

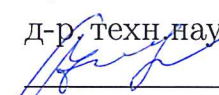
Министерство образования и науки Российской Федерации
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ (НИ ТГУ)
Факультет прикладной математики и кибернетики
Кафедра защиты информации и криптографии

УДК 004.315

ДОПУСТИТЬ К ЗАЩИТЕ В ГЭК

Руководитель ООП,

д-р техн. наук., профессор

 Г. П. Агибалов

“ 16 ” января 20 17 г.

ДИПЛОМНАЯ РАБОТА

АППАРАТНАЯ РЕАЛИЗАЦИЯ ОПЕРАЦИЙ ЯЗЫКА ЛЯПАС

по специальности 10.05.01 - Компьютерная безопасность

Висман Ян Александрович

Автор работы:

студент гр. 1115

 Висман Я. А.

Руководитель:

канд. техн. наук, доцент

 Треньяев В. Н.

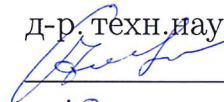
Томск 2017

Министерство образования и науки Российской Федерации
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ (НИ ТГУ)
Факультет прикладной математики и кибернетики
Кафедра защиты информации и криптографии

УТВЕРЖДАЮ

Руководитель ООП,

д-р. техн. наук, профессор

 Г. П. Агибалов

« 10 » марта 20 16 г.

ЗАДАНИЕ

по подготовке дипломной работы студенту Висману Яну Александровичу группы 1115.

1. Тема дипломной работы: «Аппаратная реализация операций языка ЛЯПАС».

2. Срок сдачи студентом выполненной дипломной работы:

а) на кафедре: 29 декабря 2016 года;

б) в ГЭК: 8 февраля 2017 года.

3. Цель работы: разработка арифметико-логического устройства (АЛУ), поддерживающего выполнение арифметических, базовых и дополнительных логических операций языка программирования ЛЯПАС.

Объект исследования: АЛУ ЛЯПАС-процессора, т.е. процессора предназначенного для выполнения программ на языке программирования ЛЯПАС.

Методы исследования: структурно-логическое проектирование АЛУ с помощью языка VHDL, функциональное моделирование и синтез АЛУ на ПЛИС с помощью САПР.

4. Основные разделы работы:

- изучение операций языка программирования ЛЯПАС (к 01.10.2016);
- разработка архитектуры АЛУ ЛЯПАС-процессора (к 01.11.2016);
- проектирование и моделирование операций АЛУ ЛЯПАС-процессора (к 29.12.2016);
- анализ полученных результатов и написание диплома (к 27.01.2017).

5. Работа выполняется по заданию кафедры защиты информации и криптографии Томского государственного университета.

6. Дата выдачи задания « 10 » марта 2016 г.

Руководитель дипломной работы,

канд. техн. наук, доцент

 / Тренькаев В. Н. /

Задание принял к исполнению,

студент гр. 1115

 / Висман Я. А. /

РЕФЕРАТ

Дипломная работа содержит 28 страниц, 1 рисунок, 9 литературных источников.

Ключевые слова: АППАРАТНАЯ РЕАЛИЗАЦИЯ, АЛУ, ЯЗЫК ПРОГРАММИРОВАНИЯ ЛЯПАС, ПЛИС, ЯЗЫК ОПИСАНИЯ АППАРАТУРЫ VHDL, ФУНКЦИОНАЛЬНОЕ МОДЕЛИРОВАНИЕ

Объект исследования: АЛУ ЛЯПАС-процессора.

Цель работы: разработка арифметико-логического устройства (АЛУ), поддерживающего выполнение арифметических, базовых и дополнительных логических операций языка программирования ЛЯПАС.

Метод исследования: структурно-логическое проектирование операций АЛУ с помощью языка VHDL, функциональное моделирование и синтез операций АЛУ на ПЛИС с помощью САПР Vivado и Xilinx ISE WebPACK.

Результаты работы: спецификация на языке VHDL операций АЛУ ЛЯПАС-процессора, моделирование с помощью САПР операций АЛУ ЛЯПАС-процессора, результаты синтеза на ПЛИС операций АЛУ ЛЯПАС-процессора.

ОГЛАВЛЕНИЕ

Введение	5
1 Язык программирования ЛЯПАС	6
2 ЛЯПАС-процессор	8
3 Арифметико-логическое устройство ЛЯПАС-процессора	9
3.1 Архитектура АЛУ	9
3.2 Операции АЛУ	10
3.3 Описание алгоритмов операций	11
4 Результаты реализации на ПЛИС логических и арифметических операций языка программирования ЛЯПАС	22
Заключение	27
Список литературы	28

ВВЕДЕНИЕ

В последнее время на кафедре ЗИиК НИ ТГУ разрабатывается криптографическое расширение русского языка программирования ЛЯПАС, на базе которого возможно реализовывать доверенное программно-аппаратное обеспечение для систем логического управления критически важными объектами, такими, как космические системы, энергетические установки, ядерное оружие, подводные лодки, беспилотники и т.п. При этом для эффективного использования ЛЯПАСа имеется необходимость в разработке спецпроцессора, поддерживающего данный язык программирования на аппаратном уровне.

Описание и общая архитектура спецпроцессора представлены в работах [1; 2]. Данная дипломная работа посвящена разработке архитектуры одного из важных блоков данного спецпроцессора – арифметико-логического устройства (АЛУ) и реализации арифметических и логических операций АЛУ на ПЛИС (программируемой логической интегральной схеме). При этом для проектирования, моделирования, верификации и синтеза АЛУ использовались САПР Vivado [3] и Xilinx ISE WebPACK [4].

В главе 1 приведен обзор основных элементов и особенностей языка программирования ЛЯПАС. Глава 2 посвящена описанию архитектуры ЛЯПАС-процессора. В главе 3 описывается арифметико-логическое устройство ЛЯПАС-процессора: архитектура АЛУ; набор арифметических и логических операций и способы их реализации. В главе 4 содержатся результаты синтеза операций АЛУ на ПЛИС, приводится сравнение аппаратной и программной реализаций этих операций. Приложение содержит разработанную спецификацию операций АЛУ на языке VHDL.

1 ЯЗЫК ПРОГРАММИРОВАНИЯ ЛЯПАС

Язык программирования ЛЯПАС был разработан в начале 1960-х годов в Томском государственном университете под руководством А.Д. Закревского и предназначен для решения математических, вычислительных и логико-комбинаторных задач [5].

Операнды языка представляют собой константы, переменные, комплексы и элементы комплексов. В памяти компьютера операнды представляются как логические векторы или массивы логических векторов. Они используются для представления булевых векторов, целых неотрицательных чисел, символов и их последовательностей.

Длина логического вектора фиксирована и составляет 32 разряда, компоненты нумеруются справа налево начиная с нуля. Целые числа не превосходят величину $2^{32} - 1$.

Константы подразделяются на натуральные, символьные и единичные. Натуральные константы представляют собой десятичные, шестнадцатеричные, восьмеричные и двоичные числа. Единичная константа является единичным вектором. Символьная константа представляет последовательность символов.

Переменные в ЛЯПАСе принимают значения булевых векторов длины 32. Особенностью языка является то, что количество доступных переменных равно 27. Первые 26 переменных обозначаются маленькими буквами латинского алфавита $a, b, \dots, z$. Переменная под номером 27 обозначается как Z и используется в специальных целях. Так же, есть виртуальная переменная, которая называется внутренней переменной и обозначается как τ . Она ассоциируется с регистром-аккумулятором данных, в который отправляются все результаты вычислений.

Комплексы это упорядоченные множества элементов, которые бывают двух видов - символьные и логические. В символьном комплексе элементами являются булевы векторы длины 8 (байты), в логическом - булевы векторы длины 32 (слова). Количество элементов комплекса называется его мощностью, максимально возможное количество элементов - емкостью.

Все операции языка ЛЯПАС разделяются на 2 больших класса: элементарные и дополнительные. Элементарные представляют собой фиксированный набор операций, дополнительные могут дополняться в процессе эксплуатации языка. Полный набор операций языка ЛЯПАС приведен в статье [1].

Элементарные операции по сложности их реализации можно условно разделить на простые и сложные. К сложным операциям относятся операции работы с комплексами, операции ввода-вывода и вызов программных функций. Простые элементарные операции подразделяются на 4 группы: операции переходов, операции передачи значения, логические и арифметические операции.

2 ЛЯПАС-ПРОЦЕССОР

ЛЯПАС–Процессор, описанный в статье [1], содержит следующие блоки: Устройство управления (CD - Control Device), Арифметико-логическое устройство (АЛУ), Память Инструкций (ИМ - Instruction Memory), Память Данных (DM - Data Memory), Регистр инструкций (IR - Instruction Register), Счётчик инструкций (IC - Instruction Counter), Дешифратор адреса (ADec) и Дешифратор операции (ODec).

Основные функции устройства управления (CD) - получение и анализ сигналов от других блоков процессора, формирование управляющих сигналов к другим исполнительным блокам, выбор очередной инструкции из памяти инструкций (ИМ) и определение ее адреса, определение адреса операнда в памяти данных (DM).

Арифметико-логическое устройство (АЛУ) - блок процессора выполняющий арифметические и логические операции над операндами, под контролем CD.

Память инструкций (ИМ) - блок предназначенный для запоминания набора команд в исполняемом коде некоторой программы на языке ЛЯПАС.

Память данных (DM) - блок используемый для хранения данных для программы на ЛЯПАСе, таких как единичные константы, переменные, комплексы для каждой подпрограммы и параметры этих комплексов (мощность, ёмкость и адрес).

Счетчик инструкций (IC) - регистр процессора, который содержит адрес выполняемой на данный момент команды или смещение в памяти, по которому хранится следующая выполняемая инструкция.

Регистр инструкций (IR) - регистр процессора, предназначенный для хранения кода команды на период времени, необходимый для её выполнения.

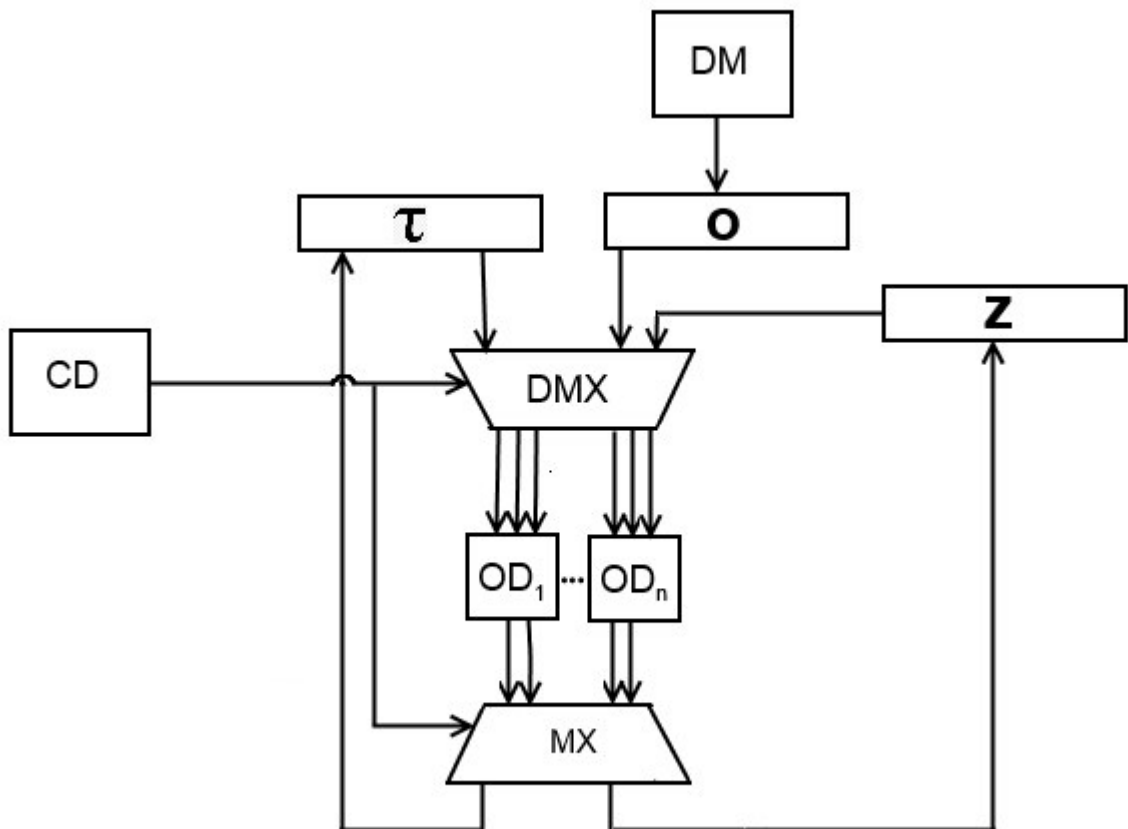
Для того, чтобы программа на ЛЯПАСе могла быть исполнена на ЛЯПАС-процессоре она должна быть сначала представлена в виде набора команд в исполняемом коде. Каждая команда состоит из кода операции, типа операции, адреса комплекса и переменной в памяти данных.

3 АРИФМЕТИКО-ЛОГИЧЕСКОЕ УСТРОЙСТВО ЛЯПАС-ПРОЦЕССОРА

3.1 АРХИТЕКТУРА АЛУ

Арифметико-логическое устройство – один из главных узлов любого спецпроцесса. АЛУ служит для вычисления результатов логических и арифметических операций над входными операндами. Операндами обычно являются значения регистров процессора, в нашем случае τ, o . Выбор операции выполняется на основании поступающего на вход АЛУ кода операции.

Рисунок 1 — Архитектура АЛУ



Архитектура АЛУ представлена на рис.1., где τ, o, Z - регистры общего назначения с длинами 32, 160 и 32 бит соответственно. Данные регистры предназначены для хранения значения внутренней (виртуальной) переменной τ , специальной переменной Z , которая в частности может служить для хранения остатка от деления, операнда o , который считывается из памяти данных (DM). Длина 32 для регистров τ и Z выбраны такими, т.к. переменные в ЛЯ-

ПАСе принимают значения булевых векторов длины 32. Длина регистра o выбрана таковой, исходя из минимальной длины операнда, необходимой для реализации одной из дополнительных логических операций, а именно перестановки. На вход этой операции поступает 32 целых числа, имеющих значение в промежутке $[0,31]$. Каждое из этих чисел можно представить пятью битами в двоичном представлении, поэтому $5 \text{ бит} * 32 = 160 \text{ бит}$ – минимально необходимая длина операнда o .

Алгоритм работы АЛУ следующий. Устройство управления CD выбирает очередную инструкцию из памяти инструкций IM и передает ее код в АЛУ. Демультимплексор DMX получает на вход код очередной операции и передает значение операндов операционному устройству OD, которое отвечает за выполнение требуемой операции. Имеется набор операционных устройств OD, каждое из которых выполняет одну арифметическую или логическую операцию. После вычисления операции мультимплексор MX, получая на вход результаты операции, помещает их в регистры τ и Z .

3.2 ОПЕРАЦИИ АЛУ

В данной работе реализован следующий набор операций.

1. Основные логические и арифметические операции:

- вычисление номера правой единицы;
- отрицание;
- вычисление веса булева вектора;
- дизъюнкция;
- конъюнкция;
- сложение по модулю 2;
- левый сдвиг;
- правый сдвиг;
- сложение по модулю 2^{32} ;
- вычитание по модулю 2^{32} ;

- умножение по модулю 2^{32} ;
- частное и остаток от деления;
- увеличение на 1;
- уменьшение на 1;

2. Дополнительные логические операции:

- перестановка: компоненты булева вектора τ переставляются в соответствии с их порядковыми номерами, указанными в элементах логического комплекса L ;
- проекция: выбирается часть булева вектора τ с компонентами, имеющими номера в интервале (ξ_1, ξ_2) ;
- вставка: часть булева вектора λ с компонентами, имеющими номера в интервале (ξ_2, ξ_3) , вставляются в τ перед ξ_1 -й компонентой (в отсутствие ξ_3 вставляется правая часть булева вектора λ длины ξ_2 , в отсутствие (ξ_2, ξ_3) - весь булев вектор λ);
- редукция: часть булева вектора τ с компонентами, имеющими номера в интервале (ξ_1, ξ_2) , вычёркивается из вектора;
- конкатенация: булев вектор λ присоединяется к τ ;
- левый и правый циклические сдвиги: часть булева вектора τ с компонентами, имеющими номера в интервале (ξ_2, ξ_3) , циклически сдвигается на ξ_1 бит влево или вправо (в отсутствие ξ_3 сдвигается правая часть булева вектора τ длины ξ_2 , в отсутствие (ξ_2, ξ_3) - весь булев вектор λ);

3.3 ОПИСАНИЕ АЛГОРИТМОВ ОПЕРАЦИЙ

Операции языка ЛЯПАС, которые описаны выше, были реализованы на языке VHDL [6; 7]. Далее приводятся алгоритмы операций.

1. OD_1 - вычисление номера правой единицы.

Вход: $\bar{\tau} = (\tau_0, \tau_1, \dots, \tau_{31})$;

$temp$ – переменная типа *bit_vector* с единицей в младшем бите
 $result = 0$; – переменная типа *integer* для хранения результата

Если $(\bar{\tau} \geq (temp \text{ сдвинуть влево на } 16))$ Тогда

$temp = temp \text{ сдвинуть влево на } 16$;

$result = result + 16$;

КонецЕсли;

Если $(\bar{\tau} \geq (temp \text{ сдвинуть влево на } 8))$ Тогда

$temp = temp \text{ сдвинуть влево на } 8$;

$result = result + 8$;

КонецЕсли;

Если $(\bar{\tau} \geq (temp \text{ сдвинуть влево на } 4))$ Тогда

$temp = temp \text{ сдвинуть влево на } 4$;

$result = result + 4$;

КонецЕсли;

Если $(\bar{\tau} \geq (temp \text{ сдвинуть влево на } 2))$ Тогда

$temp = temp \text{ сдвинуть влево на } 2$;

$result = result + 2$;

КонецЕсли;

Если $(\bar{\tau} \geq (temp \text{ сдвинуть влево на } 1))$ Тогда

$temp = temp \text{ сдвинуть влево на } 1$;

$result = result + 1$;

КонецЕсли;

Выход: $\bar{\tau} = result$ - номер правой единицы вектора $\bar{\tau}$.

2. OD_2 - отрицание.

Вход: $\bar{\tau} = (\tau_0, \tau_1, \dots, \tau_{31})$;

$\bar{\tau} = \text{not } \bar{\tau}$;

Выход: $\bar{\tau} = \neg\bar{\tau} = (\neg\tau_0, \neg\tau_1, \dots, \neg\tau_{31})$

3. OD_3 - вычисление веса булева вектора.

Вход: $\bar{\tau} = (\tau_0, \tau_1, \dots, \tau_{31})$;

$$\begin{aligned}\bar{\tau} &= (\bar{\tau} \text{ and } 0x55555555) + \\ &\quad + ((\bar{\tau} \text{ сдвинуть вправо на } 1) \text{ and } 0x55555555); \\ \bar{\tau} &= (\bar{\tau} \text{ and } 0x33333333) + \\ &\quad + ((\bar{\tau} \text{ сдвинуть вправо на } 2) \text{ and } 0x33333333); \\ \bar{\tau} &= (\bar{\tau} \text{ and } 0x0F0F0F0F) + \\ &\quad + ((\bar{\tau} \text{ сдвинуть вправо на } 4) \text{ and } 0x0F0F0F0F); \\ \bar{\tau} &= (\bar{\tau} \text{ and } 0x00FF00FF) + \\ &\quad + ((\bar{\tau} \text{ сдвинуть вправо на } 8) \text{ and } 0x00FF00FF); \\ \bar{\tau} &= (\bar{\tau} \text{ and } 0x0000FFFF) + \\ &\quad + ((\bar{\tau} \text{ сдвинуть вправо на } 16) \text{ and } 0x0000FFFF); \end{aligned}$$

Выход: $\bar{\tau}$ = вес вектора $\bar{\tau}$ в двоичном представлении;

Алгоритм работает следующим образом: разделяем исходное значение на пары смежных бит, вычисляем сумму в пределах каждой пары и помещаем её на место самой пары; поступим аналогичным образом с только что полученными значениями, увеличив длины битовых полей вдвое; будем повторять предыдущий шаг, пока не достигнем исходной битовой длины значения.

4. OD_4 - дизъюнкция.

Вход: $\bar{\tau} = (\tau_0, \tau_1, \dots, \tau_{31})$, $\bar{o} = (o_0, o_1, \dots, o_{31})$;

$$\bar{\tau} = \bar{\tau} \text{ or } \bar{o};$$

Выход: $\bar{\tau} = \bar{\tau} \vee \bar{o} = (\tau_0 \vee o_0, \tau_1 \vee o_1, \dots, \tau_{n-1} \vee o_{31})$.

5. OD_5 - конъюнкция.

Вход: $\bar{\tau} = (\tau_0, \tau_1, \dots, \tau_{31})$, $\bar{o} = (o_0, o_1, \dots, o_{31})$;

$$\bar{\tau} = \bar{\tau} \text{ and } \bar{o};$$

$$\text{Выход: } \bar{\tau} = \bar{\tau} \wedge \bar{o} = (\tau_0 \wedge o_0, \tau_1 \wedge o_1, \dots, \tau_{n-1} \wedge o_{n-1}).$$

6. OD_6 - сложение по модулю 2.

$$\text{Вход: } \bar{\tau} = (\tau_0, \tau_1, \dots, \tau_{31}), \bar{o} = (o_0, o_1, \dots, o_{31});$$

$$\bar{\tau} = \bar{\tau} \text{ xor } \bar{o};$$

$$\text{Выход: } \bar{\tau} = \bar{\tau} \oplus \bar{o} = (\tau_0 \oplus o_0, \tau_1 \oplus o_1, \dots, \tau_{31} \oplus o_{31}).$$

7. OD_7 - левый сдвиг.

$$\text{Вход: } \bar{\tau} = (\tau_0, \tau_1, \dots, \tau_{31}), \bar{o} = (o_0, o_1, \dots, o_4);$$

$$\bar{\tau} = \bar{\tau} \ll \bar{o};$$

$$\text{Выход: } \bar{\tau} = \text{вектор } \tau, \text{ сдвинутый на } \bar{o} \text{ бит влево.}$$

На вход в $\bar{o}$ подается число, на которое нужно сдвинуть вектор τ влево, представленное в двоичном виде.

8. OD_8 - правый сдвиг.

$$\text{Вход: } \bar{\tau} = (\tau_0, \tau_1, \dots, \tau_{31}), \bar{o} = (o_0, o_1, \dots, o_4);$$

$$\bar{\tau} = \bar{\tau} \gg \bar{o};$$

$$\text{Выход: } \bar{\tau} = \text{вектор } \tau, \text{ сдвинутый на } \bar{o} \text{ бит вправо.}$$

На вход в $\bar{o}$ подается число, на которое нужно сдвинуть вектор τ вправо, представленное в двоичном виде.

9. OD_9 - сложение по модулю 2^{32} .

$$\text{Вход: } \bar{\tau} = (\tau_0, \tau_1, \dots, \tau_{31}), \bar{o} = (o_0, o_1, \dots, o_{31});$$

$$\bar{\tau} = \bar{\tau} + \bar{o};$$

$$\text{Выход: } \bar{\tau} = \text{результат сложения } \bar{\tau} \text{ и } \bar{o}.$$

10. OD_{10} - вычитание по модулю 2^{32} .

$$\text{Вход: } \bar{\tau} = (\tau_0, \tau_1, \dots, \tau_{31}), \bar{o} = (o_0, o_1, \dots, o_{31});$$

$$\bar{\tau} = \bar{\tau} - \bar{o};$$

$$\text{Выход: } \bar{\tau} = \text{результат сложения } \bar{\tau} \text{ и } \bar{o}.$$

11. OD_{11} - умножение по модулю 2^{32} .

Вход: $\bar{\tau} = (\tau_0, \tau_1, \dots, \tau_{31})$, $\bar{o} = (o_0, o_1, \dots, o_{31})$;

result(63 downto 0); – вспомогательный сигнал размера 64 бит

result = $\bar{\tau} * \bar{o}$;

$\bar{\tau} = \text{result}(31 \text{ downto } 0)$; – младшая часть результата помещается в $\bar{\tau}$

$\bar{Z} = \text{result}(63 \text{ downto } 32)$; – старшая часть результата помещается в $\bar{Z}$

Выход: $\bar{\tau}$ = результат операции умножения $\bar{\tau}$ и $\bar{o}$, $\bar{Z}$ = переполнение возникшее при операции умножения $\bar{\tau}$ и $\bar{o}$.

12. OD_{12} - частное целочисленного деления.

Вход: $\bar{\tau} = (\tau_0, \tau_1, \dots, \tau_{31})$, $\bar{o} = (o_0, o_1, \dots, o_{31})$;

$\bar{\tau} = \bar{\tau} / \bar{o}$;

$\bar{Z} = \bar{\tau} \% \bar{o}$;

Выход: $\bar{\tau}$ = частное от деления $\bar{\tau}$ на $\bar{o}$, $\bar{Z}$ = остаток от деления $\bar{\tau}$ на $\bar{o}$.

13. OD_{13} - остаток целочисленного деления.

Вход: $\bar{\tau} = (\tau_0, \tau_1, \dots, \tau_{31})$, $\bar{o} = (o_0, o_1, \dots, o_{31})$;

$\bar{\tau} = \bar{\tau} \% \bar{o}$;

$\bar{Z} = \bar{\tau} / \bar{o}$;

Выход: $\bar{\tau}$ = остаток от деления $\bar{\tau}$ на $\bar{o}$, $\bar{Z}$ = частное от деления $\bar{\tau}$ на $\bar{o}$.

14. OD_{14} - увеличение на 1.

Вход: $\bar{\tau} = (\tau_0, \tau_1, \dots, \tau_{31})$.

Выход: $\bar{\tau} = \bar{\tau} + 1$.

15. OD_{15} - уменьшение на 1.

Вход: $\bar{\tau} = (\tau_0, \tau_1, \dots, \tau_{31})$.

Выход: $\bar{\tau} = \bar{\tau} - 1$.

16. OD_{16} - перестановка.

Вход: $\bar{\tau} = (\tau_0, \tau_1, \dots, \tau_{31})$, $\bar{o} = (o_0, o_1, \dots, o_{159})$;

$$\tau_0 = \tau_0(o_{4\dots 0});$$

$$\tau_1 = \tau_1(o_{9\dots 5});$$

...

$$\tau_{31} = \tau_{31}(o_{159\dots 155});$$

Выход: $\bar{\tau} = \bar{\tau}'$, где $\bar{\tau}'$ - вектор $\bar{\tau}$ в котором компоненты переставлены в соответствии с порядком, указанным в $\bar{o}$. Весь вектор $\bar{o}$ разделен на 32 части по 5 бит, где каждая часть является представлением целого числа в двоичном виде.

17. OD_{17} - проекция.

Вход: $\bar{\tau} = (\tau_0, \tau_1, \dots, \tau_{31})$, $\bar{o} = (o_0, o_1, \dots, o_9)$;

$$\bar{\tau} = \bar{\tau} \gg \bar{o}_{9\dots 5}.$$

$$\bar{\tau} = \text{len младших бит от } \bar{\tau}, \text{ где } \text{len} = \bar{o}_{4\dots 0} - \bar{o}_{9\dots 5}.$$

Выход: $\bar{\tau}$ = выбранные компоненты булева вектора $\bar{\tau}$ из требуемого интервала. Компоненты $\bar{o}_{9\dots 5}$ являются целым числом в двоичной системе счисления и представляют собой начало интервала, а компоненты $\bar{o}_{4\dots 0}$ его конец.

18. OD_{18} - вставка.

Вход: $\bar{\tau} = (\tau_0, \tau_1, \dots, \tau_{31})$, $\bar{o} = (o_0, o_1, \dots, o_{46})$;

– объявление вспомогательных переменных типа *bit_vector*

$$\text{tempo} = 0;$$

$$\text{temp} = t;$$

Случай, когда компонента ξ_3 отсутствует

– компоненты $\bar{o}_{9...5}$ являются целым числом записанным в двоичном коде и представляют собой длину правой части булева вектора $\bar{o}_{46...15}$, которую необходимо вставить в $\bar{\tau}$ перед ξ_1 -й компонентой (представляется целым числом $\bar{o}_{4...0}$ записанным в двоичном коде)

Если ($\bar{o}_{(14...9)}$ не определено и $\bar{o}_{(11...6)}$ содержит значение) Тогда

Часть переменной $\text{temp}(31...\bar{o}_{(4...0)})$ сдвигаем влево на количество бит, которое нужно вставить (указывается в $\bar{o}_{(9...5)}$);

– вычисляем часть вектора $\bar{o}$, которую нужно вставить в $\bar{\tau}$

$\text{tempo} = \text{tempo}$ сдвинуть вправо на $15 + 32 - \bar{o}_{(9...5)}$;

$\text{temp1}(\bar{o}_{9...5} - 1 \text{ до } 0) = \text{tempo}(\bar{o}_{(9...5)} - 1 \text{ до } 0)$;

– вставляем необходимую часть вектора $\bar{o}$ в $\bar{\tau}$

$\text{temp} = (\text{temp1}(31 \text{ до } 0) \text{ сдвинуть влево на } \bar{o}_{(5...0)} - 1) \text{ or } \text{temp}$;

КонецЕсли;

Случай, когда компоненты ξ_1, ξ_2, ξ_3 присутствуют

– компоненты $\bar{o}_{9...5}$ и $\bar{o}_{(14...9)}$ являются целыми числами записанными в двоичном коде и представляют собой интервал компонент вектора $\bar{o}_{46...15}$, который необходимо вставить в $\bar{\tau}$ перед ξ_1 -й компонентой (представляется целым числом $\bar{o}_{4...0}$ записанным в двоичном коде)

Если ($\bar{o}_{(14...9)}$ содержит значение и $\bar{o}_{(11...6)}$ содержит значение) Тогда

Часть переменной $\text{temp}(31...\bar{o}_{(4...0)})$ сдвигаем влево на количество бит, которое нужно вставить (разность между концом и началом интервала $\bar{o}_{(14...9)} - \bar{o}_{(9...5)}$);

– вычисляем часть вектора $\bar{o}$, которую нужно вставить в $\bar{\tau}$.

$\text{tempo} = \text{tempo}$ сдвинуть вправо на $15 + \bar{o}_{(9...5)}$;

$\text{temp1}((\bar{o}_{(14...9)} - \bar{o}_{(9...5)}) + 1 \text{ до } 0) = \text{tempo}((\bar{o}_{(14...9)} - \bar{o}_{(9...5)}) + 1 \text{ до } 0)$;

– вставляем необходимую часть вектора $\bar{o}$ в $\bar{\tau}$

temp = (temp1(31 до 0) сдвинуть влево на $\bar{o}_{(5...0)}$) or temp;

КонецЕсли;

Случай, когда компоненты ξ_2, ξ_3 отсутствуют

– компоненты $\bar{o}_{4...0}$ являются целым числом записанным в двоичном коде и представляет собой номер компоненты вектора $\bar{\tau}$, перед которым требуется вставить вектор $\bar{o}_{46...15}$

Если ($\bar{o}_{(14...9)}$ не определено и $\bar{o}_{(9...5)}$ не определено) Тогда
в temp оставляем компоненты temp($\bar{o}_{(4...0)}...0$);

– вставляем необходимую часть вектора $\bar{o}$ в $\bar{\tau}$

temp(31 до $\bar{o}_{(4...0)}$) := tempo(46- $\bar{o}_{(4...0)}$ до 15);

КонецЕсли;

Выход: $\bar{\tau} = \overline{temp}$ - результат операции вставки.

19. OD_{19} - редукция.

Вход: $\bar{\tau} = (\tau_0, \tau_1, \dots, \tau_{31})$, $\bar{o} = (o_0, o_1, \dots, o_{46})$;

– вспомогательные переменные типа *bit_vector*

temp, temp1, temp2, temp3;

one = "11111111111111111111111111111111";

– вычисляем необходимые векторные маски

temp2 = one сдвинуть вправо на $32-\bar{o}_{(4...0)}$;

temp3 = one сдвинуть влево на $\bar{o}_{(4...0)}-1$;

– получаем часть $\bar{\tau}$, с номерами компонент до вырезаемого интервала

temp2 = temp2 and $\bar{\tau}$;

– получаем часть $\bar{\tau}$ с номерами компонент после вырезаемого интервала

temp = $\bar{\tau}$ сдвинуть вправо на $\bar{o}_{(9...5)}-\bar{o}_{(4...0)}+1$;

temp = temp and temp3;

– совмещаем обе части $\bar{\tau}$, которые были получены выше

temp1 = temp2 or temp;

Выход: $\bar{\tau} = \overline{temp1}$ - результат операции редукции.

20. OD_{20} - конкатенация.

Вход: $\bar{\tau} = (\tau_0, \tau_1, \dots, \tau_{31})$, $\bar{o} = (o_0, o_1, \dots, o_{31})$;

– объявляем переменную типа *integer* для хранения длины вектора $\bar{\tau}$

len;

– вычисляем длину вектора $\bar{\tau}$

Для i от 31 до 0 Цикл

Выйти если $(\tau_i = '1'$ или $\tau_i = '0')$;

len := len+1;

КонецЦикла;

Выход: $\bar{\tau} = \bar{o}_{31-len\dots 0} \ \& \ \bar{\tau}_{len\dots 0}$ - конкатенация $\bar{\tau}$ и $\bar{o}$.

21. OD_{21} - циклический сдвиг влево.

Вход: $\bar{\tau} = (\tau_0, \tau_1, \dots, \tau_{31})$, $\bar{o} = (o_0, o_1, \dots, o_{46})$;

– объявление вспомогательных переменных типа *unsigned*

temp, temp2, temp3;

– случай, когда компоненты ξ_2, ξ_3 отсутствуют

– компоненты $\bar{o}_{4\dots 0}$ являются целым числом записанным в двоичном коде и представляет собой целое число на которое необходимо сдвинуть циклически влево вектор $\bar{\tau}$

Если $(\bar{o}_{(14\dots 9)})$ не определено и $\bar{o}_{(9\dots 5)}$ не определено) Тогда

$\bar{\tau} = \bar{\tau}$ циклически сдвинуть вправо на $\bar{o}_{(5\dots 0)}$;

КонецЕсли;

- случай, когда компоненты ξ_1, ξ_2, ξ_3 присутствуют
- компоненты $\bar{o}_{9...5}$ и $\bar{o}_{(14...9)}$ являются целыми числами записанными в двоичном коде и представляют собой интервал компонент вектора $\bar{\tau}$ который необходимо сдвинуть на ξ_1 бит влево (представляется целым числом $\bar{o}_{4...0}$ записанным в двоичном коде)

Если ($\bar{o}_{(14...9)}$ содержит значение и $\bar{o}_{(11...6)}$ содержит значение)
Тогда

```
temp = t;
temp3 = t;
temp2(31 до  $\bar{o}_{(14...10)}$ ) = temp(31 до  $\bar{o}_{(14...10)}$ );
temp2( $\bar{o}_{(9...5)}$  до 0) = temp( $\bar{o}_{(9...5)}$  до 0);
temp(31 до 0) = (others => '0');
temp( $\bar{o}_{(14...10)} - 1$  до  $\bar{o}_{(9...5)} + 1$ ) = temp3( $\bar{o}_{(14...10)} - 1$  до  $\bar{o}_{(9...5)}$ 
+ 1);
temp(31 до 0) = temp(31 до 0) сдвинуть вправо на  $\bar{o}_{(9...5)}$ ;
temp( $\bar{o}_{(14...10)} - \bar{o}_{(9...5)}$  до 0) = temp( $\bar{o}_{(14...10)} - \bar{o}_{(9...5)}$  до 0) сдвинуть циклически вправо на  $\bar{o}_{(4...0)}$ ;
temp(31 до 0) = temp(31 до 0) сдвинуть право на  $\bar{o}_{(9...5)}$ ;
 $\bar{\tau}$  = temp(31 до 0) or temp2(31 до 0);
```

КонецЕсли;

- случай, когда компонента ξ_3 отсутствует
- компоненты $\bar{o}_{9...5}$ являются целым числом записанным в двоичном коде и представляют собой длину правой части булева вектора $\bar{\tau}$, который необходимо циклически сдвинуть влево на ξ_1 (представляется целым числом $\bar{o}_{4...0}$ записанным в двоичном коде) бит

Если ($\bar{o}_{(14...9)}$ не определено и $\bar{o}_{(11...6)}$ содержит значение) Тогда

```
temp = t;
```

$\text{temp}(31 \text{ до } 31 - \bar{o}_{(9...5)}) = \text{temp}(31 \text{ до } 31 - \bar{o}_{(9...5)})$ циклически
сдвинуть вправо на $\bar{o}_{(4...0)}$;

$\text{tout}(31 \text{ до } 0) = \text{temp}(31 \text{ до } 0)$;

КонецЕсли;

Выход: в $\bar{\tau}$ помещается результат операции вставки.

22. OD_{22} - циклический сдвиг вправо.

Реализуется аналогично циклическому сдвигу влево, только все
сдвиги в операциях производятся в обратную сторону.

4 РЕЗУЛЬТАТЫ РЕАЛИЗАЦИИ НА ПЛИС ЛОГИЧЕСКИХ И АРИФМЕТИЧЕСКИХ ОПЕРАЦИЙ ЯЗЫКА ПРОГРАММИРОВАНИЯ ЛЯПАС

Логические и арифметические операции АЛУ, алгоритмы которых описаны выше, специфицированы на языке VHDL (см. приложение). Каждая операция промоделирована и синтезирована на ПЛИС XC7160КТ фирмы Xilinx семейства Kintex 7 с помощью САПР Vivado [8] и Xilinx ISE WebPACK [4]. Результаты синтеза приведены в Таблице 1 и Таблице 2.

Т а б л и ц а 1

Результаты синтеза основных арифметических и логических операций

Операция	Количество LUT, шт	Задержка, ns
Вычисление номера правой единицы	104	7,758
Отрицание	32	0,612
Вычисление веса вектора	59	4,083
Дизъюнкция	32	0,666
Конъюнкция	32	0,666
Сложение по модулю 2	32	0,666
Левый сдвиг	97	1,988
Правый сдвиг	101	1,960
Сложение по модулю 2^{32}	32	1,522
Вычитание по модулю 2^{32}	32	1,522
Частное целочисленного деления	3301	53,985
Остаток целочисленного деления	3301	53,985
Увеличение на 1	31	1,468
Уменьшение на 1	32	1,468
Умножение по модулю 2^{32}	DSP48E1s - 4	6,142

Результаты синтеза дополнительных логических операций

Операция	Количество LUT, шт.	Задержка, ns
Перестановка	320	1,576
Конкатенация	308	2,573
Редукция	301	4.373
Левый циклический сдвиг	1532	11,836
Правый циклический сдвиг	1501	12,319
Проекция	178	1,927
Вставка	LUTs - 1321 из них: LUT-FF pairs - 31 Registers - 31	6,783

Как видно из таблиц 1 и 2 большинство операций реализовано с использованием только такого базового логического блока ПЛИС типа FPGA как LUT. Типичный блок LUT (Look Up Table) строится на основе ПЗУ, в ячейках которого записан вектор значений булевой функции от небольшого количества переменных (не более шести). Кроме того, некоторые операции реализованы с использованием DSP48E1 – блока цифровой обработки сигналов. Каждый блок DSP48E1 состоит из 25х18 умножителя, 48-бит аккумулятора, предсумматора, регистров для конвейеризации и может рассматриваться как аппаратный умножитель, способный выполнять умножение с накоплением.

Также из таблиц 1 и 2 видно, что большинство операций имеет небольшую задержку (не более 10 нс), т.е. может быть реализовано частоте, сравнимой с системной тактовой частоте ПЛИС (600 МГц для Virtex-6, 450 МГц для Kintex-7, 325 МГц для Spartan-6). Однако есть операции, например нахождение частного от деления или циклический сдвиг, которые реализуются с большей задержкой (десятки нс), т.е. могут быть реализованы с рабочей частотой на порядок меньшей, чем системная тактовая частота самой ПЛИС.

Если сравнивать операции по количеству потребляемых ресурсов ПЛИС, то имеется большой разброс от малой ресурсоемкости, так сложение требует только десятки блоков LUT, до относительно большой ресурсоемкости, так

редукция и циклические сдвиги требует несколько тысяч блоков LUT. Тем не менее, ввиду досточно большого количества логических блоков у современных ПЛИС (для семейства Kintex-7 имеем сотни тысяч логических ячеек), разработанное АЛУ может быть реализовано на современных ПЛИС.

Также проведено сравнение быстродействия аппаратной реализации на ПЛИС операций языка ЛЯПАС с программной реализацией этих операций на языке СИ, т.е. на базе современного процессора общего назначения.

Т а б л и ц а 3

Сравнение задержек программной и аппаратной реализаций основных арифметических и логических операций

Операция	Процессор Intel Core i7-2630QM, ns	ПЛИС Xilinx XC7160KT семейства Kintex 7, ns
Вычисление номера правой единицы	24,219	7,758
Отрицание	3,632	0,612
Вычисление веса вектора	10,454	4,083
Дизъюнкция	3,452	0,666
Конъюнкция	3,453	0,666
Сложение по модулю 2	3,465	0,666
Левый сдвиг	3,281	1,988
Правый сдвиг	3,287	1,950
Сложение по модулю 2^{32}	3,459	1,522
Вычитание по модулю 2^{32}	3,468	1,522
Частное целочисленного деления	13,275	53,985
Остаток целочисленного деления	13,275	53,985
Увеличение на 1	3,617	1,468
Уменьшение на 1	3,606	1,468
Умножение по модулю 2^{32}	6,545	6,142

При этом использовался компьютер со следующими характеристиками: процессор Intel Core i7-2630QM с базовой тактовой частотой 2.00 ГГц, оперативная память размера 8 Гб и частотой 800 МГц, 64 разрядная операционная система Windows 10, система разработки Microsoft Visual Studio. Задержка реализации на Си операций языка ЛЯПАС замерялась следующим

образом: тысячу раз на двух случайных операндах производилось вычисление операции миллион раз, подсчитывалось среднее значение задержки.

Результаты сравнения задержек разных реализаций можно увидеть в таблицах 3 и 4.

Т а б л и ц а 4

Сравнение задержек программной и аппаратной реализаций дополнительных логических операций

Операция	Процессор Intel Core i7-2630QM, ns	ПЛИС Xilinx XC7160KT семейства Kintex 7, ns
Перестановка	90,417	1,576
Конкатенация	72,345	2,573
Редукция	52,239	4,373
Левый циклический сдвиг	70,82	11,836
Правый циклический сдвиг	72,153	12,319
Проекция	22,109	1,927
Вставка	76,370	6,783

Как видно из таблицы 3 в большинстве случаев реализация базовых арифметических и логических операций на ПЛИС превосходит или сравнима по быстродействию с реализацией на процессоре общего назначения. Есть две операции (частное целочисленного деления и остаток целочисленного деления), которые проигрывают по быстродействию.

Из таблицы 4 видно, что реализация дополнительных логических операций на ПЛИС на порядок превосходит по быстродействию реализацию на процессоре общего назначения.

Для операции деления и остатка целочисленного деления большая задержка скорее всего объясняется тем, что основу целочисленного деления составляет общепринятый способ деления с помощью операций вычитания или сложения и сдвига, т.е. не используется какой-либо способ для ускорения деления [9]:

- замена делителя обратной величиной, с последующим ее умножением на делимое;

- сокращение времени вычисления частичных остатков в традиционных методах деления за счет ускорения операций суммирования (вычитания);
- сокращение времени вычисления за счет уменьшения количества операций суммирования (вычитания) при расчете значения частного остатка;
- вычисление частного в избыточной системе счисления.

ЗАКЛЮЧЕНИЕ

В дипломной работе проведены исследования по выявлению эффективности аппаратной реализации операций языка ЛЯПАС. Создан проект АЛУ с поддержкой арифметических и логических операций языка ЛЯПАС. Операции АЛУ промоделированы и синтезированы на ПЛИС XC7160КТ фирмы Xilinx семейства Kintex 7 с помощью САПР Vivado и Xilinx ISE WebPACK. Также проведено сравнение быстродействия реализации операций языка ЛЯПАС на ПЛИС с реализацией этих операций на базе процессора общего назначения.

Большинство операций АЛУ при реализации на ПЛИС имеет задержку не более 10 нс, т.е. могут быть реализованы с частотой, сравнимой с системной тактовой частотой ПЛИС (600 МГц для Virtex-6, 450 МГц для Kintex-7, 325 МГц для Spartan-6). Имеется несколько медленных операций, которые реализуются с частотой на порядок меньшей, чем тактовая частота самой ПЛИС. Также большинство операций имеют малую ресурсоемкость. Показано, что большинство операций АЛУ при реализации на ПЛИС работают быстрее, чем при реализации на процессоре общего назначения, т.е. АЛУ ЛЯПАС-процессора "помещается" на современную ПЛИС и при этом "выигрывает" у процессора общего назначения.

СПИСОК ЛИТЕРАТУРЫ

1. *Азибалов Г. П., Липский В. Б., Панкратова И. А.* О криптографическом расширении и его реализации для русского языка программирования // Прикладная дискретная математика. — 2013. — 3 (21). — С. 93—104.
2. *Солдатов С. Е.* Процессор аппаратной поддержки первого уровня языка ЛЯПАС. — дипл. раб. / С. Е. Солдатов. — Томск, 2013.
3. Vivado Design Suite User Guide : Using the Vivado IDE. — руководство пользователя / Xilinx Inc., 2016.
4. ISE WebPACK Design Software. — URL:
<https://www.xilinx.com/products/design-tools/ise-design-suite/ise-Webpack.html> — Дата доступа: 24 января 2017. Электрон. дан.
5. *Закревский А. Д., Торопов Н. Р.* Система программирования ЛЯПАС-М. — Минск : Наука и техника, 1978.
6. *Бибило П. Н.* Основы языка VHDL. — Москва : СОЛОН-Р, 2002.
7. *Поляков А. К.* Языки VHDL и VERILOG в проектировании цифровой аппаратуры. — Москва : СОЛОН-Пресс, 2002.
8. Vivado Design Suite User Guide : Synthesis. — руководство пользователя / Xilinx Inc., 2016.
9. *Сорокин Н. Ю.* Модули деления на ПЛИС // Микропроцессорные и цифровые системы. — 2006. — 2 (12). — С. 151—159.

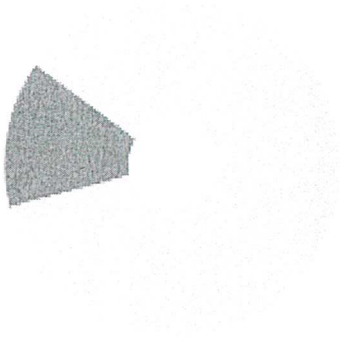
Уважаемый пользователь! Обращаем ваше внимание, что система «Антиплагиат» отвечает на вопрос, является ли тот или иной фрагмент текста заимствованным или нет. Ответ на вопрос, является ли заимствованный фрагмент именно плагиатом, а не законной цитатой, система оставляет на ваше усмотрение.

Отчет о проверке № 1

ФИО: Visman Yan
дата выгрузки: 03.02.2017 08:28:51
пользователь: visman.yan@yandex.ru / ID: 2624303
отчет предоставлен сервисом «Антиплагиат»
на сайте <http://www.antiplagiat.ru>

Информация о документе

№ документа: 2
Имя исходного файла: visman_diplom.pdf
Размер текста: 295 кБ
Тип документа: Не указано
Символов в тексте: 30538
Слов в тексте: 3686
Число предложений: 149



Информация об отчете

Дата: Отчет от 03.02.2017 08:28:51 - Последний готовый отчет
Комментарии: не указано
Оценка оригинальности: 86.2%
Заимствования: 13.8%
Цитирование: 0%

Оригинальность: 86.2%
Заимствования: 13.8%
Цитирование: 0%

Источники

Доля в тексте	Источник	Ссылка	Дата	Найдено в
10.28%	[1] Скачать полнотекстовую версию	http://journals.tsu.ru	26.11.2016	Модуль поиска Интернет
10.28%	[2] О КРИПТОГРАФИЧЕСКОМ РАСШИРЕНИИ И ЕГО РЕАЛИЗАЦИИ ДЛЯ РУССКОГО ЯЗЫКА ПРОГРАММИРОВАНИЯ	http://cyberleninka.ru	08.10.2015	Модуль поиска Интернет
1.65%	[3] ..lics_pdfN12_19.pdf	http://amursu.ru	19.09.2012	Модуль поиска Интернет

Подтверждаю :
Руководитель ААУ Треняев В.К.