Лабораторная работа. Конечный автомат: Вычислитель

Базовая структура ЭВМ определена фон Нейманом (1945 г.). Для того, чтобы ЭВМ была универсальным и эффективным устройством обработки информации, она должна строиться в соответствии со следующими принципами:

1. Информация кодируется в двоичной форме (специфика электронных схем – простота и надежность работы) и разделяется на элементы информации, называемые словами. Слово обрабатывается в ЭВМ как одно целое и представляет собой машинный элемент информации.
2. Разнотипные слова информации хранятся в одной и той же памяти (?) и различаются по способу использования, но не по способу кодирования. Т.е., команды или данные выглядят совершенно одинаково, и только порядок использования слов в программе вносит различие в слова. В результате и сами команды могут модифицироваться и обрабатываться как числа.
3. Слова информации размещаются в ячейках памяти машины и идентифицируются номерами ячеек, называемыми адресами слов. Структурно память состоит из перенумерованных ячеек. Чтобы записать слово в память или выбрать слово из памяти необходимо указать адрес ячейки. Таким образом, адрес ячейки служит машинным идентификатором (именем) хранимой величины или команды. При этом выборка из памяти (чтение) не разрушает информацию, хранимую в ячейке.
4. Алгоритм представляется в форме последовательности управляющих слов, называемых командами, которые определяют наименование операции и слова информации, участвующие в операции (операнды). Алгоритм, представленный в терминах машинных команд, называется программой. В общем случае алгоритм в ЭВМ представляется в виде упорядоченной последовательности команд следующего вида:

Разряды полей	**bb…b**	**bb…b**	**bb…b**	**bb…b**	**bb…b**
Тип данных	**код операции**	**A1**	**A2**		**Ak**

   Составные части команды называют полями. Вверху указаны номера разрядов полей. Здесь b – двоичная переменная, принимающая значения 0 или 1. Определенное число первых разрядов слова команды характеризует код операции (КОП). Последующие наборы двоичных переменных определяют адреса A1, A2…Ak операндов (аргументов и результатов), участвующих в операции, заданной своим кодом КОП.
5. Выполнение вычислений, предписанных алгоритмом, сводится к последовательному выполнению команд в порядке, однозначно определяемом программой. Первой выполняется команда, заданная пусковым адресом программы. Адрес следующей команды определяется в процессе выполнения текущей команды. Он может быть следующим по порядку (принцип следования) или адресом любой другой команды (переход). Процесс выполнения программы продолжается до тех пор, пока не будет подана команда остановки.

Следует подчеркнуть, что вычисления и все действия, проводимые машиной, определяются программой. Это позволяет изменить функции, выполняемые ЭВМ.

Перечисленные принципы функционирования ЭВМ предполагают наличие следующих устройств (базовая архитектура фон Неймана):

- арифметико-логическое устройство (АЛУ), выполняющее арифметические и логические операции;
- устройство управления (УУ), которое организует процесс выполнения программы;
- запоминающее устройство (ЗУ), или память для хранения программ и данных;
- устройство для ввода и вывода (ВУ) информации.

Архитектура ЭВМ определяет ее логическую организацию, состав и назначение функциональных устройств. Структура ЭВМ – совокупность элементов и связей между ними.

Обобщенная структура ЭВМ представлена на рис. 4.52.

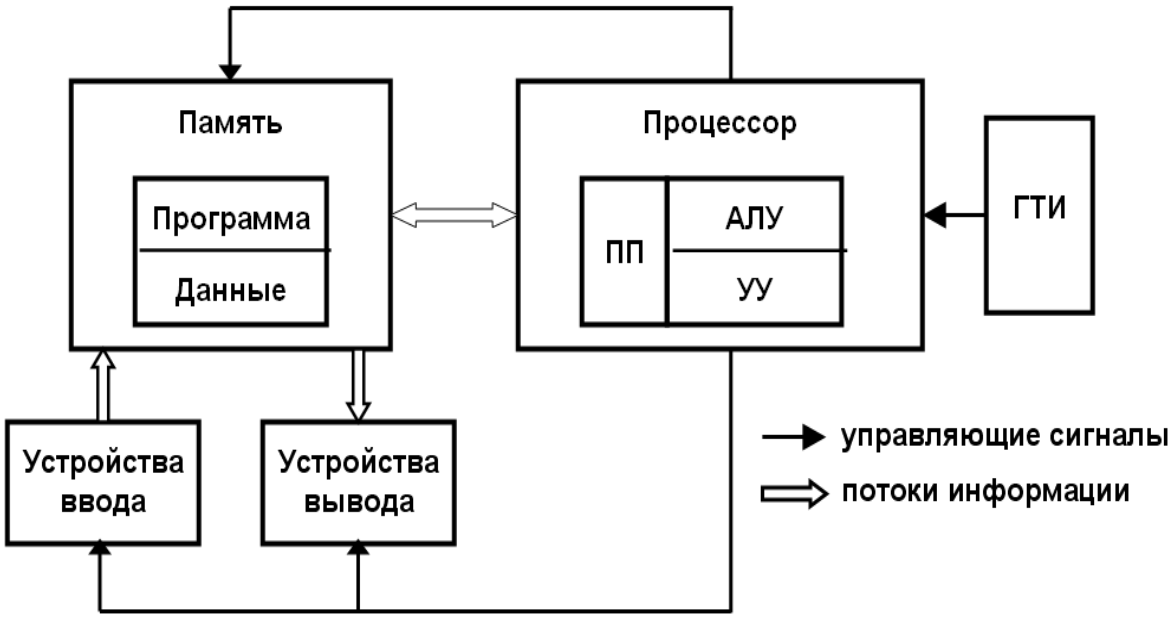

Рисунок 4.52 – Типовая архитектура ЭВМ.

ГТИ (генератор тактовых импульсов), осуществляет синхронизацию процессов передачи информации и выполнения команд (по фронтам тактовых импульсов). Промежуток времени между тактовыми импульсами определяет быстродействие ЭВМ.

УУ (устройство управления) и АЛУ (арифметико-логическое устройство) – объединены в процессор, предназначенный для обработки данных и управления работой ПК по заданной программе.

ПП (процессорная память) – специальные ячейки памяти называемые регистрами, которые располагаются внутри микропроцессора и служат для временного хранения информации.

В общем случае *регистры* – элементы памяти, каждый из которых может находиться в одном из двух устойчивых состояний (транзистор проводит или закрыт, конденсатор заряжен или разряжен, поляризованность в одном или другом направлении и т.п.). Регистр характеризуется числом битов информации, которые в нем могут храниться (разрядность). Пока включено питание, помещенная в нем информация остается до тех пор, пока она не будет заменена другой, процесс чтения информации из регистра не влияет на его содержимое и означает, что копия содержимого регистра будет создана и где-то сохранена.

Каждый бит регистра передается по отдельному проводнику. Вся совокупность этих проводников образует шину.

Регистры, в частности, служат для ускорения работы микропроцессора и выполняют специальные функции:

- *аккумулятор,* в котором располагаются один из операндов или результат операции;
- *счетчик команд* – специальный регистр, содержимое которого увеличивается на единицу в момент выборки из памяти исполняемой команды и, если выбрана команда перехода, то оно может быть заменено на содержимое адресной части команды перехода; в конце цикла исполнения команды в этом счетчике всегда должен находиться адрес команды, которая выполняется за текущей (т.е. следующая по порядку команда или другая, к которой требуется перейти при выполнении условий, заданных кодом операции команды перехода);
- *регистр команд,* в котором размещается исполняемая команда;
- *регистр адреса*, содержащего адрес ячейки памяти, из которой будет считана команда, операнд или записан результат обработки;
- *регистр состояния*, специальный регистр, в котором хранятся признаки результата выполненной операции, по которым осуществляются переходы (передача управления).

Память ЭВМ – обеспечивает хранение команд и данных. Состоит из блоков одинакового размера (ячеек памяти), хранящих одно слово информации, все ячейки нумеруются, и адрес ячейки однозначно идентифицирует данные или команду в ней хранящуюся. При считывании содержимое ячейки не изменяется. При записи в ячейку хранимое в ней слово информации заменяется на новое.

Системный интерфейс – часть открытой архитектуры, набор цепей, связывающих процессор с памятью и контроллерами вычислительного устройства, алгоритм передачи сигналов по этим цепям, их электрические параметры и конструктивные элементы.

Исполнение команд процессором: определяется системой команд (микропрограмм), которые может распознавать и исполнять устройство управления (УУ). Выполняется принцип следования, согласно которому все команды выбираются из памяти последовательно. За адресом следующей команды следит специальный регистр – счетчик команд, в который УУ помещает адрес следующей команды. Обычно его содержимое увеличивается (инкрементируется) на одно или два слова памяти команд.

Для решения задачи на такой ЭВМ следует:

1. Через устройство ввода информации загрузить в память ЭВМ программу решения задачи (алгоритм, написанный на языке ЭВМ, т.е. машинный код и исходные данные). Программа и исходные данные могут быть размещены в любой области памяти, начиная с ячейки с нулевым адресом или другим.
2. Сообщить процессору адрес ячейки памяти, в которой размещена первая (очередная) команда программы, для чего занести адрес этой ячейки в счетчик команд.
3. Каким либо способом заставить процессор приступить к исполнению программы (включить питание, нажать кнопку «Выполнить» или кнопку «Сброс»). В результате в память процессора поступит адрес первой (очередной) команды программы и произойдет пересылка содержимого этой ячейки адреса в регистр команд. С этого момента процессор начнет циклически выполнять стандартные операции выполнения программы.

Таким образом, работу вычислительной системы можно описать как циклическое повторение следующих операций:

- получить адрес памяти из счетчика команд PC;
- считать команду по данному адресу и поместить ее в регистр устройства управления;
- интерпретировать команду при необходимости считать из памяти значения операндов;
- по типу команды определить адрес следующей команды и соответствующим образом инкрементировать счетчик команд PC;
- выполнить команду в соответствии с существующим набором инструкций микропроцессора.

Если результат выполнения операции не пересылать в память, а сохранять в специальном регистре (аккумуляторе) то, можно построить систему команд, используя в каждой не более одного адреса. Команды в этом случае можно условно разделить на три группы:

- арифметические и логические (команды обработки), которые дают приказ на выполнение какой-либо арифметической или логической операции, используя в качестве операндов содержимое аккумулятора и содержимое адресуемой ячейки памяти (регистра данных);
- команды пересылки, которые дают приказ на обмен информацией между аккумулятором и памятью (через регистр данных), т.е. на загрузку аккумулятора содержимым адресуемой ячейки памяти или на запись в эту ячейку содержимого аккумулятора (результата обработки);
- команды установки битов слова состояния процессора и передачи управления, обеспечивающие переход к новой ячейке памяти с командой, которая должна быть выполнена следующей в программе при выполнении какого-либо условия или независимо от него.

Типовая структура процессора для такой системы команд приведена на рис. 4.52.

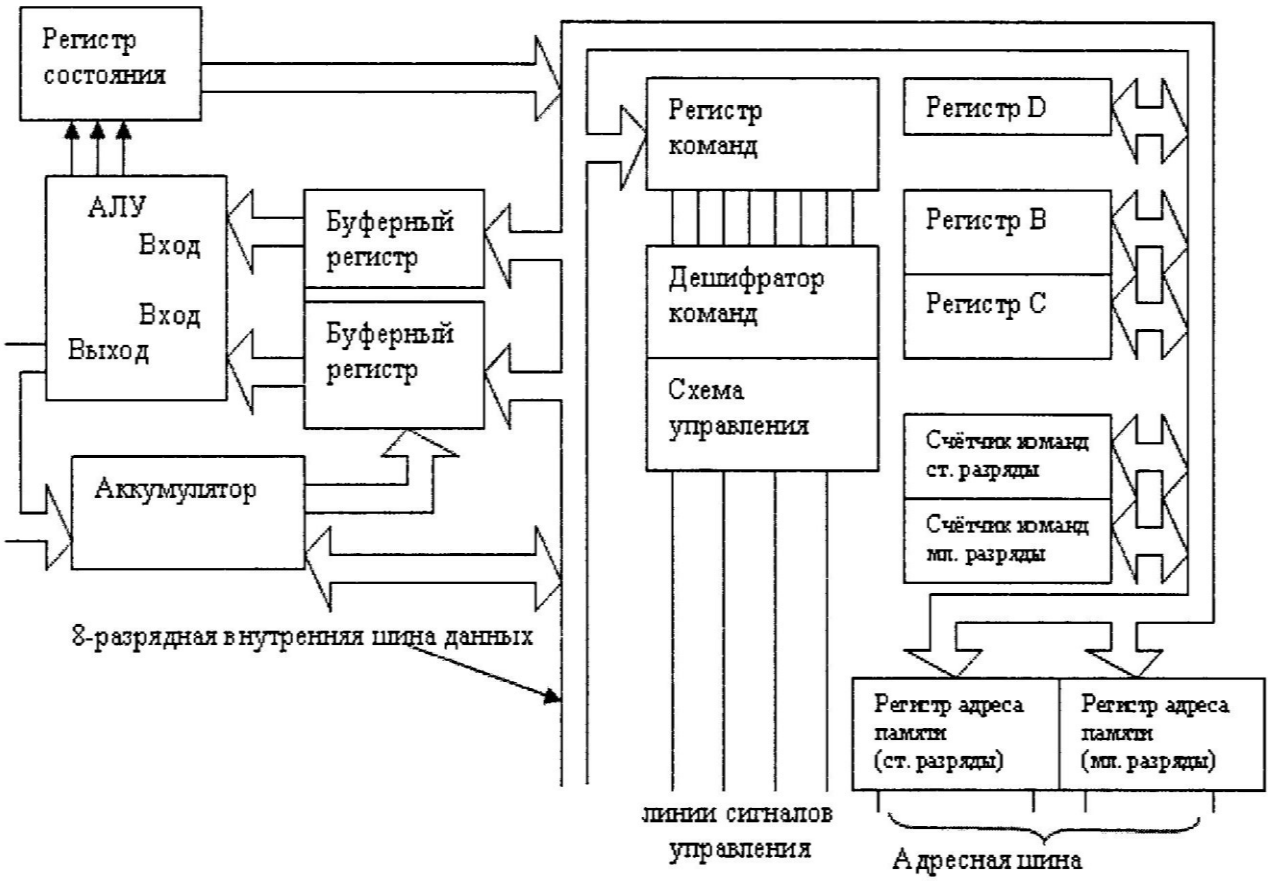

Рисунок 4.52 – Структурная схема микропроцессора

Задание для самопроверки:

Структура простейшего вычислителя, проект которого предлагается выполнить в качестве задания для самопроверки, аналогична типовой архитектуре простейшей вычислительной системы (см. рис. 4.53).

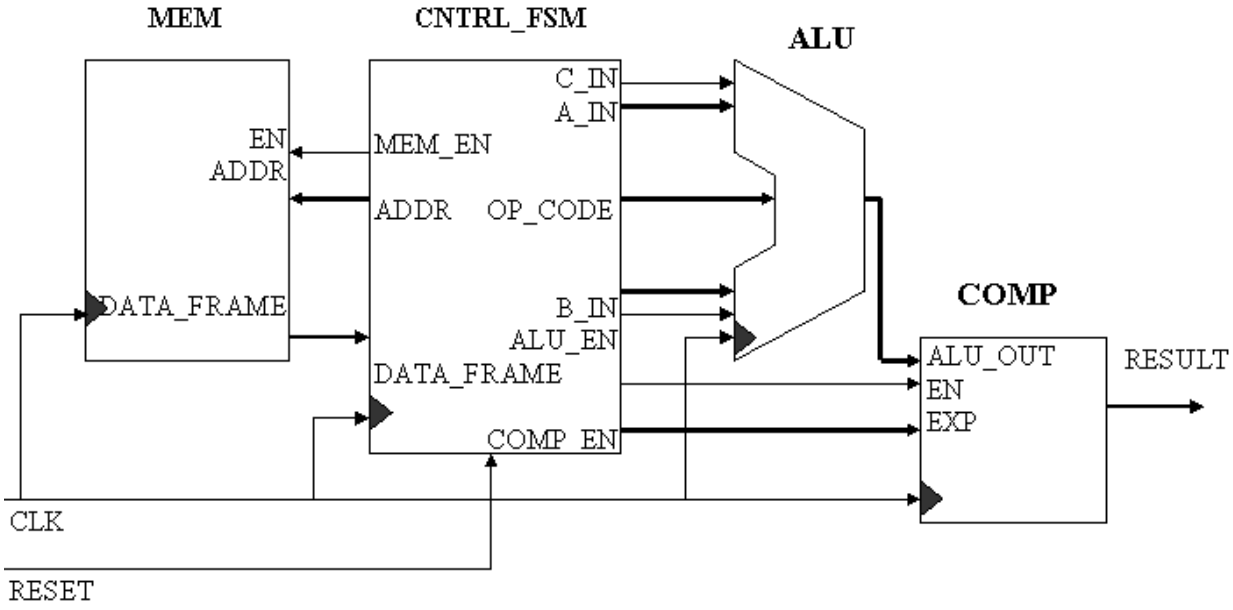

Рисунок 4.53 – Структурная схема устройства «Вычислитель».

Для изучения типовой архитектуры вычислительных устройств, совершенствования навыков разработки и моделирования устройств и систем с помощью HDL-кода на примерах проектирования модуля памяти, конечного автомата, арифметико-логического устройства, компаратора, объединенных модулем верхнего уровня, представляющем собой простейшую вычислительную структуру, выполнить структурное описание для объекта вычислителя SIMPLE_CALC, находящегося на верхнем уровне иерархии и состоящего из следующих компонентов более низкого уровня:

- модуля памяти MEM (задание 4.5.1.),
- одного из компараторов COMP_BEH или COMP_RTL в соответствии с тем как используется HDL-описание: для моделирования или для синтеза (задание 4.5.2),
- модуль арифметико-логического устройства ALU (задание 4.5.3),
- модуль конечного автомата CNTRL_FSM (задание 4.5.4).
- разработать устройство «Вычислитель», включающее в приведенные выше модули (задание 4.5.5) и используя возможности демонстрационной платы ATLYS и провести аппаратное тестировании его работы.

Вычислитель должен последовательно считать данные и коды операций из памяти, декодировать и выполнить команды над операндами, хранящимися в той же памяти и сравнить результат операции с эталонными значениями, также считанными из памяти. Выходным сигналом устройства является одноразрядный сигнал RESULT, который равен логической единице, в случае, если результат операции на выходе ALU совпадает с эталонным значением, считанным из памяти.

Задание 4.5.1. Проектирование памяти

Разработайте RTL-код для модуля памяти MEM с использованием файла, содержащего параметры настройки объема памяти. Выполните моделирование работы памяти, анализируя данные на выходе при чтении первых шести значений слов памяти.

***Указания**:* Модуль синхронной памяти MEM является одним из компонентов проекта «Вычислитель» (SIMPLE_CALC). Параметры памяти: размер пространства адресов ADDR=3 и разрядность слова, хранящегося в одной ячейке WIDTH=17 удобно

задавать во внешнем текстовом файле MY_HEADER. Эти параметры затем следует включить в проект с помощью директивы компилятора **`*include "MY_HEADER.txt"*`**. Для создания текстового файла значений параметров из навигатора проекта используйте пункт меню ***New Source Wizard***, в котором выберете тип модуля ***User Document.*** После открытия шаблона модуля введите в него следующий текст:

```
` define WIDTH 17
` define ADDR 3
```

Затем сохраните файл и убедитесь, что в окне навигатора проекта во вкладке явился файл ***MY_HEADER.txt***.

Каждое слово в ПЗУ содержит данные (операнды) и команды для организации работы арифметико-логического устройства вычислителя. Формат слова данных и команд приведен в табл. 4.11:

Таблица 4.11 – Формат 17-ти разрядного слова памяти модуля MEM.

Назначение данных из памяти	Первый операнд	Второй операнд	Бит переноса	Код операции	Эталонное значение результата
Обозначение на структурной схеме	A_IN	B_IN	C_IN	OP_CODE	EXP
Разряды слова	16…13	12…9	8	7…4	3…0

Добавьте в проект новый файл с кодом ПЗУ MEM.v и отредактируйте шаблон модуля таким образом, чтобы входной сигнал EN =1 разрешал считывание данных из памяти, синхронизированное сигналом CLK. Применяя операторы always, if и case, напишите код, реализующий выдачу данных на 17-ти разрядный выход DATA_FRAME в соответствии с табл. 4.12. При выборе любого другого адреса ПЗУ, считать, что значения данных не определены (17'bx).

Таблица 4.12 – Данные памяти, передающиеся на выход DATA_FRAME в соответствии с заданным адресом.

Адрес	Данные	Десятеричное значение
3'b000	17'b0001_1000_1_0000_0010	12546
3'b001	17'b1001_0100_0_0001_1101	75805
3'b010	17'b0101_1010_1_0001_0000	46352
3'b011	17'b0001_0010_0_0010_1110	9262
3'b100	17'b0011_0001_1_0010_0010	25378
3'b101	17'b0000_1001_0_0111_1111	4735

Для моделирования работы ПЗУ примените конструкцию **initial**, внутри которой через определённые промежутки времени задавайте адреса ПЗУ и контролируйте значение данных на выходе памяти при каждом фронте сигнала CLK. Для анализа данных

используйте представление DATA_FRAME в десятичном формате. Для этого выделите этот сигнал в окне симулятора и в контекстном меню в выпадающей вкладке выберите пункты **Radix - Unsigned Decimal**.

Задание 4.5.2. Проектирование компаратора

Используя оператор ***if/else***, разработайте коды двух версий описаний для модуля **COMP** (синхронный компаратор). В первом случае поведенческое описание должно быть предназначено для испытательного стенда, выдающего сообщение об ошибке с помощью директивы ***$dislplay***. Во втором случае – напишите только синтезируемый RTL код, также используя конструкцию ***if/else***. Проверьте работу компаратора с помощью испытательного стенда.

Указания для выполнения задания 5.2:

Компараторы применяются для сравнения двух входных кодов и выдачи на выходы сигналов о результатах этого сравнения. Модуль компаратор COMP является одним из компонентов проекта «Вычислитель» (SIMPLE_CALC). В окончательном варианте в код SIMPLE_CALC модуль компаратора должен быть подключен при помощи процедуры условного подключения модуля в зависимости от того, решается ли задача синтеза всего устройства или задача тестирования его работоспособности.

Используйте разрядности и обозначения для портов компаратора, приведенные в табл. 4.13.

Таблица 4.13 – Разрядности и обозначения для портов модуля компаратора COMP.

Имя порта	Разрядность, бит	Описание
COMP_EN	1	вход, цепь разрешения работы компаратора, генерируется конечным автоматом
EXP	4	вход, эталонное значение, генерируется выходным сигналом памяти
ALU_OUT	4	вход, результат операции ALU
RESULT	1	выход, результат сравнения выходного сигнала ALU с эталонным значением.
CLK	1	вход синхронизации

Для выдачи сообщения об отрицательных результатах моделирования запишите следующую строку кода:

```
$display ($time, "_Simulation mismatch occurred, ALU_OUT is not equal to EXP");
```

согласно которой в момент времени **time** на экран монитора выдается сообщение «Обнаружено несоответствие: выходной сигнал ALU_COMP не совпадает с значением EXP».

Используя процедуру **View RTL Schematic,** изучите, как разработанный код синтезирован в схему. Обратите внимание, что сообщение об ошибке не синтезируется.

Для проверки функционирования модуля COMP последовательно задайте два варианта выходных сигналов ALU_OUT и убедитесь, что в случае несовпадения его значения с EXP на монитор выдается соответствующее сообщение.

Задание 4.5.3. Проектирование арифметико-логического устройства

Напишите RTL-код для описания арифметико-логического устройства ALU, представляющего собой один из модулей устройства «Вычислитель» (SIMPLE_CALC). С помощью испытательного стенда проверьте работоспособность разработанного описания.

Указания Арифметико-логическое устройство (АЛУ) предназначено для выполнения логических и арифметико-логических операций над операндами. Как правило, АЛУ является составной частью интегральной схемы микропроцессора, и перечень функций АЛУ определяется набором команд конкретного типа микропроцессора.

Пример АЛУ, выполненного как отдельная микросхема, – четырехразряное скоростное АЛУ К155ИП3. В проекте вычислителя Verilog-код модуля АЛУ должен описывать четырёхразрядные входы операндов A и B, четырёхразрядный вход выбора (кода) операций OP_CODE, вход переноса C_IN, вход разрешения выполнения операций EN и вход сигнала синхронизации CLK. Результат операции вырабатывается на выходе Y (рис. 4.54).

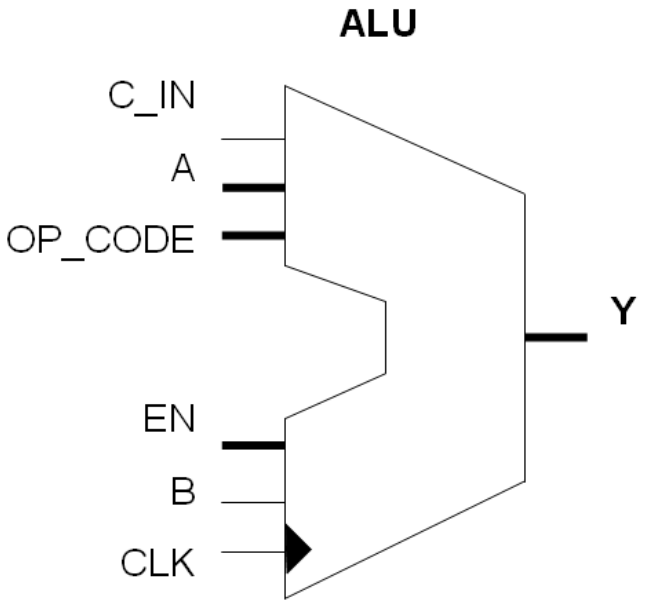

Рисунок 4.54 – Входные и выходные сигналы модуля ALU.

При написании кода для выбора конкретной операции используйте табл. 4.14 и оператор **case**, аргументами которого служит результат конкатенации значений кода операции OP_CODE и бита переноса C_IN:

```
assign OP_CODE_CI = {OP_CODE,C_IN};
```

Таблица 4.14 – Функции АЛУ в зависимости от состояния входов OP_CODE и C_IN.

OP_CODE	C_IN	Операция	Примечание
0000	0	Y = A	-
0000	1	Y = A +1	Инкремент
0001	0	Y = A + B	Сложение
0001	1	Y = A + B + 1	Сложение с переносом
0010	0	Y = A + (~B)	Сложение с инверсией
0010	1	Y = A + (~B) + 1	Вычитание
0011	0	Y = A – 1	Декремент
0011	1	Y = B	-
0100	0	Y = A & B	И
0101	0	Y = A | B	ИЛИ
0110	0	Y = A ^ B	ИСКЛЮЧАЮЩЕЕ ИЛИ
0111	0	Y = ~ A	Инверсия
1000	0	Y = 0	Установка нуля

Для задания входных сигналов при моделировании используйте шаблон испытательного стенда, в котором входное воздействие передается 5-ти разрядной цепью OP_CODE_CI_SIG, разделяющейся затем на две цепи OP_CODE и C_IN в соответствии с примером, приведенным на рис. 4.55.

Для проверки функционирования разработанного модуля составьте проверочную таблицу (например, аналогичную, табл. 4.15), в которой для нескольких значений пар входных операндов A и B, при заданных кодах операций вычислите ожидаемые значения.

Занесите результаты, полученные в ходе моделирования, в таблицу и сравните их с вычисленными значениями сигнала Y.

Таблица 4.15 – Ожидаемые и полученные в ходе моделирования результаты работы АЛУ (тестовая таблица).

A (операнд 1)	B (операнд 2)	Операция	OP_CODE	C_IN	Y	Ожидаемый результат
0001	1000	Y = A	0000	0		**0001**
0001	1000	Y = A +1	0000	1		**0010**
0001	1000	Y = A + B	0000	0		**1001**
…	…	…	…	…	…	…
0001	1000		1000	0		**0000**

```
`timescale 1ns / 1ps

module ALU_TB_v;

  wire[3:0] OP_CODE;
  reg [3:0] A = 4'b0001;
  reg [3:0] B = 4'b1000;
  wire C_IN;
  reg CLK = 1'b0;
  reg EN = 1'b1;

  wire [3:0] Y;

  reg [4:0] OP_CODE_CI_SIG ;

    assign  OP_CODE = OP_CODE_CI_SIG[4:1];
    assign  C_IN = OP_CODE_CI_SIG[0];

  ALU uut (
       .CLK(CLK),
       .OP_CODE(OP_CODE),
       .A(A),
       .B(B),
       .C_IN(C_IN),
       .EN(EN),
       .Y(Y));

always #10 CLK = ~CLK ;

initial
begin
  #100 OP_CODE_CI_SIG = 5'b00000;
  #100 OP_CODE_CI_SIG = 5'b00001;
  #100 OP_CODE_CI_SIG = 5'b00010;
  #100 OP_CODE_CI_SIG = 5'b00011;
  #100 OP_CODE_CI_SIG = 5'b00100;
  #100 OP_CODE_CI_SIG = 5'b00101;
  #100 OP_CODE_CI_SIG = 5'b00110;
  #100 OP_CODE_CI_SIG = 5'b00111;
  #100 OP_CODE_CI_SIG = 5'b01000;
  #100 OP_CODE_CI_SIG = 5'b01010;
  #100 OP_CODE_CI_SIG = 5'b01100;
  #100 OP_CODE_CI_SIG = 5'b01110;
  #100 OP_CODE_CI_SIG = 5'b10000;
end

endmodule
```

Рисунок 4.55 – Пример кода тестового файла для моделирования ALU.

Задание 4.5.4. Проектирование конечного автомата

Напишите RTL-код, реализующий конечный автомат устройства «Вычислитель» (SIMPLE_CALC). С помощью испытательного стенда проверьте работоспособность разработанного описания.

Указания: Конечный автомат CNTRL_FSM является главным компонентом проекта «Вычислитель» (SIMPLE_CALC) поскольку он управляет работой остальных модулей

устройства. В данном случае для организации машинного цикла используется автомат Мура, в котором выходные сигналы зависят только от текущего состояния автомата. Описание работы конечного автомата в виде схемы графов, отражающих все его состояния, приведено на рис. 4.56.

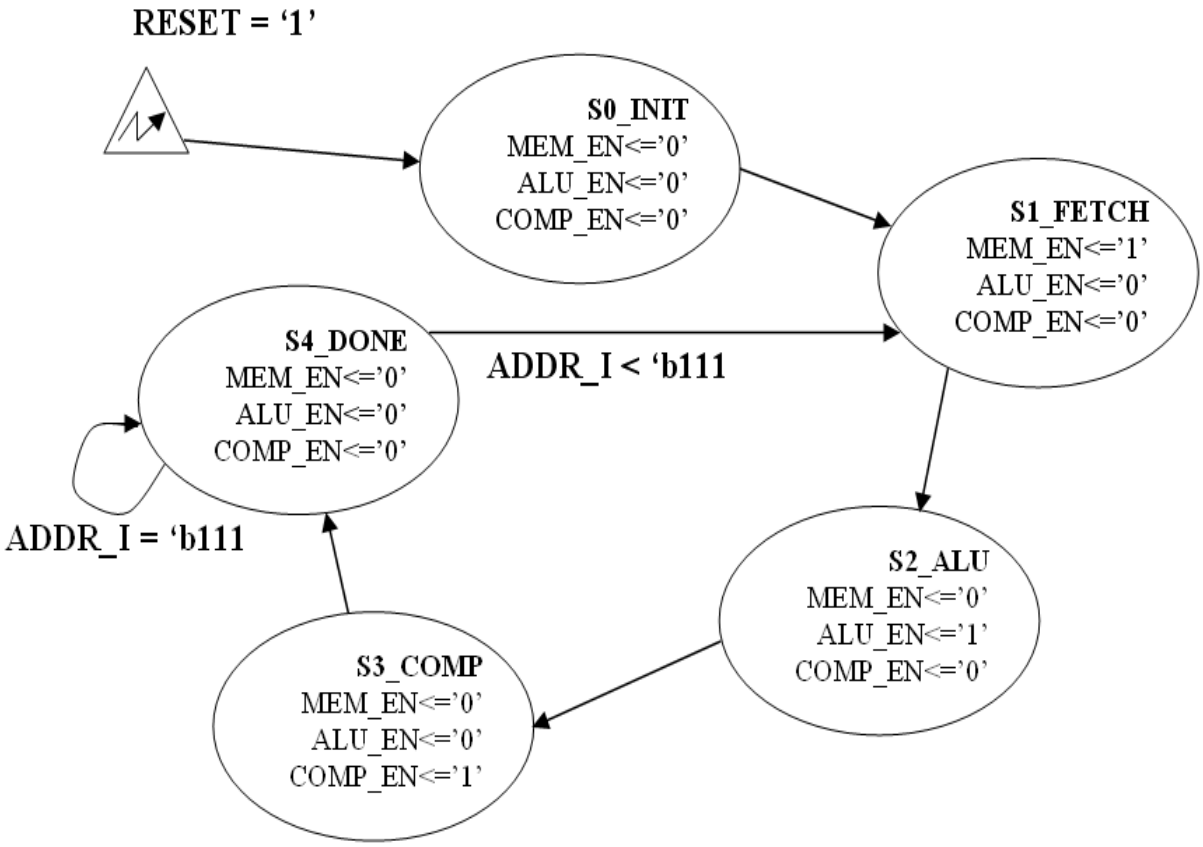

Рисунок 4.56 – Состояния и переходы конечного автомата, организующего работы устройства «Вычислитель».

Назначение конечного автомата – последовательное автоматическое выполнение всех операций внутреннего цикла работы вычислительной системы. В данном случае внутренний цикл вычислительной системы состоит в следующем: 1) начальное состояние; 2) считывание слова памяти, и выделение из считанного слова данных значений операндов, кода операции, значения бита переноса и эталонного значения результата; 3) выдача команды ALU выполнить ту или другую операцию в соответствии с ее кодом; 4) выдача компаратору команды разрешения сравнить полученный в АЛУ результат с эталонным значением и вывести результат сравнения; 5) определение следующего адреса для считывания слова из памяти и переход в состояние ожидания **`S4_DONE`** или в состояние начала нового внутреннего цикла **`S1_FETCH`**. Из состояние ожидания автомат может выйти только после получения сигнала RESET. Все переходы из одного состояния в другое осуществляются синхронно по переднему фронту сигнала CLK.

По условию задания команды и данные для работы устройства содержатся в семи первых словах памяти. Таким образом, автомат должен последовательно считать семь слов памяти, выделить во внутреннем цикле из считанных данных операнды, коды операций, бит переноса и эталонное значение результата, дать команду ALU выполнить операцию,

затем разрешить компаратору сравнить полученный результат с эталонным значением.

Рекомендуемая последовательность действий при программировании описания конечного автомата из проекта «вычислитель» приведена ниже.

Создайте в новом проекте файл с описанием конечного автомата ***CNTRL_FSM.v*** и определите для него входные и выходные порты в соответствии с рисунком 5.5.3:

```
`timescale 1ns / 1ps
module CNTRL_FSM(
     input CLK, RESET,
     input [16:0] DATA_FRAME,
     output reg [3:0] A_IN, B_IN, OP_CODE, EXP,
     output reg [2:0] ADDR,
     output reg COMP_EN, ALU_EN, C_IN, MEM_EN
     );
```

Задайте 5-ти разрядные локальные параметры для каждого состояния автомата в соответствии с рисунком 4.56:

```
localparam [4:0]
     S0_INIT   = 5'b00001,
     S1_FETCH  = 5'b00010,
     S2_ALU    = 5'b00100,
     S3_COMP   = 5'b01000,
     S4_DONE   = 5'b10000;
```

Объявите пятиразрядные переменные CURR_STATE и NEXT_STATE (тип ***reg***), предназначенные для хранения значений текущего состояния автомата:

```
reg [4:0] CURR_STATE, NEXT_STATE;
```

Объявите трехразрядную переменную ADDR_I (тип ***reg***), играющую в вычислителе роль внутреннего инкрементного счетчика машинных циклов и предназначенную для хранения определения момента перехода автомата в режим ожидания (состояние **S4_DONE)**:

```
reg [2:0] ADDR_I;
```

Запишите фрагмент кода синхронизации всех внутренних событий сигналом CLK, и то, что сигнал RESET, установленный в единицу активизирует асинхронный сброс автомата в начальное состояние:

```
always @ ( posedge CLK, posedge RESET)
     if (RESET == 1'b1)
          begin
          CURR_STATE <= SO_INIT;
          ADDR <= 3'b000;
          end
     else
          begin
          CURR_STATE <= NEXT_STATE;
          ADDR <= ADDR_I;
          end
```

При написании кода примите во внимание следующие особенности входных и выходных сигналов **DATA_FRAME** и **ADDR**:

- DATA_FRAME – 17-ти битный вектор, считываемый из памяти и содержащий всю информацию для выполнения одного машинного цикла в соответствии с таблицей 5.5.1. Для установки значений сигналов используйте конструкцию **always @ *** и переназначение индексов:

```
always @ *
     begin
     A_IN      = DATA_FRAME[16:13] ;
     B_IN      = DATA_FRAME[12:9] ;
     C_IN      = DATA_FRAME[8] ;
     OP_CODE   = DATA_FRAME[7:4];
     EXP       = DATA_FRAME[3:0] ;
```

Сокращенная конструкция «**always @ *** » означает, что в списке чувствительности присутствуют все входные сигналы, которые могут повлиять на значения выходов. Эта конструкция используется в том случае, когда требуется описать элементы, которые изменяются, как только изменяются значения одного из их входов, и только для описания блоков содержащих комбинаторную логику или логические вентили. Внутри этой конструкции допускается использование только блокирующих (=) присваиваний.

- ADDR – выходной сигнал, который должен инкрементироваться с каждым выполнением внутреннего цикла автомата для организации обращения к следующей ячейки памяти и считывания следующего слова с операндами и кодом операции. Адрес памяти также используется внутри цикла как условие для определения момента остановки работы конечного автомата. Поэтому в коде следует использовать дополнительную переменную - внутренний адрес **ADDR_I**, также увеличивающийся на единицу после завершения каждого полного внутреннего цикла работы автомата.

Следует принять меры, чтобы предотвратить возможность защелкивания внутреннего адреса при неопределенности текущего состояния автомата. Для этого одновременно с инициализацией входных сигналов, следует задать также конкретное значение внутреннего адреса, равное текущему внешнему адресу:

```
ADDR_I = ADDR;
```

Для переходов конечного автомата из одного состояния в другое целесообразно использовать оператор case, условием для которого служит его текущее состояние:

```
case (CURR_STATE)

     S0_INIT:
     begin
          MEM_EN  = 1'b0;
          ALU_EN  = 1'b0;
          COMP_EN = 1'b0;

          NEXT_STATE = S1_FETCH ;
     end

     S1_FETCH:
     begin
          MEM_EN  = 1'b1;
          ALU_EN  = 1'b0;
          COMP_EN = 1'b0;

          NEXT_STATE = S2_ALU ;
     end

    S2_ALU:
     begin
          MEM_EN  = 1'b0;
          ALU_EN  = 1'b1;
          COMP_EN = 1'b0;

          NEXT_STATE = S3_COMP ;
     end
     S3_COMP:
     begin
          MEM_EN  = 1'b0;
          ALU_EN  = 1'b0;
          COMP_EN = 1'b1;

          NEXT_STATE = S4_DONE ;
     end
     S4_DONE:
     begin
          MEM_EN  = 1'b0;
          ALU_EN  = 1'b0;
```

```
                    COMP_EN = 1'b0;
                    if ( ADDR == 3'b111 )
                         begin
                              NEXT_STATE = S4_DONE;
                         end
                    else
                         begin
                              NEXT_STATE = S1_FETCH ;
                              ADDR_I = ADDR + 1;
                         end
               end
          default:
               begin
                    NEXT_STATE = S0_INIT;
                    MEM_EN = 1'b0;
                    COMP_EN = 1'b0;
                    ALU_EN =  1'b0;
               end
          endcase
end                 // относится к always @ *
endmodule           // относится к module CNTRL_FSM
```

При написании кода испытательного стенда ***CNTRL_FSM_TB.v*** задайте входные воздействия так, чтобы проверить следующие события:

- по сигналу **RESET**==1, автомат приходит в свое начальное состояние;

- сигнал **ADDR** должным образом инкрементируется (каждый раз за три периода сигнала **CLK**);

- сигналы **MEM_EN, ALU_EN** и **COMP_EN** выдаются на выход в нужные моменты времени);

- переходит ли FSM в состояние **S4_DONE**, если сигнал **ADDR_I** достигает своего предельного значения?

Для проверки функционирования задайте периодический тактовый сигнал с частотой 50 МГц (**`forever #10 CLK = ~CLK)`** и изменяйте через определенные промежутки времени значения слова данных **DATA_FRAME,** также как это было выполнено при выполнении заданий 4.5.1 и 4.5.3.

Пример кода испытательного стенда приведен на рис. 4.57:

```
initial
   begin
        // Initialize Inputs
        CLK = 0;
        RESET = 0;
        DATA_FRAME = 0;
        forever #10 CLK = ~CLK ;
   end

initial
   begin
        #50 RESET = 1'b1;
        #25 RESET = 1'b0;
   end

initial
   begin
        #1    DATA_FRAME = 17'b1100_0011_1_0000_1111 ;
        #100  DATA_FRAME = 17'b0011_0011_1_1110_1100 ;
        #100  DATA_FRAME = 17'b1101_1011_1_1000_1111 ;
        #100  …
   end
endmodule // это относится к module CNTRL_FSM_TB
```

Рисунок 4.57 – Пример кола испытательного стенда для конечного автомата, использующегося в проекте вычислителя

При анализе результатов моделирования учтите, что сигналы модулей низкого уровня **UUT** не отображаются в окне временной диаграммы автоматически. Если требуется отобразить сигналы модулей нижних уровней иерархии, следует найти сигналы соответствующего компонента **UUT** в иерархическом списке сигналов (вкладка **Instants and Process Name)** и назначить нужные сигналы для вывода принудительно в ручном режиме, поместив соответствующие имена из окна **Objects** в окно отображаемых сигналов **Name** (пример приведен на рис. 5.5.7).

Задание 4.5.5. Проектирование модуля верхнего уровня «Вычислитель»

Напишите Verilog-код структурного описания модуля ***SIMPLE_CALC***, находящегося на верхнем уровне иерархии проекта «Вычислитель», и состоящего из модулей более низкого уровня, разработанных при выполнении заданий 5.1…5.4.

Для включения в объект ***SIMPLE_CALC*** один из объектов ***COMP_BEH*** или ***COMP_RTL*** используйте специальную процедуру **generate,** проверяющую состояние специально введенного в код однобитового флага **`SINTH`** (см. рис. 4.58).

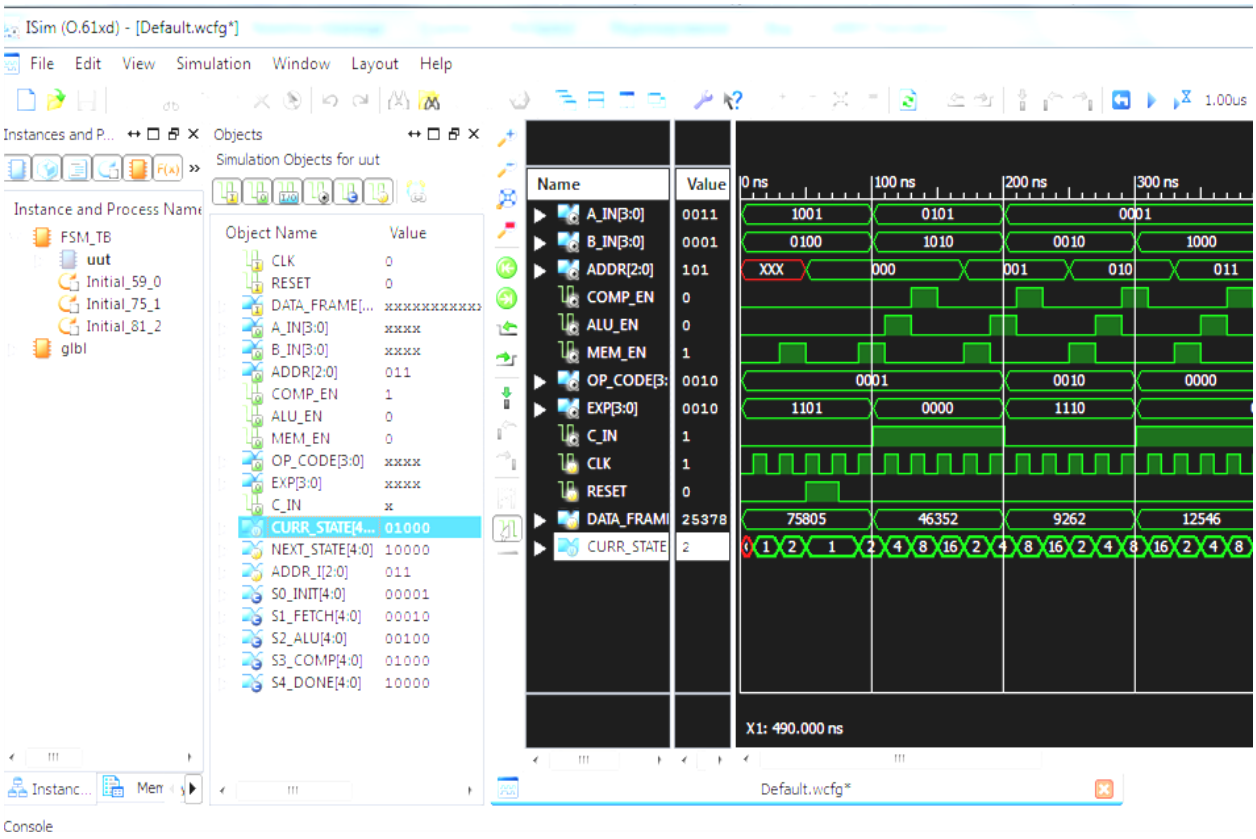

Рисунок 4.58 – Назначение внутренних сигналов UUT для отображения во временной диаграмме.

Измените содержимое памяти **MEM** так, чтобы выполнить проверку работоспособности проекта с помощью испытательного стенда и проверьте работоспособность разработанного описания.

Реализуйте схему вычислителя на ПЛИС (FPGA Spartan 6), используя возможности демонстрационной платы ATLYS.

Структурная схема объекта верхнего уровня SIMPLE_CALC представлена на рис. 4.59. Простейший вычислитель можно представить в видё «чёрного ящика» с входами синхронизации CLK и сброса RESET и одним выходом RESULT, сигнализирующим о правильности работы калькулятора. Работу простейшего вычислителя можно описать следующим образом.

Конечный автомат CNTRL_FSM обращается к модулю памяти MEM по адресу 0, и происходит считывание операндов, кода операции, значения переноса, ожидаемого результата из памяти. Автомат обращается к АЛУ, загружая в него необходимую информацию, в котором выполняется определённая операция. Конечный автомат загружает значение ожидаемого результата в компаратор. Результат из АЛУ подаётся на компаратор, где происходит сравнение ожидаемого результата и значения, выработанного АЛУ. Если они совпали, сигнал RESULT на выходе компаратора принимает значение «1», и «0» - в обратном случае в случае ошибки. Далее автомат переходит в начальное состояние и происходит считывание операндов из памяти по адресу 2 и т.д. Вышеописанные действия выполняются до тех пор, пока не будут обработаны операнды, считанные из памяти по адресу 5. Обращения конечного автомата к модулям MEM, ALU и COMP тактированы синхросигналом CLK. Сигнал RESET служит для сброса конечного автомата в начальное

состояние, и считывание из памяти в этом случае начинается вновь по адресу 0.

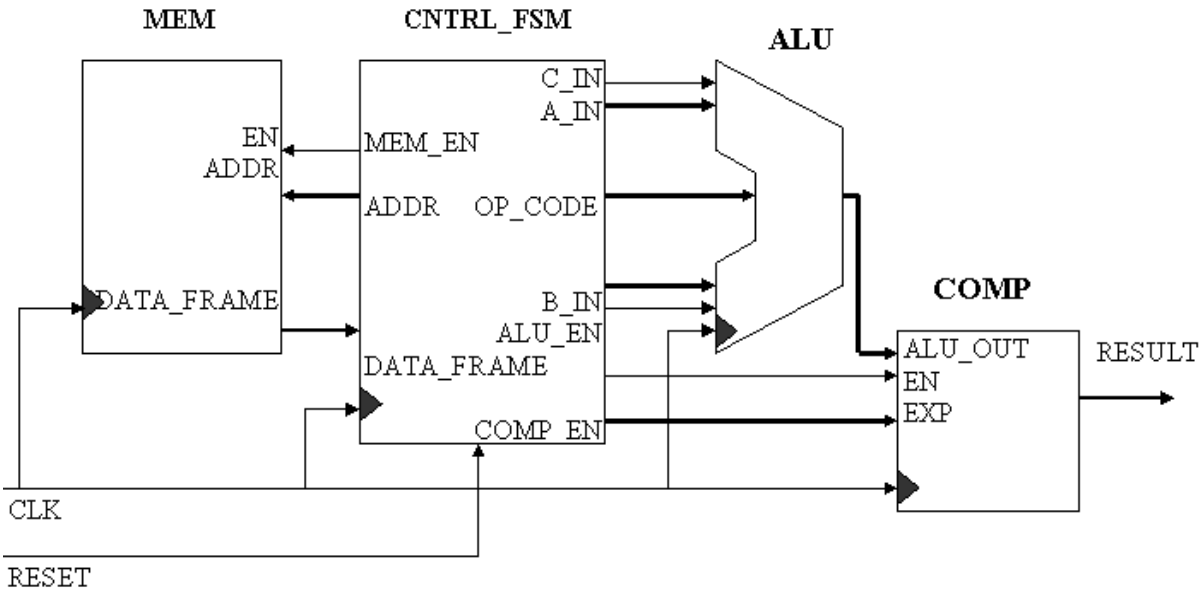

Рисунок 4.59 – Структурная схема устройства «Вычислитель».

Указания для выполнения задания 4.5.5: При выполнении этого задания рекомендуется следующая последовательность действий:

1) Создание нового проекта и включение в него файлов MEM.v, COMP.v, ALU.v и FSM.v с кодами устройств, разработанных ранее. Примите во внимание, чтобы обозначения портов ввода и вывода модуля верхнего уровня соответствовали рисунку 4.59. Для включения в проект копий составляющих модулей нижнего уровня используйте функции **Project – Add Copy of Source** и опцию **All** (или **Synthesis/Imp + Simulation**) для того, чтобы проводить синтез и моделирование подключенных модулей в данном проекте (см. рис. 4.60).

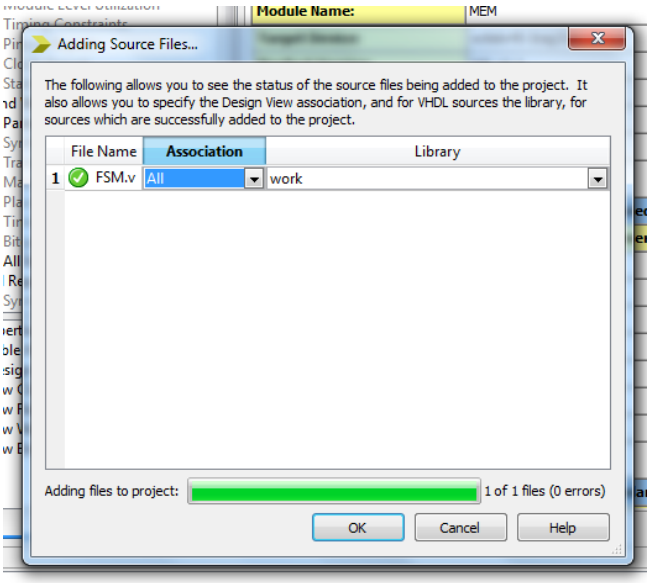

Рисунок 4.60 – Окно для дабвления копий уже созданных кодов в проект.

2) Модификация кода модуля MEM, для загрузки тестового содержимого ROM используя данные заданий 4.5.1 и 4.5.3.

Для проверки реакции на ошибку внесите изменение в значение одного или нескольких разрядов сигнала EXP, входящего в состав сигнала памяти MEM

DATA_FRAME. Например, так:

```
…
3'b011:DATA_FRAME = 17'b0001_0010_0_0010_1010;  // 9258 , а не 9262
…
```

3) Написание Verilog-кода, проверка его синтаксиса и анализ реализации на RTL уровне и в ПЛИС. Пример фрагмента кода, для включения в проект модулей нижнего уровня приведен на рис. 4.61.

```
`timescale 1ns / 1ps
`define SINTH 0
module SIMPLE_CALC(input CLK, RESET, output RESULT);

wire [3:0] A_IN, B_IN, OP_CODE, EXP_OUT, ALU_OUT ;
wire C_IN, ALU_EN, COMP_EN, MEM_EN ;
wire [2:0] ADDR_SIG ;
wire [16:0] DATA_FRAME ;

 ALU ALU_INST0 (
     .OP_CODE(OP_CODE),
     .A(A_IN),
     .B(B_IN),
     .C_IN(C_IN),
     .EN(ALU_EN),
     .Y(ALU_OUT),
    .CLK(CLK)
     );
 CNTRL_FSM FSM_INST0 (
     .CLK(CLK),
     .RESET(RESET),
     .DATA_FRAME(DATA_FRAME),
     .A_IN(A_IN),
     .B_IN(B_IN),
     .OP_CODE(OP_CODE),
     .EXP(EXP_OUT),
     .ADDR(ADDR_SIG),
     .COMP_EN(COMP_EN),
     .ALU_EN(ALU_EN),
    .MEM_EN(MEM_EN),
     .C_IN(C_IN)
     );
 MEM MEM_INST0 (
     .ADDR(ADDR_SIG),
     .DATA_FRAME(DATA_FRAME),
    .CLK(CLK),
    .EN(MEM_EN)
     );
… // здесь нужно включить процедуру generate
endmodule
```

Рисунок 4.61 – Фрагмент кода модуля верхнего уровня проекта «Вычислитель».

Для реализации одного из двух возможных сценариев использования кодов модуля компаратора COMP_BEH или COMP_RTL примените процедуру **generate** проверяющую значение флага '**SINTH**:

```
generate

  case (`SINTH )
```

```
    1'b0:
    begin : U
COMP_BEH COMP_INST_BEH (CLK, COMP_EN, EXP_OUT, ALU_OUT, RESULT );
    end
    default:
    begin: K
COMP_RTL COMP_INST_RTL (CLK, COMP_EN, EXP_OUT, ALU_OUT, RESULT );
    end
  endcase

endgenerate
```

После синтеза следует провести анализ отчета проекта о реализации проекта в конкретном исполнении в ПЛИС, а также просмотреть RTL–реализацию (функция **View RTL Schematic)** и вариант реализации из компонентов ПЛИС (**View Technology Schematic**).

Примеры соответствующих схем приведены на рисунках 4.61 и 4.62.

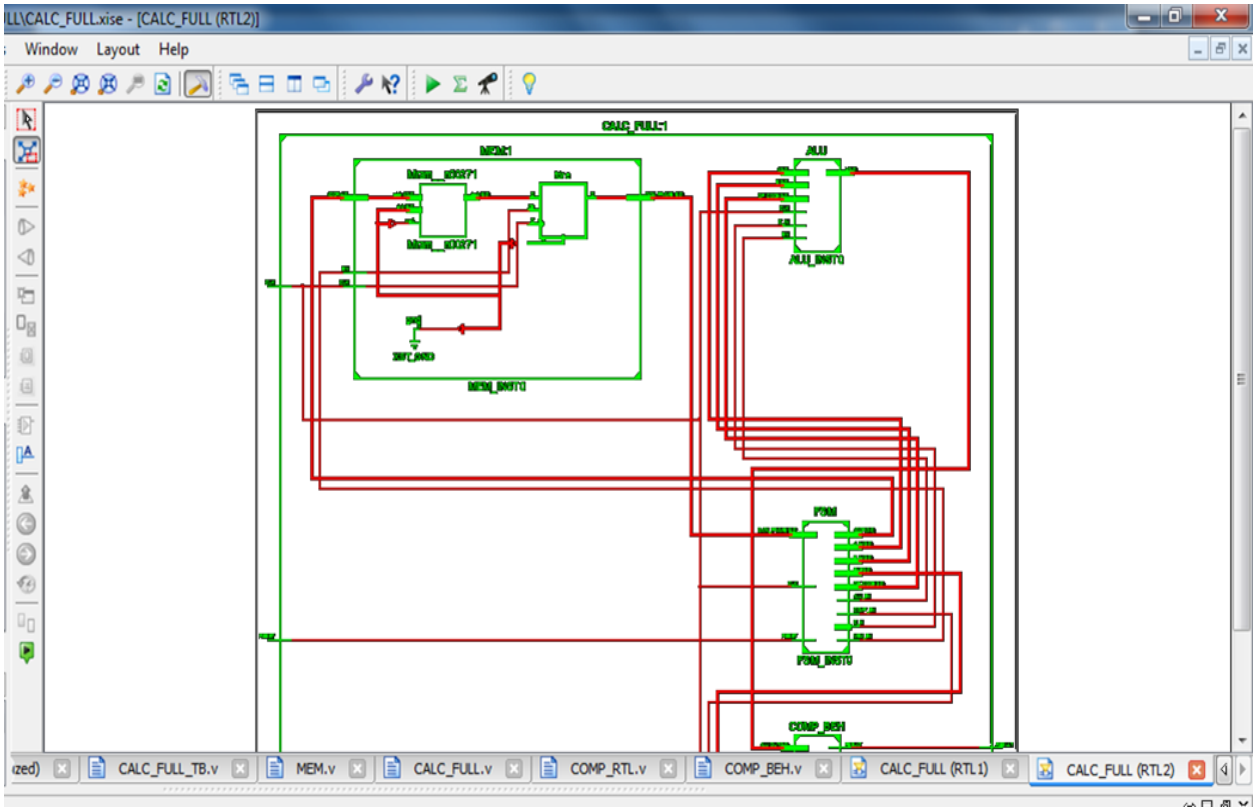

Рисунок 4.61 – RTL-схема устройства «Вычислитель».

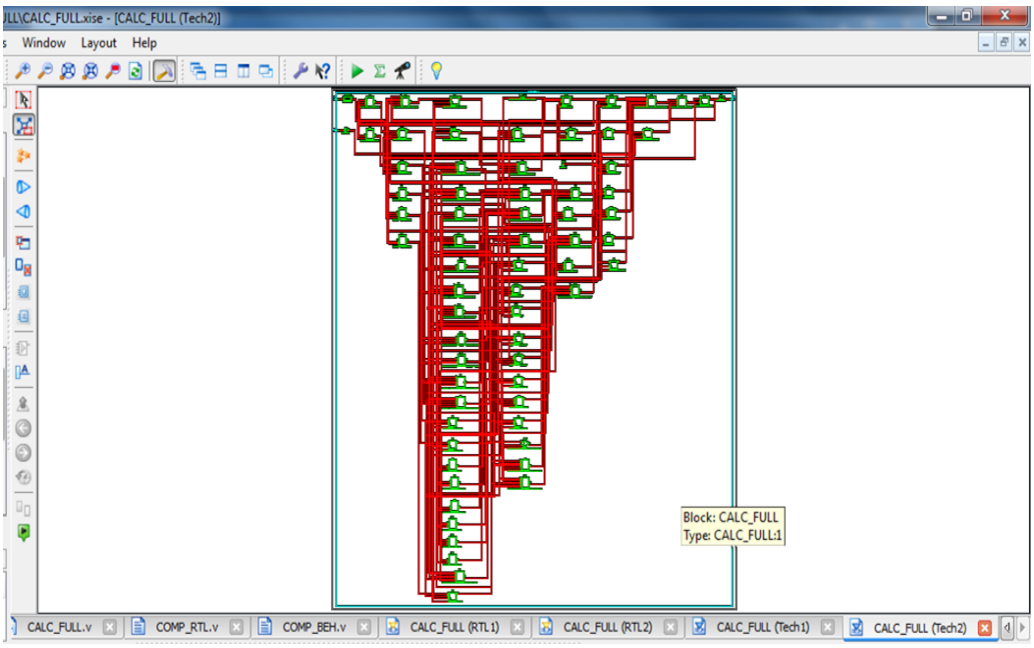

Рисунок 4.62 – Схема устройства «Вычислитель», реализованная на ресурсах ПЛИС.

4) Разработка кода испытательного стенда, моделирование и проверка правильности функционирования вычислителя. Пример кода для моделирования работы вычислителя приведен на рис. 4.63.

```
`timescale 1ns / 1ps

module SIMPLE_CALC_TB_v;

   reg CLK_TB = 1'b0;
   reg RESET_TB = 1'b0;

   wire RESULT_TB;

   SIMPLE_CALC uut (
         .CLK(CLK_TB),
         .RESET(RESET_TB),
         .RESULT(RESULT_TB)
   );

   initial
   forever #10 CLK_TB = ~ CLK_TB ;

   initial
   begin
         #10 RESET_TB = 1'b1 ;
         #25 RESET_TB = 1'b0 ;
         #650 RESET_TB = 1'b1 ;
         #25 RESET_TB = 1'b0 ;
   end

endmodule
```

Рисунок 4.63 – Пример кода испытательного стенда для вычислителя.

5) Выполнение моделирования и анализ результатов. Пример диаграммы с результатами моделирования работы вычислителя приведен на рис. 4.64

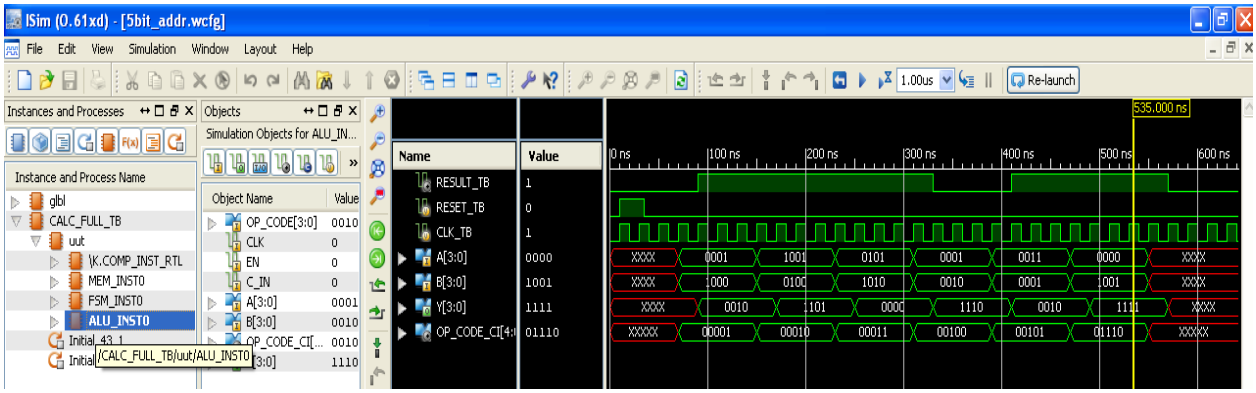

Рисунок 4.64 – Окно программы iSim с примером временной диаграммы моделирования работы устройства «Вычислитель».

Кроме отображения входных сигналов CLK и RESET и выходного сигнала RESULT для анализа работы модулей нижнего уровня в диаграмме также назначены для воспроизведения сигналы OP_CODE[3:0], C_IN, OP_CODE_C_I[4:0], A[3:0], B[3:0], Y[3:0]. Для этого нужно открыть модуль UUT на закладке **Instants and Process Name,** перенести нужные сигналы из окна **Objects** в окно **Name**. Из диаграммы моделирования видно:

- считывание данных происходит семь раз, после чего устройство переходит в режим ожидания, и после прихода сигнала RESET, цикл повторяется;

- при считывании седьмого слова данных значения сигналов не определены;

- сигнал RESULT принимает значения логического нуля в период обработки слова памяти, содержащегося по третьему адресу, что соответствует внесенной нами преднамеренно ошибке в значение сигнала EXP;

Таким образом, результаты моделирования позволяют сделать вывод о том, что проект выполнен успешно.

6) На заключительном этапе проекта следует получить данные отчетов о затраченных ресурсах при реализации в ПЛИС. Пример такого отчета приведен в табл. 4.16.

Таблица 4.16 – Данные из отчета о ресурсах ПЛИС FPGA 6SLX45CS.

Наименование узла ПЛИС	Использовано	Доступно	% исп.
количество задействованных ячеек Slices Registers	26	54576	~0
количество задействованных блоков Slice LUTs	39	27288	~0
количество полностью задействованных пар с триггерами FF (Slice LUTs-FF)	20	45	44
количество ячеек ввода-вывода (Bonded IOBs)	3	218	~1

7) Для упражнения в качестве дополнительного задания можно реализовать проект в ПЛИС FPGA 6SLX45CS в корпусе G324, используя демонстрационную плату ATLYS. Для этого в файле пользовательских ограничений следует назначить подключение сигнала CLK к выводу15, сигнал RESET – подключить к одной из командных кнопок, а выход RESULT – к одному из светодиодов. После этого следует выполнить процедуру имплементации проекта подобно тому, как это подробно описано в п 5.4.1.