set = '1' then
                        state <= s0;
                elsif (rising_edge(clk)) then -- по положительному фронту переключаются состояния
                        state <= next_state;
                end if;
         end process;

NS: process (state,sno,cop,x,i) -- этот процесс определяет следующее состояние МУУ, управляющие сигналы для БО
         begin
                        case state is
                                when s0=> -- переходы из s0
                                 if  (sno = '1' and cop='1') then       -- если вторая операция
                                                next_state <= s1; y<="110101100110";
                                        elsif (sno = '1' and cop='0') then      --
                                                next_state <= s4; y<="110011100110";
                                                elsif (sno = '0' ) then
                                                next_state <= s0; y<="000000000000";
                                        end if;


                                when s1=>
                                next_state <= s2;

                                if cop='0' and x(2) = '1'  then
                                                y<="000001101000"; -- rr=rr +RA

                                        elsif cop='0' and x(2) = '0'  then
                                                y<="000000000000"; -- rr=rr+0 - пропускаем и делаем сдвиг
                                        elsif  cop='1' then     -- если приращение
                                                y<="000101101000";

                                        end if;
                                        --end if;

                                when s2=>
                                        if i = n-1 then
                                                next_state <= s3; y<="000001100001"; -- формируем сигнал конца
операции
                                        elsif cop='0' then
        -- если умножение
                                                next_state <= s1; y<="100000000100";  -- иначе сдвиг ra, сдвиг
RB
                                        elsif cop='1' then                      -- если 2оп
                                           next_state <= s3; y<="001100000000";  -- иначе запись признака в RPR
                                        end if;

                                when s3 =>
                                                next_state <= s0; y<="000000000000";  -- иначе запись признака
в RPR
                                when s4 =>
                                                next_state <= s1; y<="100000000000";  -- формируем сигнал конца
операции

                        end case;
        end process;

        sko<='1' when (state=s3 and (i=n-1))  or state =s3 else -- формирование sko
                        '0';
        incr_i<='1' when state=s2 and cop='0' and i/=n-1 else --инкремент i, когда умножение и не последний
разряд множителя
                        '0';

count_i:   process (sno, clk) -- этот процесс определяет поведение счетчика i

        begin
                if (sno='1') then i<=1; --устанавливаем в начальное состояние
                elsif clk'event and clk='1' then
                  if (incr_i='1') then i<=i+1; -- инкремент счетчика
                  end if;
                 end if;
        end process;



pr_RA: process (clk) -- этот процесс описывает логику работы регистра RA сдвиговый
        begin
      if clk'event and clk='1' then -- по положительному фронту
                 if y(12)='1' and  y(11)='1' and y(9) = '0' then -- если есть разрешение тактирования
                 RA(2*n-1 downto n) <= a;
                                        RA(n-1 downto 0) <=(others => '0');
```

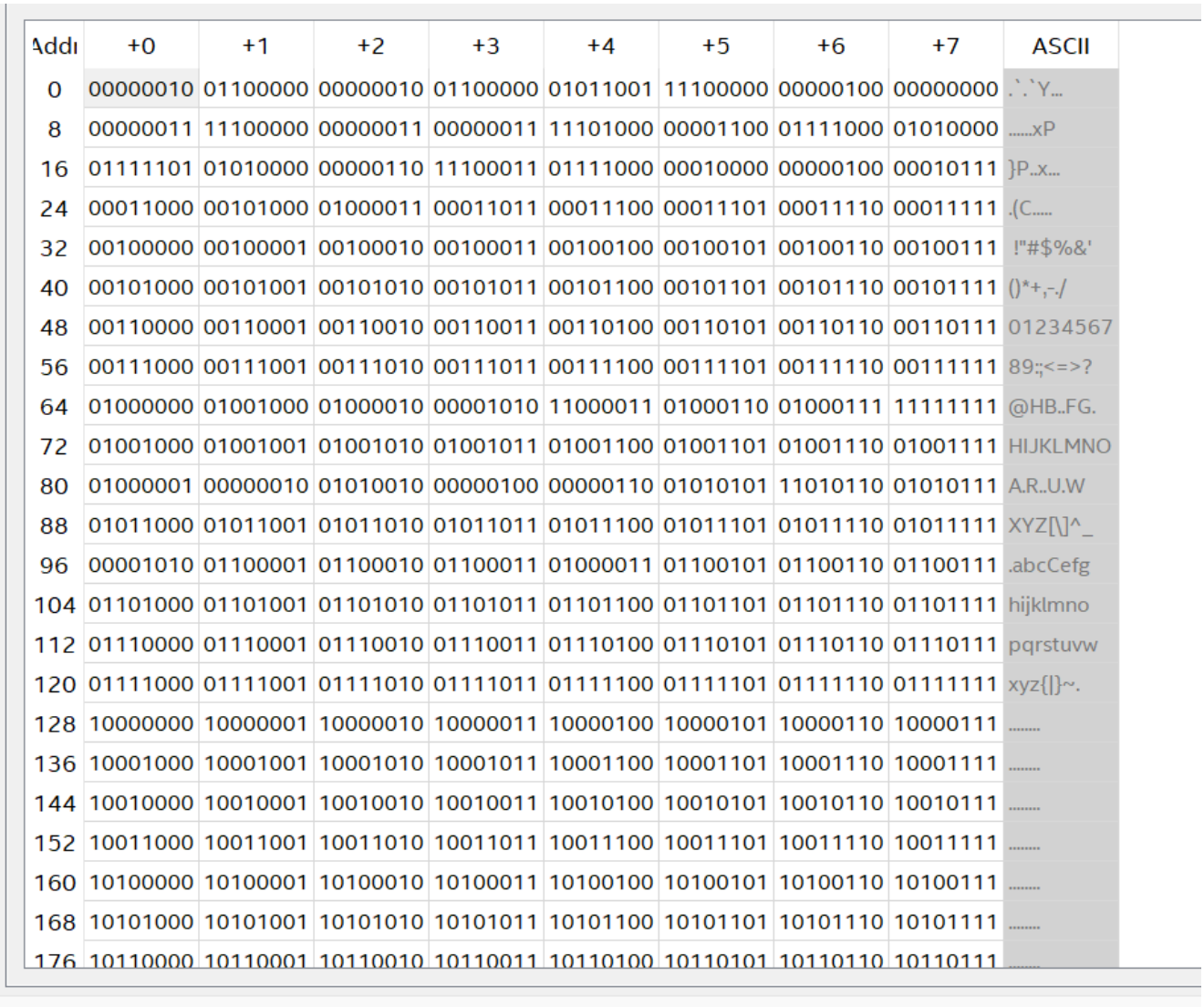
```
                elsif y(11)='0' and y(12)='1' then  RA <= RA(2*n-1) & RA(2*n-1 downto 1); -- циклический сдвиг

                end if;
                        if y(12)='1' and  y(11)='1'  and y(9)='1' then -- если есть разрешение тактирования
                 RA(2*n-1 downto n) <= (others => '0') ;
                                        RA(n-1 downto 0) <= a;
                                    end if;
                            end if;
        end process pr_RA;

pr_RB: process(clk) -- этот процесс описывает логику работы регистра RB
    begin
        if clk'event and clk='1' then  -- по положительному фронту
            if y(3) = '1' then -- если есть разрешение тактирования
                if y(2) = '1' then
                    RB <= b; -- если разрешена загрузка, то прием второго операнда
                else
                    RB <=RB(n-1) & RB(n-3 downto 0)&'0'; -- иначе сдвиг с сохранением знака
                end if;
            end if;
        end if;
    end process pr_RB;

-- KC1
with y(4) select
        d(2*n-1 downto 0)<= '0' & RA(2*n-2 downto 0)   when '1',-- передаем +A(в старшие)если y4=1
        (others=>'0') when '0'; -- ноль в остальных случаях

-- KC2
q(2*n-1 downto n)<=RR(2*n-1 downto n) when y(9)='0' else -- RR когда умножение
                        (others=>'0'); -- старшая часть 0

q(n-1 downto 0)<=RR(n-1 downto 0) when y(9)='0' else -- когда умножение
                        (not RB(n-1)) & RB(n-2 downto 0); -- когда приращение

SM: process(d,q)
variable sym:STD_LOGIC_VECTOR (2*n downto 0); -- для вычисления суммы
begin

sym:=('0'&d)+('0'&q);-- сложение
        if (sym(2*n)='1')then sym(2*n) :='0'; sym :=sym;
   end if;

s <=sym(2*n-1 downto 0);
end process SM;

-- KC5 - знак результата
ZN: process(RA, RB, y, s)
    variable z: STD_LOGIC;
begin
z := RA(2*n-1) xor RB(n-1); -- автоматически считает знак
    if y(1) = '1' then
rrr  <= z & s(2*n-2 downto 0);
else
rrr  <= s(2*n-1 downto 0);

    end if;
end process ZN;

--znak<=z;

pr_RR: process (clk) -- этот процесс описывает работу регистра результата

begin
        if clk'event and clk='1' then   -- по положительному фронту синхросигнала
                if y(8)='1' then rr<=(others=>'0');   --очистка rr
                elsif (y(7)='1') then -- если есть разрешение тактирования
                        if y(6)='1' then rr<=rrr;--загрузка rr


                        end if;
                end if;
        end if;
end process pr_RR;

--ниже приводится описание KC3, которая формирует признак результата
        pr<="00" when rr(n downto 0 ) = 0  else -- результат равен нулю, нет переноса
                "01" when (rr (n) = '0') and rr(n-2 downto 0 ) > 0 else -- не 0 и нет переполнения
                "10" when (rr (n) = '1') and rr(n-1 downto 0 ) = 0  else -- переполнение
                "11"  ; -- результат меньше 0
```

```
pr_RPR: process(clk) --этот процесс описывает работу регистра признака
begin
        if clk'event and clk='1' then -- по положительному фронту
                if y(10)='1' then priznak<=pr; -- запоминаем признак результата
                end if;
        end if;
end process pr_RPR;
-- ниже приводится описание логических условий
x(1)<= RB(n-1);   --знак множителя
x(2)<= RB(n-2); --      анализируемый разряд множителя
x(3)<= '0' when RR (n downto 0)=(2**(n+1))-1 else -- признак отрицательного нуля
                        '0'; -- иначе ноль



--      s_out<=0 when state=s0 else
--                       1 when state=s1 else
--                       3;
--      next_state_out<=0 when next_state=s0 else
--                       1 when next_state=s1 else
--                       2 when next_state=s2 else
--                       3;

end arch;
```

Memory. Этот файл создается в самом САПР QUARTUS II и используем для подключения портов и файла для оперативной памяти.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

LIBRARY altera_mf;
USE altera_mf.altera_mf_components.all;

ENTITY memory IS
        PORT
        (
                address         : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
                clock           : IN STD_LOGIC  := '1';
                data            : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
                wren            : IN STD_LOGIC ;
                q               : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
        );
END memory;


ARCHITECTURE SYN OF memory IS

        SIGNAL sub_wire0: STD_LOGIC_VECTOR (7 DOWNTO 0);

BEGIN
        q    <= sub_wire0(7 DOWNTO 0);

        altsyncram_component : altsyncram
        GENERIC MAP (
                clock_enable_input_a => "BYPASS",
                clock_enable_output_a => "BYPASS",
                init_file => "mif.mif",
                intended_device_family => "Cyclone IV E",
                lpm_hint => "ENABLE_RUNTIME_MOD=YES,INSTANCE_NAME=hgh",
                lpm_type => "altsyncram",
                numwords_a => 256,
                operation_mode => "SINGLE_PORT",
                outdata_aclr_a => "NONE",
                outdata_reg_a => "UNREGISTERED",
                power_up_uninitialized => "FALSE",
                read_during_write_mode_port_a => "NEW_DATA_NO_NBE_READ",
                widthad_a => 8,
                width_a => 8,
                width_byteena_a => 1
        )
        PORT MAP (
                address_a => address,
                clock0 => clock,
                data_a => data,
                wren_a => wren,
                q_a => sub_wire0
```

```
	);

END SYN;
```



Addı	+0	+1	+2	+3	+4	+5	+6	+7	ASCII
0	00000010	01100000	00000010	01100000	01011001	11100000	00000100	00000000	.`.`Y...
8	00000011	11100000	00000011	00000011	11101000	00001100	01111000	01010000	.....xP
16	01111101	01010000	00000110	11100011	01111000	00010000	00000100	00010111	}P..x...
24	00011000	00101000	01000011	00011011	00011100	00011101	00011110	00011111	.(C.....
32	00100000	00100001	00100010	00100011	00100100	00100101	00100110	00100111	!"#$%&'
40	00101000	00101001	00101010	00101011	00101100	00101101	00101110	00101111	()*+,-./
48	00110000	00110001	00110010	00110011	00110100	00110101	00110110	00110111	01234567
56	00111000	00111001	00111010	00111011	00111100	00111101	00111110	00111111	89:;<=>?
64	01000000	01001000	01000010	00001010	11000011	01000110	01000111	11111111	@HB..FG.
72	01001000	01001001	01001010	01001011	01001100	01001101	01001110	01001111	HIJKLMNO
80	01000001	00000010	01010010	00000100	00000110	01010101	11010110	01010111	A.R..U.W
88	01011000	01011001	01011010	01011011	01011100	01011101	01011110	01011111	XYZ[\]^_
96	00001010	01100001	01100010	01100011	01000011	01100101	01100110	01100111	.abcCefg
104	01101000	01101001	01101010	01101011	01101100	01101101	01101110	01101111	hijklmno
112	01110000	01110001	01110010	01110011	01110100	01110101	01110110	01110111	pqrstuvw
120	01111000	01111001	01111010	01111011	01111100	01111101	01111110	01111111	xyz{|}~.
128	10000000	10000001	10000010	10000011	10000100	10000101	10000110	10000111	.....
136	10001000	10001001	10001010	10001011	10001100	10001101	10001110	10001111	.....
144	10010000	10010001	10010010	10010011	10010100	10010101	10010110	10010111	.....
152	10011000	10011001	10011010	10011011	10011100	10011101	10011110	10011111	.....
160	10100000	10100001	10100010	10100011	10100100	10100101	10100110	10100111	.....
168	10101000	10101001	10101010	10101011	10101100	10101101	10101110	10101111	.....
176	10110000	10110001	10110010	10110011	10110100	10110101	10110110	10110111	.....

Рис. 14. Файл с оперативной памятью

Ram_2port_11. Этот файл создается в самом САПР QUARTUS II и используем для подключения портов и файла для регистровой памяти.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

LIBRARY altera_mf;
USE altera_mf.altera_mf_components.all;

ENTITY Ram_2port_11 IS
	PORT
	(
		address_a		: IN STD_LOGIC_VECTOR (2 DOWNTO 0);
		address_b		: IN STD_LOGIC_VECTOR (2 DOWNTO 0);
		clock		: IN STD_LOGIC  := '1';
		data_a		: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
		data_b		: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
		wren_a		: IN STD_LOGIC  := '0';
		wren_b		: IN STD_LOGIC  := '0';
		q_a		: OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
		q_b		: OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
	);
END Ram_2port_11;
```

```
ARCHITECTURE SYN OF ram_2port_11 IS

        SIGNAL sub_wire0: STD_LOGIC_VECTOR (7 DOWNTO 0);
        SIGNAL sub_wire1: STD_LOGIC_VECTOR (7 DOWNTO 0);

BEGIN
        q_a    <= sub_wire0(7 DOWNTO 0);
        q_b    <= sub_wire1(7 DOWNTO 0);

        altsyncram_component : altsyncram
        GENERIC MAP (
                address_reg_b => "CLOCK0",
                clock_enable_input_a => "BYPASS",
                clock_enable_input_b => "BYPASS",
                clock_enable_output_a => "BYPASS",
                clock_enable_output_b => "BYPASS",
                indata_reg_b => "CLOCK0",
                init_file => "file_init_RP.mif",
                intended_device_family => "Cyclone IV GX",
                lpm_type => "altsyncram",
                numwords_a => 8,
                numwords_b => 8,
                operation_mode => "BIDIR_DUAL_PORT",
                outdata_aclr_a => "NONE",
                outdata_aclr_b => "NONE",
                outdata_reg_a => "UNREGISTERED",
                outdata_reg_b => "UNREGISTERED",
                power_up_uninitialized => "FALSE",
                read_during_write_mode_mixed_ports => "OLD_DATA",
                read_during_write_mode_port_a => "NEW_DATA_NO_NBE_READ",
                read_during_write_mode_port_b => "NEW_DATA_NO_NBE_READ",
                widthad_a => 3,
                widthad_b => 3,
                width_a => 8,
                width_b => 8,
                width_byteena_a => 1,
                width_byteena_b => 1,
                wrcontrol_wraddress_reg_b => "CLOCK0"
        )
        PORT MAP (
                clock0 => clock,
                wren_a => wren_a,
                address_b => address_b,
                data_b => data_b,
                wren_b => wren_b,
                address_a => address_a,
                data_a => data_a,
                q_a => sub_wire0,
                q_b => sub_wire1
        );

END SYN;
```

ıdd	+0	+1	+2	+3	+4	+5	+6	+7	ASCII
0	80	168	84	60	96	4	3	1	P.T<`...

Рис. 15. Файл с регистровой памятью

CYY_AU_OP_RP_F. В данном файле описывается вся работа центрального устройства управления. Здесь описывается взаимодействия оперативной и регистровой памяти. А также алгоритм записи двух операндов и вывод результата по адресу одного из операндов.

```
---  В этом файле приводится описание центрального устройства управления ЦУУ.
-- Оно должно формировать управляющие сигналы для ОП, РП и спроектированного ранее арифметического устройства
-- Внешними сигналами для ЦУУ являются сигналы set, по которому ЦУУ, устанавливается в исходное состояние и
тактовый сигнал clk
-- Для арифметического устройства оно готовит операнды A и B, задает cop и подает сигнал начала операции sno,
после их подготовки
-- Из арифметического устройства оно забирает результат,
```

```
-- для умножения это 2n-разрядное произведение, для сложения -n-разрядная сумма и двухразрядный признак
результата.
-- Сигналом, подтверждающим выполнение операции в арифметическом устройстве, является сигнал конца операции sko
-- Для оперативной памяти оно формирует следующие сигналы:
-- data_in_OP [7:0] - данные для записи в ОП
-- address_OP [7:0] - адрес, для обращения к ОП
-- wr_en_OP - сигнал записи в ОП, если этот сигнал не активен ОП выполняет чтение
-- Из ОП в ЦУУ поступает сигнал
-- data_out_OP [7:0] - данные, считанные из ОП
-- Для регистровой памяти РП ЦУУ формирует следующие сигналы
-- data_a_RP - данные для записи в РП, через порт a
-- address_a_RP [2:0] - адрес, для обращения к РП, через порт a
-- wren_a_RP - сигнал записи через порт a в РП, если этот сигнал не активен РП выполняет чтение
-- data_b_RP - данные для записи в РП, через порт b
-- address_b_RP [2:0] - адрес, для обращения к РП, через порт b
-- wren_b_RP - сигнал записи через порт b в РП, если этот сигнал не активен РП выполняет чтение
-- q_a - данные, считываемые из РП, через порт a
-- q_b - данные, считываемые из РП, через порт b



library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
--use IEEE.numeric_std.all; -- добавляем библиотеку
LIBRARY work;

--LIBRARY altera_mf;

--USE altera_mf.altera_mf_components.all;

entity CYY_AU_OP_RP_F is
generic (n:integer:=8);     -- n параметр, задает разрядность операндов
        port
        (
                clk              : in   std_logic; -- тактовый сигнал
                set              : in   std_logic; --  сигнал начальной установки
                f_com     : buffer std_logic_vector(1 downto 0); -- пока задает формат команды: 2 - УП; 1 - РР;
0 - ПР
-- Для взаимодействия с АУ
                a                        : buffer  STD_LOGIC_VECTOR (n-1 downto 0);-- первый операнд для АУ

                b                        : buffer  STD_LOGIC_VECTOR (n-1 downto 0);-- второй операнд для АУ


                cop              : buffer       std_logic; --  код операции 1-умножение,0 - сложение для АУ
                sno              : buffer       std_logic; -- сигнал начала операции для АУ

                rr               : buffer  STD_LOGIC_VECTOR (2*n-1 downto 0);-- результат из АУ
      priznak    : buffer  STD_LOGIC_VECTOR (1 downto 0); -- признак результата из АУ
                sko              : buffer       std_logic; -- сигнал конца операции из АУ
-- Для наблюдения внутренних сигналов во время отладки проекта
                signal RB : buffer STD_LOGIC_VECTOR (7 downto 0);-- регистр адреса, для адресации операнда в ОП
                signal CK : buffer STD_LOGIC_VECTOR (7 downto 0);-- счетчик команд, для адресации текущей
команды в ОП
                signal RK : buffer STD_LOGIC_VECTOR (7 downto 0);-- регистр команд, для хранения адреса
                signal RI : buffer STD_LOGIC_VECTOR (7 downto 0);-- регистр команд, для хранения операнда
                s_out            : out STD_LOGIC_VECTOR(3 downto 0);    -- отладочный выход для наблюдения
состояний БМК
-- Для взаимодействия с ОП
                data_in_OP : buffer STD_LOGIC_VECTOR (7 downto 0); -- данные для записи в ОП
                address_OP : buffer STD_LOGIC_VECTOR (7 downto 0); -- адрес, для обращения к ОП
                wr_en_OP   : buffer std_logic; -- сигнал записи в ОП, если этот сигнал не активен, ОП выполняет
чтение
                data_out_OP: buffer STD_LOGIC_VECTOR (7 downto 0); -- данные, считанные из ОП
--              address_b_OP : buffer STD_LOGIC_VECTOR (7 downto 0); -- адрес, для обращения к ОП
-- Для взаимодействия с РП
--              data_a_RP    : buffer STD_LOGIC_VECTOR (7 downto 0); -- данные для записи в РП, через порт a
                address_a_RP : buffer STD_LOGIC_VECTOR (2 downto 0); -- адрес, для обращения к РП, через порт a
                wr_en_a_RP       : buffer std_logic;                                                    -- сигнал
записи через порт a в РП, если этот сигнал не активен РП выполняет чтение
--              data_b_RP        : buffer STD_LOGIC_VECTOR (7 downto 0); -- данные для записи в РП, через порт b
                address_b_RP : buffer STD_LOGIC_VECTOR (2 downto 0); -- адрес, для обращения к РП, через порт b
                wr_en_b_RP       : buffer std_logic;                                                        --
сигнал записи через порт b в РП, если этот сигнал не активен РП выполняет чтение
                q_a                      : buffer STD_LOGIC_VECTOR (7 downto 0);-- данные из РП с порта a

                q_b                      : buffer STD_LOGIC_VECTOR (7 downto 0) -- данные из РП с порта b
        );
```

```
end entity CYY_AU_OP_RP_F;

architecture arch of CYY_AU_OP_RP_F is

-----------------------------Декларация компонента ОП на 256 байт -----------------------------------------
----------------------

component memory
        PORT
        (
                address : IN STD_LOGIC_VECTOR (7 DOWNTO 0); -- адресный вход
                clock           : IN STD_LOGIC ;                                                        -- тактовый
вход
                data            : IN STD_LOGIC_VECTOR (7 DOWNTO 0); -- вход данных
                wren            : IN STD_LOGIC ;                                                -- разрешение записи
                q               : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)        -- выход данных
        );
end component;
-------------------------------------------------------------------------------------------------------------------
-----------------------
-- Следующим компонентом является память регистровая RP
-- Декларация компонента регистровой памяти на 8 байт
-- Содержит два порта a и b
-- Создан в QII версии 13.1 Как его создать, есть в методичке
COMPONENT Ram_2port_11
        PORT
        (
                address_a               :       IN STD_LOGIC_VECTOR(2 DOWNTO 0); -- адресный вход порта a
                address_b               :       IN STD_LOGIC_VECTOR(2 DOWNTO 0); -- адресный вход порта b
                clock           :       IN STD_LOGIC;
                 -- тактовый сигнал
                data_a          :       IN STD_LOGIC_VECTOR(7 DOWNTO 0);        -- вход данных для записи
через порт a
                data_b          :       IN STD_LOGIC_VECTOR(7 DOWNTO 0);        -- вход данных для записи
через порт b
                wren_a          :       IN STD_LOGIC;
         -- разрешение записи через порт a
                wren_b          :       IN STD_LOGIC;
         -- разрешение записи через порт b
                q_a             :       OUT STD_LOGIC_VECTOR(7 DOWNTO 0);               -- выходная шина порта
a
                q_b             :       OUT STD_LOGIC_VECTOR(7 DOWNTO 0)                -- выходная шина порта
b
        );
END COMPONENT;

-------------------------------------------------------------------------------------------------------------------
-----------------------------------
---- Компонент Арифметическое устройство, спроектированное ранее
-- Взят из седьмого проекта

COMPONENT ctrl_un_BO
        GENERIC ( n : INTEGER );
        PORT
        (
                a               :       IN STD_LOGIC_VECTOR(n-1 DOWNTO 0); -- вход первого операнда
                b               :       IN STD_LOGIC_VECTOR(n-1 DOWNTO 0); -- вход второго операнда
                clk             :       IN STD_LOGIC; -- синхросигнал
                set             :       IN STD_LOGIC; -- сигнал начальной установки
                cop             :       IN STD_LOGIC; -- код операции
                sno             :       IN STD_LOGIC; -- сигнал начала операции
                rr                      :       OUT STD_LOGIC_VECTOR(2*n-1 DOWNTO 0); -- результат
                priznak :        OUT STD_LOGIC_VECTOR(1 DOWNTO 0); -- признак результата
                sko             :       OUT STD_LOGIC -- сигнал конца операции
        );
END COMPONENT;
-------------------------------------------------------------------------------------------------------------------
--------------------------------
-- Декларация сигналов, используемых в проекте
type state_type is (s0, s1, s2, s3, s4, s5, s6, s7, s8); -- определяем состояния БМК
        signal next_state, state : state_type; -- следующее состояние, текущее состояние

signal incr_CK  : STD_LOGIC:='0';-- разрешение инкремента CK
signal summ_CK  : STD_LOGIC:='0';-- вычисление адреса перехода
signal load_RK  : STD_LOGIC:='0';-- загрузка команды
signal load_RB  : STD_LOGIC:='0';-- загрузка адреса
signal load_RI  : STD_LOGIC:='0';-- загрузка адреса
signal IA               : STD_LOGIC_VECTOR (7 downto 0);-- исполнительный адрес операнда в ОП
signal incr_RA  : STD_LOGIC:='0';-- разрешение инкремента PA
signal incr_RI  : STD_LOGIC:='0';-- разрешение инкремента PA
-------------------------------------------------------------------------------
```

```
begin
-- устанавливаются экземпляры компонентов OP,RP,AU
Comp_OP: memory
port map ( address_OP, clk, data_in_OP, wr_en_OP, data_out_OP);

-------------------------------------------------------------------------------------------------------------------

Comp_RP: Ram_2port_11 PORT MAP (
                address_a        => address_a_RP, -- RK(5 downto 3), -- адрес R1
           address_b     => address_b_RP, -- RK(2 downto 0), -- адрес R2
                clock    => clk,
                data_a   => rr(7 downto 0),  -- младшую часть результата для записи суммы
                data_b   => rr(15 downto 8),   -- записывать через второй порт пока не надо
                wren_a   => wr_en_a_RP,         -- разрешение записи в РП
                wren_b   => wr_en_b_RP,                                  -- через порт b запись не выполняем
                q_a      => q_a,                                         -- для наблюдения на вд
                q_b      => q_b                                          -- для наблюдения на вд
        );

-------------------------------------------------------------------------------------------------------------------
-------------
Comp_AY: ctrl_un_BO
generic map
        (n => 8)
port map ( a,b,clk,set,cop,sno,rr,priznak,sko);

-----------------------------------------------------------------------------------------
pr_CK:   process (set, clk) -- этот процесс определяет поведение счетчика команд СК

        begin
                if (set='1') then CK<=(others=>'0'); --устанавливаем в начальное состояние
                elsif clk'event and clk='1' then
                  if (incr_CK='1') then CK<=CK+"00000001"; -- инкремент счетчика
                        elsif (summ_CK='1' and (priznak = "00" or priznak = "01")) then CK<=RI ; -- вычисление
адреса перехода
                   end if;
                 end if;
        end process pr_CK;
-----------------------------------------------------------------------------------------
pr_RK: process (clk) -- этот процесс определяет поведение регистра команд
        begin
                if clk'event and clk='1' then -- по положительному фронту clk
                        if load_RK='1' then -- если есть разрешение на прием команды
                        RK<=data_out_OP; -- выполняется прием команды с выхода ОП
                        end if;
                end if;
        end process pr_RK;
-----------------------------------------------------------------------------------------
pr_RB: process (clk)-- этот процесс описывает логику работы регистра адреса RA
        begin
                if clk'event and clk='1' then -- по положительному фронту
                 if load_RB='1' then  RB<= data_out_OP; -- если есть разрешение, то загружаем исполнительный
адрес первого операнда
                                --elsif incr_RA='1' then RA<=RA-1; --инкремент адреса"00000001"
                 end if;
                end if;
end process pr_RB;
-----------------------------------------------------------------------------------------
pr_RI: process (clk)-- запоминает значение операнда
        begin
                if clk'event and clk='1' then -- по положительному фронту
                 if load_RI='1' then  RI<=data_out_OP; -- если есть разрешение, то загружаем исполнительный
адрес первого операнда
                        elsif incr_RI='1' then RI<=RI+1;
                        end if;
                end if;
end process pr_RI;
-----------------------------------------------------------------------------------------
-- Ниже приводится описание устройства управления для ЦУУ.Реализованы три формата команд ПР,РР и УП
TS: process (clk,set) -- этот процесс определяет текущее состояние МУУ
         begin
                if set = '1' then
                        state <= s0;
                elsif (rising_edge(clk)) then -- по положительному фронту переключаются состояния
                        state <= next_state;
                end if;
         end process TS;

NS: process (state,set,f_com,sko,priznak) -- этот процесс определяет следующее состояние МУУ, управляющие
сигналы
```

```
        begin
--

                        case state is
                                when s0=> -- переходы из s0

                                        if (set = '0') then
                                                next_state <= s1;
                                        else
                                                next_state <= s0;
                                        end if;
                                when s1=>
                                if f_com = "01" then
                                                next_state <= s4;
                                        else
                                                next_state <= s2;
                                                end if;

                                when s2=>
                                if f_com = "11" then
                                                next_state <= s7;
                                        else
                                                next_state <= s3;
                                                end if;

                                when s3=>
                                                next_state <= s4;

                                when s4=>
                                                next_state <= s5;


                                when s5 =>
                                                if (sko='1') then
                                                        next_state <= s6;
                                                else
                                                        next_state <= s5;

                                        end if;

                                when s6 =>
                                                        next_state <= s7;
                                when s7 =>
                                                next_state <= s8;

                                when s8 =>
                                                next_state <= s0;

                        end case;
        end process NS;
-------------------------------------------------------------------------------------------------------------
-- ниже приводится описание управляющих сигналов для БМК

incr_CK<='1' when ( state=s0 or (state = s2 and (f_com = "00" or f_com = "11")) ) else
                        '0';
summ_CK<='1' when (state=s7 and ( f_com = "11") ) else
                        '0';
load_RK<='1' when (state=s0) else -- загрузка команды в RK всегда в s0 для любой операции
                        '0';
load_RB<='1' when (state=s3) else -- загрузка значения операнда B
                        '0';
incr_RI <='1' when (state = s6 and f_com = "00") else -- инкремент RA для записи старшей части результата в ОП,
только для умножения
                        '0';

load_RI<='1' when (state = s2) else -- загрузка ИА в RA в s2 для операции умножения
                        '0';

sno <='1' when (state=s4) else -- когда извлекли операнды на шину A и B
                        '0';
data_in_OP<= rr(2*n-1 downto n) when (state= s7 and f_com = "00") else
                                 rr(n-1 downto 0) when (state = s6 and f_com = "00");


-- адреса
address_OP <= IA when (state = s2 and f_com = "00") else
                                  RI when ((state = s3 or (state = s6 or state=s7)) and f_com = "00") else
                                  CK;

IA<= q_b ; -- исполнительный адрес для 2 операнда
```

```
wr_en_a_RP<='1' when (sko = '1' and f_com = "01") else '0'; -- запись в РП, если формат Р-Р, инвертирование
wr_en_OP<='1' when (cop = '0' and (state = s6 or state = s7)) and f_com = "00" else '0'; -- запись в рп при
умножение



a<= data_out_OP when (state = s4 and f_com = "00") else
         q_a when (state = s4 and f_com = "01") else
         (others=>'0') ;

b<= RB when (state = s4 and f_com = "00") else
         q_b when (state = s4 and f_com = "01") else
         (others=>'0');



f_com<="00" when RK(7 downto 6)="00" else -- если умн
                 "01" when RK(7 downto 6)="01" else -- если инвертирование, то РР
                 "11";           -- если переход

cop<=RK(6); -- этот разряд определяет операцию в арифметическом устройстве
-------------------------------------------------------------------------------------------------------

address_a_RP<= RK(5 downto 3)  ; -- поле R1  в команде
address_b_RP<= RK (2 downto 0); -- поле R2  в команде


-------------------------------------------------------------------------------------------------------

-------------------------------------------------------------------------------------------------------------
        --  отладочный выход для наблюдения текущего состояния
                s_out<="0000" when state=s0 else
                                "0001" when state=s1 else
                                "0010" when state=s2 else
                                "0011" when state=s3 else
                                "0100" when state=s4 else
                                "0101" when state=s5 else
                                "0110" when state=s6 else
                                "0111" when state=s7 else
                                "1000" when state=s8; --else
--                              "1001" when state=s9 else
--                              "1010";
end arch;
```

3.6. Верификация центрального устройства управления

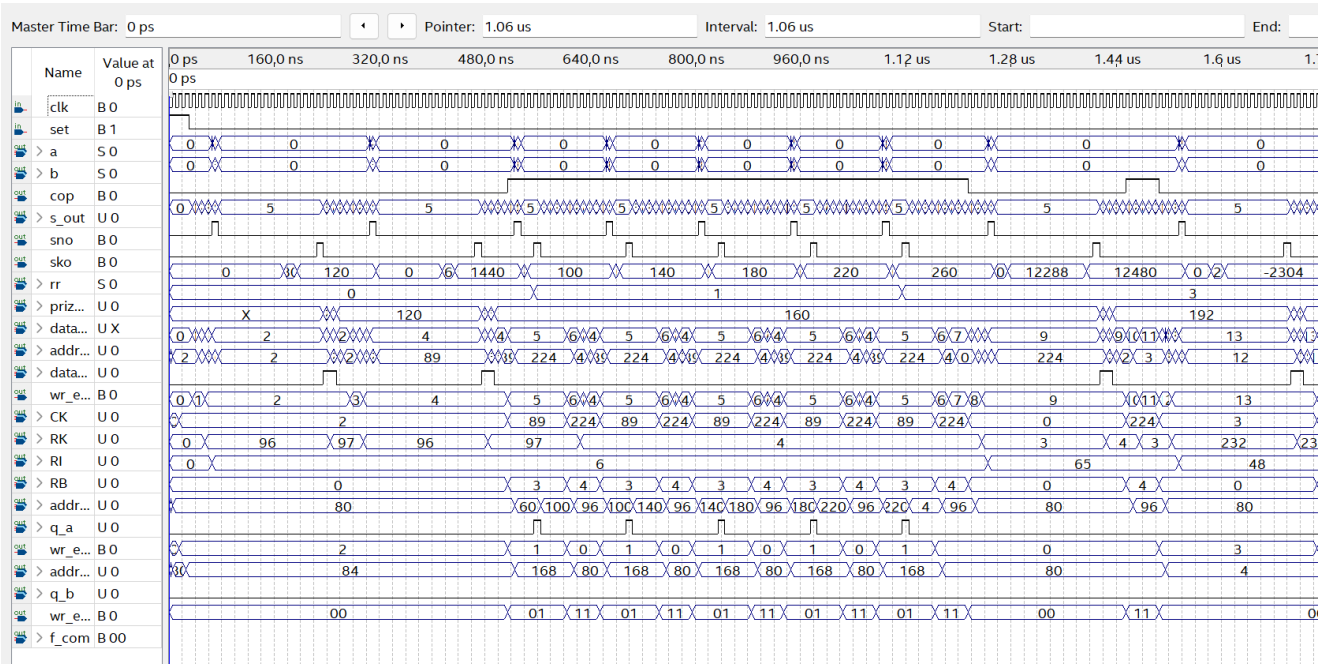


Рис. 16. Работа центрального устройства управления

3.7. Составление тестовой программы. Тестирование процессора

Кодируем команды и составляем тестовую программу. Желательно в программе использовать несколько команд умножения, выполняемых последовательно. Причем каждая следующая использует результат, сформированный предыдущей командой. Команду сложения рекомендуется выполнять в цикле.

Выполняем моделирование, убеждаемся в правильной работе процессора. В отчет помещаем свои комментарии, подтверждающие правильную работу процессора.

Начальное содержимое РП

R0	R1	R2	R3	R4	R5	R6	R7
80	168	84	60	96	4	3	1

Табл.2. Регистровая память

Адрес в ОП	Команда	bin			hex	dec	После выполнения
		cop	R1	R2			
0	mul ОП(RP(RK[2:0])), ОП(RA)	00	000	010	2	2	ОП[96,97]:=120,0
1		01	100	000	5	96	
2	mul ОП(RP(RK[2:0])), ОП(RA)	00	000	010	2	2	ОП[96,97]:= 160, 5
3		01	100	000	A	96	
4	Отрицательное приращение	00	011	001	19	25	РП[3]:=100
5	Условный переход	11	010	000	E0	224	ПР = 01 = КМ
6		00	000	100	4	4	СК = RA
4	Отрицательное приращение	00	011	001	19	25	РП[3]:=140
5	Условный переход	11	010	000	E0	224	ПР = 01 = КМ
6		00	000	100	4	4	СК = RA
4	Отрицательное приращение	00	011	001	19	25	РП[1]:=180
5	Условный переход	11	010	000	E0	224	ПР = 01 = КМ
6		00	000	100	4	4	СК = RA
4	Отрицательное приращение	00	011	001	19	25	РП[1]:=220
5	Условный переход	11	010	000	E0	224	ПР = 01 = КМ
6		00	000	100	4	4	СК = RA
4	Отрицательное приращение	00	011	001	19	25	РП[1]:= 260
5	Условный переход	11	010	000	E0	224	ПР = 11 != КМ
6		00	000	100	4	4	СК +1

Табл.3. тестовая программа

Заключение

В ходе выполнения курсовой работы была успешно достигнута цель проектирования блока операций и местного устройства управления, что включало их верификацию и сопряжение. Важным аспектом этого процесса стало тестирование модулей с использованием временных диаграмм, что обеспечило корректное отображение результатов работы и подтверждение функциональности системы.

Работа над проектом существенно обогатила навыки, связанные с составлением функционального алгоритма, его программной реализацией и отладкой. Эти навыки представляют собой необходимый опыт, который будет полезен в будущей профессиональной деятельности, особенно в области проектирования цифровых систем и микропроцессоров.

В процессе проектирования был подробно изучен процесс взаимодействия центрального устройства управления (ЦУУ) с компонентами памяти, такими как оперативная память (ОП) и регистровая память (РП), а также с арифметическим устройством. Эти взаимосвязи оказались критически важными для успешного выполнения заданий, определенных в техническом задании.

Ключевыми функциональными узлами ЦУУ были определены счетчик команд (СК), регистр команд (RK), регистр адреса (RO) и комбинаторные схемы. Взаимодействие между этими узлами, осуществляемое через управляющие сигналы, позволило создать эффективную структуру процесса обработки команд.

Применение САПР QUARTUS II и использования модуля синхронной памяти altsyncram значительно упростило весь процесс проектирования и отладки системы, что позволило сосредоточиться на развитии логики и архитектуры процессора.

Таким образом, реализация курсовой работы способствовала углубленному пониманию проектирования цифровых систем и дало ценную возможность применения теоретических знаний на практике.

Список литературы

1. Язык VHDL. Видеолекции

   https://mf.bmstu.ru/info/faculty/kf/caf/k3/subjects/automata_theory/course_project/video/vhdl/
2. МОСКОВСКИЙ ИНЖЕНЕРНО-ФИЗИЧЕСКИЙ ИНСТИТУТ (ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ). Кафедра «Компьютерные системы и технологии». Б.Н. Ковригин. АЛГОРИТМЫ УМНОЖЕНИЯ. Москва 2007.
3. Организация ЭВМ и систем. Кафедра К3. Прикладная математика, информатика и вычислительная техника. https://mf.bmstu.ru/info/faculty/kf/caf/k3/subjects/computer_organization/
4. Ефремов Н. В. Введение в систему автоматизированного проектирования QuartusII. https://mf.bmstu.ru/info/faculty/kf/caf/k3/discip/Automata_theory/main.shtml
5. Ефремов Н.В. Логические основы цифровых автоматов. Учебное пособие-3-е изд.-М.: МГУЛ, 2008 – 40стр.. https://mf.bmstu.ru/info/faculty/kf/caf/k3/discip/Automata_theory/main.shtml