Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: СиФО ЭВМ

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему:

РАЗРАБОТКА МИКРО-ЭВМ НА ПЛИС

БГУИР КП 1-40 02 01 1 ПЗ

Студент: гр. 550504 Нарейко В.И.

Руководитель: Третьяков А.Г.

Минск 2018

**СОДЕРЖАНИЕ**

ВВЕДЕНИЕ……………………………………………………………………..3

1РАЗРАБОТКА ОБЩЕЙ СТРУКТУРЫ МИКРО-ЭВМ……………………..4

2РАЗРАБОТКА ОСНОВНЫХ УСТРОЙСТВ МИКРО-ЭВМ……………….7

3ФУНКЦИОНАЛЬНОЕ МОДЕЛИРОВАНИЕ……………………………...16

ЗАКЛЮЧЕНИЕ……………………………………………………………….24

**ВВЕДЕНИЕ**

Целью данного курсового проекта является реализация микро\_ЭВМ на ПЛИС. В качестве инструмента разработки используется программа Quartus II 9.1 Web Edition. В этой программе находятся все необходимые элементы для разработки схем любой сложности. Программа позволяет тестировать разрабатываемую схему при помощи .vpf диаграмм на любом этапе разработки. Также программа позволяет объединять участки схем в блоки, которые в дальнейшем можно использовать в других схемах.

Архитектура разрабатываемой схема – Пристонская. Это значит, что команды и данные будут находится в одной памяти, в ROM. Также будет использоваться RAM для хранения промежуточных значений. В целом устройство будет выглядеть как RAM, ROM и устройство управления, которое будет включать в себе такие модули, как кэш, арифметически-логическое устройство, стек. Схема будет способна распознавать команды.

1. **РАЗРАБОТКА ОБЩЕЙ СТРУКТУРЫ МИКРО-ЭВМ**

В данном разделе будут расписаны задание, структурная схема микро-ЭВМ, а также разработка архитектуры команд.

* 1. **Задание**

Вариант: 1.

Тип архитектуры: Пристонская.

Разрядность шин: ША – 8 бит, ШД – 8 бит.

Тип памяти: ПЗУ – синхр., ОЗУ – асинхр.

Тип адресации: Непосредственная.

Команда условного перехода: JS.

Количество РОН: 4.

Команды АЛУ: INCS, NOR, NAND, SLA.

Стек: 11 элементов, рост вверх.

КЭШ: k = 2, алгоритм замещения строк – по наиболее давнему хранения, синхронизация с памятью – сквозная без отображения.

Арбитраж шин: децентрализованный параллельный.

Автомат предсказания: тип – А3, 3 бит, тип шаблона – PC + GHR.

* 1. **Структурная схема микро-ЭВМ**

Структурная схема будет состоять из устройства управления CU и памяти MEMORY. Устройство управления CU будет содержать в себе блок РОН, АЛУ и стек. А память будет оперативная и постоянная. Структурная схема представлена на рисунке 1.2.1.

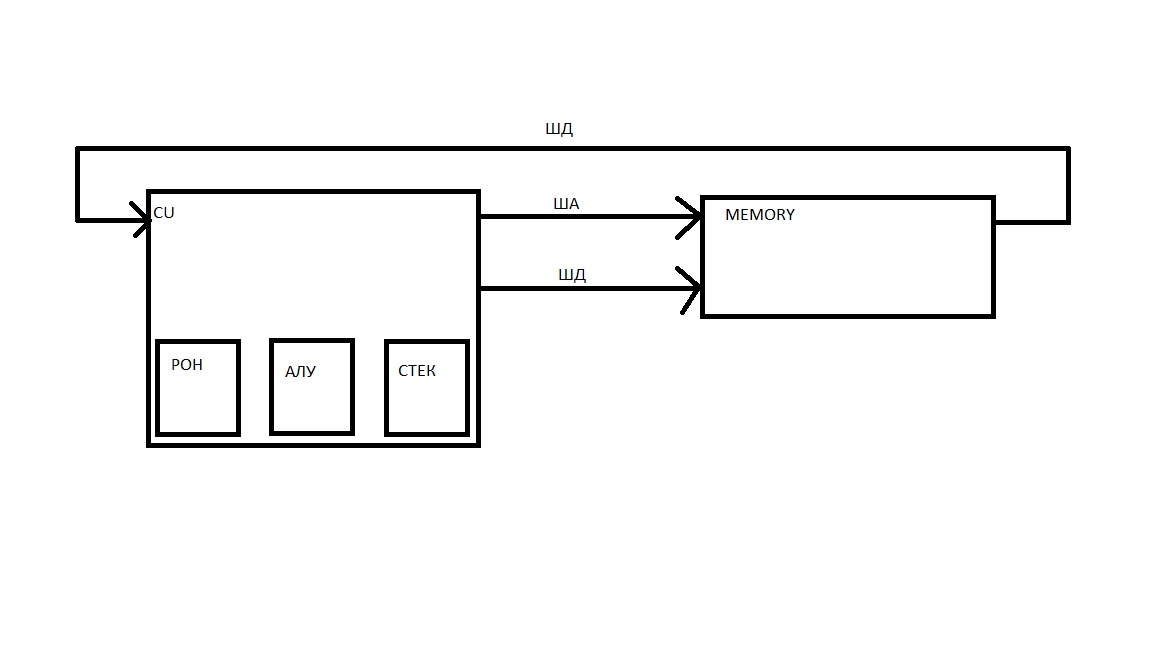


Рис. 1.2.1

Работа устройства будет начинаться при подаче на вход управляющего устройства CU тактового сигнала CLK, каждый такт устройство управления будет обрабатывать и управлять внутренними компонентами (АЛУ, стек) и работой с памятью. В управляющем устройстве существует цикл обработки команд. В его начале устройство управления обращается к ПЗУ для чтения команды и операндов, затем расшифровывая команду начинается её выполнение. При выполнении команды устройство управления будет обращаться к стеку, АЛУ, регистрам. Затем начинается выполнение новой команды. Сами команды хранятся в ПЗУ в .hex-файле ROM.hex.

* 1. **Разработка архитектуры команд**

Команды, которые требуется реализовать:

MOV reg, $mem

MOV $mem, reg

PUSH reg

POP reg

JMP $mem

JS $mem

HLT

INCS reg, reg

NOR reg, reg

NAND reg, reg

Исходя из вида команд, можно сделать вывод, что будут

использоваться три формата команд:

Таблица 1.3.1

|  |  |  |
| --- | --- | --- |
| КОП | Первый операнд | Второй операнд |

Таблица 1.3.2

|  |  |
| --- | --- |
| КОП | Операнд |

Таблица 1.3.3

|  |
| --- |
| КОП |

Исходя из исходных данных, а именно количества РОН, разрядности шин, а также количества требуемых команд, было решено отводить под определение типа команды 4 бита, 7 битов на адрес первого операнда, и 7 битов на адрес второго операнда. При первом типе команд(таблица 1.3.1) будет три обращения к памяти, из которых при первом обращении первые четыре бита будут опеределять тип команды, при втором обращении 7 бит определят адрес первого операнда, при третьем обращении адрес второго операнда. При втором типе команд(таблица 1.3.2) будет два обращения к памяти, из которых четыре первых бита первого обращения – тип команды, при втором обращении к памяти будет определен адрес операнда. При третьем типе команды(таблица 1.3.3) – такая команда только одна(HLT) – нам достаточно только одно обращение к памяти, в котором мы обнаружим тип команды и просто остановим работу микро-ЭВМ. Итого получается три типа систем команд.

Таблица 1.3.4

|  |  |  |
| --- | --- | --- |
| КОП | Первый операнд | Второй операнд |
| ХХХХ000 | ХХХХХХХ | ХХХХХХХ |

Таблица 1.3.5

|  |  |
| --- | --- |
| КОП | Операнд |
| ХХХХ000 | ХХХХХХХ |

Таблица 1.3.6

|  |
| --- |
| КОП |
| ХХХХ000 |

1. **РАЗРАБОТКА ОСНОВНЫХ УСТРОЙСТВ МИКРО-ЭВМ**

В данном разделе будут описаны основные устройства, которые

были разработаны для реализации проекта.

* 1. **Память**

Исходя из того, что была выбрана пристонская архитектура, где

память команд и данных объединена, было принято следующее решение: все так же использовать асинхронный rom для хранения команд, а синхронный ram для хранения данных, но объединить их в одно адресное пространство следующим образом: для адресации в rom использовались адреса с 0-63, а для адресации в ram – 63-127 (диапазоны были выбраны исходя из разрядности ША). Организация блока памяти представлена на рисунке 2.1.1.

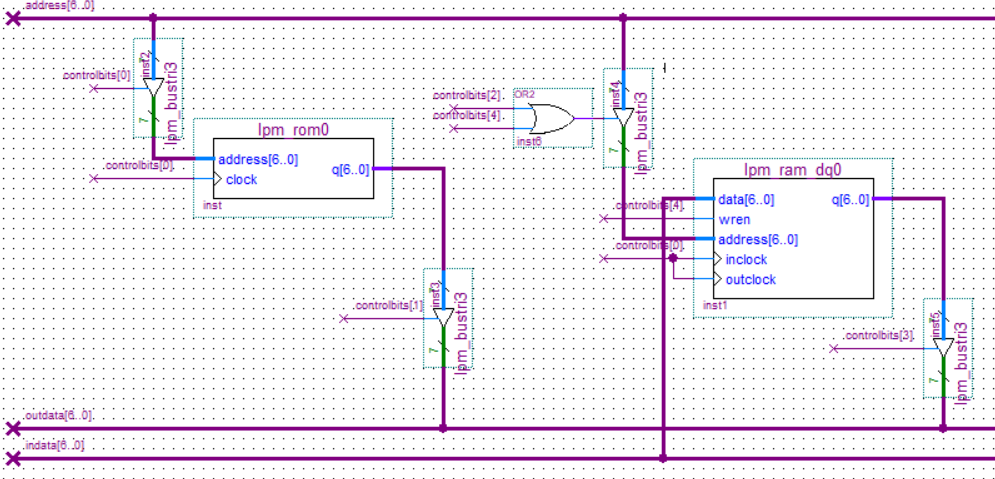


Рис. 2.1.1

Управляющие биты controlbits[0..4] будут сформированы и приходить из CU для корректной работы с памятью. Шина address[6..0] определяет адрес, откуда будем считывать или куда будем записывать данные. Шины indata[6..0] и outdata[6..0] – шины приходящих в ram данных и выходящих из rom или из ram данных соответсвенно.

* 1. **АЛУ**

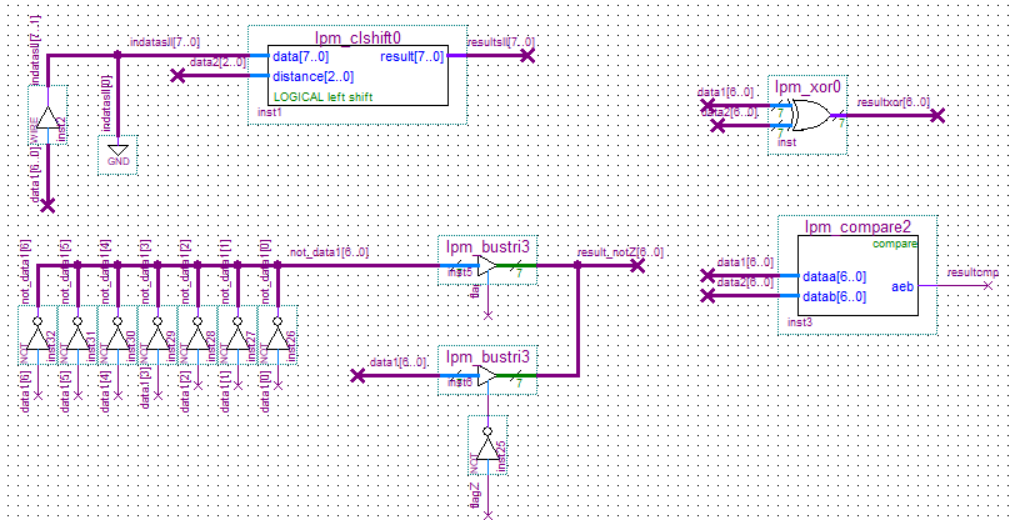


Рис. 2.2.1

АЛУ будет выполнять 4 различных команд: команду сдвига, команду инверсии по флагу Z, xor и команду сравнения двух операндов.

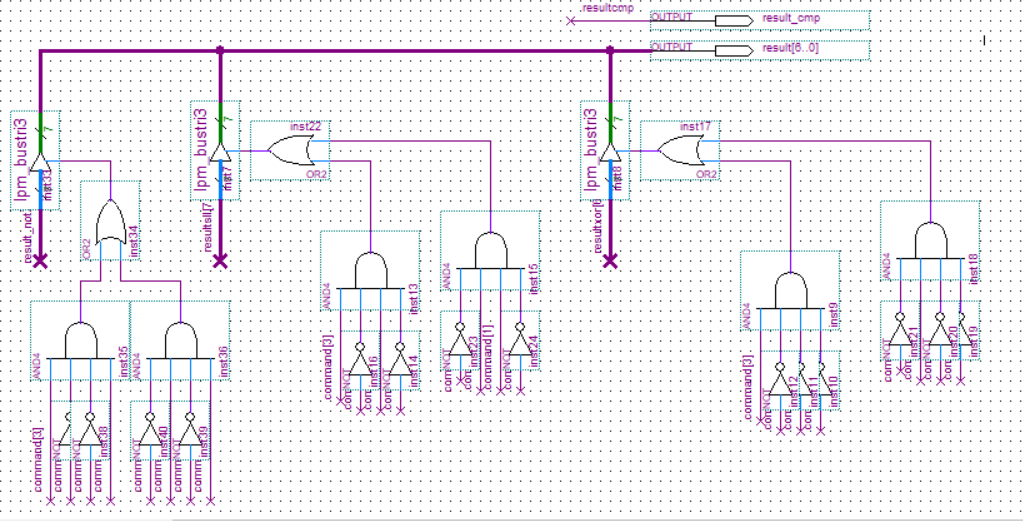


Рис. 2.2.2

В зависимости от типа адресации тот или иной результат будет подан на выход.

* 1. **Стек**

Сам стек состоит из 9 элементов lpm\_dff, на вход которого в

зависимости от входящих сигналов будут подаваться либо данные из следующего элемента стека (ничего для последнего), либо данные из предыдущего элемента стека(входные данные для первого).

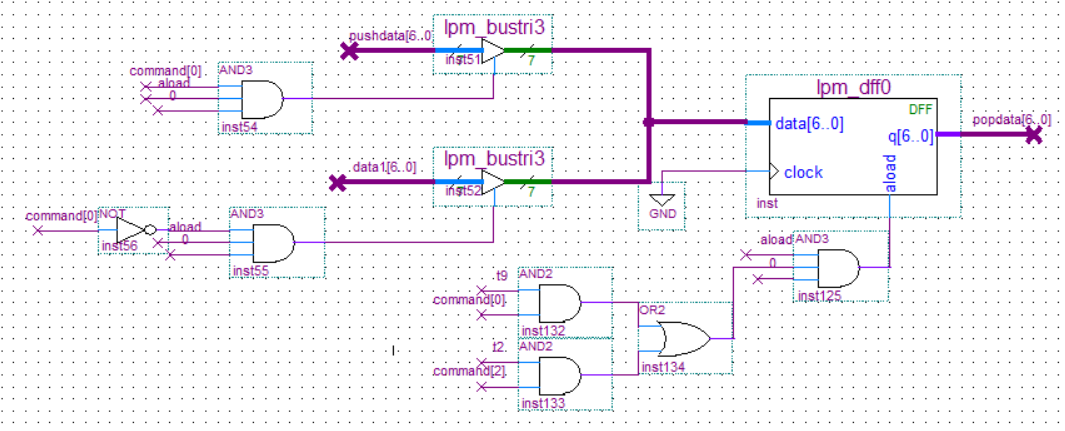


Рис. 2.3.1

Так как стек по варианту растет вверх, требуется реализовать

правильный инкремент SP. Если на вход блока стека пришла команда push, требуется проверить, а не заполнен ли стек полностью. Если заполнен, то инкрементацию выполнять не надо. То же самое требуется выполнить при подаче команды pop. При этом всем на вход блока приходят и управляющие сигналы controlbits, которые управляют, в какой момент требуется записать новое значение в регистр SP.

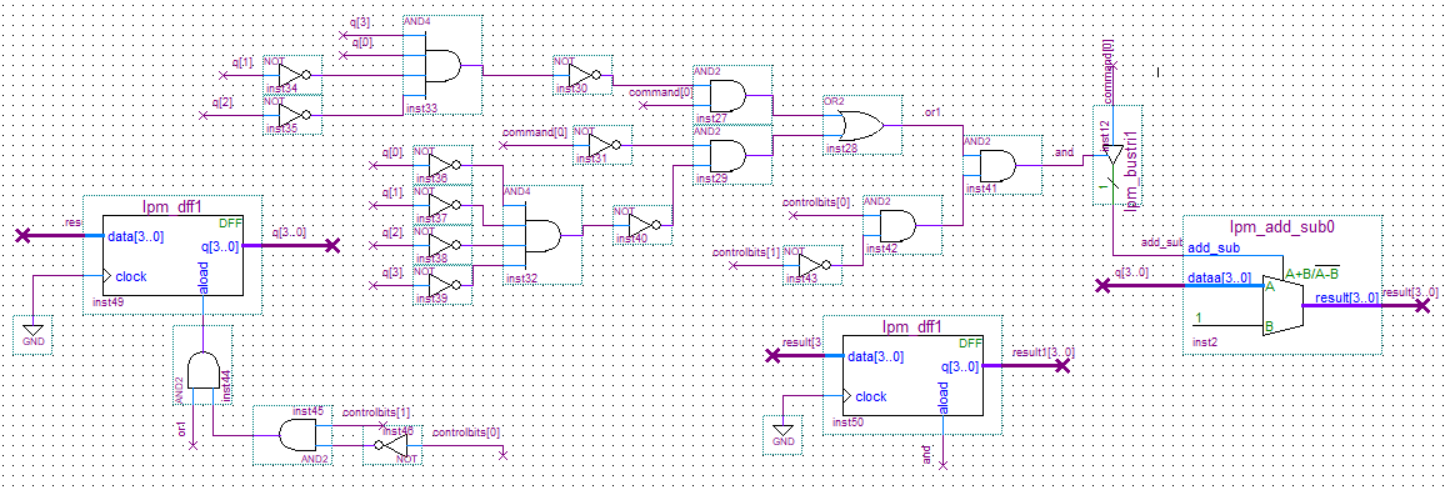


Рис. 2.3.2

Кроме этого для корректной работы стека требуется вырабатыват

дополнительные сигналы, которые будут управлять самими элементами стека, чтобы данные в определенный момент времени записывались или считывались в нужном регистре.

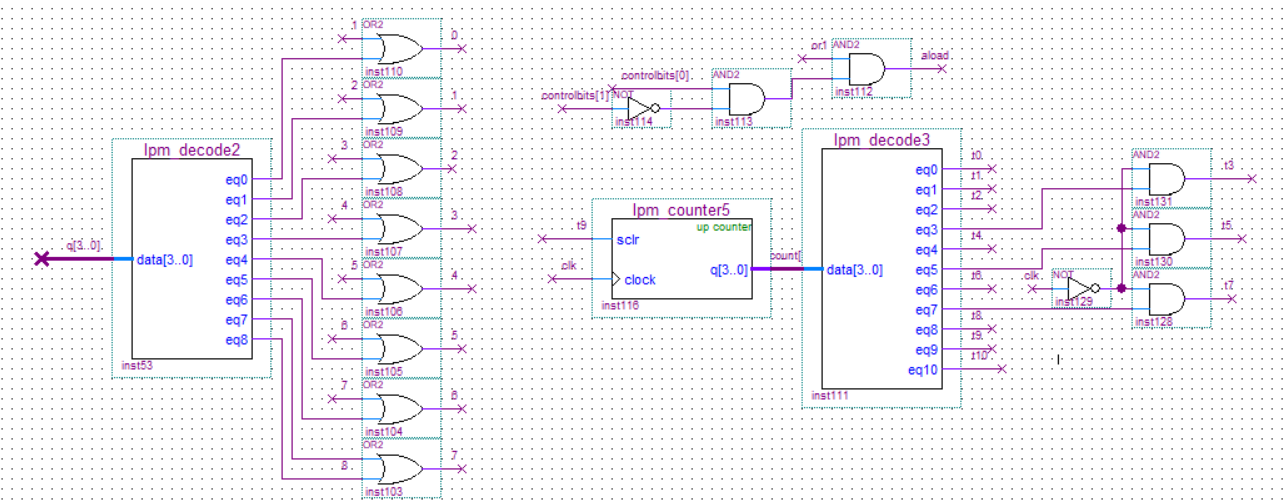


Рис. 2.3.3

* 1. **РОН**

РОН так же состоит из 16 элементов lpm\_dff и в зависимости от

приходящих на вход блока сигналов, в один из регистр будут либо записываться, либо считываться из него данные. Если пришла команда, которая требует записи в регистр, сравниваются адреса регистров и адреса, которые пришли на вход блока. При совпадении адреса регистра некоторым командам требуется еще сравнить такт, в который все совпало. Если совпал и такт, то на вход асинхронной загрузки в регистр будет подаваться единичный сигнал, что будет говорить о том, что в регистр требуется произвести запись (разработка сигналов изображены на рисунке 2.4.1 и 2.4.2). Если на выход блока приходит команда, требующая считывание информации из регистра, сверяется адрес на считывание и адрес регистра. Если совпадает, то на шину outdata[6..0] выдаются данные, которые идут сразу в CU(сигналы изображены на рисунке 2.4.1 и 2.4.3).

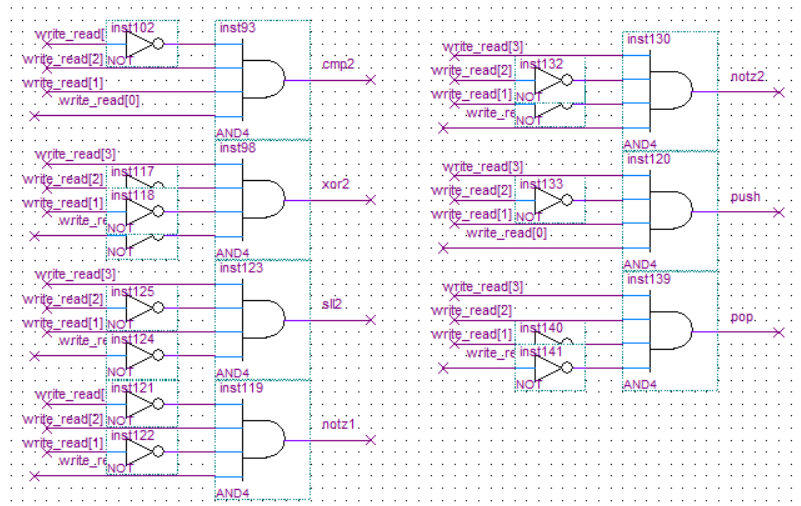


Рис. 2.4.1

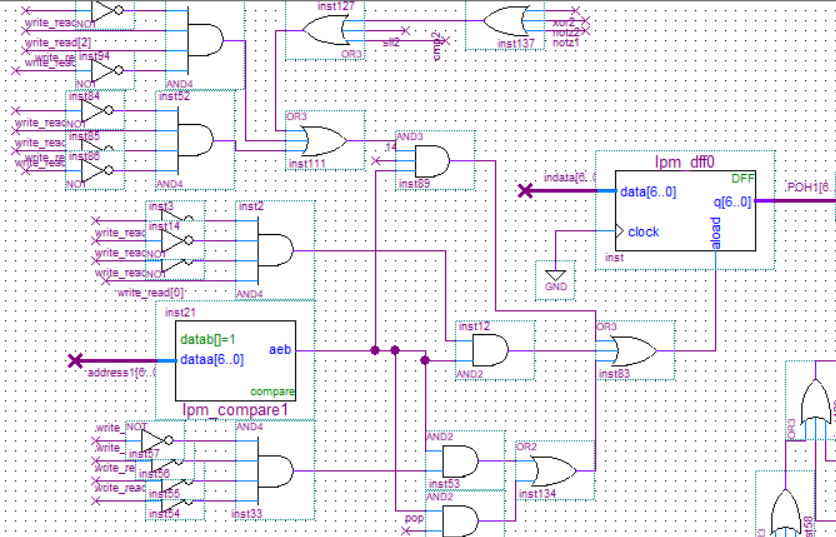


Рис. 2.4.2

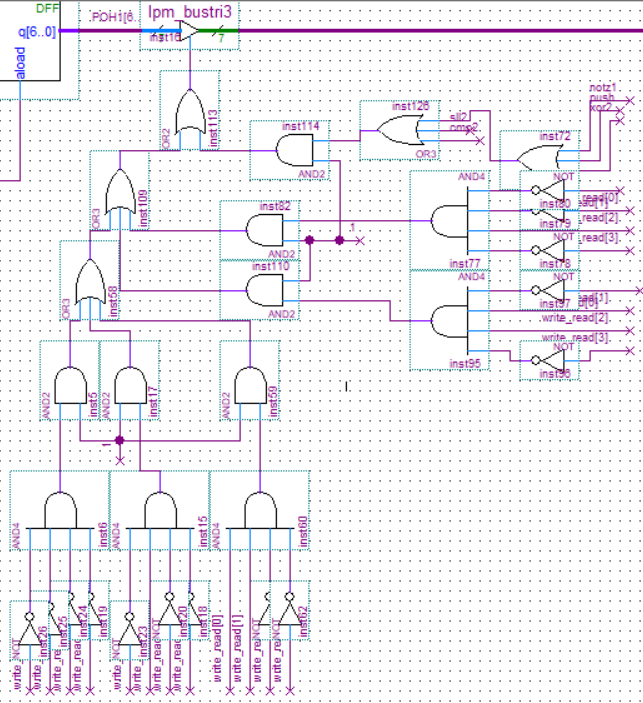


Рис. 2.4.3

* 1. **CU**

Сам CU состоит из регистров IP, IR, AR, DR, FR. IP является

счетчиком, который посылает адреса в rom для считывания команд и адресов операндов. IR хранит в себе код команды, из него происходит расшифровка команды и выбор дальнейших действий. AR хранит адреса операндов. DR сами операнды. FR хранит флаги, которые потребуются для реализации условных переходов. В зависимости от команд, в регистр IP подаются разные значения.

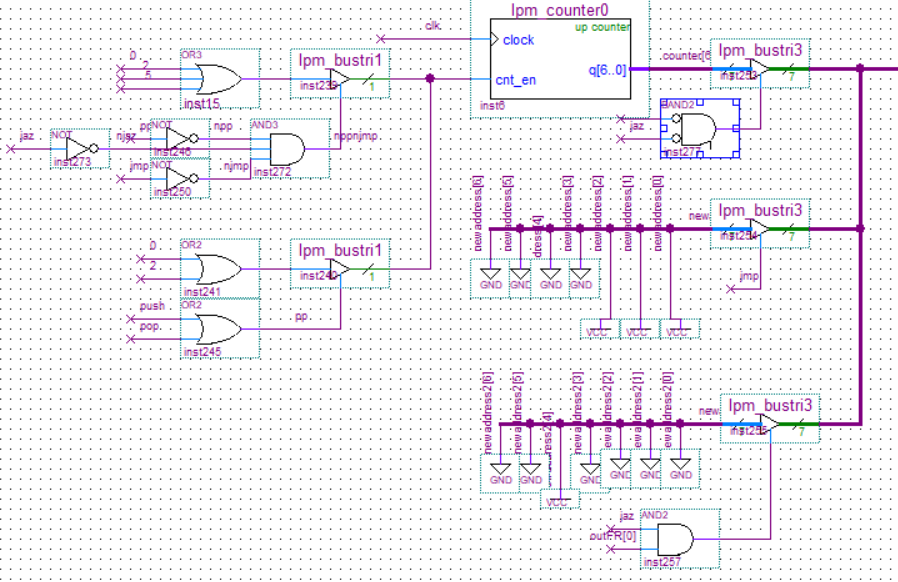


Рис. 2.5.1

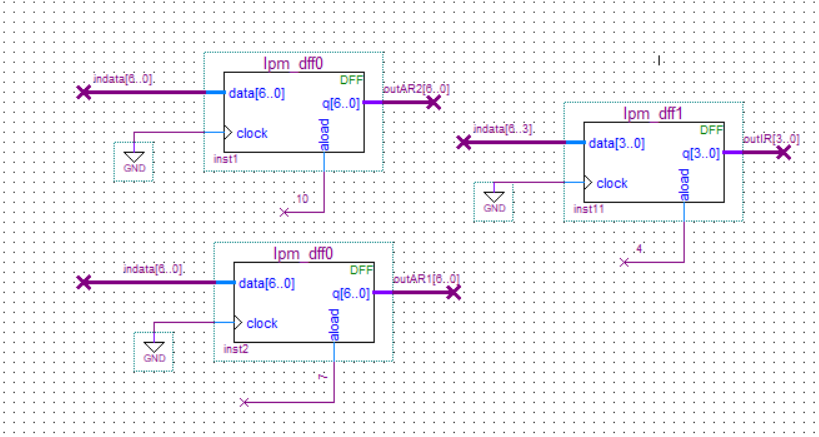


Рис. 2.5.2

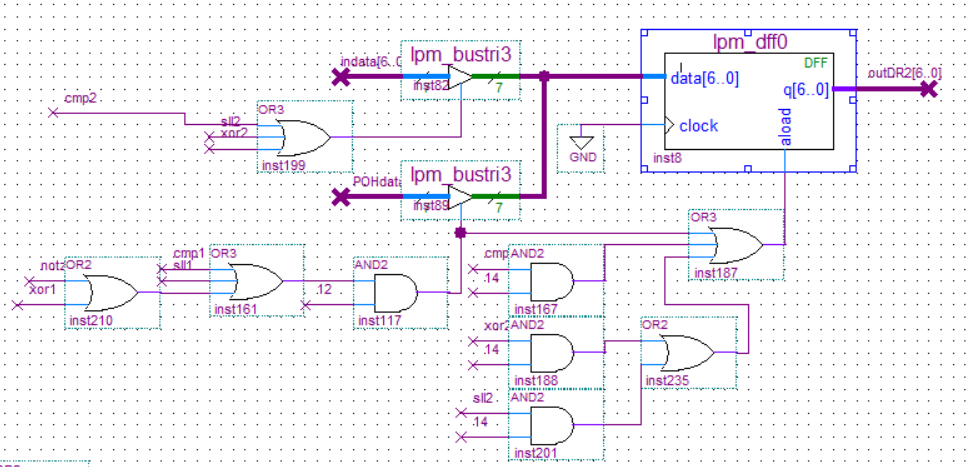


Рис. 2.5.3

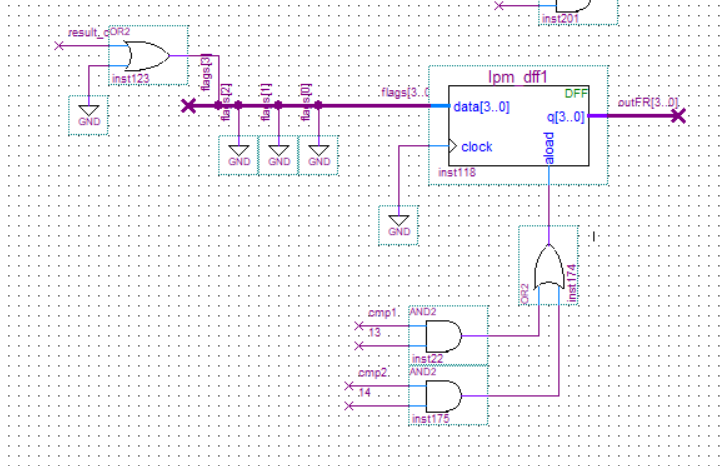


Рис. 2.5.4

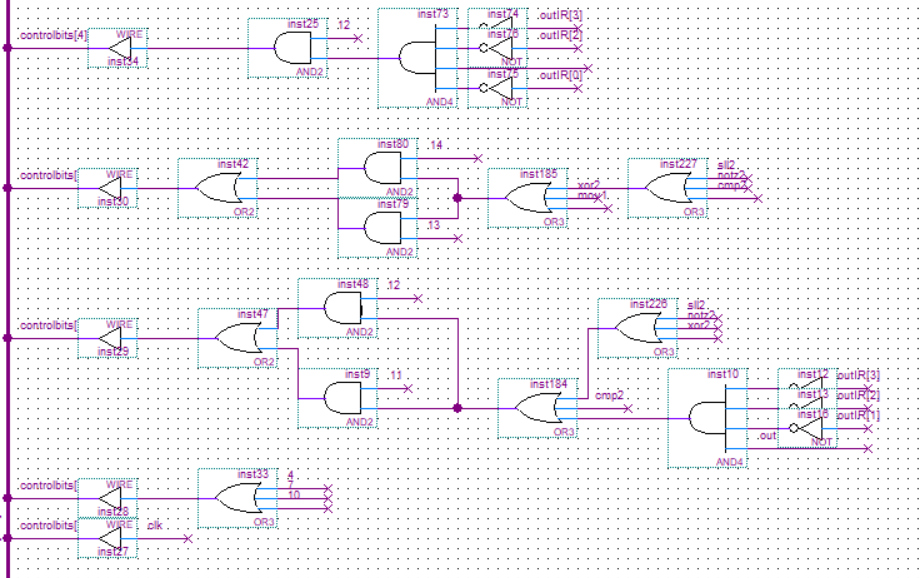


Рис. 2.5.5

1. **ФУНКЦИОНАЛЬНОЕ МОДЕЛИРОВАНИЕ**

Результат функционального моделирования блока АЛУ:

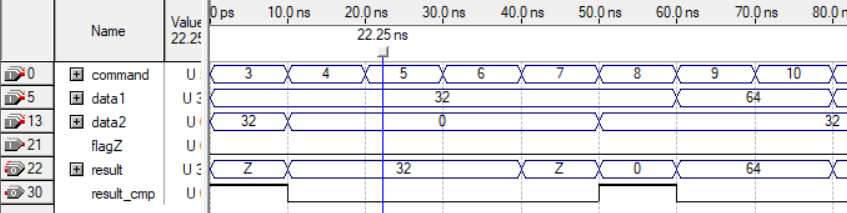


Рис. 3.1

Результат функционального моделирования блока стек:

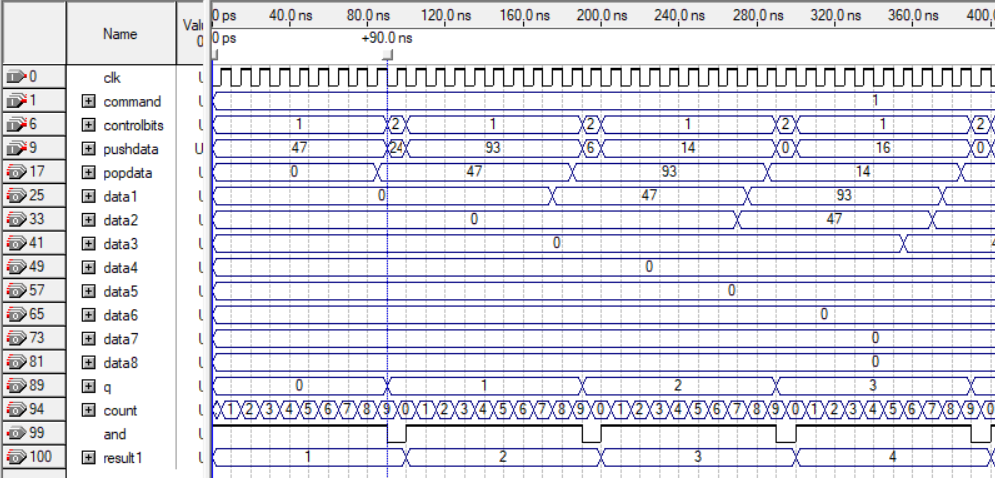


Рис. 3.2

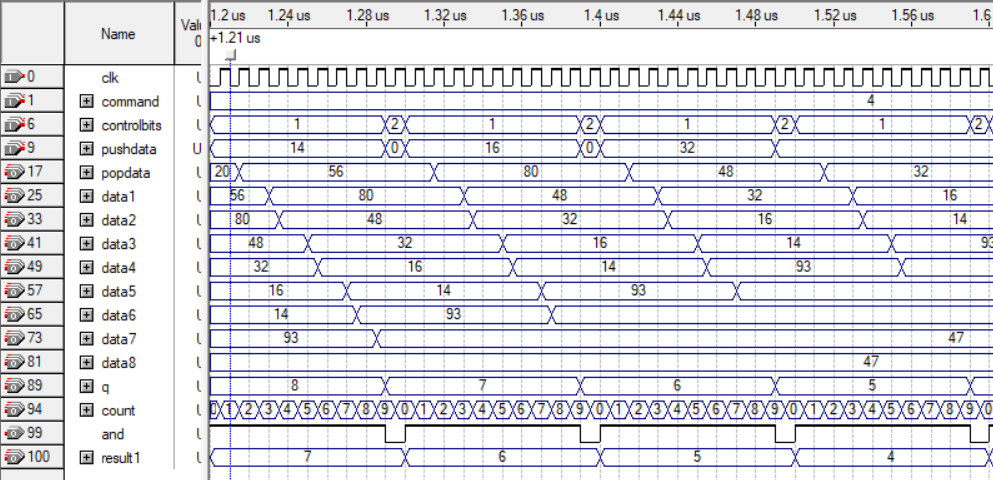


Рис. 3.3

Код реализуемой программы:

1. На языке асемблера:

mov AX, $70

mov BX, $71

cmp AX, BX

xor AX, BX

sll AX, BX

cmp AX, $69

xor AX, BX

sll AX, $69

notz AX, BX

notz AX, $66

push AX

pushBX

push AX

Push BX

Pop AX

pop BX

pop AX

jmp $7

pop BX

jaz $16

sll BX, AX

hlt

1. Двоичная система:

0001000 0000001 1000101

0001000 0000010 1000110

0011000 0000001 0000010

0100000 0000001 0000010

0110000 0000001 0000010

0111000 0000001 1000100

1000000 0000001 0000010

1010000 0000001 1000100

0101000 0000001 0000020

0101000 0000001 1000100

1011000 0000001

1011000 0000010

1011000 0000001

1011000 0000010

1100000 0000001

1100000 0000010

1100000 0000001

1101000

1100000 0000010

1110000

0110000 0000010 0000001

0000000

1. Шестнадцатеричная система:

А 1 46

А 2 47

18 1 2

20 1 2

30 1 2

38 1 45

40 1 2

50 1 45

28 1 2

28 1 45

58 1

58 2

58 1

58 2

60 1

60 2

60 1

68

60 2

70

30 2 1

0

Общее моделирование:

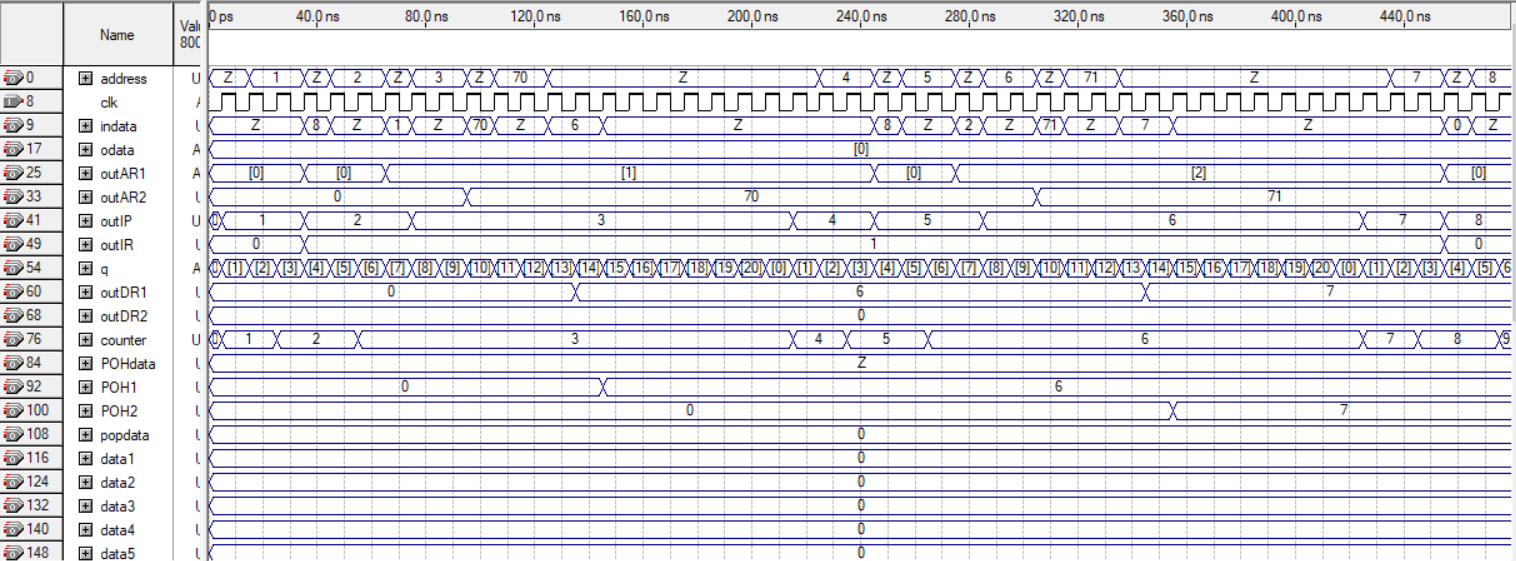


Рис. 3.4

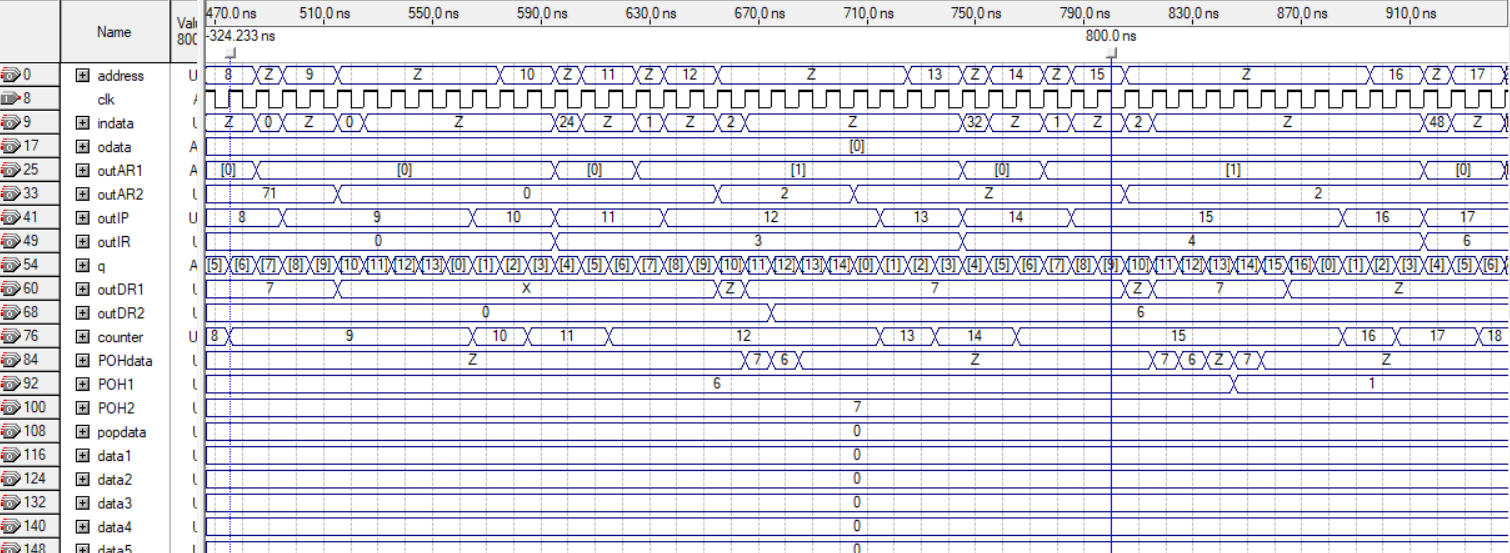


Рис. 3.5

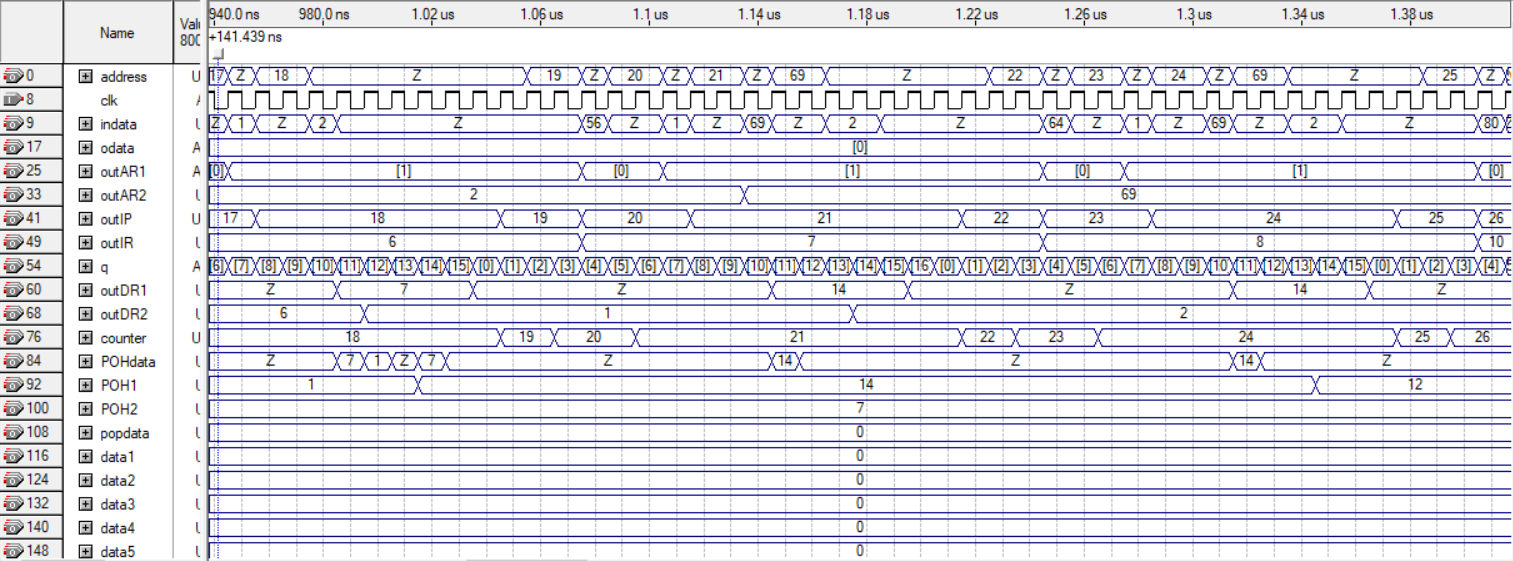


Рис. 3.6

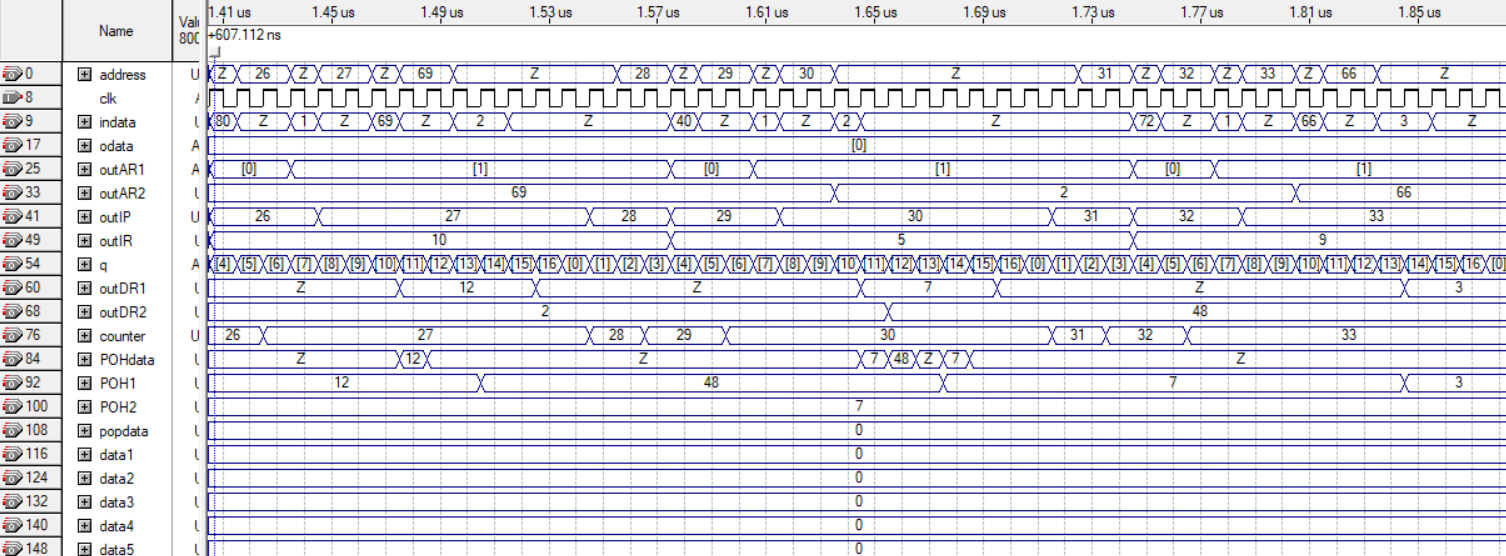


Рис. 3.7

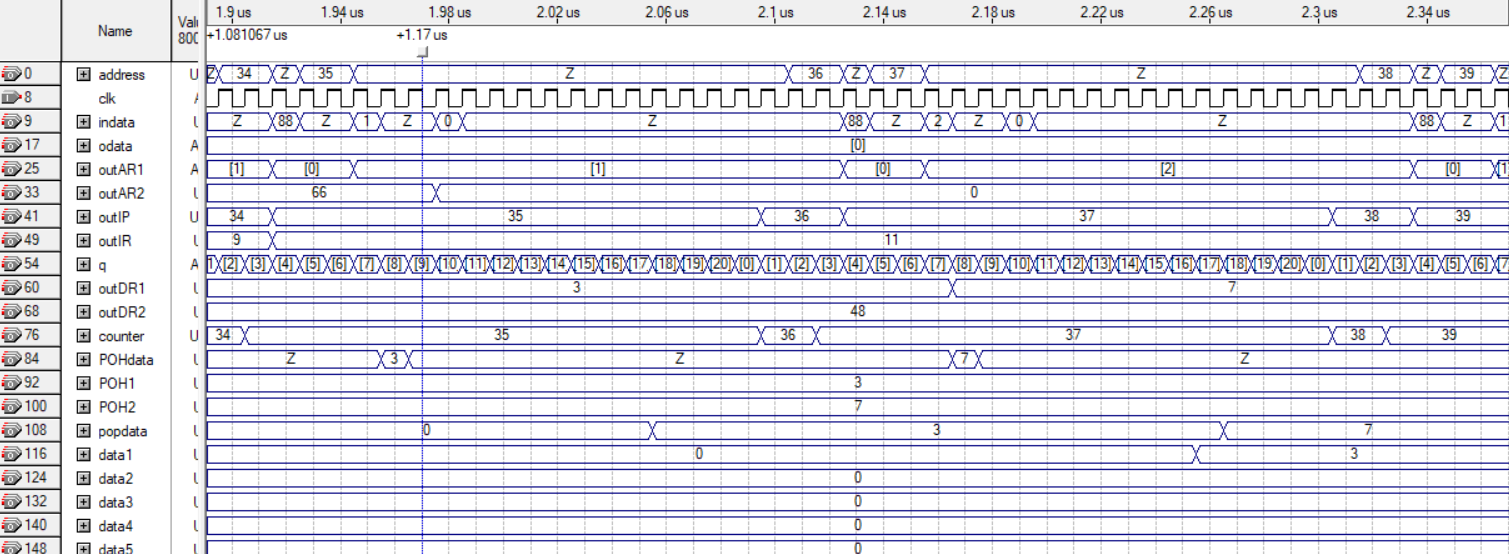


Рис. 3.8

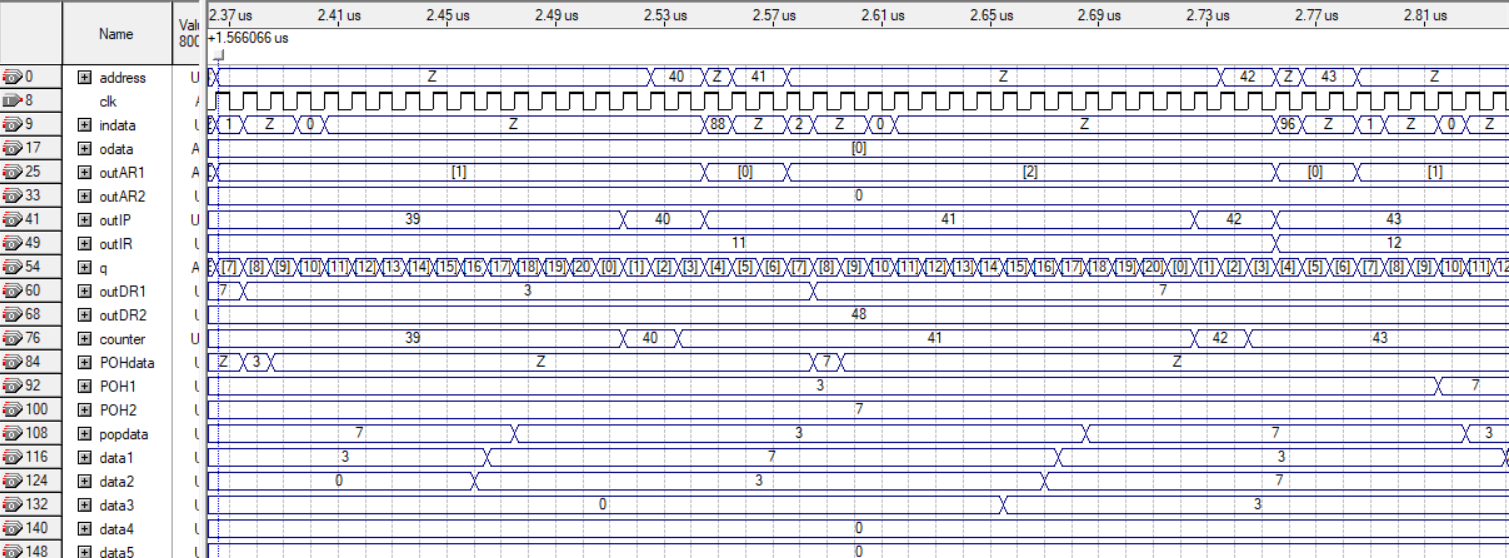


Рис. 3.9

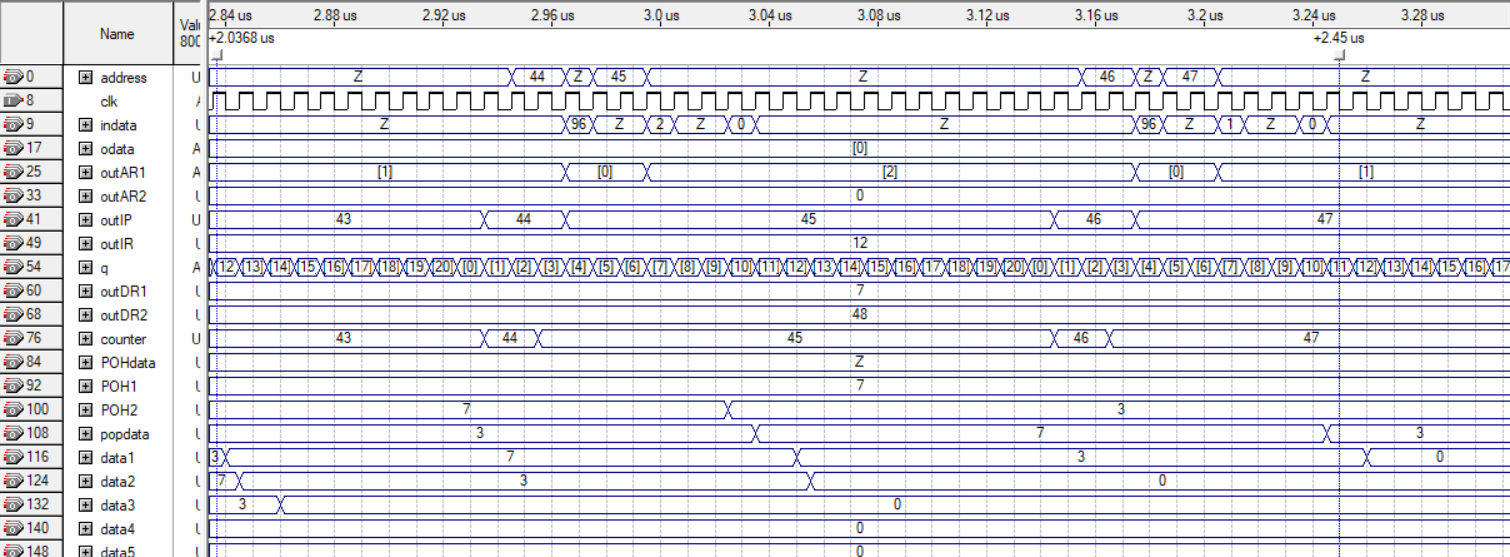


Рис. 3.10

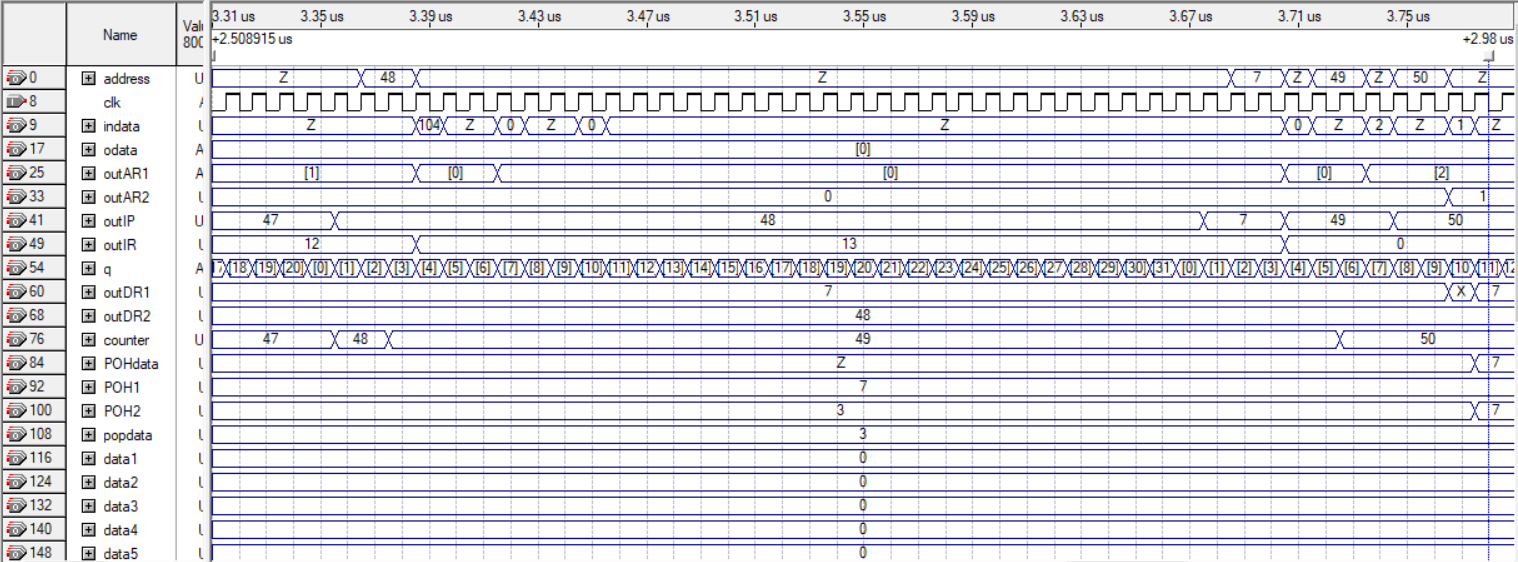


Рис. 3.11

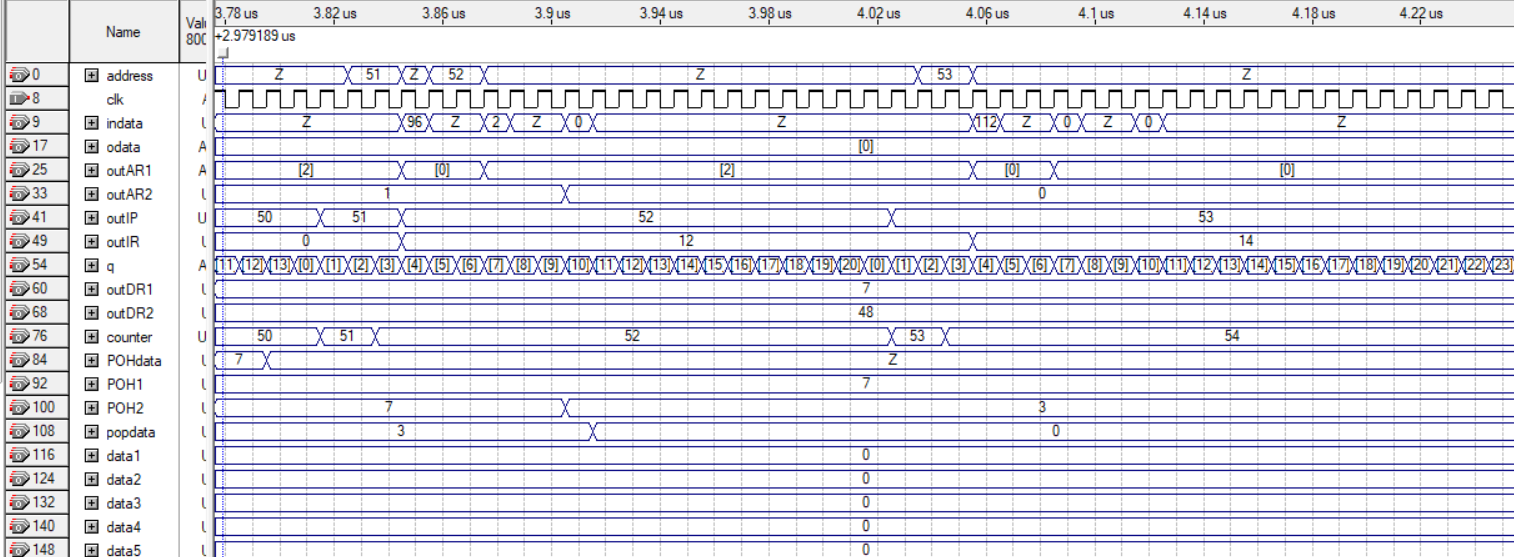


Рис. 3.12

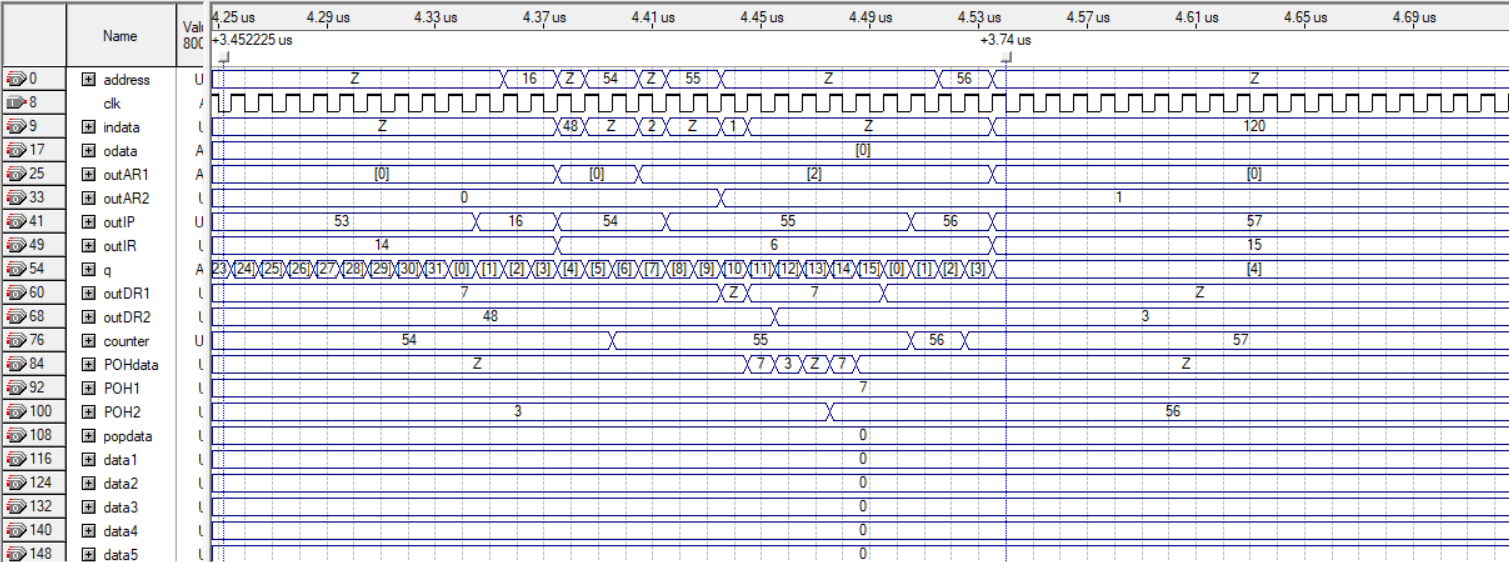


Рис. 3.13

Дамп памяти rom:

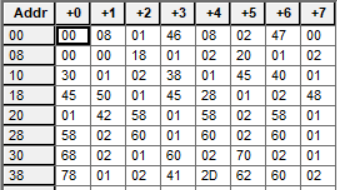


Рис 3.14

Дамп памяти ram до моделирования:

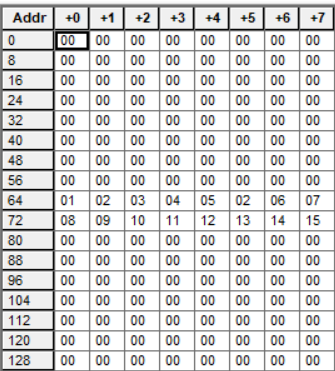


Рис. 3.15

Дамп памяти ram после моделирования:

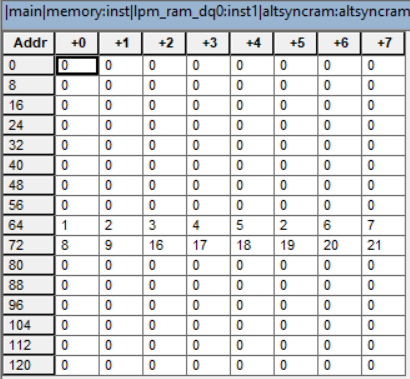


Рис. 3.16

**ЗАКЛЮЧЕНИЕ**

В ходе выполнения задания была спроектирована микро-ЭВМ на

ПЛИС, которая выполняет различные команды. Микро-ЭВМ легко управляется путем добавления кода описанных команд в память rom. Кроме этого микро-ЭВМ легко модернизируется. В нее можно добавить много полезных функций: от добавления новых команд до конвейера, КПДП арбитража шин.