Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
Рязанский государственный радиотехнический университет
имени В.Ф. Уткина

Кафедра ЭВМ

К защите
Руководитель работы:
_____
дата, подпись

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К КУРСОВОЙ РАБОТЕ
по дисциплине
«Проектирование цифровых устройств»
Тема:
«Проектирование микропроцессорных систем на основе ПЛИС»

Выполнил студент группы 045
Вашкулатов Н.А
_____
дата сдачи на проверку, подпись

Руководитель работы
доцент кафедры ЭВМ
Устюков Д.И.
_____ _____
оценка дата защиты, подпись

Рязань 2024

Задание 1. Разработайте иерархический проект синтезированной микропроцессорной системы на основе ПЛИС по описанию, приведенному в теоретической части. В отчете опишите процесс и результаты разработки.

Технические требования

1. Требование по быстродействию. Любая команда должна выполняться за один период синхроимпульсов.

2. Разрядность команд - 16 бит, разрядность данных – 8 бит.

3. Методы адресации и состав системы команд.

3.1. Команды с непосредственной адресацией. Операнды - содержимое регистра и константа из команды. Состав команд: пересылка, суммирование, суммирование с учетом переноса, логические операции И, ИЛИ, сумма по модулю два.

3.2. Двухадресные команды с регистровой адресацией. Операнды - содержимое двух регистров. Состав команд подобен п.3.1.

3.3. Одноадресные команды циклических сдвигов.

3.4. Команды обращения к памяти с косвенной адресацией.

3.5. Команды безусловных и условных переходов по признакам нуля - zf и переноса - cf с прямой адресацией.

4. Тип ПЛИС - семейство FLEX 10K.

5. Простая система прерываний без приоритетов или переопределения. Один такт на подготовку к прерыванию и один такт на выход из прерывания.

Задание 2. Разработайте программу для тестирования логических операций с непосредственной адресацией. Выполните моделирование. Определите временные задержки формирования адреса команд, чтения кода команды, а также формирования результата операции на шине данных. Номер задачи – из таблицы вариантов.

Задание 3. Создайте в памяти, начиная с адреса 00, массив из 8 чисел W0 – W7, которые вычисляются в соответствии с заданной формулой:

$W = 2 \times k + 3;$

Определите экспериментально максимальную частоту синхронизации.

Задание 4. Включите логический сдвиг вправо содержимого регистра lsr rx в систему команд, представьте в отчете результаты тестирования.

Задание 5. Протестируйте систему прерываний.

Содержание

Введение

Проектирование цифровых вычислительных устройств неразрывно связан с использованием программируемых логических интегральных схем (ПЛИС). Эти универсальные компоненты допускают создание синтезированных микропроцессоров, так называемых систем на кристалле (*System On Chip, SoC*), которые объединяют в себе процессорные ядра, память, модули обработки данных и интерфейсные средства.

Синтезированные процессоры, разработанные средствами САПР, могут быть значительно компактнее и эффективнее по сравнению с универсальными микропроцессорами, особенно когда речь идёт о выполнении конкретного набора задач. Их использование позволяет добиться оптимизации аппаратных затрат и улучшения быстродействия.

Синтез микропроцессора на ПЛИС представляет собой особо интересную задачу цифрового проектирования.

Целью данной курсовой работы является разработка микропроцессорной системы на базе ПЛИС, а также проверка возможностей и работоспособности.

1 Выбор архитектуры микропроцессорной системы и конфигурации процессорного ядра

Архитектура микропроцессорной системы отражает логическое построение системы, устройства и связей между ними.

Существуют две архитектуры для разработки МП. Джона фон Неймана и Гарвардская. Их основное различие в том, что в первой связь между процессором, устройствами памяти, ввода и вывода выполняет общая системная шина, которая содержит группы проводников - шины адреса, данных и управления. А во второй для хранения команд и данных используются независимые адресные пространства, полученные в результате использования отдельных устройств памяти и отдельных системных шин. В гарвардской архитектуре для хранения команд и данных можно использовать блоки памяти различных типов, емкости и разрядности. В микроконтроллерах для хранения программы, которая в процессе работы изменяться не должна, используют ПЗУ, а для хранения данных – ОЗУ

Для микропроцессорной системы выбрана гарвардская архитектура (рисунок 1). С устройствами памяти команд и данных процессор связан посредством четырех отдельных шин. Это шина адреса команд **ak**, шина команды **k**, шина адреса данных **ad** и шина данных **d-bus**.

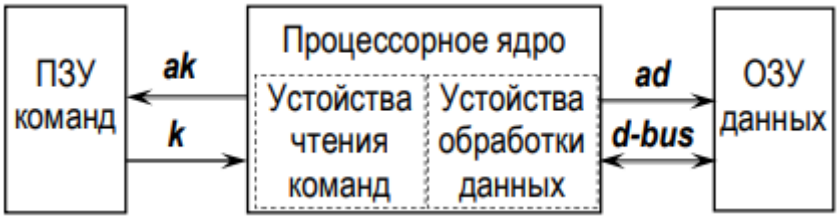

Рисунок 1 - Гарвардская архитектура микропроцессорной системы

Конфигурация процессорного ядра основана на совместном использовании арифметико-логическое устройства (АЛУ) и блока регистров общего назначения (РОН). Блок РОН выполнен как двухадресная (двухпортовая) память, которая позволяет одновременно выдать в АЛУ содержимое двух регистров - rx и ry, в которых хранятся первый и второй

операнды двухадресной команды. Регистр первого операнда rx впоследствии используется как получатель результата, исходное значение первого операнда теряется. Содержимое регистра второго операнда rx сохраняется. Код операции и адреса регистров ax и ax заданы в команде. Результат операции, полученный в АЛУ, через шину данных d_bus передается на блок РОН для записи в rx и на другие устройства процессора.

Пример использования архитектуры АЛУ и блока РОН представлен на рисунке 2.

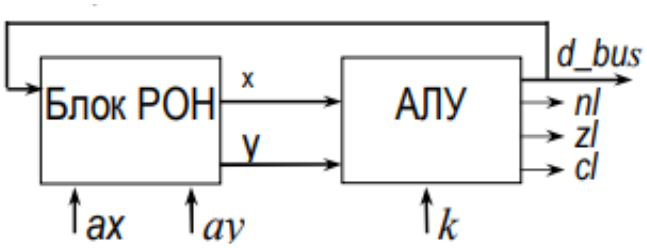

Рисунок 2 - Конфигурация процессора с блоком РОН

2 Разработка системы команд

Благодаря тому, что мы используем гарвардскую архитектуру у нас есть возможность без сложностей реализовать RISC систему команд. В RISC все команды имеют одинаковую разрядность и выполняются за один машинный цикл.

Для разрабатываемого процессора выберем два варианта деления команды на поля. Первый вариант – два 4-разрядных и одно 8-разрядное поле (например, команды с непосредственной адресацией). При этом код команды в 16-ричной системе счисления содержит 4 цифры, 4-разрядному полю соответствует одна цифра, а байт кодируют две цифры. Второй вариант – команда содержит 4 поля, по 4 разряда (пример – команды с регистровой адресацией).

2.1 Команды с непосредственной адресацией

Структура команд с непосредственной адресацией представлена на рисунке 3.

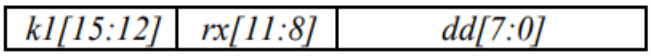

Рисунок 3 - структура команд с непосредственной адресацией

k1 - код операции;

rx - номер регистра первого операнда, в который после выполнения операции записывается результат;

d8 –байт данных - второй операнд, поступающий из команды.

Поле k1 должно иметь признак команд с непосредственной адресацией, например, старший бит, равный нулю. В этом случае значения кода k1 будут принадлежать диапазону от 0 до 7, а возможное количество команд с непосредственной адресацией составит 8.

Таблица 1. Команды с непосредственной адресацией.

k1	Команда	Операция	Пояснение
0	movi rx, #d8	rx = d8	Пересылка в регистр константы
1	addi rx, #d8	rx = rx + d8	Суммирование
2	subi rx, #d8	rx = rx + d8	Вычитание
3	andi rx, #d8	rx = rx *d8	Логическая операция И
4	ori rx, #d8	rx = rx ∧ d8	Логическая операция ИЛИ
5	xori rx, #d8	rx = rx ⊕ d8	Логическая операция Исключающее ИЛИ

2.2 Двухадресные команды с регистровой адресацией

Структура двухадресных команд с регистровой адресацией представлена на рисунке 4.

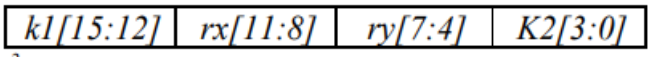

Рисунок 4 - структура двухадресных команд с регистровой адресацией

k1 - код операции равный 8;

rx - номер регистра первого операнда, в который после выполнения операции записывается результат;

ry – номер регистра второго операнда

k2 – код исполняемой операции

Таблица 2. Двухадресные команды с регистровой адресацией.

k1	Команда	Операция	Пояснение
0	mov rx, ry	rx = ry	Пересылка в регистр из регистра
1	add rx, ry	rx = rx + ry	Суммирование
2	sub rx, ry	rx = rx - ry	Вычитание
3	and rx, ry	rx = rx *ry	Логическая операция И
4	or rx, ry	rx = rx ∧ ry	Логическая операция ИЛИ

5	xor rx, ry	rx = rx ⊕ ry	Логическая операция Исключающее ИЛИ

2.3 Одноадресные команды с регистровой адресацией

Структура одноадресных команд с регистровой адресацией представлена на рисунке 5.

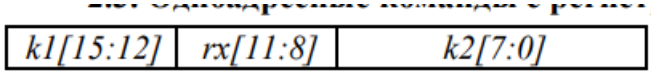

Рисунок 5 - одноадресных команд с регистровой адресацией

k1 - код операции равный 9;

rx - номер регистра первого операнда

k2 – код исполняемой операции

Таблица 3. Одноадресные команды с регистровой адресацией.

k1	Команда	Операция	Пояснение
0	inc rx	rx = rx + 1	Инкремент
1	dec rx	rx = rx - 1	Декремент
2	not rx	rx = !rx	Отрицание
3	asr rx	rx(n) = rx(n + 1) C = r(0) n = 0..6	Логическая сдвиг вправо
4	lsl rx	rx(n + 1) = rx(n), rd(0) = 0	Логический сдвиг влево

2.4 Команды обращения к памяти с косвенной адресацией

Структура команд обращения к памяти представлена на рисунке 6.

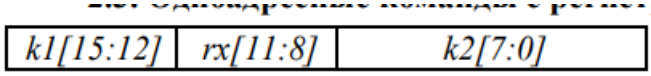

Рисунок 6 – структура команд обращения к памяти

k1 - код операции равный 0xA;

rx - номер регистра в котором находится операнд;

ra - номер регистра в котором находится адрес ячейки;

k2 – код исполняемой операции;

Таблица 4. Команды обращения к памяти

k1	Команда	Операция	Пояснение
0	ld rx, [ra]	rx = mem[ra]	Загрузка в rx из памяти
1	st rx, [ra]	mem[ra] = rx	Сохранение в памяти

2.5 Команды ветвления с прямой адресацией

Структура команд ветвления с прямой адресацией представлена на рисунке 7.

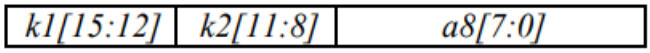

Рис. 8. Формат команд ветвления
с прямой адресацией

Рисунок 7 - Структура команд ветвления с прямой адресацией

k1 - код операции равный 0xF;

a8 – адрес перехода;

k2 – код исполняемой операции;

Таблица 5. Одноадресные команды с регистровой адресацией.

k1	Команда	Операция	Условие
0	b a8	PC = a8	Безусловный переход
1	beq a8	Z = 1, то PC = a8	Результат 0
2	bne a8	Z = 0, то PC = a8	Результат 1
3	bcs a8	C = 1, то PC = a8	Был перенос
4	bcc	C = 0, то PC = a8	Не было переноса
5	bpl a8	N = 0, то PC = a8	Результат > 0
6	bmi a8	N = 1, то PC = a8	Результат < 0

3 Разработка системы синхронизации

Система синхронизации обеспечивает устранение ошибок, обусловленных гонками, возникающими при выполнении операций. Работа процессора сводится к циклическому исполнению определенных операций, которым должны соответствовать определенные событиям и состояниям синхросигнала.

Период синхросигнала содержит два состояния (clk = 0 и clk = 1) и два изменения состояний – события - (спад и фронт импульса) (Рисунок 8). Примем за начало цикла команды (машинного цикла), который должен составлять один период синхросигнала, спад синхроимпульса. Выполнение команды во времени представим в виде последовательности следующих действий.

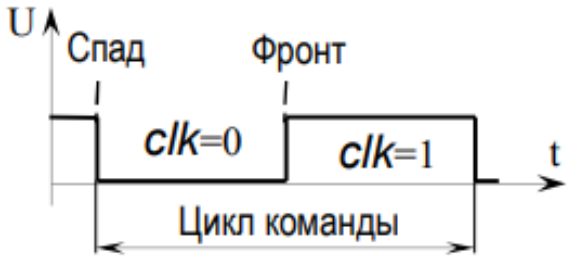

Рисунок 8 – Импульсы синхронизации

1. Спад синхросигнала clk - запись адреса следующей команды с выхода программного счетчика в регистр адреса ПЗУ. С этого момента начинается цикл выполнения команды, при этом команда k содержится на шине команд в течение всего периода синхросигнала и подается в блоки АЛУ, РОН, устройства управления и синхронизации.

2. Состояние clk = 0. Устройства и блоки обработки данных выполняют операцию, предусмотренную в команде под управлением кодов, содержащиеся в определенных полях команды, которые выбираются при

разработке форматов команд и учитываются в последствии при разработке принципиальной схемы.

На блок РОН поступают адреса регистров, используемых в данной команде, в соответствии с которой выдаются операнды, содержащиеся в rx и в ry.

На входы АЛУ будут поступать все данные, необходимые для выполнения операции – операнды из блока РОН и код операции из команды. На выходе АЛУ, и на шине «d_bus», будет сформирован результат выполнения команды.

Схема управления переходами формирует адреса новой команды для записи в программный счетчик.

Схема синхронизации записи сформирует сигналы разрешения записи результата операции в те устройства памяти и регистры, которые предусмотрены в данной команде.

Обработку данных выполняют устройства комбинационного типа (АЛУ, мультиплексоры, дешифраторы), которые не связаны с импульсами синхронизации, работают асинхронно. Формирование результата будет сопровождаться задержками сигналов в логических элементах.

3. Фронт синхросигнала clk - момент записи результата в регистр, или память, запись в программный счетчик адреса следующей команды, запись признаков результата в регистр состояния. Модуль разрешения записи формирует сигналы разрешения записи данных, полученных на шине «d_bus» при выполнении текущей команды, только в те устройства, для которых эта запись предусмотрена при проектировании модуля.

4. Интервал С=1. Запить адреса следующей команды, поступающего с выхода программного счетчика в регистр адреса ПЗУ.

Выполнение команды за один такт синхроимпульса становится возможным, если выполнять операции записи синхронно с фронтом или спадом импульсов, а все остальные операции выполнять посредством

комбинационных схем, которые не содержат элементов памяти и работают асинхронно.

4 Разработка системы прерываний

Прерывание – событие, возникающее в работе системы, требующие остановки всех действий, сохранение текущего прогресса и обработка этого события.

В системе будут присутствовать два вида прерывания: INT_1 и INT_2. За начало обработки прерывания отвечает отдельный модуль – контроллер прерываний, который формирует сигнал о необходимости сохранить PC, РОН и регистр флагов в любую память и заменяет PC на тот, который указан для конкретного прерывания. Для завершения обработки и перехода в нормальный режим обработчик завершается специальной командой 0xdddd при обработке которой, контроллер сформирует импульс, оповещающий о необходимости восстановить сохраненные ранее данные. Все действия контроллер прерываний выполняет на спаде синхроимпульса, а сигнал для сохранения / загрузки в устройствах обрабатывается на фронте, что позволяет перейти из нормального состояния в обработку или наоборот за один такт.

5 Разработка проекта микропроцессорной системы

Функциональная схема представлена на рисунке 9.

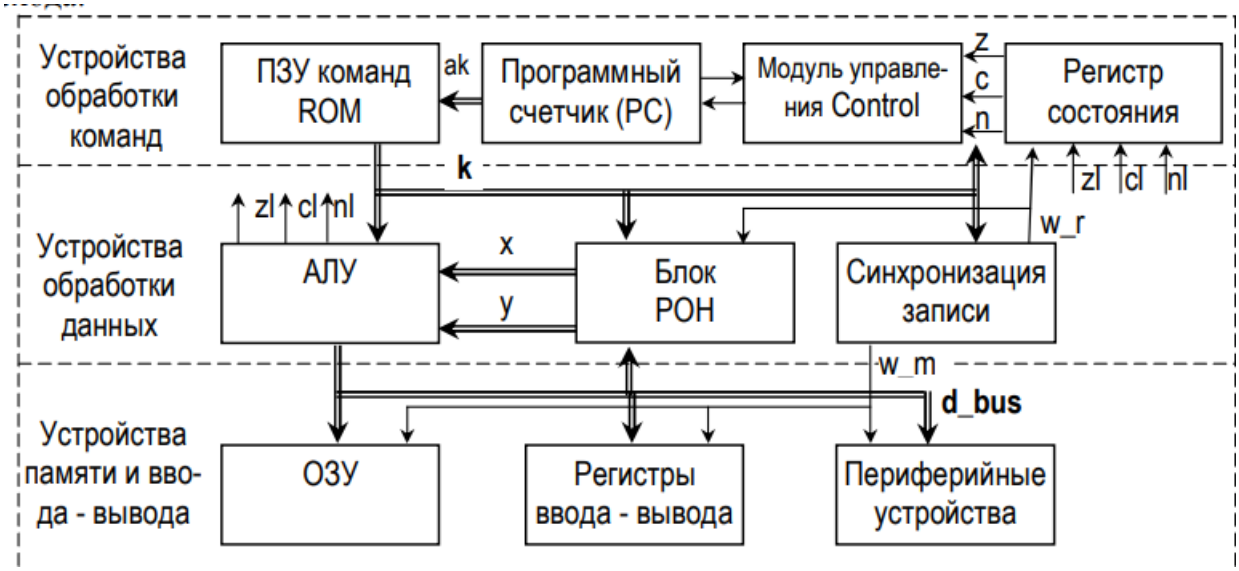

Рисунок 9 – Функциональная схема

5.1 Модуль Control

Модуль Control формирует адрес следующей команды, в зависимости от признаков и сигналов от модуля прерываний о сохранении и загрузке состояния:

```
module control(clk, k, c, z, n, branch, q_pc, int_save, int_load,
int_v);
input clk, c, z, n, int_save, int_load;
input [15:0] k;
input [7:0] int_v;
output [7:0] q_pc; reg [7:0] q_pc;
reg [7:0] prev_q_pc;

output branch;
assign branch =
      (k[15:12]==4'hf)&((k[11:8]==4'h0)|
      (k[11:8]==1)& z| (k[11:8]==2)& ~z|
      (k[11:8]==3)& c| (k[11:8]==4)&~c|
      (k[11:8]==5)& n|(k[11:8]==6)& ~n);
always @ (posedge clk)
begin
      if (branch) q_pc = k;
            else q_pc = q_pc+1;

      if(int_save)
      begin
            prev_q_pc = q_pc;
            q_pc = int_v;
      end
      else if (int_load)
      begin
            q_pc = prev_q_pc;
```

```
      end
end
```

5.2 Модуль РОН

В разрабатываемом микропроцессоре используется двухадресный (двухпортовый) блок регистров общего назначения (РОН), имеющий режим одновременного чтения содержимого двух регистров. Для записи используют адрес первого регистра. В этом модуле так же содержится сохранение и загрузка состояния:

```
module blok_ron (c,wreg,
d_bus, ax,ay,x,y, int_save, int_load);
input c,wreg, int_save, int_load ;
input [7:0] d_bus;
input [3:0] ax, ay;
output [7:0] x,y;
 reg [7:0] x,y;
reg [7:0] ron [3:0];
reg [7:0] prev_ron [3:0];
 always begin
 x=ron[ax]; y=ron[ay]; end
 always @(posedge c)
 begin
      if (wreg)
      begin
            ron[ax] = d_bus;
            if(int_save)
            begin
                  prev_ron[0] <= ron[0];
                  prev_ron[1] <= ron[1];
                  prev_ron[2] <= ron[2];
                  prev_ron[3] <= ron[3];


                  ron[0] <= 8'b0;
                  ron[1] <= 8'b0;
                  ron[2] <= 8'b0;
                  ron[3] <= 8'b0;
            end
            else if (int_load)
            begin
                  ron[0] <= prev_ron[0];
                  ron[1] <= prev_ron[1];
                  ron[2] <= prev_ron[2];
                  ron[3] <= prev_ron[3];
            end
      end
end
endmodule
```

5.3 Модуль АЛУ

Модуль АЛУ выполняет арифметические и логические команды обработки данных с непосредственной и регистровой адресацией. Кроме того, через данный модуль выполняются команды обращения к ОЗУ.

```
module alu (k,x,y,dm, d_bus, cl,zl,nl);
input [15:0] k;
input [7:0] x, y, dm;
output [7:0] d_bus; reg [7:0] d_bus;
output cl; reg cl; output zl, nl;
always if (k [15]==0) case(k [14:12])
      0: {cl,d_bus} = {1'b0,k [7:0]};
      1: {cl,d_bus} = x + k [7:0];
      2: {cl,d_bus} = x - k [7:0] ;
      3: {cl,d_bus} = {1'b0,( x & k [7:0])};
      4: {cl,d_bus} = {1'b0,( x | k [7:0])};
      5: {cl,d_bus} = {1'b0,( x ^ k [7:0])};
endcase else
if (k[15:12] == 8) case (k [2:0])
      0: d_bus=y;
      1: {cl,d_bus} = x + y;
      2: {cl,d_bus} = x - y;
      3: {cl,d_bus} = {1'b0,( x & y)};
      4: {cl,d_bus} = {1'b0,( x | y)};
      5: {cl,d_bus} = {1'b0,( x ^ y)};
endcase else
if (k[15:12] == 9) case (k [2:0])
      0: {cl,d_bus,}={1'b0,( x+1)};
      1: {cl,d_bus}= {1'b0,( x-1)};
      2: {cl,d_bus}= {1'b0,( ~x)};
      3: {d_bus,cl}= {x[7], x[7:0]};
      4: {cl,d_bus}= {x,1'b0};
      5: {d_bus,cl} = {x};
endcase else
if (k[15:12]==4'b1010) case (k [0])
      0: d_bus=dm;
      1: d_bus=x;
endcase
else d_bus=0;
assign zl=(d_bus==0);
assign nl = d_bus[7];
endmodule
```

5.3 Устройство синхронизации записи данных

Результат операции, выполненной, а АЛУ, выдается на шину выходных данных d_bus, к которой подключаются различные устройства памяти, принимающие данные, это РОН, ОЗУ, другие периферийные устройства. Данные подаются на все устройства. Однако, запись результата выполняется только в определенное устройство, в соответствии с кодом команды. Устройство синхронизации записи подобно дешифратору, на вход которого

поступают коды команд, а выходами являются сигналы разрешения записи в регистры (wreg), и в память (wmem).

```
module sync_wr (k, wreg, wmem);
input [15:0] k ;
output wreg, wmem ;
assign wreg = ~k[15] |( k[15] & ~k[14] & ~k[13] );
assign wmem = k[15] & ~k[14] & k[13] &~ k[12]& k[0];
```

5.4 Регистр состояния

Хранит признаки нуля (z), отрицательного результата (n) и переноса (z) а также по команде от контроллера прерываний сохраняет все признаки.

```
module status_reg(clk, wreg,zl,cl,nl,z,c,n, int_save, int_load);
input clk, wreg, nl, zl, cl, int_load, int_save;
output n; reg n;
output z; reg z;
output c; reg c;
reg prev_n;
reg prev_z;
reg prev_c;
always @ (posedge clk)
begin
      if (wreg==1) begin
             n=nl; z=zl; c=cl;
            if(int_save)
            begin
                  prev_n = n;
                  prev_c = c;
                  prev_z = z;
                  n=0; z=0; c=0;
            end
            else if (int_load)
            begin
                  n = prev_n;
                  c = prev_c;
                  z = prev_z;
            end

      end
end
endmodule
```

5.5 Контроллер прерываний

К контроллеру подключены два источника прерываний. По спаду проверяется сигнал на этих источниках. Если обнаружено прерывание, то формируется адрес перехода, флаг обработки прерывания и команда на сохранение состояния. Адрес перехода может указывать, например, на первые две ячейки памяти, в которых содержится безусловный переход на обработчики. В данном случае таблица векторов находится прямо в модуле.

Если прерывание было зарегистрировано, то контроллер переходит в состояние проверки на спец команду, при нахождении которой будет отправка сигнала на переход в нормальный режим, а также переход к ожиданию прерывания.

```
module int_control(
    input clk,
    input INT_1,
    input INT_2,
    input [15:0] k,
    output reg int_handle,
    output reg [7:0] int_v,
    output reg int_save,
    output reg int_load
);


parameter IDLE = 0,
          CHECK_K = 1;

reg [2:0] state, next_state;

always @(negedge clk) begin
        case (state)
            IDLE: begin
                if (INT_1) begin
                    int_v <= 8'h08;
                    int_save <= 1;
                    int_handle <= 1;
                              next_state <= CHECK_K;
                end else if (INT_2) begin
                    int_v <= 8'h02;
                    int_save <= 1;
                    int_handle <= 1;
                              next_state <= CHECK_K;
                end
            end
            CHECK_K: begin
                if (k[15:12]==4'hd) begin
                    next_state <= IDLE;
                    int_load <= 1;
                    int_handle <= 0;
                end
            end
            default: next_state <= IDLE;
        endcase

        if (next_state != state) begin
            int_save <= 0;
            int_load <= 0;
            state <= next_state;
        end
    end
endmodule
```

Принципиальная схема представлена на рисунке 10.

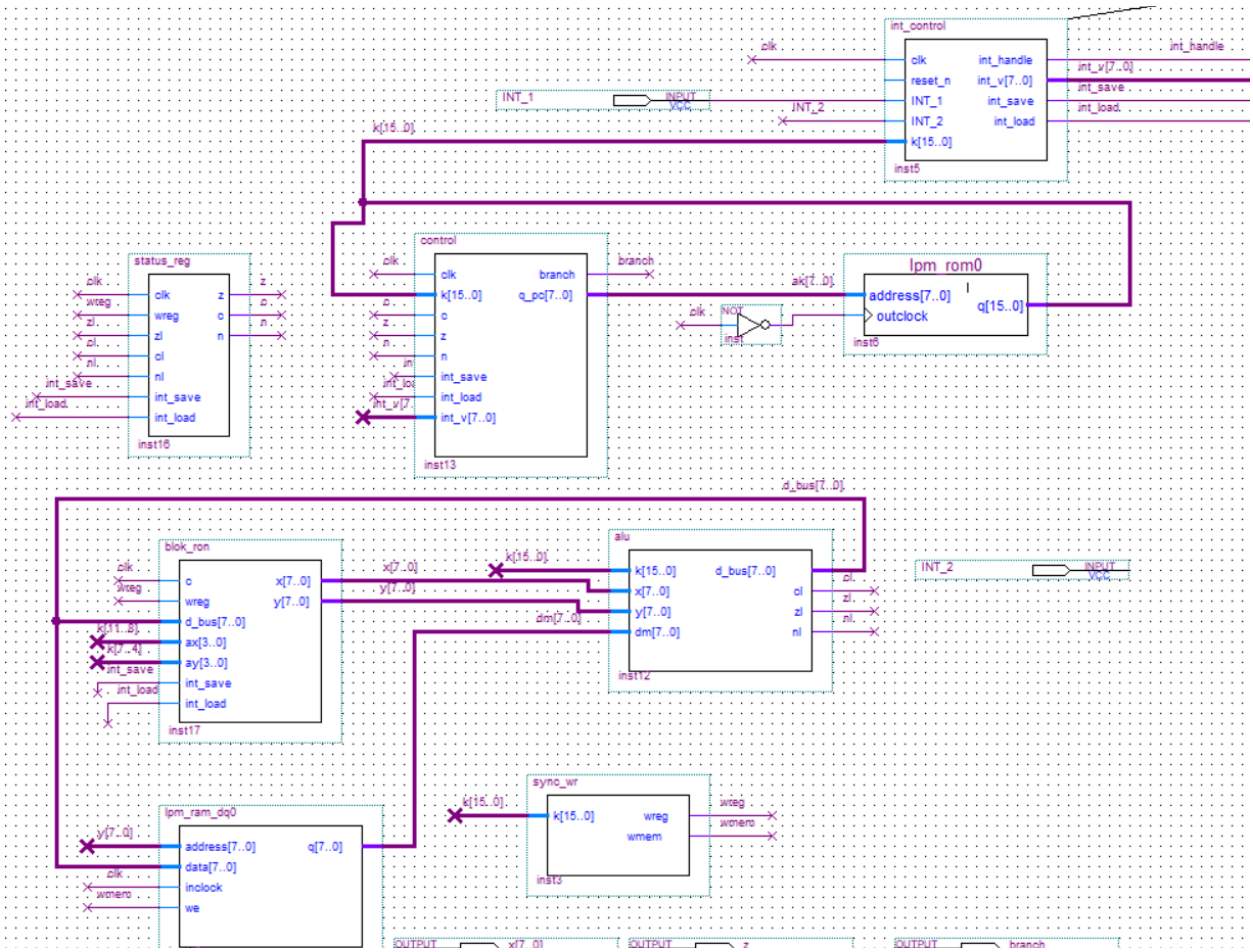

Рисунок 10 – Принципиальная схема

6. Экспериментальная часть

Задание 2. Разработайте программу для тестирования логических операций с регистровой адресацией. Выполните моделирование. Определите временные задержки формирования адреса команд, чтения кода команды, а также формирования результата операции на шине данных.

Таблица 6. Программа тестирования логических операций с регистровой адресацией.

Программа	ak	k	q_bus	nzc
movi r1, 0b10101010	00	01AA	AA	100
movi r2, 0b01010101	01	0255	55	000
and r1, r2	02	8123	00	010
movi r1, 0b10101010	03	01AA	AA	100
or r1, r2	04	8124	FF	100
movi r1, 0b10101010	05	01AA	AA	100
xor r1, r2	06	8125	FF	100
xor r1, r1	07	8115	00	010

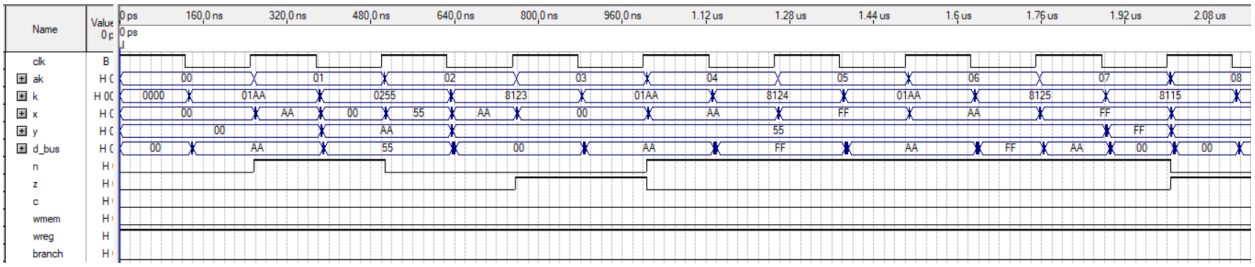

Рисунок 11 – Результат моделирования

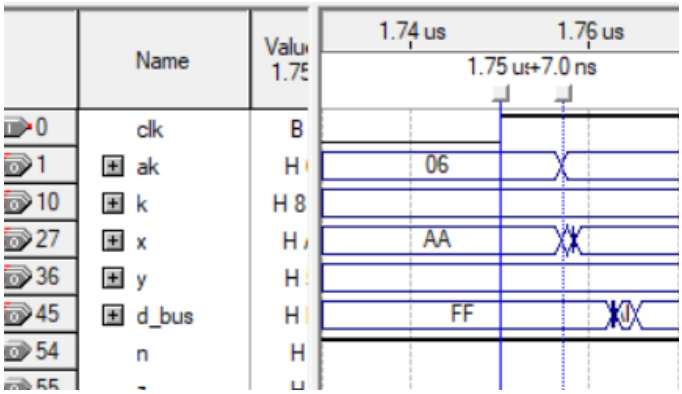

Рисунок 12 – Задержка формирования адреса команд

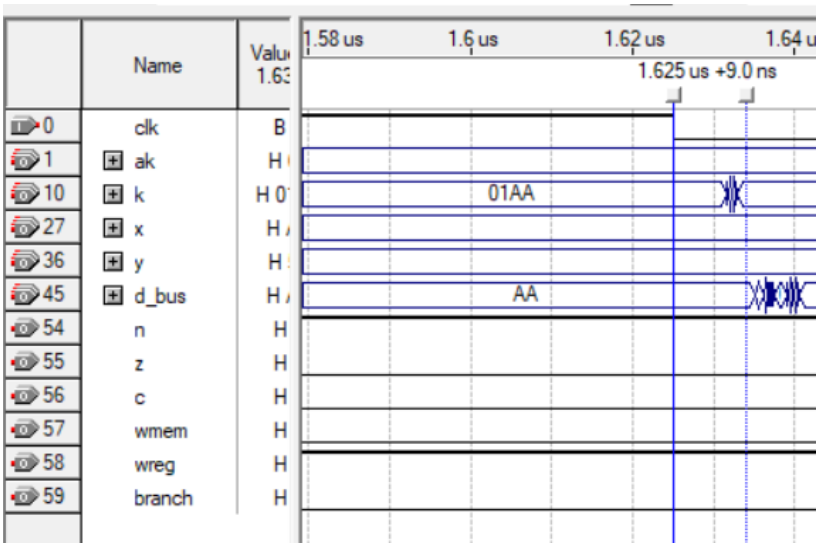

Рисунок 13 – Время чтения команды

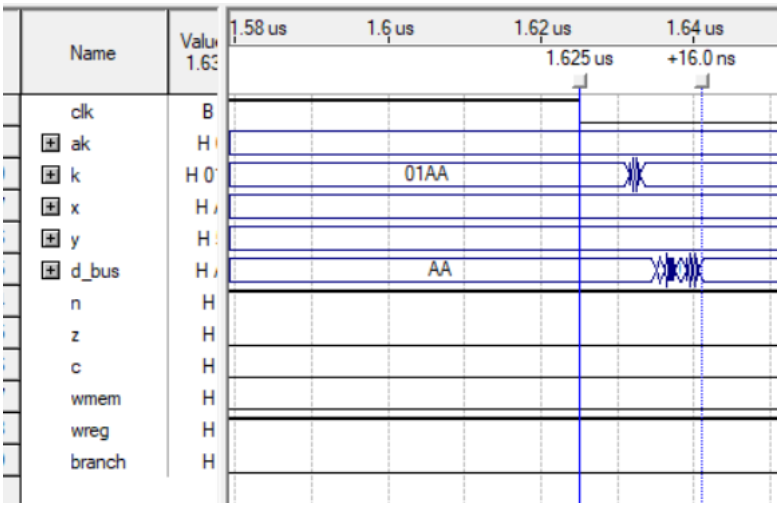

Рисунок 14 – Время формирования данных на шине d_bus

Результат:

- Формирования адреса - 7 нс;
- Чтение команды – 9 нс;
- Получение данных на шине 16 – 9 = 7 нс.

Задание 3. В памяти, начиная с адреса 00, запишите массив из 8 чисел W0 – W7, которые вычисляются в соответствии с заданной формулой: W = 2× k + 1. Определите экспериментально максимальную частоту синхронизации.

Пусть r0 – счетчик циклов. r1 – аккумулятор. r2 – разность прогрессии, r3 – указатель адреса в памяти

Таблица 7. Программа тестирования записи в память

Программа	ak	k
movi r0, #8	00	0008
movi r1, #1	01	0101
movi r2, #2	02	0202
movi r3, #0	03	0300
st r1, [r3]	04	a131
add r1, r2	05	8121
inc r3	06	9300
dec r0	07	9001
jz 0a	08	f10a
jmp 04	09	f004
jmp 0a	0a	f00a

Установим период равный 100 нс:

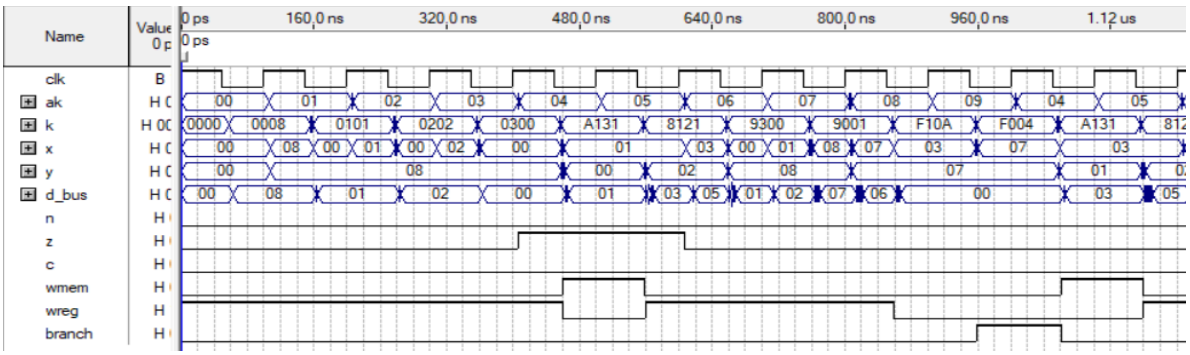

Рисунок 15 – Моделирование программы при 100нс

|cpu|lpm_ram_dq0:inst15|lpm_ram_dq:lpm_ram_dq_component|altram:sr

Addr	+0	+1	+2	+3	+4	+5	+6	+7
0	01	03	05	07	09	0B	0D	0F
8	00	00	00	00	00	00	00	00

Рисунок 16 – Результат выполнения программы при 100 нс

Результат формируется корректно. Уменьшим до 30 нс.

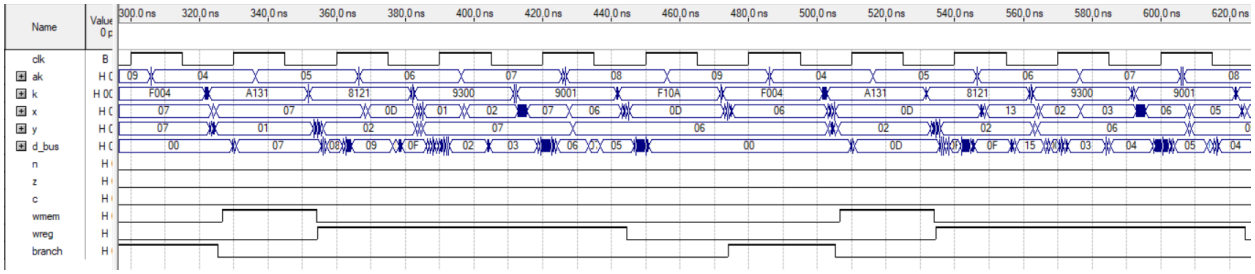

Рисунок 17 – Моделирование программы при 30 нс

Addr	+0	+1	+2	+3	+4	+5	+6	+7
0	01	07	0D	13	19	1F	25	2B
8	00	00	00	00	00	00	00	00

Рисунок 18 – Результат выполнения программы при 30 нс

Как видно результат не соответствует ожидаемому. Увеличим до 40 нс.

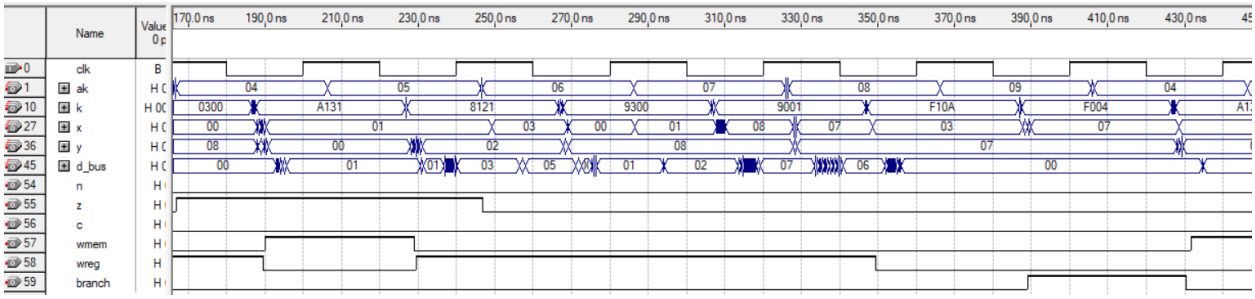

Рисунок 19 – Моделирование программы при 40 нс

|cpu|lpm_ram_dq0:inst15|lpm_ram_dq:lpm_ram_dq_component|altram:sram|conte

Addr	+0	+1	+2	+3	+4	+5	+6	+7
0	01	03	05	07	09	0B	0D	0F
8	00	00	00	00	00	00	00	00

Рисунок 20 – Результат выполнения программы при 40 нс

Экспериментальным путем выяснили максимальную частоту работы микропроцессора при выполнении этой программы. Она составляет 25МГц.

Задание 4. Включите дополнительную команды логического сдвига вправо содержимого регистра lsr rx.

Для добавления команды модифицируем модуль АЛУ (рисунок 21).

```
endcase else
if (k[15:12] == 9) case (k [2:0])
    0: {cl,d_bus,}={1'b0,( x+1)};
    1: {cl,d_bus}= {1'b0,( x-1)};
    2: {cl,d_bus}= {1'b0,( ~x)};
    3: {d_bus,cl}= {x[7], x[7:0]};
    4: {cl,d_bus}= {x,1'b0};
    5: {d_bus,cl} = {x};
endcase else
```

Рисунок 21 – Новая команда в АЛУ

Таблица 8. Программа тестирования операции логического сдвига вправо.

Программа	ak	k	q_bus	nzc
movi r1, 0b10101010	00	01AA	AA	100
lsr r1	01	9105	55	000
lsr r1	02	9105	2A	001
jmp 01	03	f001	xx	xxx

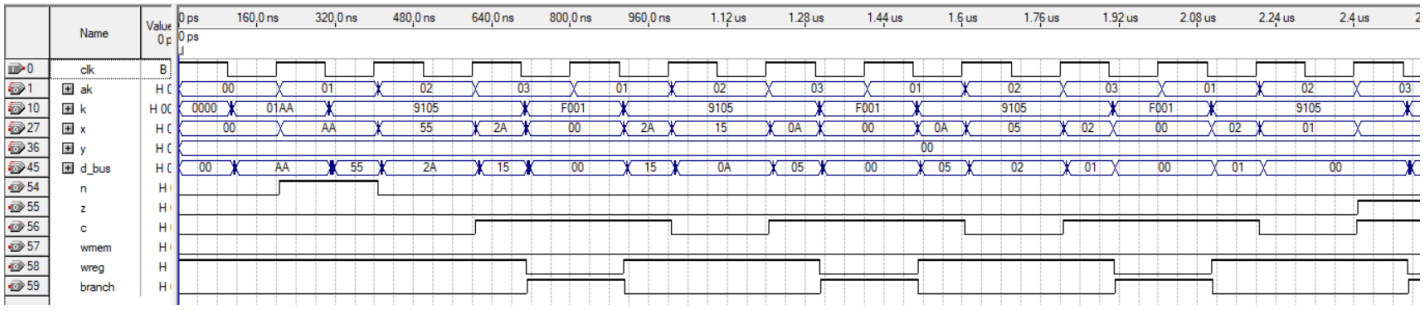

Рисунок 22 – Результат моделирования программы

Как видно из результатов моделирования, операция работает корректно.

Задача 5. Протестировать систему прерываний.

Для проверки системы прерываний напишем программу, представляющую собой бесконечный цикл, а также обработчик прерывания INT_1 по адресу 0x08.

Таблица 9. Бесконечный цикл.

Программа	ak	k	q_bus	nzc
movi r1, 0b00000010	00	0102	02	000
add r0,r1	01	8011	02	000
jmp 01	03	f001	xx	xxx

Таблица 10. Обработчик прерывания.

Программа	ak	k	q_bus	nzc
movi r1, 0b00000001	08	0101	01	000
movi r0, 0b00000001	08	0001	01	000
sub r0,r1	09	8012	00	010
sub r0,r1	a	8012	ff	100
sub r0,r1	b	8012	fe	100
end int	c	dddd	xxx	xxx

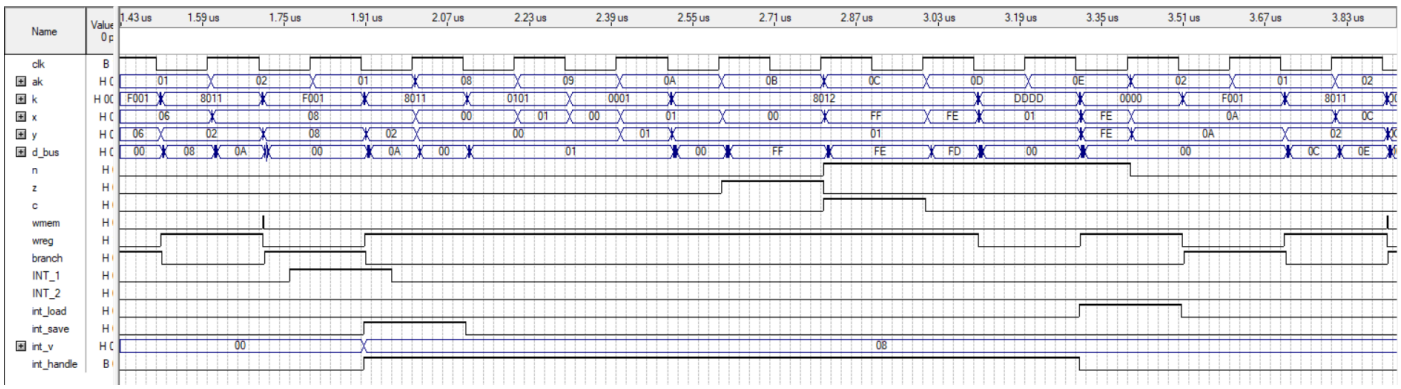

Рисунок 23 – Результат моделирования проверки системы прерываний

Как видно из моделирования в программе выполняется бесконечный цикл, в котором происходит увеличение r0 на 2 в каждом такте. При r0 = 8 происходит прерывание, выполняется команда по адресу 1, появляется сигнал для сохранения РОН, признаков и адреса команды, после чего выполняется переход к ak = 8. В обработчике используются те же регистры,

что и в основной программе, при этом это никак не влияет на выполнение программы. Все действия выполняются корректно. После команды DDDD выполняется загрузка всего состояния обратно в микропроцессор и продолжается обработка с правильной команды, следующей за той, которая была до прерывания, с правильным значением регистра.

Заключение

В ходе курсовой работы была разработана и проверена микропроцессорная система на основе ПЛИС. В полученном микропроцессоре используется гарвардская архитектура, RISC команды с разрядностью данных 8 бит и разрядностью команд 16 бит. Так же дополнительно была разработана система прерываний.

Список используемой литературы

1.Корнеев В. В., Киселев А. В. Современные микропроцессоры. — 3-е изд., перераб, и доп. — СПб.: БХВ - Петербург, 2003. - 448 е.: ил.

2.Стешенко В.Б. ПЛИС фирмы Altera: элементная база, система проектирования и языки описания. _М.: Издательский дом «Додэка-XXI» 2007.-124с.

4.Зотов В. Ю. Проектирование встраиваемых микропроцессорных систем на основе ПЛИС фирмы XILINX. - М.: Горячая линия - Телеком, 2006. - 520 с, ил.

5.В. В. Соловьев. Основы языка проектирования цифровой аппаратуры VERILOG. — М.: Горячая линия-Телеком, 2014. — 205 с: ил.