МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Политехнический институт Факультет приборостроения, информационных технологий и электроники	Кафедра «Информационная безопасность систем и технологий»
Специальность 100502 –	Информационная безопасность телекоммуникационных систем
Специализация -	Разработка защищенных телекоммуникационных систем

ДИПЛОМНЫЙ ПРОЕКТ

на тему

УЗЕЛ ОБНАРУЖЕНИЯ ИСКАЖЕНИЯ ДАННЫХ В ПАМЯТИ КОМПЬЮТЕРА

Студент Никулин Никита Сергеевич

Руководитель О.В. Кулагин

Нормоконтролёр А.Г. Фатеев

Рецензент _____ С.Л. Агафонов

Работа допущена к защите (протокол заседания кафедры от 12.01.17 № 6)

Заведующий кафедрой С.Л. Зефиров

Работа защищена с отметкой _____(протокол заседания ГЭК от _____ № ____)

Секретарь ГЭК _____ А.Г. Фатеев

Пенза 2016

Проверил
12.01.17

РЕФЕРАТ

Дипломный проект содержит 82 с., 17 рис., 8 источников, 3 прил.

ХЭШ-ФУНКЦИЯ, ПРОГРАММИРУЕМЫЕ ЛОГИЧЕСКИЕ ИНТЕГРАЛЬНЫЕ СХЕМЫ, КОНТРОЛЬ ЦЕЛОСТНОСТИ, МОДЕЛИРОВАНИЕ УЗЛА, СТРУКТУРНАЯ СХЕМА УЗЛА

Объектом исследования является узел контроля целостности данных.

Целью работы является разработка узла контроля целостности данных на программируемой логической интегральной схеме (ПЛИС).

В процессе работы были рассмотрены хэш-функции, ГОСТ 34.11-2012, принцип контроля целостности посредством хэш-функции, принцип использования в изделии узла. Были разработаны структурная схема принципа использования в изделии узла и структурная схема узла.

Результатом работы является реализация узла в среде для проектирования «Quartus II» на языке описания аппаратуры VHDL.

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное бюджетное образовательное учреждение высшего образования

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

УДК 004.056

Политехнический институт

УТВЕРЖДАЮ

Факультет приборостроения, информационных технологий и электроники

Зав. кафедрой ИБСТ
к.т.н., доцент С.Л. Зефиров

Выпускающая кафедра:
Информационная безопасность систем и технологий

«___» _____ 2016 г.

Учебная группа 11ПТ1

ТЕХНИЧЕСКОЕ ЗАДАНИЕ

НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ

студента группы 11ПТ1 Никулина Никиты Сергеевича

Специальность 100502 – Информационная безопасность телекоммуникационных систем

Тема дипломного проекта:

Узел обнаружения искажения данных в памяти компьютера

Руководитель ВКР

к.т.н., Инженер I категории АО «ПНИЭИ» О.В. Кулагин

Нормоконтролер

к.т.н., доцент кафедры ИБСТ А.Г. Фатеев

1 Цели и задачи ВКР

Цель работы – разработка узла контроля целостности данных в памяти компьютера.

2 Тактико-технические требования к выполнению ВКР

Технические требования:

- узел контроля целостности данных должен быть реализован на ПЛИС серии FLEX10K фирмы Altera;
- узел контроля целостности данных должен быть реализован на основе алгоритма формирования хэш-функции по ГОСТ Р 34.11-2012;
- узел контроля целостности данных должен подключаться параллельно памяти компьютера по следующим информационным цепям:
  - 20 бит адреса;
  - 16 бит данных;
  - 1 бит сигнала чтения данных (RD) с активным низким уровнем.
- узел контроля целостности данных должен иметь входной сигнал с активным низким уровнем, переводящим все выходы узла, имеющие связь с контролируемой памятью, в высокоимпедансное состояние;
- работа узла контроля целостности данных начинается только при отсутствии высокоимпедансных состояний на его выходах и после подачи сигнала запуска однократным единичным импульсом (оба сигнала формируются внешним устройством);
- для приема исходных данных и считывания результата узел контроля целостности данных должен иметь отдельный интерфейс, включающий:
  - 16 бит данных;
  - шину адресов;
  - сигнал записи WR (активный низкий уровень);
  - сигнал чтения RD (активный низкий уровень).

В ВКР должны быть рассмотрены следующие вопросы:

– хэш-функция, ее использование и способ получения;

– необходимость контроля целостности данных в компьютерной памяти;

– разработка схемы структурной узла контроля целостности информации;

– проверка работоспособности посредством моделирования.

3 Требования к разрабатываемой документации

Отчет по ВКР выполняется в соответствии с требованиями ГОСТ 7.32.

Перечень и содержание графической части ВКР:

– принцип формирования хэш-функции согласно ГОСТ Р 34.11-2012 – 1л., ф. А1 (плакат);

– сравнение алгоритмов криптографических преобразований ГОСТ Р 34.11-2012 и AES – 1л., ф. А1 (плакат);

– принцип использования в изделии узла контроля целостности информации –1л., ф. А1 (плакат);

– схема структурная разработанного узла контроля целостности информации – 1л., ф. А1;

– временные диаграммы моделирования работы узла контроля целостности информации – 2л., ф. А1 (плакат).

Электронная копия отчета и графической части ВКР должна быть представлена на магнитном или оптическом носителе.

4 Сроки выполнения ВКР

Тема утверждена приказом ректора ПГУ № ____ от «___» _____ 2016 г.

Дата выдачи задания «__» _____ 2016 г.

Время дипломного проектирования с _____ 2016 г. по _____ 2016 г.

Задание к исполнению принял «__ » _____ 2016 г.

Исполнитель ВКР _____ Никулин Н.С.

СОДЕРЖАНИЕ

Введение 9
1 Хэш-функции и принцип контроля целостности посредством хэш-функций 10
1.1 Назначение хэш-функций 10
1.2 ГОСТ Р 34.11-2012 13
1.3 Принцип контроля целостности посредством хэш-функции 18
2 Принцип использования в изделии узла 20
2.1 Особенности разработки защищенных информационных систем 20
2.2 Назначение узла и принципы его работы в изделии 21
3 Разработка узла контроля целостности 23
3.1 Использование ПЛИС серии FLEX10K фирмы Altera 23
3.2 Разработка структурной схемы узла 23
3.3 Моделирование узла 24
4 Описание модулей 26
5 Результаты моделирования 49
Заключение 59
Список использованных источников 60
Приложение А Схема структурная разработанного узла контроля целостности информации 61
Приложение Б Описание узла контроля целостности на языке VHDL 63
Приложение В Результаты проверки в системе «Антиплагиат» 81

ОПРЕДЕЛЕНИЯ

Хеш-функция – функция, отображающая строки бит в строки бит фиксированной длины и удовлетворяющая следующим свойствам:

– по данному значению функции сложно вычислить исходные данные, отображаемые в это значение;

– для заданных исходных данных сложно вычислить другие исходные данные, отображаемые в то же значение функции;

– сложно вычислить какую-либо пару исходных данных, отображаемых в одно и то же значение [2].

Электронная цифровая подпись – строка бит, полученная в результате процесса формирования подписи [2].

Целостность информации – способность средства вычислительной техники или автоматизированной системы обеспечивать неизменность информации в условиях случайного и (или) преднамеренного искажения (разрушения) [7].

Данные – информация, представленная в виде, удобном для передачи, интерпретации либо обработки компьютером [8].

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

ИС – интегральная схема

ОЗУ – оперативное запоминающее устройство

ПЗУ – постоянное запоминающее устройство

ПЛИС – программируемая логическая интегральная схема

САПР – система автоматизированного проектирования

ЦП – центральный процессор

ЭЦП – электронная цифровая подпись

ВВЕДЕНИЕ

К задаче контроля целостности необходимо подходить с нескольких позиций. Для начала нужно дать ответ на вопрос, для чего реализуется контроль целостности. Так как при корректной реализации разграничительной политики доступа к ресурсам их целостность не может быть несанкционированно нарушена. Из этого можно сделать вывод, что целостность ресурсов нужно контролировать тогда, когда не представляется возможным осуществить корректное разграничение доступа или когда есть сомнения в надежности разграничительной политики. Это оправданное сомнение, так как средство защиты информации от несанкционированного доступа, которое обеспечивало бы 100% защиту, реализовать невозможно даже теоретически. Также необходимо понимать, что контроль целостности очень ресурсоемкий механизм, поэтому в реальных условиях допустим контроль весьма ограниченных по объему объектов.

Принципиальная особенность защиты информации на прикладном уровне заключается в том, что реализация любой разграничительной политики доступа к ресурсам на этом уровне не допустима, поскольку без проблем преодолевается злоумышленником. На данном уровне решаются задачи контроля целостности, которые основаны на реализации функций сравнения с эталоном. Важной особенностью является то, что с эталоном могут сравниваться уже произошедшие события. То есть задача защиты на прикладном уровне заключается не в предотвращении несанкционированного действия, а в выявлении и в фиксировании факта того, что несанкционированное действие произошло.

Достоинством защиты на прикладном уровне является тот факт, что произошедшее несанкционированное действие может быть зарегистрировано практически всегда, вне зависимости от причин его возникновения.

1 Хэш-функции и принцип контроля целостности посредством хэш-функций

1.1 Назначение хэш-функции

Криптографические методы не только позволяют обеспечивать конфиденциальность, но и контролировать целостность передаваемых данных. Контроль целостности, как правило, осуществляется методом вычисления контрольной суммы данных. В настоящее разработано большое число алгоритмов, вычисляющих контрольные суммы передаваемых данных. Дело в том, что в большинстве случаев достаточно простой контрольной суммы, например, в тех случаях, когда объем информации не известен заранее или имеет значение скорость обработки.

Но при использовании простой контрольной суммы можно подобрать несколько массивов данных, у которых будет получаться одинаковая контрольная сумма. Криптостойкие контрольные суммы получаются в случае применения к исходному тексту хэш-функции [1].

Хэш-функции имеют большое значение в криптографии. Они используются в таких приложениях как, электронная подпись, аутентификация и обеспечение целостности данных.

Хэш-функции предназначены для «сжатия» набора данных или произвольного сообщения, записанных в некоторую битовую комбинацию фиксированной длины. Хэш-функции применяются при тестировании логических устройств, при проведении статистических экспериментов, при построении алгоритмов быстрого поиска и проверки целостности записей. Основным из требований к хеш-функциям является равномерность распределения их значений при случайном выборе значений аргумента.

Коллизией хэш-функции H(X) называется ситуация, когда существуют два различных текста X1 и X2, таких что H(X1) равно H(X2).

Основное требование, предъявляемое криптографическими приложениями к хэш-функциям, заключается в отсутствии эффективных алгоритмов поиска коллизий.

Криптографической хэш-функцией является хэш-функция, являющаяся криптостойкой, то есть удовлетворяющая ряду требований.

В криптографии хэш-функции применяются для решения задач:

– построения систем контроля целостности данных при их передаче или хранении;

– аутентификация источника данных.

Для решения первой задачи для каждого набора данных вычисляется значение хэш-функции (имитовставка), которое хранится и передается вместе с данными. После получения данных вычисляется значение свертки и сравнивается с имеющимся контрольным значением. При несовпадении значений можно сделать вывод, что данные были изменены.

Хэш-функция, которая служит для выработки имитовставки, должна позволять осуществлять обнаружение как случайных ошибок в наборах данных, возникающих при передаче и хранении, так и сигнализировать об активных атаках злоумышленника, пытающегося осуществить искажение информации. Чтобы злоумышленник не смог самостоятельно вычислить контрольное значение свертки и осуществить искажение информации, хэш-функция должна зависеть от секретного параметра – ключа пользователя. Ключ должен быть известен передающей и проверяющей сторонам. Такие хэш-функции называются ключевыми [1].

Имитовставки, которые формируются при помощи ключевых хэш-функций, не должны позволять злоумышленнику искажать информацию.

При решении второй задачи стороны не доверяют друг другу. В связи с этим подход, при котором стороны обладают одним секретным ключом, недопустим. В таком случае применяют схемы цифровой подписи, которые позволяют осуществлять аутентификацию источника данных. Сообщение, перед тем как быть подписанным личной подписью, основанной на секретном ключе, «сжимается» при помощи хэш-функции, выполняющей функцию кода обнаружения ошибок. В этом случае хэш-функция не зависит от секретного ключа, известна всем и может быть фиксированной.

Основными требованиями к ней являются гарантии того, что подписанный документ невозможно подменить и что два различных сообщения с одинаковым значением хэш-функции невозможно подобрать.

Для решения этих задач используются два разных типа криптографических хэш-функций – ключевые и бесключевые.

К ключевым хэш-функциям предъявляются следующие требования:

– невозможность фабрикации;

– невозможность модификации.

Первое требование означает высокую сложность подбора сообщения с верным значением свертки. Второе требование подразумевает высокую сложность подбора для сообщения с известным значением свертки другого сообщения с верным значением свертки.

Эти свойства можно объединить в одно свойство – свойство вычислительной устойчивости. Это требование означает высокую сложность подбора для множества сообщений $\{x_1, \dots, x_i\}$ с известными значениями сверток еще одного сообщения x, x не равно x_i, i равно $1, \dots, t$ с правильным значением свертки.

«Высокая сложность» означает такую вычислительную сложность задачи, при которой ее решение с использованием ВТ за реальное время не является возможным.

Ключевые хэш-функции применяются в ситуациях, когда обе стороны доверяют друг другу и имеют общий секретный ключ. В этих условиях не требуется, чтобы система обеспечивала защиту при отказе получателя от факта получения сообщения или же его подмены.

Атаки на ключевые хэш-функции заключаются в передаче сфабрикованных сообщений в пустом канале и в подмене передаваемых сообщений с целью навязывания неверных сообщений.

Из свойства вычислительной устойчивости следует невозможность определения ключа, который используется хэш-функцией, так как знание

ключа предоставляет возможность вычислять значение свертки для любых наборов данных.

1.2 ГОСТ Р 34.11-2012

ГОСТ Р 34.11-2012 Информационная технология. Криптографическая защита информации. Функция хеширования [2] – действующий российский криптографический стандарт, определяющий алгоритм и процедуру вычисления хеш-функции, которая называется «Стрибог», в честь славянского божества. Разработан Центром защиты информации и специальной связи ФСБ России с участием ОАО «ИнфоТеКС» и введенный в действие 1 января 2013 года [5].

Стандарт содержит описание алгоритма и процедуры вычисления хеш-функции для любой последовательности двоичных символов, которые применяются в криптографических методах защиты информации, в том числе в процессах формирования и проверки электронной цифровой подписи [2].

Необходимость разработки настоящего стандарта вызвана потребностью в создании хеш-функции, соответствующей современным требованиям к криптографической стойкости.

Стандарт определяет алгоритм и процедуру вычисления хеш-функции для любой последовательности двоичных символов, которые применяются в криптографических методах обработки и защиты информации, в том числе для реализации процедур обеспечения целостности, аутентичности, электронной цифровой подписи (ЭЦП) при передаче, обработке и хранении информации в автоматизированных системах [2].

Определенная в настоящем стандарте функция хеширования используется при реализации систем электронной цифровой подписи на базе асимметричного криптографического алгоритма.

Стандарт рекомендуется использовать при создании, эксплуатации и модернизации систем обработки информации различного назначения.

Стандарт обрабатывает блоки сообщений длиной 512 бит и вычисляет 256- или 512-битные хэш-значения. *l*-битное сообщение дополняется до

размера, кратного 512 битам. Присоединяется единичный бит к концу сообщения, после него следует 512 – 1 – (l mod 512) нулевых бит.

$$m = m_t||m_{t-1}|| \dots || m_1, \quad (1)$$

$$\sum = N = IV = 0^{512}, \quad (2)$$

где m – сообщение, состоящее из t блоков после дополнения, IV – предопределенное начальное значение, t – число кратное 512, $\boxplus$ обозначает операцию сложения в кольце $Z_{2^{512}}$, $g_N(h,m)$ – функция сжатия алгоритма, которая основана на 512-битном блочном шифре. Схема вычисления хэш-функции H(M) представлена на рисунке 1.

Функция сжатия $g_N(h,m)$ вычисляется по следующей формуле:

$$g_N(h,m) = E(L \circ P \circ S(h \oplus N), m) \oplus h \oplus m, \quad (3)$$

где L – линейное преобразование, P – преобразование перестановки, S – преобразование подстановки, E – функция блочного шифра.

$$E(K,m) = X[K_{13} \circ \prod_{i=1}^{12}(L \circ P \circ S \circ X[K_i](m)), \quad (4)$$

где K – раундовый ключ, который генерируется для каждого из 13 раундов генерируется по формуле:

$$K_i = L \circ P \circ S(K_{i-1} \oplus C_{i-1}), K_1 = K, i \in \{2, \dots, 13\}, \quad (5)$$

где C – итерационная константа, значения которой указаны в ГОСТ 34.11-2012.

Схема вычисления функции сжатия $g_N(h,m)$ представлена на рисунке 2.

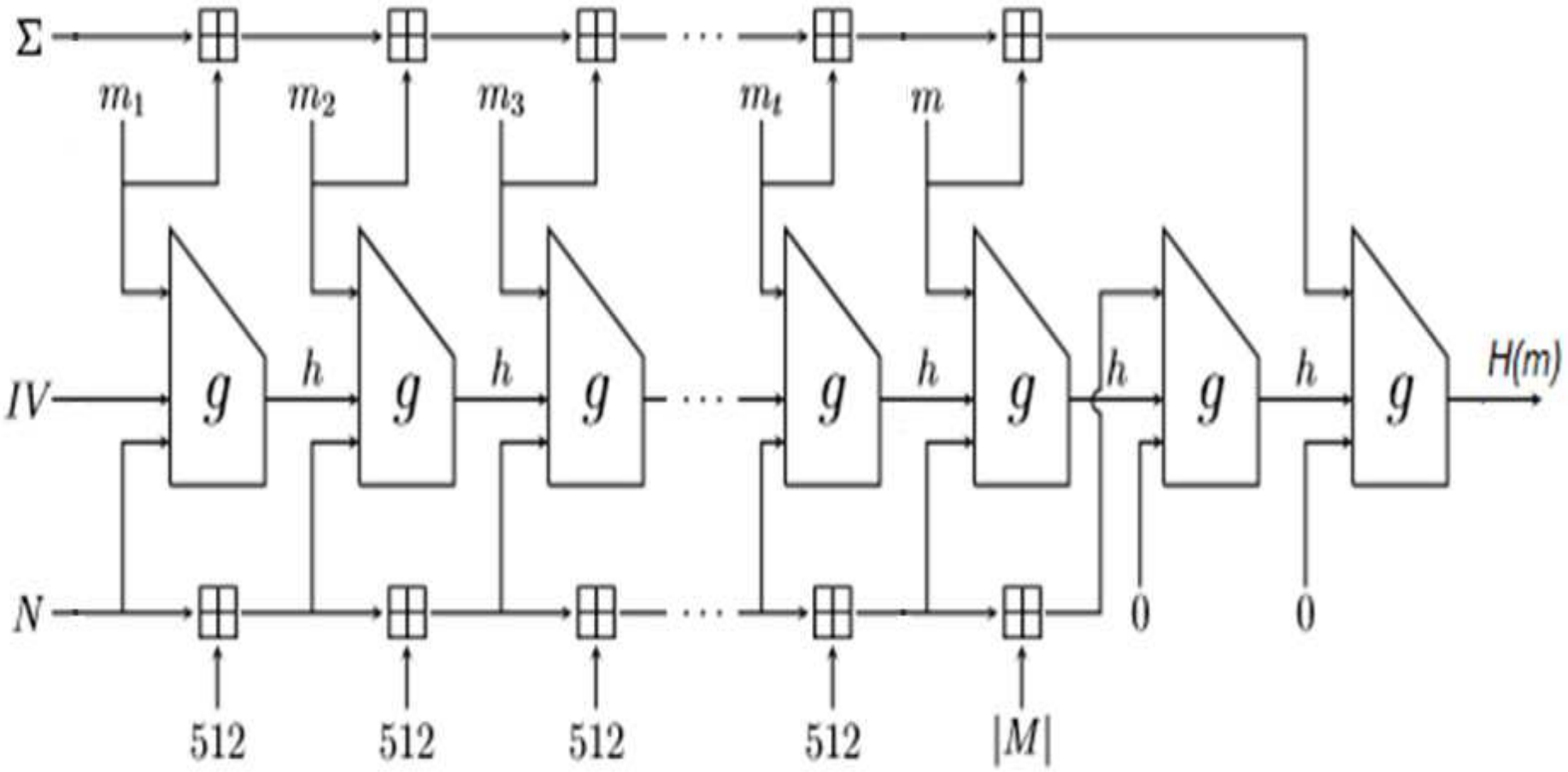

Рисунок 1 – Принцип формирования хэш-функции ГОСТ Р 34.11-2011

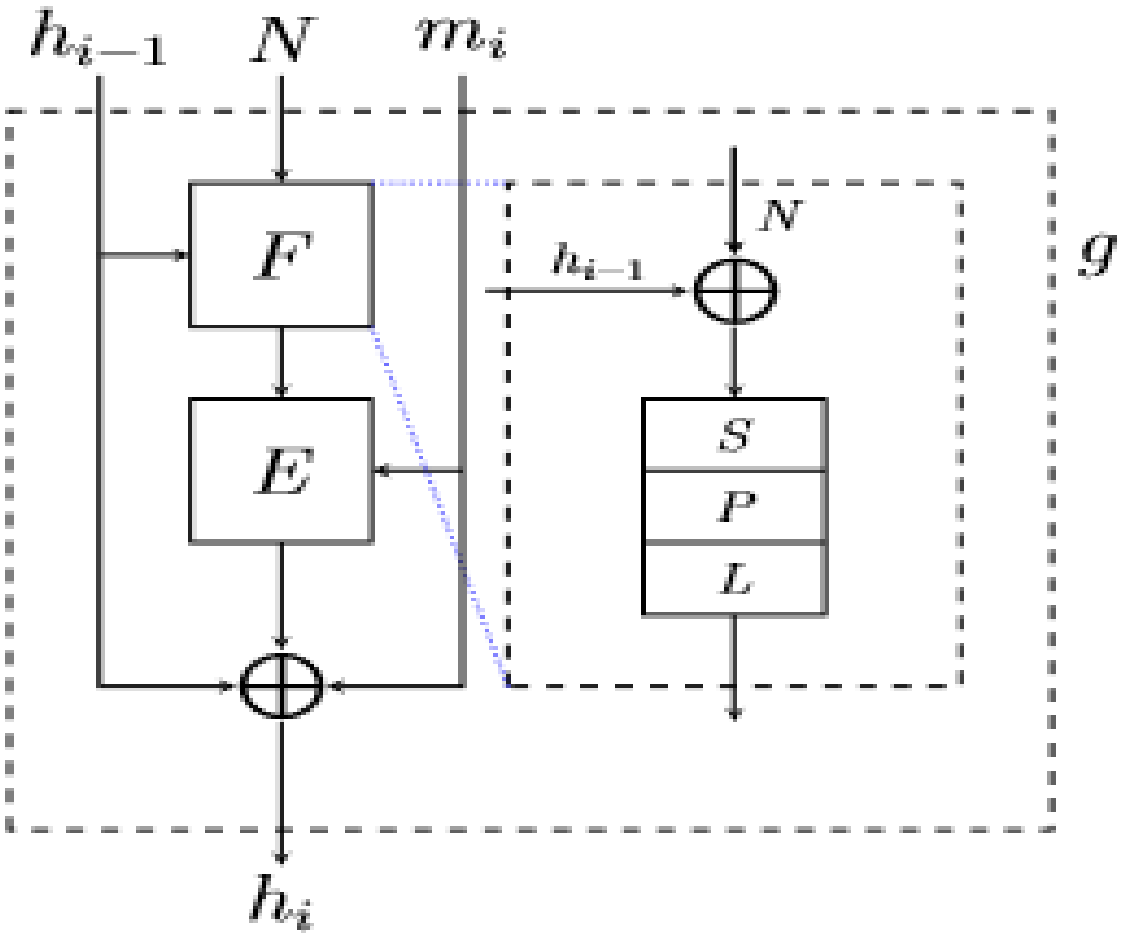

Рисунок 2 – Схема вычисления функции сжатия

На рисунке 3 приведено сравнение алгоритмов S-преобразований хэш-функции «Стрибог» и AES, взятое из статьи [3], на котором видно преимущество хэш-функции «Стрибог» перед AES.

Свойства	Стрибог	AES
Векторная булева функция		
Сбалансированность	Да	Да
Нелинейность	100	112
Абсолютный показатель	96	32
Сумма квадратов	258688	133120
Критерий распространения	0	0
Корреляционная стойкость	0	0
Минимум алгебраической степени	7	7
Устойчивость	0	0
Строгий лавинный критерий	Нет	Нет
Подстановка		
Биекция	Да	Да
Максимум дифференциации	8	4
Максимум аппроксимации	28	16
Структура циклов	252:243, 46:13	43:27, 242:87, 99:59, 124:81, 143:2
Алгебраическая стойкость	3 (441)	2 (39)

Рисунок 3 – Сравнение алгоритмов S-преобразований хэш-функции «Стрибог» и AES

1.3 Принцип контроля целостности посредством хэш-функции

Под целостностью данных подразумевается состояние информации, при котором отсутствует любое её изменение, либо изменение осуществляется только преднамеренно субъектами, имеющими на него право.

Контроль целостности данных посредством хэш-функции можно осуществлять несколькими способами:

– для обеспечения целостности хранимых данных можно добавить к ним значение хэш-функции, зависящей от ключа, известного только владельцу информации. Данный способ решает проблему аутентификации источника данных;

– для обеспечения целостности передаваемого сообщения можно использовать хэш-функцию, зависящую от секретного ключа, известного отправителю и получателю. Аутентификация источника гарантируется тем, что секретный ключ известен только двум сторонам.

В качестве контролируемой информации в рамках дипломной работы будут взяты программы, обрабатывающие конфиденциальную информацию. Программа – объект неизменяемый, и по этой причине для каждой защищаемой программы может быть заранее подсчитана хэш-функция. Помещаемую в ПЗУ программу контролировать не нужно в том случае, если запись в это ПЗУ исключено и если эта программа исполняется непосредственно из него без загрузки в другую память. Однако, зачастую оказывается удобным использование многофункциональных устройств, способных работать с разными алгоритмами, обрабатывающими конфиденциальную информацию. Это предполагает хранение программ, реализующих эти алгоритмы, в некоторой постоянной или перезаписываемой памяти и поочередную загрузку этих программ в ОЗУ исполнительного устройства.

В этом случае и возникает необходимость периодического контроля целостности этой программы (периодичность задается регламентом работы

устройства). Это и осуществляет разработанный в данной работе узел. Для того чтобы в алгоритм подсчета хэш-функции не были внесены искажения, он реализован аппаратно на ПЛИС и работает под управлением внешнего устройства, например, центрального процессора. Последний включает узел подсчета хэш-функции, загружает в него необходимые исходные данные и принимает от него вычисленный результат.

2 Принципы использования узла

2.1 Особенности разработки защищенных информационных систем

Разработка аппаратуры, обрабатывающей закрытую информацию, значительно сложнее разработки аналогичной аппаратуры, предназначенной только для открытой информации. Если в последнем случае достаточно правильно подобрать элементную базу и средства программирования, собрать и настроить аппаратуру, то в первом случае весь цикл разработки будет сопровождать контролирующая организация. Она предъявит свои требования к аппаратным и программным решениям в этой аппаратуре, а потом проведет исследования по оценке уровня и информативности побочных электромагнитных излучений и наводок, которые будет излучать вновь разработанная аппаратура. Для сокращения объема работ контролирующей организации выгодно широкое использование в новых разработках отработанных и хорошо зарекомендовавших себя заделов в программной и аппаратной частях.

Однако это требования вступает в противоречие с нынешним состоянием дел в современной элементной базе. Цифровая элементная база представлена двумя основными классами интегральных схем (ИС) – процессорами и программируемыми логическими интегральными схемами (ПЛИС). Все они отличаются высокой сложностью и жизненный цикл у всех у них очень мал и измеряется несколькими годами – обычно пять лет. Это сильно затрудняет использование задела в новых разработках и заставляет искать новые методы проектирования аппаратуры, обрабатывающей закрытую информацию, которые бы снизили трудоемкость ее разработки. Так возникает проблема переносимости аппаратных узлов (проектов для ПЛИС) и, особенно, программ на другую элементную базу – со старой на новую или на имеющуюся в наличии [5].

Для проектов ПЛИС проблема переносимости успешно решена путем использования языков описания аппаратной части – VHDL или Verilog. Системы автоматизированного проектирования (САПР) всех фирм-

производителей ПЛИС поддерживают эти языки, и поэтому для переноса проекта с одной ПЛИС на другую достаточно просто иметь описание этого проекта на языке VHDL или языке Verilog.

В данном дипломном проекте преимуществом использования ПЛИС для реализации узла контроля целостности является неуязвимость аппаратно реализованной схемы для внешних информационных воздействий.

2.2 Назначение узла и принципы его работы в системе

Разрабатываемый узел является необходимой составной частью разрабатываемой многофункциональной аппаратуры, в которой программа, обрабатывающая конфиденциальную информацию, может загружаться в ОЗУ и быть там объектом атак злоумышленников.

Работа узла контроля целостности информации начинается только по команде от центрального процессора, который отключает процессор от ОЗУ, переводя его выходы в высокоимпедансное состояние, и наоборот, подключает узел к ОЗУ снимая с его выходов высокоимпедансное состояние. Затем после загрузки необходимых исходных данных в узел контроля целостности центральный процессор подачей сигнала запуска однократным единичным импульсом инициирует работу этого узла. Узел контроля целостности данных оказывается подключенным параллельно ОЗУ и начинает считывать из него данные, целостность которых нужно проверить. После считывания узел считает хэш-функцию, сравнивает ее значение с контрольным значением и формирует выходной сигнал о совпадении или несовпадении значений подсчитанной и контрольной хэш-функции.

На рисунке 4 изображен принцип использования в изделии узла контроля целостности данных.

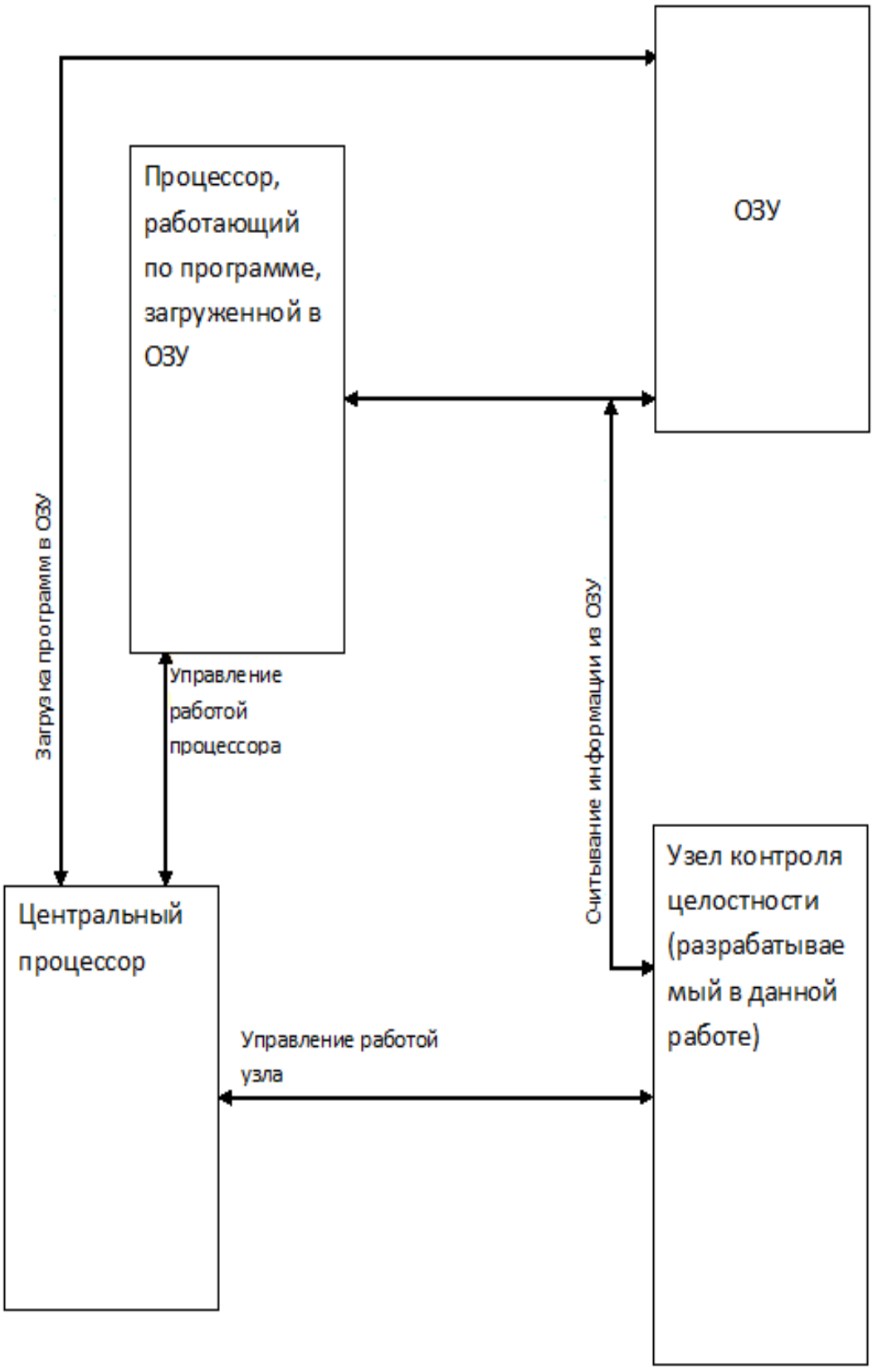

Рисунок 4 – Принцип использования в изделии узла контроля целостности данных

3 Разработка узла контроля целостности

3.1 Использование ПЛИС серии FLEX10K фирмы Altera

Разрабатываемый узел контроля целостности данных должен быть реализован на ПЛИС серии FLEX10K фирмы Altera. Данная необходимость связана с тем, что Воронежский завод АО «ВЗПП-С» выпускает отечественный аналог ПЛИС 5576ХС1Т, который является функциональным аналогом ПЛИС типа EPF10R50RI240 серии FLEX10K. Этот фактор является решением проблемы импортозамещения, так как использование зарубежных изделий для проектирования систем, связанных с обработкой защищаемой информации, недопустимо.

Особенность проектирования на основе ПЛИС 5576ХС1Т является то, что номера выводов ПЛИС 5576ХС1Т не соответствуют номерам выводов ПЛИС EPF10R50RI240. Для задания определения номеров выводов ПЛИС 5576ХС4Т существует специальная таблица соответствия.

Помимо данной особенности ПЛИС 5576ХС1Т имеет расширенную функциональность элементов ввода-вывода (режимы Pull-Up, Pull-Down, Bus-Hold), а также дополнительные режимы работы (SCRUBBING, VERIFICATION, FREQ_DIV), которые отсутствуют в аналоге. Эти режимы реализуются при помощи специального программного обеспечения «5576ХС4Т Configuration Edit Tools».

3.2 Разработка структурной схемы узла

Разработан модуль (узел) контроля целостности. Узел считывает данные из ОЗУ, считает хэш-функцию этих данных, производит сверку подсчитанной хэш-функции с контрольным значением, которое записывается извне. На выход узла передается сигнал о совпадении или несовпадении значений подсчитанной хэш-функции и контрольного значения хэш-функции.

В состав узла входят следующие составные части:

– регистр контрольного значения. Имеет разрядность 512 бит и записывается с внешней 16-разрядной шины данных пословно;

– демультиплексор шины данных (DM). По заданному в ТЗ интерфейсу, содержащему 16-битную шину данных, шину адресов, сигнал записи WR и сигнал чтения RD, демультиплексор обеспечивает запись регистра контрольного значения;

– формирователь управляющих сигналов. Формирует сигналы, необходимые для считывания информации из контролируемого ОЗУ – выборка, запись, чтение, адрес. В соответствии с этими сигналами считываемая из ОЗУ информация попадает в узел подсчета хэш-функции;

– узел подсчета хэш-функции. Считает хэш-функцию от считываемой из ОЗУ информации, передавая ее в 512-разрядный регистр результата;

– узел сравнения. Сравнивает подсчитанную хэш-функцию с контрольной и формирует результат сравнения;

– выходные шинные формирователи F. Формируют требуемые электрические уровни сигналов и обеспечивают сопряжение узла контроля целостности с внешними цепями.

Схема структурная разработанного узла представлена в приложении А.

3.3 Моделирование узла

В процессе проектирования дипломной работы был смоделирован узел контроля целостности в среде для проектирования «Quartus II» на языке описания аппаратуры интегральных схем VHDL.

В рамках дипломного проекта моделирование входных и выходных интерфейсов не представляется возможным, так как узел является составной частью разрабатываемой многофункциональной аппаратуры. Поскольку остальные части аппаратуры не реализованы, то входные данные вводятся вручную.

В качестве среды для проектирования был выбран программный продукт фирмы Altera «Quartus II». Altera является одним из крупнейших разработчиков ПЛИС, концентрирущимся на разработке схем и модулей на языках описания аппаратуры VHDL, Verilog и языке AHDL, который был

разработан фирмой. Программное обеспечение «Quartus II» предоставляет полный цикл для создания высокопроизводительных систем на кристалле. «Quartus II» объединяет в себе все этапы проектирования (ввод проекта, компиляция, верификация и программирование), связь с системами проектирования других производителей и возможность реализации проектов для любого кристалла фирмы Altera. Система «Quartus II» поддерживает такие средства описания проекта, как схемный ввод, описание проекта языками AHDL, VHDL, Verilog, также имеется графическое средство описание параметризируемых модулей. Преимуществом «Quartus II» является его бесплатная версия «Quartus II Web Edition», которая поддерживает все этапы проектирования от ввода проекта до программирования.

В качестве языка проектирования был выбран VHDL. Это мощный язык, который позволяет описывать поведение, т.е. алгоритмы функционирования цифровых систем, а также проводить иерархическое функционально-структурное описание систем. VHDL имеет средства для описания параллельных асинхронных процессов, регулярных структур и обладает возможностями языка программирования высокого уровня, т.е. позволяет создавать свои типы данных, имеет широкий набор арифметических и логических операций [6].

Описание узла в среде для проектирования «Quartus II» на языке описания аппаратуры VHDL представлено в приложении Б.

4 Описание модулей

В результате моделирования были написаны на языке описания аппаратуры VHDL восемь модулей. Каждый из модулей выполняет свою функцию.

Модуль «check_module» предназначен для контроля расчета хэш-функции «Стрибог» и проверки на совпадение рассчитанного значения с контрольным значением.

Подключение библиотек:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;
```

Описание интерфейса модуля:

```
entity check_module is

	port
	(
		clk                         : in std_logic;
		data_in                     : in std_logic_vector(511 downto
0);
		hash_check_in          : in std_logic_vector(511 downto 0);
		start                  : in std_logic;
		check_over             : out std_logic := '0';
		check_resume           : out std_logic := '0'
	);

end check_module;
```

где clk – тактовая частота, data_in – входные данные для расчета хэш-функции, hash_check_in – контрольное значение хэш-функции для сверки с полученным, start – сигнал запуска работы модуля, check_over – сигнал завершения обработки данных, check_resume – признак вердикта о совпадении рассчитанного и контрольного значений хэш-функций (1 – совпадение, 0 – несовпадение).

Описание архитектуры модуля:

```
architecture arch of check_module is

component gh_fixed512_logic

	port
	(
		clk                         : in std_logic;
```

```
          clken                : in std_logic;
          data_in                   : in std_logic_vector(511 downto
0);
          valid_out            : out std_logic := '0';
          hash_out             : out std_logic_vector(511 downto 0)
     );

end component gh_fixed512_logic;
     signal enable             : std_logic := '0';
     signal data_in_i     : std_logic_vector(511 downto 0);
     signal valid_out_i        : std_logic := '0';
     signal hash_out_i         : std_logic_vector(511 downto 0) :=
(others => '0');
     signal hash_out_i_lock    : std_logic_vector(511 downto 0) :=
(others => '0');
     signal hash_check_in_i    : std_logic_vector(511 downto 0) :=
(others => '0');
     signal step                    : integer range 0 to 7 := 0;

begin
```

Процесс управления модулем расчета хэша:

```
process(clk)

begin

     if( rising_edge(clk) ) then
```

Предустановка признака завершения обработки в 0:

```
          check_over               <= '0';
```

Автомат работы с данными:

```
          case(step) is
```

Состояние автомата "ожидание сигнала запучка работы":

```
          when 0 =>
```

Сброс признака совпадения хэшей:

```
          check_resume         <= '0';
```

Если подан сигнал начала работы, тогда дается разрешение на работу модулю расчета, подается входной вектор данных на модуль расчета, подается в буфер контрольное значение хэш-функции, осуществляется переход на следующее состояние автомата для последующего сравнения:

```
               if( start = '1' ) then

                    enable                <= '1';
                    data_in_i        <= data_in;
                    hash_check_in_i <= hash_check_in;
                    step             <= 1;

               else
```

Если нет сигнала запуска работы, тогда модуль расчета остановлен:

```
            enable                <= '0';

        end if;
```

Состояние автомата переходит в «ожидание завершения расчета значения хэш-функции»:

```
        when 1 =>
```

Если от модуля расчета пришел признак валидности выходных данных, тогда происходит остановка модуля расчета, рассчитанное значение хэш-функции подается в буфер для последующего сравнения. После следует переход на следующее состояние автомата:

```
        if( valid_out_i = '1' ) then

            enable                  <= '0';
            hash_out_i_lock <= hash_out_i;
            step            <= 2;

        end if;
```

В состоянии автомата "сравнение значений хэш-функций" выдается признак завершения обработки данных, затем сравниваются рассчитанное значение хэш-функции и контрольное значение:

```
    when 2 =>

        check_over            <= '1';
        if( hash_out_i_lock = hash_check_in_i ) then
```

При совпадении рассчитанного значения хэш-функции и контрольного значения выдается признак совпадения 1, при несовпадении – 0, после следует переход на следующее состояние автомата:

```
            check_resume    <= '1';
        else

            check_resume    <= '0';
        end if;

        step                <= 3;
```

Состояние автомата "ожидание сброса сигнала старт" (это состояние необходимо для того, чтобы расчеты были однозначно стартованы, а не

выполнялись бесконечно над одними и теми же данными), если сигнал старта сброшен, то осуществляется переход на начальное состояние автомата:

```
            when 3 =>

                  if( start = '0' ) then
                        step            <= 0;
                  end if;
            when others => null;
            end case;

      end if;

end process;


gh_fixed512_logic_inst : gh_fixed512_logic

      port map
      (
            clk                        => clk,
            clken                 => enable,
            data_in                    => data_in_i,
            valid_out             => valid_out_i,
            hash_out              => hash_out_i
      );


end arch;
```

Модуль «tb» предназначен для ввода входных данных и моделирования тактовой частоты.

Подключение библиотек:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;
```

Описание интерфейса модуля:

```
entity tb is

      port
      (
            check_over            : out std_logic := '0';
            check_resume          : out std_logic := '0'
      );

end tb;
```

Описание архитектуры модуля:

```
architecture arch of tb is

component check_module is
```

```
     port
     (
          clk                       : in std_logic;
          data_in                   : in std_logic_vector(511 downto
0);
          hash_check_in        : in std_logic_vector(511 downto 0);
          start                : in std_logic;
          check_over           : out std_logic := '0';
          check_resume         : out std_logic := '0'
     );

end component check_module;
```

Ввод входных данных:

```
     signal data_in            : std_logic_vector(511 downto 0) :=
(others => '0');
     signal hash_check_in  : std_logic_vector(511 downto 0) :=
x"D32FA1AA3DF25B8853FE3E059D25C0E17238D3027EB271557949F248267FB89C4700
CFEC58DB03CB92F69C6D9B2CE7F008BD0C6D5306C859417463B2ECED692C";
     signal start              : std_logic := '0';
     signal clk                : std_logic := '0';

begin

     data_in(15 downto 0)  <= x"0000", x"1234" after 10 us;
     start                     <= '0', '1' after 12 us, '0' after 20
us;
```

Процесс моделирования тактовой частоты (периодом 2 мксек):

```
Process

begin
     wait for 1 us; clk <= not clk;
end process;

check_module_inst : check_module

     port map

     (
          clk                       => clk,
          data_in                   => data_in,
          hash_check_in        => hash_check_in,
          start                => start,
          check_over           => check_over,
          check_resume         => check_resume
     );

end arch;
```

Модуль «gh_fixed512_logic» предназначен для расчета хэш-функции «Стрибог».

Подключение библиотек:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_arith.all;
```

Описание интерфейса модуля:

```
entity gh_fixed512_logic is

port
(
      clk                         : in std_logic;
      clken                 : in std_logic;
      data_in                     : in std_logic_vector(511 downto 0);
      valid_out             : out std_logic := '0';
      hash_out              : out std_logic_vector(511 downto 0)
);

end gh_fixed512_logic;
```

Где valid_out – признак валидности выходных данных.

Описание архитектуры модуля:

```
architecture arch of gh_fixed512_logic is

component gh_round_logic is

port
(
      clk                         : in std_logic;
      clken                 : in std_logic;
      hash_in                     : in std_logic_vector(511 downto 0);
      numbits_in            : in std_logic_vector(511 downto 0);
      data_in                     : in std_logic_vector(511 downto 0);
      valid_out             : out std_logic := '0';
      hash_out              : out std_logic_vector(511 downto 0)
);

end component gh_round_logic;
```

Присвоение значений сигналам типа std_logic_vector:

```
signal after_data : std_logic_vector(511 downto 0);
signal after_numbits : std_logic_vector(511 downto 0);
signal vector_512 : std_logic_vector(511 downto 0);
```

Создание типа данных типа матрицы 80x512:

```
type data_delay_type is array (0 to 79) of std_logic_vector(511 downto
0);

signal data_delay : data_delay_type;
signal valid_out_1 : std_logic := '0';
```

```
signal valid_out_2 : std_logic := '0';
```

Преобразование числа 512 в вектор 512-разрядный:

```
vector_512      <= conv_std_logic_vector(512, 512);
```

Подключение некоторых модулей и организация связей между ними:

```
stage_1_data : gh_round_logic

port map

(
      clk             => clk,
      clken                 => clken,
      hash_in         => (others => '0'),
      numbits_in=> (others => '0'),
      data_in         => data_in,
      valid_out       => valid_out_1,
      hash_out        => after_data
);

stage_2_numbits : gh_round_logic

port map
(
      clk             => clk,
      clken                 => valid_out_1, --clken,
      hash_in         => after_data,
      numbits_in=> (others => '0'),
      data_in         => vector_512,
      valid_out       => valid_out_2,
      hash_out        => after_numbits
);
```

Процесс промежуточного формирования данных для подключенного модуля:

```
stage_3_sumbits

process(clk)

begin
      if( rising_edge(clk) ) then
```

Проверка сигнала clken (разрешения обработки данных):

```
            if( clken = '1' ) then
```

Организация сдвигового регистра:

```
                  data_delay(1 to 79) <= data_delay(0 to 78);
                  data_delay(0)   <= data_in;
            end if;

      end if;

end process;
```

```
stage_3_sumbits : gh_round_logic

port map
(
      clk             => clk,
      clken                 => valid_out_2, --clken,
      hash_in         => after_numbits,
      numbits_in=> (others => '0'),
      data_in         => data_delay(79),
      valid_out       => valid_out,
      hash_out        => hash_out
);

end arch;
```

Модуль «gh_round_efunc_v2_logic» предназначен для расчета функции блочного шифра.

Подключение библиотек:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;
```

Описание интерфейса модуля:

```
entity gh_round_efunc_v2_logic is

port
(
     clk                        : in std_logic;
     clken                 : in std_logic;
     hash_in                    : in std_logic_vector(511 downto 0);
     data_in                    : in std_logic_vector(511 downto 0);
     valid_out             : out std_logic := '0';
     hash_out              : out std_logic_vector(511 downto 0)
);

end gh_round_efunc_v2_logic;
```

Создание типа данных типа матрицы 12x512:

```
     type const_c_type is array (1 to 12) of std_logic_vector(511
downto 0);
```

Объявление константы C, значения которой взяты из ГОСТ 34.11-2012:

```
     constant const_c : const_c_type := (

x"b1085bda1ecadae9ebcb2f81c0657c1f2f6a76432e45d016714eb88d7585c4fc4b7c
e09192676901a2422a08a460d31505767436cc744d23dd806559f2a64507",

x"6fa3b58aa99d2f1a4fe39d460f70b5d7f3feea720a232b9861d55e0f16b501319ab5
176b12d699585cb561c2db0aa7ca55dda21bd7cbcd56e679047021b19bb7",
```

```
x"f574dcac2bce2fc70a39fc286a3d843506f15e5f529c1f8bf2ea7514b1297b7bd3e2
0fe490359eb1c1c93a376062db09c2b6f443867adb31991e96f50aba0ab2",

x"ef1fdfb3e81566d2f948e1a05d71e4dd488e857e335c3c7d9d721cad685e353fa9d7
2c82ed03d675d8b71333935203be3453eaa193e837f1220cbebc84e3d12e",

x"4bea6bacad4747999a3f410c6ca923637f151c1f1686104a359e35d7800fffbdbfcd
1747253af5a3dfff00b723271a167a56a27ea9ea63f5601758fd7c6cfe57",

x"ae4faeae1d3ad3d96fa4c33b7a3039c02d66c4f95142a46c187f9ab49af08ec6cffa
a6b71c9ab7b40af21f66c2bec6b6bf71c57236904f35fa68407a46647d6e",

x"f4c70e16eeaac5ec51ac86febf240954399ec6c7e6bf87c9d3473e33197a93c90992
abc52d822c3706476983284a05043517454ca23c4af38886564d3a14d493",

x"9b1f5b424d93c9a703e7aa020c6e41414eb7f8719c36de1e89b4443b4ddbc49af489
2bcb929b069069d18d2bd1a5c42f36acc2355951a8d9a47f0dd4bf02e71e",

x"378f5a541631229b944c9ad8ec165fde3a7d3a1b258942243cd955b7e00d0984800a
440bdbb2ceb17b2b8a9aa6079c540e38dc92cb1f2a607261445183235adb",

x"abbedea680056f52382ae548b2e4f3f38941e71cff8a78db1fffe18a1b3361039fe7
6702af69334b7a1e6c303b7652f43698fad1153bb6c374b4c7fb98459ced",

x"7bcd9ed0efc889fb3002c6cd635afe94d8fa6bbbebab076120018021148466798a1d
71efea48b9caefbacd1d7d476e98dea2594ac06fd85d6bcaa4cd81f32d1b",

x"378ee767f11631bad21380b00449b17acda43c32bcdf1d77f82012d430219f9b5d80
ef9d1891cc86e71da4aa88e12852faf417d5d9b21b9948bc924af11bd720"
     );
```

Создание типа данных типа матрицы 12x512, 13x512, 13x1:

```
     type m12_type is array (1 to 12) of std_logic_vector(511 downto
0);
     type m13_type is array (1 to 13) of std_logic_vector(511 downto
0);
     type b13_type is array (1 to 13) of std_logic;
```

Присвоение значений сигналам:

```
     signal val_x : m13_type;
     signal val_kc : m12_type;

     signal val_k : m13_type;
     signal val_lps : m13_type;

     signal clken_i : b13_type := (others => '0');
     signal valid_out_i : b13_type := (others => '0');

component gh_round_lps_logic

port
(
     clk                         : in std_logic;
     clken                  : in std_logic;
     lps_arg                     : in std_logic_vector(511 downto 0);
     valid_out              : out std_logic := '0';
```

```
      lps_func              : out std_logic_vector(511 downto 0)
);

end component gh_round_lps_logic;

begin
```

Вывод выходных данных на выход:

```
      hash_out <= val_x(13);
```

Предварительная подготовка промежуточных данных:

```
      val_lps(1) <= data_in;
      val_k(1) <= hash_in;
      valid_out_i(1) <= clken;
      clken_i(1) <= clken;
```

Обработка данных:

```
generate_cycle_j: for g1 in 2 to 13 generate

      process(clk

      begin
            if(rising_edge(clk)) then
```

Проверка сигнала clken (разрешения обработки данных):

```
                  if( clken = '1' ) then

                        val_x(g1-1) <= val_k(g1-1) xor val_lps(g1-1);
                        val_kc(g1-1) <= val_k(g1-1) xor const_c(g1-1);
                        clken_i(g1) <= valid_out_i(g1-1);
                  end if;

            end if;

      end process;

      gh_round_lps_logic_main_inst : gh_round_lps_logic

      port map
      (
            clk                        => clk,
            clken                            => clken,
            --clken                          => clken_i(g1-1),
            lps_arg                    => val_x(g1-1),
            valid_out                  => valid_out_i(g1),
            lps_func                   => val_lps(g1)
      );

      gh_round_lps_logic_k_inst : gh_round_lps_logic

      port map
      (
            clk                        => clk,
            clken                            => clken,
            --clken                          => clken_i(g1-1),
            lps_arg                    => val_kc(g1-1),
```

```
            valid_out                   => open, --valid_out_i(g1),
            lps_func                    => val_k(g1)
      );

end generate generate_cycle_j;
```

Процесс предварительного формирования выходных данных и сигнала валидности этих выходных данных:

```
process(clk)

begin
      if(rising_edge(clk)) then
            -- проверка сигнала clken (разрешения обработки данных)
            if( clken = '1' ) then
                  val_x(13) <= val_k(13) xor val_lps(13);
                  valid_out <= valid_out_i(13);
            else
                  valid_out <= '0';
            end if;

      end if;

end process;

end arch;
```

Модуль «gh_round_logic» также предназначен для расчета функции блочного шифра.

Подключение библиотек:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;
```

Описание интерфейса модуля:

```
entity gh_round_logic is

port
(
      clk                         : in std_logic;
      clken                 : in std_logic;
      hash_in                     : in std_logic_vector(511 downto 0);
      numbits_in            : in std_logic_vector(511 downto 0);
      data_in                     : in std_logic_vector(511 downto 0);
      valid_out             : out std_logic := '0';
      hash_out              : out std_logic_vector(511 downto 0)
);

end gh_round_logic;
```

Описание архитектуры модуля:

```
architecture arch of gh_round_logic is
```

```
component gh_round_lps_logic is

port
(
      clk                         : in std_logic;
      clken                 : in std_logic;
      lps_arg                     : in std_logic_vector(511 downto 0);
      valid_out             : out std_logic := '0';
      lps_func              : out std_logic_vector(511 downto 0)
);

end component gh_round_lps_logic;

component gh_round_efunc_v2_logic is

port
(
      clk                         : in std_logic;
      clken                 : in std_logic;
      hash_in                     : in std_logic_vector(511 downto 0);
      data_in                     : in std_logic_vector(511 downto 0);
      valid_out             : out std_logic := '0';
      hash_out              : out std_logic_vector(511 downto 0)
);

end component gh_round_efunc_v2_logic;
```

Присвоение значений сигналу:

```
signal first_lps_func : std_logic_vector(511 downto 0);
```

Создание типа данных типа матрицы 2x512, 37x512, 37x1:

```
type delay_type is array (0 to 1) of std_logic_vector(511 downto 0);

signal data_in_delay : delay_type;
signal hash_in_delay : delay_type;

type dxorh_delay_type is array (0 to 36) of std_logic_vector(511
downto 0);

signal dxorh_delay : dxorh_delay_type;
signal efunc_out : std_logic_vector(511 downto 0);
signal hash_out_ff : std_logic_vector(511 downto 0);

signal temp_xor1 : std_logic_vector(511 downto 0);
signal temp_xor2 : std_logic_vector(511 downto 0);

type valid_out_delay_type is array (0 to 36) of std_logic;

signal valid_out_delay : valid_out_delay_type := (others => '0');

signal valid_lps : std_logic := '0';
signal valid_efunc_v2 : std_logic := '0';

begin
```

Предварительная подготовка входных данных для обработки:

```
      temp_xor1       <= hash_in xor numbits_in;


gh_round_lps_logic_inst : gh_round_lps_logic

port map
(
      clk                  => clk,
      clken          => clken,
      lps_arg              => temp_xor1,
      valid_out      => valid_lps,
      lps_func       => first_lps_func
);
```

Обработка данных:

```
process(clk)

begin
      if( rising_edge(clk) ) then
```

Проверка сигнала clken (разрешения обработки данных):

```
            if( clken = '1' ) then
                  data_in_delay(1) <= data_in_delay(0);
                  data_in_delay(0) <= data_in;
                  hash_in_delay(1) <= hash_in_delay(0);
                  hash_in_delay(0) <= hash_in;
            end if;

      end if;

end process;
```

Промежуточная подготовка данных для обработки:

```
      temp_xor2                  <= data_in_delay(1) xor
hash_in_delay(1);


process(clk)

begin
      if( rising_edge(clk) ) then
```

Проверка сигнала clken (разрешения обработки данных):

```
            if( clken = '1' ) then
```

Организация сдвигового регистра и сдвигового регистра значений валидности выходных данных из модуля gh_round_efunc_v2_logic_inst:

```
                  dxorh_delay(1 to 36) <= dxorh_delay(0 to 35);
                  dxorh_delay(0)  <= temp_xor2;
                  valid_out_delay(1 to 36) <= valid_out_delay(0 to 35);
                  valid_out_delay(0) <= valid_efunc_v2;
            else
```

Сброс значений валидности выходных данных:

```
                valid_out_delay(0 to 36) <= (others => '0');
           end if;

      end if;

end process;


gh_round_efunc_v2_logic_inst : gh_round_efunc_v2_logic

port map
(
      clk             => clk,
      --clken               => valid_lps, --clken,
      clken               => clken,
      hash_in         => first_lps_func,
      data_in         => data_in_delay(1),
           valid_out  => valid_efunc_v2,
      hash_out        => efunc_out
);
```

Процесс формирования выходных данных и сигнала валидности этих выходных данных:

```
process(clk)

begin
      if( rising_edge(clk) ) then
```

Проверка сигнала clken (разрешения обработки данных):

```
           if( clken = '1' ) then
                hash_out_ff <= efunc_out xor dxorh_delay(36);
                valid_out <= valid_out_delay(36);
           else
                valid_out <= '0';
           end if;

      end if;

end process;
      hash_out <= hash_out_ff;

      end arch;
```

Модуль «gh_round_lps_logic» предназначен для организации связей между модулями расчета L – преобразования, P – преобразования, S – преобразования.

Подключение библиотек:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;
```

Описание интерфейса модуля:

```
entity gh_round_lps_logic is

port
(
      clk                        : in std_logic;
      clken                  : in std_logic;
      lps_arg                    : in std_logic_vector(511 downto 0);
      valid_out              : out std_logic := '0';
      lps_func               : out std_logic_vector(511 downto 0)
);

end gh_round_lps_logic;
```

Описание архитектуры модуля:

```
architecture arch of gh_round_lps_logic is

component gh_round_subs_logic is

port
(
      clk                        : in std_logic;
      clken                  : in std_logic;
      lps_arg                    : in std_logic_vector(511 downto 0);
      valid_out              : out std_logic := '0';
      after_s                    : out std_logic_vector(511 downto 0)
);

end component gh_round_subs_logic;
      signal after_s             : std_logic_vector(511 downto 0);
      signal valid_out_subs : std_logic := '0';

component gh_round_perm_logic is

port
(
      after_s                    : in std_logic_vector(511 downto 0);
      after_p                    : out std_logic_vector(511 downto 0)
);
end component gh_round_perm_logic;

      signal after_p             : std_logic_vector(511 downto 0);

component gh_round_ltran_logic is

port
(
      clk                        : in std_logic;
      clken                  : in std_logic;
      after_p                    : in std_logic_vector(511 downto 0);
      valid_out              : out std_logic := '0';
      lps_func               : out std_logic_vector(511 downto 0)
);

end component gh_round_ltran_logic;
```

Подключение модулей расчета преобразований и организация связей между ними:

```
gh_round_subs_logic_inst : gh_round_subs_logic

port map
(
      clk                        => clk,
      clken                           => clken,
      lps_arg                    => lps_arg,
      valid_out                  => valid_out_subs,
      after_s                    => after_s
);

gh_round_perm_logic_inst : gh_round_perm_logic

port map
(
      after_s                    => after_s,
      after_p                    => after_p
);

gh_round_ltran_logic_inst : gh_round_ltran_logic

port map
(
      clk                        => clk,
      clken                           => clken,
      --clken                         => valid_out_subs,
      after_p                    => after_p,
      valid_out                  => valid_out,
      lps_func                   => lps_func
);

end arch;
```

Модуль «gh_round_ltran_logic» предназначен для расчета L – преобразования (линейное преобразование).

Подключение библиотек:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;
```

Описание интерфейса модуля:

```
entity gh_round_ltran_logic is

port
(
      clk                        : in std_logic;
      clken                 : in std_logic;
      after_p                    : in std_logic_vector(511 downto 0);
      valid_out             : out std_logic := '0';
```

```
      lps_func              : out std_logic_vector(511 downto 0)
);

end gh_round_ltran_logic;

architecture arch of gh_round_ltran_logic is
```

Создание типа данных типа матрицы 64x64:

```
type table_a_type is array (0 to 63) of std_logic_vector(63 downto 0);
```

Значения таблицы коэффициентов:

```
constant table_a: table_a_type := (
x"8e20faa72ba0b470", x"47107ddd9b505a38", x"ad08b0e0c3282d1c",
x"d8045870ef14980e",
x"6c022c38f90a4c07", x"3601161cf205268d", x"1b8e0b0e798c13c8",
x"83478b07b2468764",
x"a011d380818e8f40", x"5086e740ce47c920", x"2843fd2067adea10",
x"14aff010bdd87508",
x"0ad97808d06cb404", x"05e23c0468365a02", x"8c711e02341b2d01",
x"46b60f011a83988e",
x"90dab52a387ae76f", x"486dd4151c3dfdb9", x"24b86a840e90f0d2",
x"125c354207487869",
x"092e94218d243cba", x"8a174a9ec8121e5d", x"4585254f64090fa0",
x"accc9ca9328a8950",
x"9d4df05d5f661451", x"c0a878a0a1330aa6", x"60543c50de970553",
x"302a1e286fc58ca7",
x"18150f14b9ec46dd", x"0c84890ad27623e0", x"0642ca05693b9f70",
x"0321658cba93c138",
x"86275df09ce8aaa8", x"439da0784e745554", x"afc0503c273aa42a",
x"d960281e9d1d5215",
x"e230140fc0802984", x"71180a8960409a42", x"b60c05ca30204d21",
x"5b068c651810a89e",
x"456c34887a3805b9", x"ac361a443d1c8cd2", x"561b0d22900e4669",
x"2b838811480723ba",
x"9bcf4486248d9f5d", x"c3e9224312c8c1a0", x"effa11af0964ee50",
x"f97d86d98a327728",
x"e4fa2054a80b329c", x"727d102a548b194e", x"39b008152acb8227",
x"9258048415eb419d",
x"492c024284fbaec0", x"aa16012142f35760", x"550b8e9e21f7a530",
x"a48b474f9ef5dc18",
x"70a6a56e2440598e", x"3853dc371220a247", x"1ca76e95091051ad",
x"0edd37c48a08a6d8",
x"07e095624504536c", x"8d70c431ac02a736", x"c83862965601dd1b",
x"641c314b2b8ee083"
      );
```

Создание типа данных типа матрицы 64x8:

```
type mass_64x7_type is array (0 to 63) of std_logic_vector(7 downto
0);

signal state_in : mass_64x7_type;
signal state_out : mass_64x7_type;
signal state_ff : mass_64x7_type;
```

Создание функции для обработки данных:

```
function func_many_cycles(state_in: mass_64x7_type) return
```

```
mass_64x7_type is

    variable v : std_logic_vector(63 downto 0);
    variable temp1 : std_logic_vector(7 downto 0);
    variable temp2 : std_logic_vector(7 downto 0);
    variable temp3 : std_logic_vector(7 downto 0);
    variable temp11 : integer range 0 to 63;
    variable temp12 : integer range 0 to 63;
     variable state_out_v : mass_64x7_type;
```

Циклы обработки данных:

```
for i in 0 to 7 loop
     v := (others => '0');
     for k in 0 to 7 loop
          for j in 0 to 7 loop
```

Проверка байт данных на то, чтобы они были информативными и не были равны 0 или неопределенными:

```
                if (state_in(63-(i*8+k))(7-j) /= '0' and state_in(63-
(i*8+k))(7-j) /= 'U' and state_in(63-(i*8+k))(7-j) /= 'Z' and
state_in(63-(i*8+k))(7-j) /= 'X' ) then
                     v := v xor table_a(k*8+j);
                end if;

          end loop;

     end loop;
```

Осуществляется расчет линейного преобразования:

```
     for kk in 0 to 7 loop

          temp11 := (7-kk)*8;
          temp3 := v(temp11+7 downto temp11);

          state_out_v(63-(i*8+kk)) := temp3;
     end loop;

end loop;

return state_out_v;
end func_many_cycles;

begin
```

Осуществляется переписывание байтов из входного массива в байты двумерной матрицы для того, чтобы входной массив 512 разрядов обрабатывать побайтово:

```
fill_inputs_cycle_j: for in_j in 0 to 63 generate
     state_in(in_j)  <= after_p((7+in_j*8) downto in_j*8);
end generate fill_inputs_cycle_j;
```

Осуществляется переписывание данных побайтово из матрицы в выходной массив для того, чтобы сформировать из двумерной матрицы данных входной 512-разрядный массив:

```
fill_outputs_cycle_j: for in_j in 0 to 63 generate
     lps_func((7+in_j*8) downto in_j*8) <= state_ff(in_j);
end generate fill_outputs_cycle_j;
```

Процесс обработки данных:

```
process(clk)

begin
     if( rising_edge(clk) ) then
```

Проверка сигнала clken (разрешения обработки данных):

```
          if( clken = '1' ) then
```

Вызов функции func_many_cycles():

```
               state_ff <= func_many_cycles(state_in);
```

Установка признака валидности выходных данных:

```
               valid_out <= '1';
          else
```

Сброс признака валидности выходных данных:

```
               valid_out <= '0';
          end if;

     end if;

end process;


end arch;
```

Модуль «gh_round_perm_logic» предназначен для расчета P – преобразования (преобразование перестановки).

Подключение библиотек:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;
```

Описание интерфейса модуля:

```
entity gh_round_perm_logic is

port
(
     after_s                    : in std_logic_vector(511 downto 0);
     after_p                    : out std_logic_vector(511 downto 0)
);
```

```
end gh_round_perm_logic;
```

Задается двумерная матрица из байт:

```
type mass_7x7_type is array (0 to 7) of std_logic_vector(7 downto 0);
```

Задается трехмерная матрица из байт:

```
type mass_7x7x7_type is array (0 to 7) of mass_7x7_type;


signal after_s_int : mass_7x7x7_type := (others => (others => (others
=> '0')));
signal after_p_int : mass_7x7x7_type := (others => (others => (others
=> '0')));

begin
```

Осуществляется переписывание байтов из входного массива в байты трехмерной матрицы для того, чтобы входной массив 512 разрядов обрабатывать побайтово:

```
      fill_inputs_cycle_i: for in_i in 0 to 7 generate
      fill_inputs_cycle_j: for in_j in 0 to 7 generate
            after_s_int(in_i)(in_j)    <= after_s((7+in_i*64+in_j*8)
downto in_i*64+in_j*8);

      end generate fill_inputs_cycle_j;

end generate fill_inputs_cycle_i;
```

Осуществляется побайтовая обработка данных:

```
rotate_cycle_i: for in_i in 0 to 7 generate
      rotate_cycle_j: for in_j in 0 to 7 generate
            after_p_int(in_j)(in_i) <= after_s_int(in_i)(in_j);
      end generate rotate_cycle_j;

end generate rotate_cycle_i;
```

Осуществляется переписывание данных побайтово из матрицы в выходной массив для того, чтобы сформировать из трехмерной матрицы данных входной 512-разрядный массив:

```
fill_outputs_cycle_i: for in_i in 0 to 7 generate
      fill_outputs_cycle_j: for in_j in 0 to 7 generate
            after_p((7+in_i*64+in_j*8) downto in_i*64+in_j*8) <=
after_p_int(in_i)(in_j);
      end generate fill_outputs_cycle_j;

end generate fill_outputs_cycle_i;

end arch;
```

Модуль «gh_round_subs_logic» предназначен для расчета S – преобразования (преобразование подстановки).

Подключение библиотек:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;
```

Описание интерфейса модуля:

```
entity gh_round_subs_logic is

port
(
     clk                        : in std_logic;
     clken                 : in std_logic;
     lps_arg                    : in std_logic_vector(511 downto 0);
     valid_out             : out std_logic := '0';
     after_s                    : out std_logic_vector(511 downto 0)
);

end gh_round_subs_logic;
```

Создание типа данных типа матрицы 256x8:

```
type sbox_table_type is array (0 to 255) of std_logic_vector(7 downto
0);
```

Ввод значений подстановки:

```
constant sbox_table   : sbox_table_type := (
     x"FC", x"EE", x"DD", x"11", x"CF", x"6E", x"31", x"16", x"FB",
x"C4", x"FA", x"DA", x"23", x"C5", x"04", x"4D",
     x"E9", x"77", x"F0", x"DB", x"93", x"2E", x"99", x"BA", x"17",
x"36", x"F1", x"BB", x"14", x"CD", x"5F", x"C1",
     x"F9", x"18", x"65", x"5A", x"E2", x"5C", x"EF", x"21", x"81",
x"1C", x"3C", x"42", x"8B", x"01", x"8E", x"4F",
     x"05", x"84", x"02", x"AE", x"E3", x"6A", x"8F", x"A0", x"06",
x"0B", x"ED", x"98", x"7F", x"D4", x"D3", x"1F",
     x"EB", x"34", x"2C", x"51", x"EA", x"C8", x"48", x"AB", x"F2",
x"2A", x"68", x"A2", x"FD", x"3A", x"CE", x"CC",
     x"B5", x"70", x"0E", x"56", x"08", x"0C", x"76", x"12", x"BF",
x"72", x"13", x"47", x"9C", x"B7", x"5D", x"87",
     x"15", x"A1", x"96", x"29", x"10", x"7B", x"9A", x"C7", x"F3",
x"91", x"78", x"6F", x"9D", x"9E", x"B2", x"B1",
     x"32", x"75", x"19", x"3D", x"FF", x"35", x"8A", x"7E", x"6D",
x"54", x"C6", x"80", x"C3", x"BD", x"0D", x"57",
     x"DF", x"F5", x"24", x"A9", x"3E", x"A8", x"43", x"C9", x"D7",
x"79", x"D6", x"F6", x"7C", x"22", x"B9", x"03",
     x"E0", x"0F", x"EC", x"DE", x"7A", x"94", x"B0", x"BC", x"DC",
x"E8", x"28", x"50", x"4E", x"33", x"0A", x"4A",
     x"A7", x"97", x"60", x"73", x"1E", x"00", x"62", x"44", x"1A",
x"B8", x"38", x"82", x"64", x"9F", x"26", x"41",
     x"AD", x"45", x"46", x"92", x"27", x"5E", x"55", x"2F", x"8C",
x"A3", x"A5", x"7D", x"69", x"D5", x"95", x"3B",
```

```
     x"07", x"58", x"B3", x"40", x"86", x"AC", x"1D", x"F7", x"30",
x"37", x"6B", x"E4", x"88", x"D9", x"E7", x"89",
     x"E1", x"1B", x"83", x"49", x"4C", x"3F", x"F8", x"FE", x"8D",
x"53", x"AA", x"90", x"CA", x"D8", x"85", x"61",
     x"20", x"71", x"67", x"A4", x"2D", x"2B", x"09", x"5B", x"CB",
x"9B", x"25", x"D0", x"BE", x"E5", x"6C", x"52",
     x"59", x"A6", x"74", x"D2", x"E6", x"F4", x"B4", x"C0", x"D1",
x"66", x"AF", x"C2", x"39", x"4B", x"63", x"B6"
     );
```

Создание типа данных типа матрицы 64x8, 64x1:

```
type mass_64x7_type is array (0 to 63) of std_logic_vector(7 downto
0);
signal data_in_int : mass_64x7_type;
signal data_out_ff : mass_64x7_type;
type mass_64x1_type is array (0 to 63) of std_logic;
signal valid_i : mass_64x1_type := (others => '0');

begin
```

Осуществляется переписывание байтов из входного массива в байты двумерной матрицы для того, чтобы входной массив 512 разрядов обрабатывать побайтово:

```
fill_inputs_cycle_j: for in_j in 0 to 63 generate
     data_in_int(in_j)     <= lps_arg((7+in_j*8) downto in_j*8);
end generate fill_inputs_cycle_j;
```

Осуществляется переписывание данных побайтово из матрицы в выходной массив для того, чтобы сформировать из двумерной матрицы данных входной 512-разрядный массив:

```
fill_outputs_cycle_j: for in_j in 0 to 63 generate
     after_s((7+in_j*8) downto in_j*8) <= data_out_ff(in_j);
end generate fill_outputs_cycle_j;
```

Заданный выходной признак валидности равен 1 при условии, что все промежуточные признаки валидности равны 1:

```
     valid_out <= '1' when valid_i(0 to 63) = x"FFFFFFFFFFFFFFFF";
```

Осуществляется побайтовая обработка данных:

```
g_cycle: for g1 in 0 to 63 generate
     process( clk )
     begin
          if( rising_edge(clk) ) then
```

Проверка сигнала clken (разрешения обработки данных):

```
               if( clken = '1' ) then
```

Проверка байт данных на то, чтобы они были информативными и не были равны нулю или неопределенными:

```
                    if( data_in_int(g1) /= "UUUUUUUU" and
data_in_int(g1) /= "ZZZZZZZZ" and data_in_int(g1) /= "XXXXXXXX" ) then
                        data_out_ff(g1) <=
sbox_table(conv_integer(data_in_int(g1)));
```

Установка признака валидности выходных данных:

```
                        valid_i(g1) <= '1';
                    end if;
                else
```

Сброс признака валидности выходных данных:

```
                    valid_i(g1) <= '0';
                end if;
            end if;
        end process;
        end generate;

end arch;
```

5 Результаты моделирования

Для большей наглядности работоспособности смоделированного в «Quartus II» узла в качестве симулятора была использована программа ModelSim Altera Starter Edition.

Для корректной симуляции необходимо расположить модули в правильном порядке, как показано на рисунке 5.

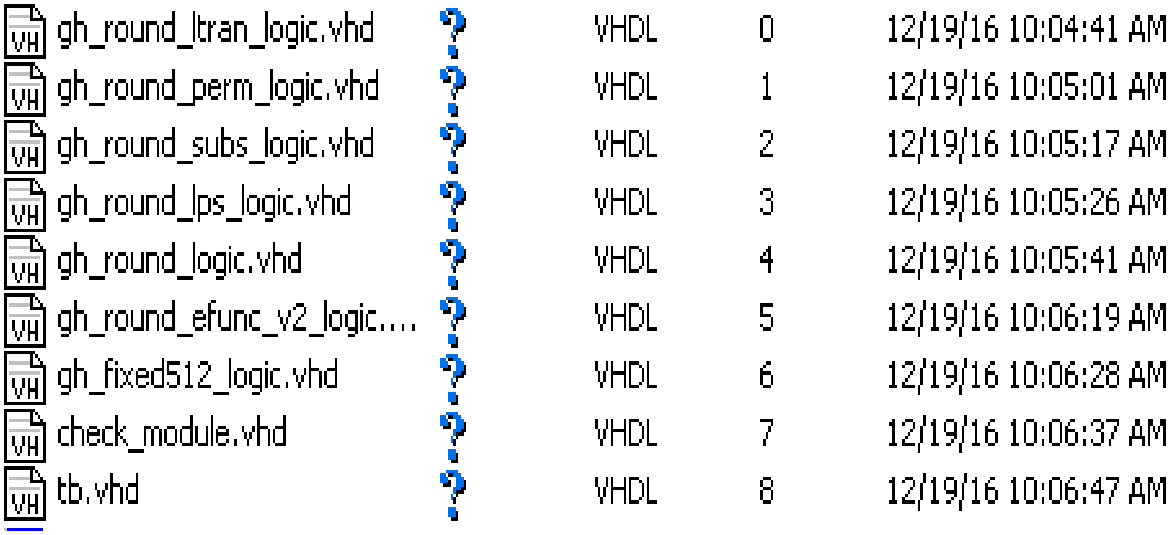

Рисунок 5 – Порядок расположения модулей

Для дальнейших действий необходимо выполнить компиляцию. Компиляция модулей выполняется нажатием на кнопку, выделенную красной рамкой на рисунке 6.

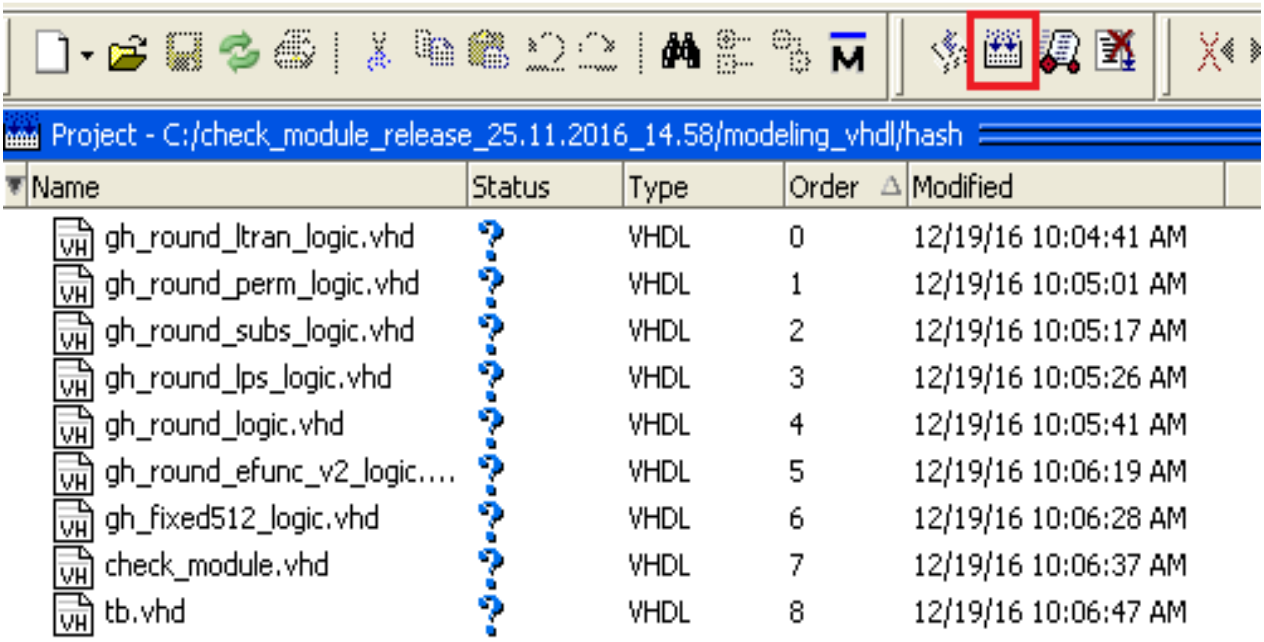

Рисунок 6 – Компиляция проекта

Результат успешной компиляции представлен на рисунке 7.

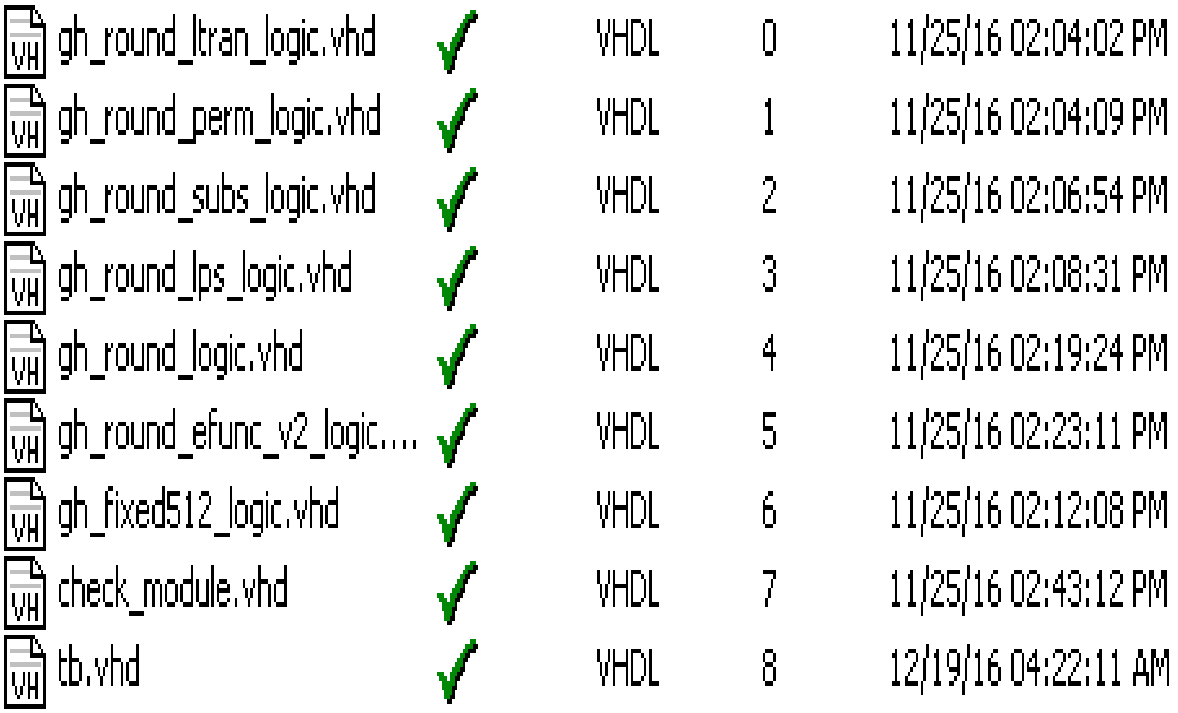

Рисунок 7 – Результат успешной компиляции

Для симуляции необходимо создать файл симуляции главного модуля «tb.vhd». Созданный файл симуляции представлен на рисунке 8.

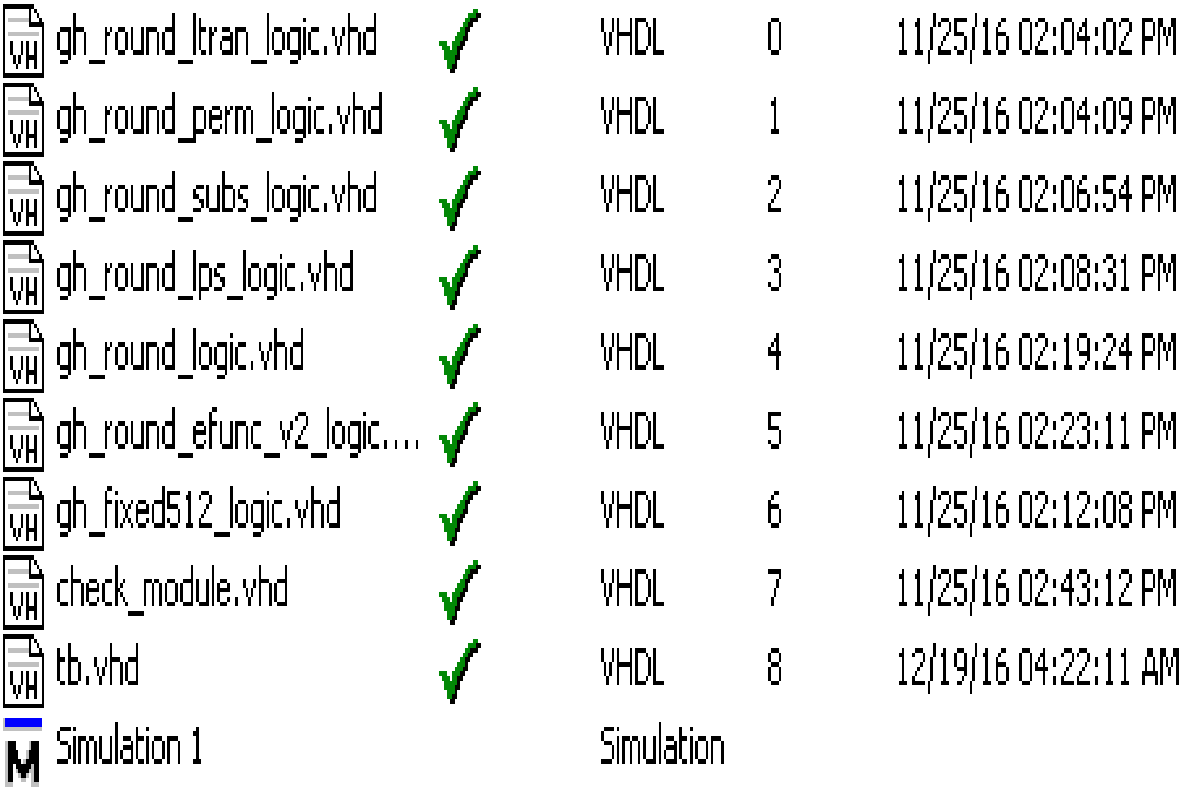

Рисунок 8 – Файл симуляции

После открытия этого файла необходимо добавить в симуляцию объекты модулей, как показано на рисунке 9.

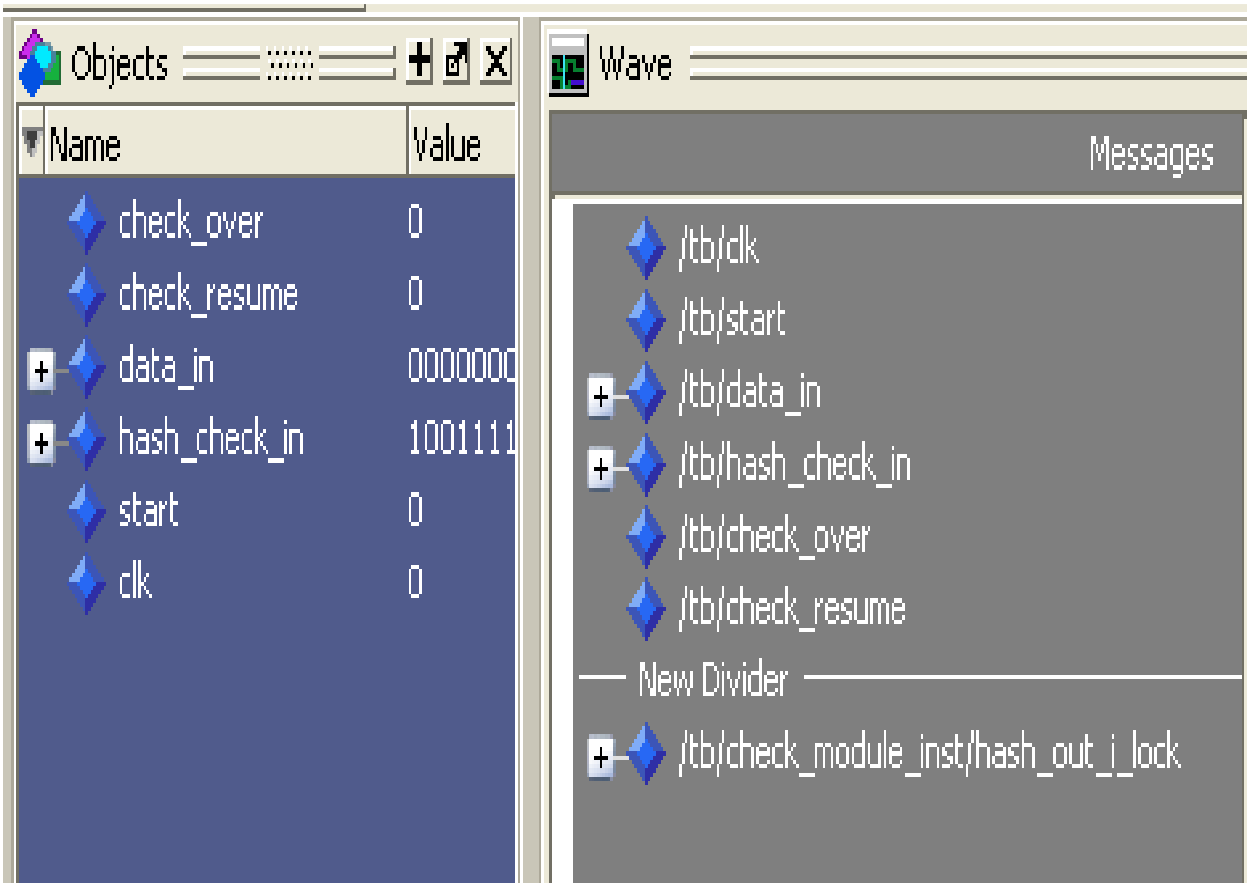

Рисунок 9 – Объекты модулей, добавленные в симуляцию

На рисунке 9 используются обозначения: «clk» – тактовая частота, «start» – сигнал запуска работы модуля, «data_in» – входные данные для расчета хэш-функции, «hash_check_in» – контрольное значение хэш-функции, «check_over» – сигнал завершения обработки данных, «check_resume» – признак вердикта о совпадении рассчитанного значения хэш-функции и контрольного значения (1 – совпадение, 0 – несовпадение), «hash_out_i_lock» – рассчитанное значение хэш-функции.

Для наглядной демонстрации работоспособности смоделированного узла входящие данные и контрольное значение хэш-функции были взяты из контрольного примера ГОСТ 34.11-2012 и введены вручную.

Результат симуляции с совпадением рассчитанного значения хэш-функции и контрольного значения представлен на рисунке 10.

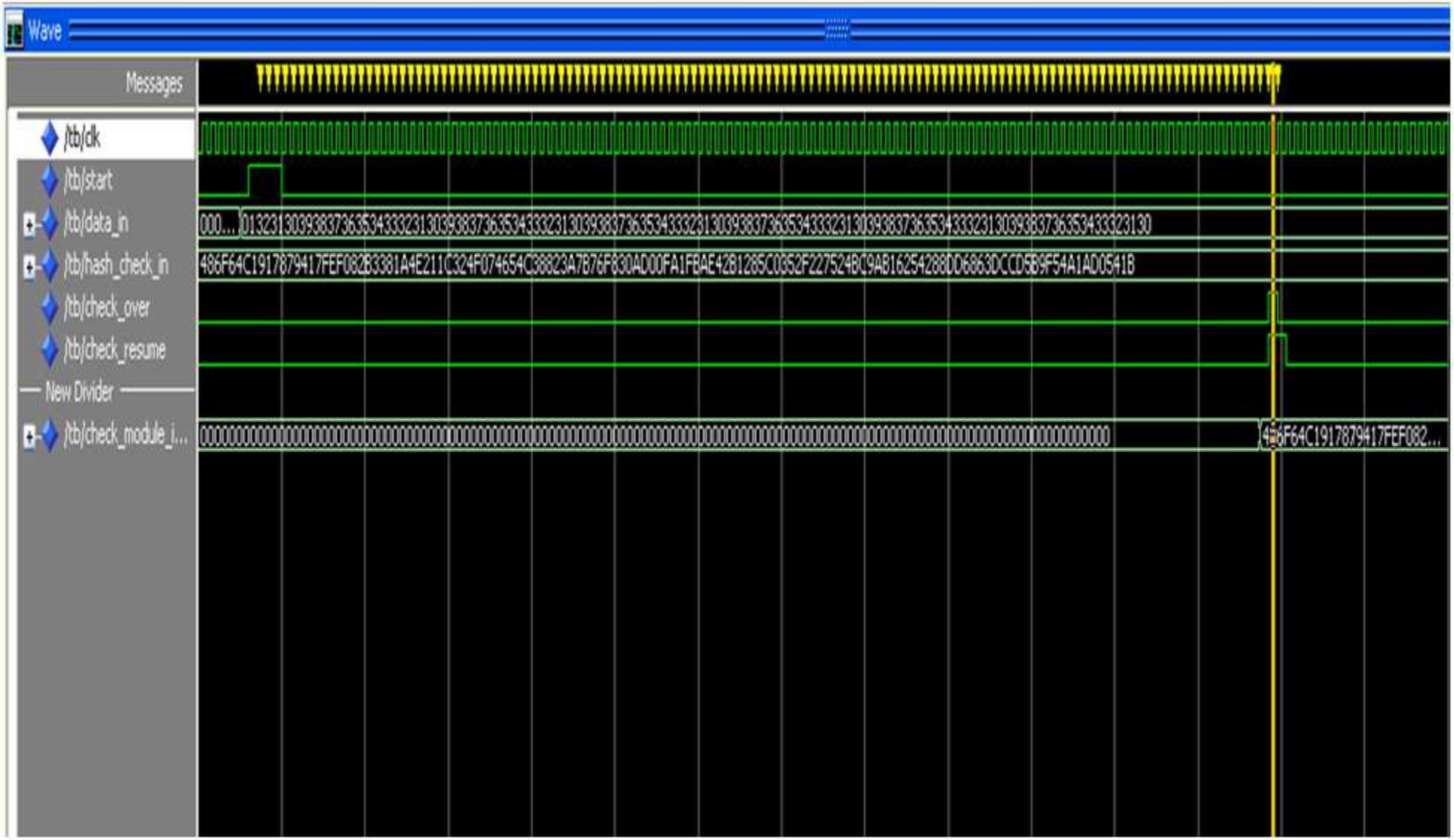

Рисунок 10 – Результат симуляции с совпадением рассчитанного значения хэш-функции и контрольного значения

На рисунке видно, что после сигнала запуска работы модуля произошло считывание входных данных. После сигнала завершения обработки данных признак вердикта о совпадении рассчитанного значения хэш-функции и контрольного значения установился в единицу, что означает, что рассчитанное значение хэш-функции и контрольное значение совпали. Более детально значения входных данных, контрольного значения, рассчитанного значения хэш-функции представлены на рисунках 11, 12, 13 соответственно.

Рисунок 11 – Входные данные

Для демонстрации несовпадения рассчитанного значения хэш-функции и контрольного значения во входных данных вручную были изменены входящие данные.

Результат симуляции с несовпадением рассчитанного значения хэш-функции и контрольного значения представлен на рисунке 14.

На рисунке 14 видно, что после сигнала запуска работы модуля произошло считывание входных данных. После сигнала завершения обработки данных признак вердикта о совпадении рассчитанного значения хэш-функции и контрольного значения установился в ноль, что означает, что рассчитанное значение хэш-функции и контрольное значение не совпали.

Более детально значения входных данных, контрольного значения, рассчитанного значения хэш-функции представлены на рисунках 15, 16, 17 соответственно.

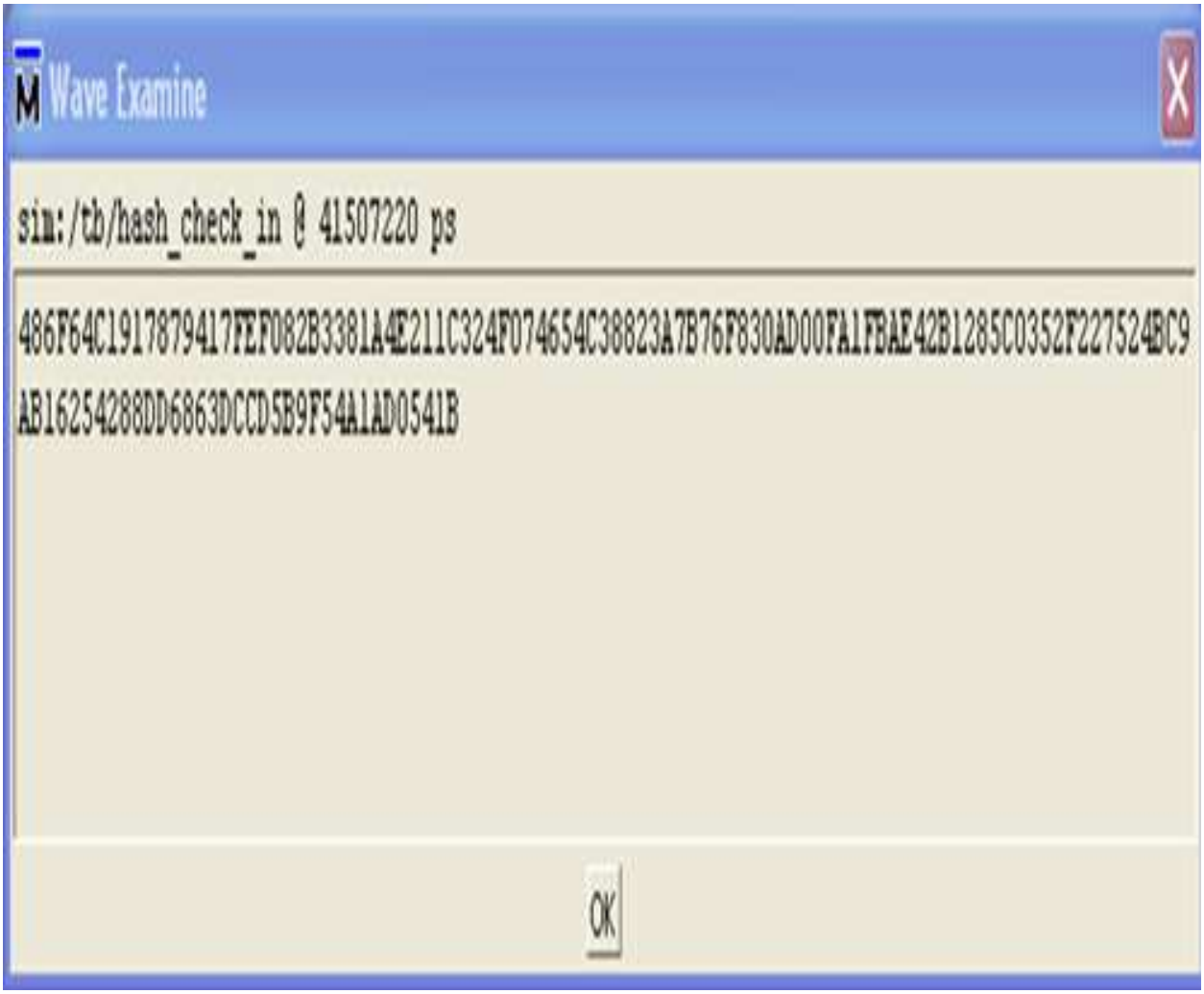

Рисунок 12 – Контрольное значение

Рисунок 13 – Рассчитанное значение хэш-функции

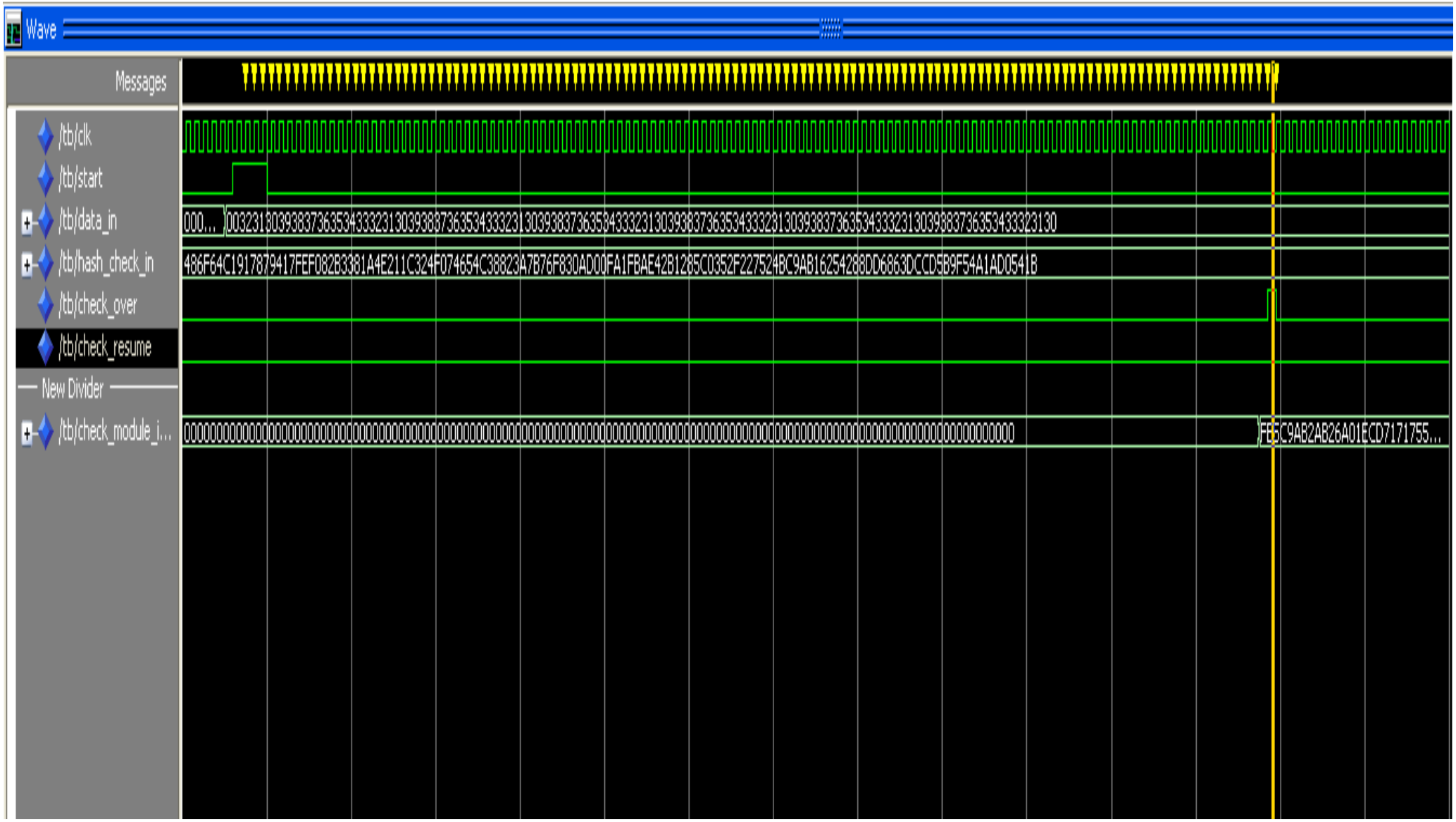

Рисунок 14 – Результат симуляции с несовпадением рассчитанного значения хэш-функции и контрольного значения

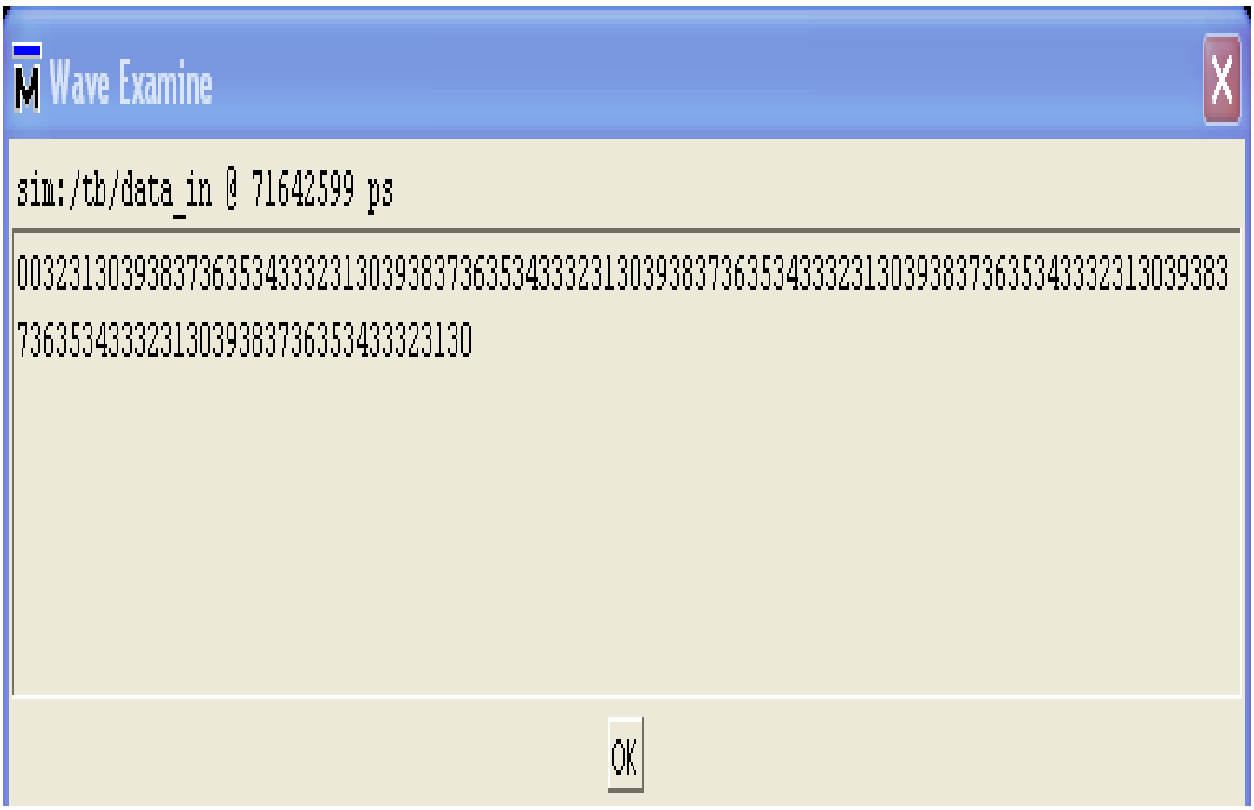

Рисунок 15 – Входные данные

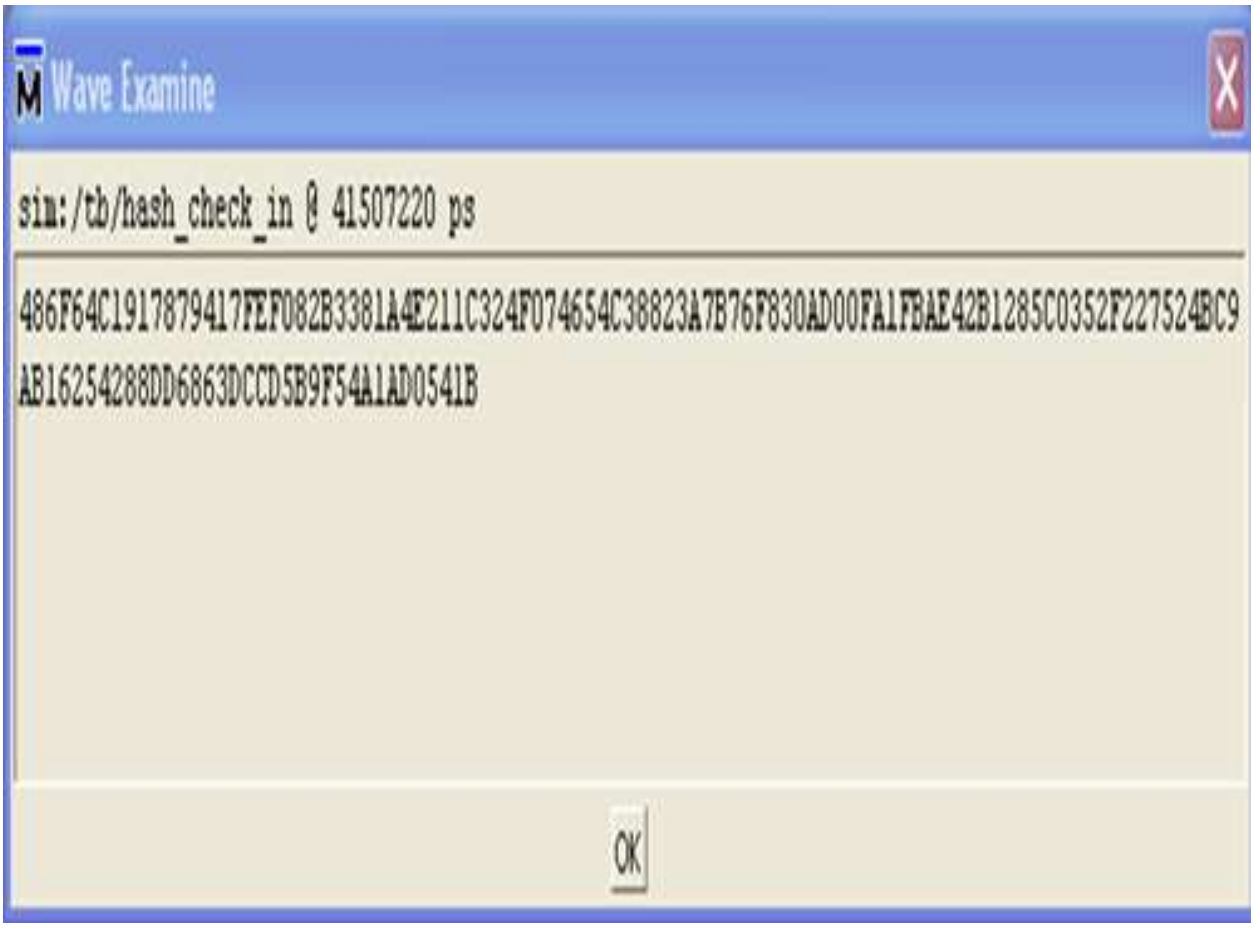

Рисунок 16 – Контрольное значение

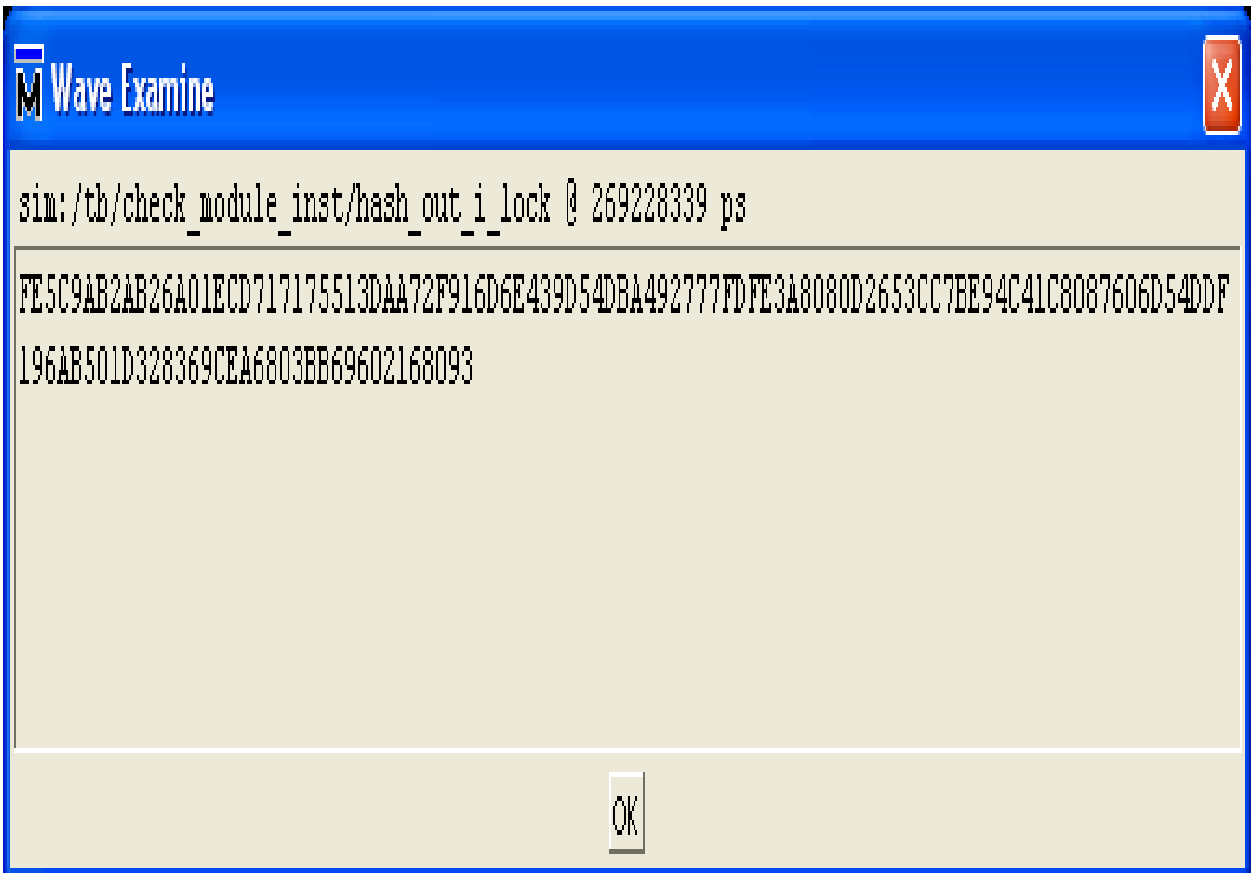

Рисунок 17 – Рассчитанное значение хэш-функции

ЗАКЛЮЧЕНИЕ

В результате выполнения ВКР был разработан узел контроля целостности данных на ПЛИС, который контролирует целостность программ, находящихся в ОЗУ.

Были разработаны структурная схема принципа использования в изделии узла, структурная схема узла. Был смоделирован узел контроля целостности данных в среде для проектирования «Quartus II» на языке описания аппаратуры VHDL. Были описаны модули, представлен результат моделирования узла.

Требования технического задания на дипломное проектирование выполнены в полном объеме.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Алферов А.П., Зубов А.Ю., Кузьмин А.С., Черемушкин А.В. Основы криптографии. Учебное пособие. Изд. 2-е, Гелиос АРВ, 2005. – 480 с.

2 ГОСТ 34.11-2012 «Информационная технология. Криптографическая защита информации. Функция хэширования».

3 Kazymyrov O., Kazymyrova V. Algebraic Aspects of the Russian Hash Standard GOST R 34.11-2012.

4 Матюхин Д. В., Шишкин В. А., Рудский В. И. Перспективный алгоритм хеширования.

5 Угрюмов Е. П. Цифровая схемотехника. Учебное пособие. Изд. 2-е, БХВ-Петербург, 2004. – 357 с.

6 Бибило П.Н. Основы языка VHDL. Изд. 3-е, ЛКИ, 2007. – 328 с.

7 ГТК РФ. Руководящий документ. «Защита от несанкционированного доступа к информации. Термины и определения».

8 ГОСТ Р МЭК 61508-4-2012 «Функциональная безопасность систем электрических, электронных, программируемых электронных связанных с безопасностью. Часть 4. Термины и определения».

ПРИЛОЖЕНИЕ А

(обязательное)

Схема структурная разработанного узла контроля целостности информации

ПРИЛОЖЕНИЕ Б

(обязательное)

Описание узла контроля целостности на языке VHDL

Модуль «tb»

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;


entity tb is
     port
     (
          check_over           : out std_logic := '0';
          check_resume         : out std_logic := '0'
     );
end tb;

architecture arch of tb is

component check_module is
     port
     (
          clk                       : in std_logic;
          data_in                   : in std_logic_vector(511 downto
0);
          hash_check_in        : in std_logic_vector(511 downto 0);
          start                : in std_logic;
          check_over           : out std_logic := '0';
          check_resume         : out std_logic := '0'
     );
end component check_module;
     signal data_in            : std_logic_vector(511 downto 0) :=
(others => '0');
     signal hash_check_in  : std_logic_vector(511 downto 0) :=
x"D32FA1AA3DF25B8853FE3E059D25C0E17238D3027EB271557949F248267FB89C4700
CFEC58DB03CB92F69C6D9B2CE7F008BD0C6D5306C859417463B2ECED692C";
     signal start              : std_logic := '0';
     signal clk                : std_logic := '0';



begin

     data_in(15 downto 0)  <= x"0000", x"1234" after 10 us;
     start                      <= '0', '1' after 12 us, '0' after 20
us;


process
begin
     wait for 1 us; clk <= not clk;
end process;

check_module_inst : check_module
     port map
     (
          clk                       => clk,
          data_in                   => data_in,
          hash_check_in        => hash_check_in,
```

```
            start                   => start,
            check_over              => check_over,
            check_resume            => check_resume
      );


end arch;
```

Модуль «check_module»

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;

entity check_module is
      port
      (
            clk                         : in std_logic;
            data_in                     : in std_logic_vector(511 downto
0);
            hash_check_in         : in std_logic_vector(511 downto 0);
            start                 : in std_logic;
            check_over            : out std_logic := '0';
            check_resume          : out std_logic := '0'
      );
end check_module;

architecture arch of check_module is

component gh_fixed512_logic
      port
      (
            clk                         : in std_logic;
            clken                 : in std_logic;
            data_in                     : in std_logic_vector(511 downto
0);
            valid_out             : out std_logic := '0';
            hash_out              : out std_logic_vector(511 downto 0)
      );
end component gh_fixed512_logic;
      signal enable               : std_logic := '0';
      signal data_in_i      : std_logic_vector(511 downto 0);
      signal valid_out_i          : std_logic := '0';
      signal hash_out_i           : std_logic_vector(511 downto 0) :=
(others => '0');
      signal hash_out_i_lock      : std_logic_vector(511 downto 0) :=
(others => '0');
      signal hash_check_in_i      : std_logic_vector(511 downto 0) :=
(others => '0');
      signal step                       : integer range 0 to 7 := 0;


begin

process(clk)
begin
```

```
      if( rising_edge(clk) ) then

            check_over                <= '0';

            case(step) is
            when 0 =>
                  check_resume        <= '0';
                        if( start = '1' ) then
                        enable                <= '1';
                        data_in_i       <= data_in;
                        hash_check_in_i <= hash_check_in;
                        step            <= 1;
                  else
                        enable                <= '0';
                  end if;
            when 1 =>
                  if( valid_out_i = '1' ) then
                        enable                <= '0';
                        hash_out_i_lock <= hash_out_i;
                        step            <= 2;
                  end if;
            when 2 =>
                  check_over            <= '1';
                  if( hash_out_i_lock = hash_check_in_i ) then
                        check_resume    <= '1';
                  else
                        check_resume    <= '0';
                  end if;
                  step                  <= 3;
            when 3 =>
                  if( start = '0' ) then
                        step            <= 0;
                  end if;
            when others => null;
            end case;


      end if;


end process;


gh_fixed512_logic_inst : gh_fixed512_logic

      port map
      (
            clk                         => clk,
            clken                 => enable,
            data_in                     => data_in_i,
            valid_out             => valid_out_i,
            hash_out              => hash_out_i
      );


end arch;
```

Модуль «gh_fixed512_logic»

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_arith.all;

entity gh_fixed512_logic is
port
(
     clk                        : in std_logic;
     clken                 : in std_logic;
     data_in                    : in std_logic_vector(511 downto 0);
     valid_out             : out std_logic := '0';
     hash_out              : out std_logic_vector(511 downto 0)
);
end gh_fixed512_logic;

architecture arch of gh_fixed512_logic is


component gh_round_logic is
port
(
     clk                        : in std_logic;
     clken                 : in std_logic;
     hash_in                    : in std_logic_vector(511 downto 0);
     numbits_in            : in std_logic_vector(511 downto 0);
     data_in                    : in std_logic_vector(511 downto 0);
     valid_out             : out std_logic := '0';
     hash_out              : out std_logic_vector(511 downto 0)
);
end component gh_round_logic;



signal after_data : std_logic_vector(511 downto 0);
signal after_numbits : std_logic_vector(511 downto 0);
signal vector_512 : std_logic_vector(511 downto 0);

type data_delay_type is array (0 to 79) of std_logic_vector(511 downto
0);
signal data_delay : data_delay_type;

signal valid_out_1 : std_logic := '0';
signal valid_out_2 : std_logic := '0';




begin

     vector_512      <= conv_std_logic_vector(512, 512);
```

```
stage_1_data : gh_round_logic
port map
(
      clk             => clk,
      clken                 => clken,
      hash_in         => (others => '0'),
      numbits_in=> (others => '0'),
      data_in         => data_in,
      valid_out       => valid_out_1,
      hash_out        => after_data
);

stage_2_numbits : gh_round_logic
port map
(
      clk             => clk,
      clken                 => valid_out_1, --clken,
      hash_in         => after_data,
      numbits_in=> (others => '0'),
      data_in         => vector_512,
      valid_out       => valid_out_2,
      hash_out        => after_numbits
);

stage_3_sumbits

process(clk)

begin
     if( rising_edge(clk) ) then
          if( clken = '1' ) then
               data_delay(1 to 79) <= data_delay(0 to 78);
               data_delay(0)   <= data_in;
          end if;
     end if;
end process;



stage_3_sumbits : gh_round_logic

port map
(
      clk             => clk,
      clken                 => valid_out_2, --clken,
      hash_in         => after_numbits,
      numbits_in=> (others => '0'),
      data_in         => data_delay(79),
      valid_out       => valid_out,
      hash_out        => hash_out
);


end arch;
```

Модуль «gh_round_efunc_v2_logic»

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;

entity gh_round_efunc_v2_logic is
port
(
     clk                        : in std_logic;
     clken                 : in std_logic;
     hash_in                    : in std_logic_vector(511 downto 0);
     data_in                    : in std_logic_vector(511 downto 0);
     valid_out             : out std_logic := '0';
     hash_out              : out std_logic_vector(511 downto 0)
);
end gh_round_efunc_v2_logic;

architecture arch of gh_round_efunc_v2_logic is

     type const_c_type is array (1 to 12) of std_logic_vector(511
downto 0);
     constant const_c : const_c_type := (

x"b1085bda1ecadae9ebcb2f81c0657c1f2f6a76432e45d016714eb88d7585c4fc4b7c
e09192676901a2422a08a460d31505767436cc744d23dd806559f2a64507",

x"6fa3b58aa99d2f1a4fe39d460f70b5d7f3feea720a232b9861d55e0f16b501319ab5
176b12d699585cb561c2db0aa7ca55dda21bd7cbcd56e679047021b19bb7",

x"f574dcac2bce2fc70a39fc286a3d843506f15e5f529c1f8bf2ea7514b1297b7bd3e2
0fe490359eb1c1c93a376062db09c2b6f443867adb31991e96f50aba0ab2",

x"ef1fdfb3e81566d2f948e1a05d71e4dd488e857e335c3c7d9d721cad685e353fa9d7
2c82ed03d675d8b71333935203be3453eaa193e837f1220cbebc84e3d12e",

x"4bea6bacad4747999a3f410c6ca923637f151c1f1686104a359e35d7800fffbdbfcd
1747253af5a3dfff00b723271a167a56a27ea9ea63f5601758fd7c6cfe57",

x"ae4faeae1d3ad3d96fa4c33b7a3039c02d66c4f95142a46c187f9ab49af08ec6cffa
a6b71c9ab7b40af21f66c2bec6b6bf71c57236904f35fa68407a46647d6e",

x"f4c70e16eeaac5ec51ac86febf240954399ec6c7e6bf87c9d3473e33197a93c90992
abc52d822c3706476983284a05043517454ca23c4af38886564d3a14d493",

x"9b1f5b424d93c9a703e7aa020c6e41414eb7f8719c36de1e89b4443b4ddbc49af489
2bcb929b069069d18d2bd1a5c42f36acc2355951a8d9a47f0dd4bf02e71e",

x"378f5a541631229b944c9ad8ec165fde3a7d3a1b258942243cd955b7e00d0984800a
440bdbb2ceb17b2b8a9aa6079c540e38dc92cb1f2a607261445183235adb",

x"abbedea680056f52382ae548b2e4f3f38941e71cff8a78db1fffe18a1b3361039fe7
6702af69334b7a1e6c303b7652f43698fad1153bb6c374b4c7fb98459ced",

x"7bcd9ed0efc889fb3002c6cd635afe94d8fa6bbbebab07612001802114846679 8a1d
71efea48b9caefbacd1d7d476e98dea2594ac06fd85d6bcaa4cd81f32d1b",
```

```
x"378ee767f11631bad21380b00449b17acda43c32bcdf1d77f82012d430219f9b5d80
ef9d1891cc86e71da4aa88e12852faf417d5d9b21b9948bc924af11bd720"
      );

      type m12_type is array (1 to 12) of std_logic_vector(511 downto
0);
      type m13_type is array (1 to 13) of std_logic_vector(511 downto
0);
      type b13_type is array (1 to 13) of std_logic;

      signal val_x : m13_type;
      signal val_kc : m12_type;

      signal val_k : m13_type;
      signal val_lps : m13_type;

      signal clken_i : b13_type := (others => '0');
      signal valid_out_i : b13_type := (others => '0');

component gh_round_lps_logic
port
(
      clk                         : in std_logic;
      clken                 : in std_logic;
      lps_arg                     : in std_logic_vector(511 downto 0);
      valid_out             : out std_logic := '0';
      lps_func              : out std_logic_vector(511 downto 0)
);
end component gh_round_lps_logic;



begin

      hash_out <= val_x(13);

      val_lps(1) <= data_in;
      val_k(1) <= hash_in;
      valid_out_i(1) <= clken;
      clken_i(1) <= clken;

generate_cycle_j: for g1 in 2 to 13 generate
      process(clk)
      begin
            if(rising_edge(clk)) then
                              if( clken = '1' ) then
                        val_x(g1-1) <= val_k(g1-1) xor val_lps(g1-1);
                        val_kc(g1-1) <= val_k(g1-1) xor const_c(g1-1);
                        clken_i(g1) <= valid_out_i(g1-1);
                  end if;
            end if;
      end process;

      gh_round_lps_logic_main_inst : gh_round_lps_logic

      port map
      (
```

```
            clk                         => clk,
            clken                             => clken,
            lps_arg                     => val_x(g1-1),
            valid_out                   => valid_out_i(g1),
            lps_func                    => val_lps(g1)
      );

      gh_round_lps_logic_k_inst : gh_round_lps_logic
      port map
      (
            clk                         => clk,
            clken                             => clken,
            --clken                           => clken_i(g1-1),
            lps_arg                     => val_kc(g1-1),
            valid_out                   => open, --valid_out_i(g1),
            lps_func                    => val_k(g1)
      );

end generate generate_cycle_j;

process(clk)
begin
      if(rising_edge(clk)) then
            if( clken = '1' ) then
                  val_x(13) <= val_k(13) xor val_lps(13);
                  valid_out <= valid_out_i(13);
            else
                  valid_out <= '0';
            end if;
      end if;
end process;

end arch;
```

Модуль «gh_round_logic»

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;

entity gh_round_logic is
port
(
      clk                               : in std_logic;
      clken                       : in std_logic;
      hash_in                           : in std_logic_vector(511 downto 0);
      numbits_in                  : in std_logic_vector(511 downto 0);
      data_in                           : in std_logic_vector(511 downto 0);
      valid_out                   : out std_logic := '0';
      hash_out                    : out std_logic_vector(511 downto 0)
);
end gh_round_logic;

architecture arch of gh_round_logic is


component gh_round_lps_logic is
```

```
port
(
      clk                        : in std_logic;
      clken                 : in std_logic;
      lps_arg                    : in std_logic_vector(511 downto 0);
      valid_out             : out std_logic := '0';
      lps_func              : out std_logic_vector(511 downto 0)
);
end component gh_round_lps_logic;

component gh_round_efunc_v2_logic is
port
(
      clk                        : in std_logic;
      clken                 : in std_logic;
      hash_in                    : in std_logic_vector(511 downto 0);
      data_in                    : in std_logic_vector(511 downto 0);
      valid_out             : out std_logic := '0';
      hash_out              : out std_logic_vector(511 downto 0)
);
end component gh_round_efunc_v2_logic;



signal first_lps_func : std_logic_vector(511 downto 0);

type delay_type is array (0 to 1) of std_logic_vector(511 downto 0);
signal data_in_delay : delay_type;
signal hash_in_delay : delay_type;

type dxorh_delay_type is array (0 to 36) of std_logic_vector(511
downto 0);
signal dxorh_delay : dxorh_delay_type;
signal efunc_out : std_logic_vector(511 downto 0);
signal hash_out_ff : std_logic_vector(511 downto 0);

signal temp_xor1 : std_logic_vector(511 downto 0);
signal temp_xor2 : std_logic_vector(511 downto 0);

type valid_out_delay_type is array (0 to 36) of std_logic;
signal valid_out_delay : valid_out_delay_type := (others => '0');

signal valid_lps : std_logic := '0';
signal valid_efunc_v2 : std_logic := '0';


begin

      temp_xor1       <= hash_in xor numbits_in;

gh_round_lps_logic_inst : gh_round_lps_logic
port map
(
      clk                        => clk,
      clken                 => clken,
      lps_arg                    => temp_xor1,
      valid_out             => valid_lps,
      lps_func              => first_lps_func
```

```
);


process(clk)
begin
     if( rising_edge(clk) ) then
          if( clken = '1' ) then
               data_in_delay(1) <= data_in_delay(0);
               data_in_delay(0) <= data_in;
               hash_in_delay(1) <= hash_in_delay(0);
               hash_in_delay(0) <= hash_in;
          end if;
     end if;
end process;

     temp_xor2                 <= data_in_delay(1) xor
hash_in_delay(1);

process(clk)
begin
     if( rising_edge(clk) ) then
          if( clken = '1' ) then
               dxorh_delay(1 to 36) <= dxorh_delay(0 to 35);
               dxorh_delay(0)  <= temp_xor2;
               valid_out_delay(1 to 36) <= valid_out_delay(0 to 35);
               valid_out_delay(0) <= valid_efunc_v2;
          else
               valid_out_delay(0 to 36) <= (others => '0');
          end if;
     end if;
end process;

gh_round_efunc_v2_logic_inst : gh_round_efunc_v2_logic
port map
(
      clk            => clk,
      clken               => clken,
      hash_in        => first_lps_func,
      data_in        => data_in_delay(1),
          valid_out  => valid_efunc_v2,
      hash_out       => efunc_out
);

process(clk)
begin
     if( rising_edge(clk) ) then
          -- проверка сигнала clken (разрешения обработки данных)
          if( clken = '1' ) then
               hash_out_ff <= efunc_out xor dxorh_delay(36);
               valid_out <= valid_out_delay(36);
          else
               valid_out <= '0';
          end if;
     end if;
end process;

     hash_out <= hash_out_ff;
```

```
end arch;
```

Модуль «gh_round_lps_logic»

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;

entity gh_round_lps_logic is
port
(
     clk                        : in std_logic;
     clken                 : in std_logic;
     lps_arg                    : in std_logic_vector(511 downto 0);
     valid_out             : out std_logic := '0';
     lps_func              : out std_logic_vector(511 downto 0)
);
end gh_round_lps_logic;

architecture arch of gh_round_lps_logic is

component gh_round_subs_logic is
port
(
     clk                        : in std_logic;
     clken                 : in std_logic;
     lps_arg                    : in std_logic_vector(511 downto 0);
     valid_out             : out std_logic := '0';
     after_s                    : out std_logic_vector(511 downto 0)
);
end component gh_round_subs_logic;
     signal after_s             : std_logic_vector(511 downto 0);
     signal valid_out_subs : std_logic := '0';

component gh_round_perm_logic is
port
(
     after_s                    : in std_logic_vector(511 downto 0);
     after_p                    : out std_logic_vector(511 downto 0)
);
end component gh_round_perm_logic;
     signal after_p             : std_logic_vector(511 downto 0);

component gh_round_ltran_logic is
port
(
     clk                        : in std_logic;
     clken                 : in std_logic;
     after_p                    : in std_logic_vector(511 downto 0);
     valid_out             : out std_logic := '0';
     lps_func              : out std_logic_vector(511 downto 0)
);
end component gh_round_ltran_logic;


begin
```

```
gh_round_subs_logic_inst : gh_round_subs_logic
port map
(
      clk                        => clk,
      clken                           => clken,
      lps_arg                    => lps_arg,
      valid_out                  => valid_out_subs,
      after_s                    => after_s
);

gh_round_perm_logic_inst : gh_round_perm_logic
port map
(
      after_s                    => after_s,
      after_p                    => after_p
);

gh_round_ltran_logic_inst : gh_round_ltran_logic
port map
(
      clk                        => clk,
      clken                           => clken,
      --clken                         => valid_out_subs,
      after_p                    => after_p,
      valid_out                  => valid_out,
      lps_func                   => lps_func
);


end arch;
```

Модуль «gh_round_ltran_logic»

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;

entity gh_round_ltran_logic is
port
(
     clk                         : in std_logic;
     clken                  : in std_logic;
     after_p                     : in std_logic_vector(511 downto 0);
     valid_out              : out std_logic := '0';
     lps_func               : out std_logic_vector(511 downto 0)
);
end gh_round_ltran_logic;

architecture arch of gh_round_ltran_logic is

type table_a_type is array (0 to 63) of std_logic_vector(63 downto 0);
constant table_a: table_a_type := (
x"8e20faa72ba0b470", x"47107ddd9b505a38", x"ad08b0e0c3282d1c",
x"d8045870ef14980e",
x"6c022c38f90a4c07", x"3601161cf205268d", x"1b8e0b0e798c13c8",
x"83478b07b2468764",
```

```
x"a011d380818e8f40", x"5086e740ce47c920", x"2843fd2067adea10",
x"14aff010bdd87508",
x"0ad97808d06cb404", x"05e23c0468365a02", x"8c711e02341b2d01",
x"46b60f011a83988e",
x"90dab52a387ae76f", x"486dd4151c3dfdb9", x"24b86a840e90f0d2",
x"125c354207487869",
x"092e94218d243cba", x"8a174a9ec8121e5d", x"4585254f64090fa0",
x"accc9ca9328a8950",
x"9d4df05d5f661451", x"c0a878a0a1330aa6", x"60543c50de970553",
x"302a1e286fc58ca7",
x"18150f14b9ec46dd", x"0c84890ad27623e0", x"0642ca05693b9f70",
x"0321658cba93c138",
x"86275df09ce8aaa8", x"439da0784e745554", x"afc0503c273aa42a",
x"d960281e9d1d5215",
x"e230140fc0802984", x"71180a8960409a42", x"b60c05ca30204d21",
x"5b068c651810a89e",
x"456c34887a3805b9", x"ac361a443d1c8cd2", x"561b0d22900e4669",
x"2b838811480723ba",
x"9bcf4486248d9f5d", x"c3e9224312c8c1a0", x"effa11af0964ee50",
x"f97d86d98a327728",
x"e4fa2054a80b329c", x"727d102a548b194e", x"39b008152acb8227",
x"9258048415eb419d",
x"492c024284fbaec0", x"aa16012142f35760", x"550b8e9e21f7a530",
x"a48b474f9ef5dc18",
x"70a6a56e2440598e", x"3853dc371220a247", x"1ca76e95091051ad",
x"0edd37c48a08a6d8",
x"07e095624504536c", x"8d70c431ac02a736", x"c83862965601dd1b",
x"641c314b2b8ee083"
     );

type mass_64x7_type is array (0 to 63) of std_logic_vector(7 downto
0);
signal state_in : mass_64x7_type;
signal state_out : mass_64x7_type;
signal state_ff : mass_64x7_type;

function func_many_cycles(state_in: mass_64x7_type) return
mass_64x7_type is
    variable v : std_logic_vector(63 downto 0);
    variable temp1 : std_logic_vector(7 downto 0);
    variable temp2 : std_logic_vector(7 downto 0);
    variable temp3 : std_logic_vector(7 downto 0);
    variable temp11 : integer range 0 to 63;
    variable temp12 : integer range 0 to 63;
     variable state_out_v : mass_64x7_type;
begin
for i in 0 to 7 loop
     v := (others => '0');

     for k in 0 to 7 loop
          for j in 0 to 7 loop
               if (state_in(63-(i*8+k))(7-j) /= '0' and state_in(63-
(i*8+k))(7-j) /= 'U' and state_in(63-(i*8+k))(7-j) /= 'Z' and
state_in(63-(i*8+k))(7-j) /= 'X' ) then
                    v := v xor table_a(k*8+j);
               end if;
          end loop;
     end loop;
```

```
      for kk in 0 to 7 loop
            temp11 := (7-kk)*8;
            temp3 := v(temp11+7 downto temp11);
            state_out_v(63-(i*8+kk)) := temp3;
      end loop;
end loop;
return state_out_v;
end func_many_cycles;

begin

fill_inputs_cycle_j: for in_j in 0 to 63 generate
      state_in(in_j)  <= after_p((7+in_j*8) downto in_j*8);
end generate fill_inputs_cycle_j;

fill_outputs_cycle_j: for in_j in 0 to 63 generate
      lps_func((7+in_j*8) downto in_j*8) <= state_ff(in_j);
end generate fill_outputs_cycle_j;

process(clk)
begin
      if( rising_edge(clk) ) then
            if( clken = '1' ) then
                  state_ff <= func_many_cycles(state_in);
                  valid_out <= '1';
            else
                  valid_out <= '0';
            end if;
      end if;
end process;

end arch;
```

Модуль «gh_round_perm_logic»

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;

entity gh_round_perm_logic is
port
(
      after_s                    : in std_logic_vector(511 downto 0);
      after_p                    : out std_logic_vector(511 downto 0)
);
end gh_round_perm_logic;

architecture arch of gh_round_perm_logic is

type mass_7x7_type is array (0 to 7) of std_logic_vector(7 downto 0);
type mass_7x7x7_type is array (0 to 7) of mass_7x7_type;

signal after_s_int : mass_7x7x7_type := (others => (others => (others
=> '0')));
signal after_p_int : mass_7x7x7_type := (others => (others => (others
=> '0')));
```

```
begin

fill_inputs_cycle_i: for in_i in 0 to 7 generate
     fill_inputs_cycle_j: for in_j in 0 to 7 generate
          after_s_int(in_i)(in_j)    <= after_s((7+in_i*64+in_j*8)
downto in_i*64+in_j*8);
     end generate fill_inputs_cycle_j;
end generate fill_inputs_cycle_i;

rotate_cycle_i: for in_i in 0 to 7 generate
     rotate_cycle_j: for in_j in 0 to 7 generate
          after_p_int(in_j)(in_i) <= after_s_int(in_i)(in_j);
     end generate rotate_cycle_j;
end generate rotate_cycle_i;

fill_outputs_cycle_i: for in_i in 0 to 7 generate
     fill_outputs_cycle_j: for in_j in 0 to 7 generate
          after_p((7+in_i*64+in_j*8) downto in_i*64+in_j*8) <=
after_p_int(in_i)(in_j);
     end generate fill_outputs_cycle_j;
end generate fill_outputs_cycle_i;

end arch;
```

Модуль «gh_round_subs_logic»

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;

entity gh_round_subs_logic is
port
(
     clk                         : in std_logic;
     clken                  : in std_logic;
     lps_arg                     : in std_logic_vector(511 downto 0);
     valid_out              : out std_logic := '0';
     after_s                     : out std_logic_vector(511 downto 0)
);
end gh_round_subs_logic;

architecture arch of gh_round_subs_logic is

type sbox_table_type is array (0 to 255) of std_logic_vector(7 downto
0);
constant sbox_table   : sbox_table_type := (
     x"FC", x"EE", x"DD", x"11", x"CF", x"6E", x"31", x"16", x"FB",
x"C4", x"FA", x"DA", x"23", x"C5", x"04", x"4D",
     x"E9", x"77", x"F0", x"DB", x"93", x"2E", x"99", x"BA", x"17",
x"36", x"F1", x"BB", x"14", x"CD", x"5F", x"C1",
     x"F9", x"18", x"65", x"5A", x"E2", x"5C", x"EF", x"21", x"81",
x"1C", x"3C", x"42", x"8B", x"01", x"8E", x"4F",
     x"05", x"84", x"02", x"AE", x"E3", x"6A", x"8F", x"A0", x"06",
x"0B", x"ED", x"98", x"7F", x"D4", x"D3", x"1F",
     x"EB", x"34", x"2C", x"51", x"EA", x"C8", x"48", x"AB", x"F2",
x"2A", x"68", x"A2", x"FD", x"3A", x"CE", x"CC",
```

```
      x"B5", x"70", x"0E", x"56", x"08", x"0C", x"76", x"12", x"BF",
x"72", x"13", x"47", x"9C", x"B7", x"5D", x"87",
      x"15", x"A1", x"96", x"29", x"10", x"7B", x"9A", x"C7", x"F3",
x"91", x"78", x"6F", x"9D", x"9E", x"B2", x"B1",
      x"32", x"75", x"19", x"3D", x"FF", x"35", x"8A", x"7E", x"6D",
x"54", x"C6", x"80", x"C3", x"BD", x"0D", x"57",
      x"DF", x"F5", x"24", x"A9", x"3E", x"A8", x"43", x"C9", x"D7",
x"79", x"D6", x"F6", x"7C", x"22", x"B9", x"03",
      x"E0", x"0F", x"EC", x"DE", x"7A", x"94", x"B0", x"BC", x"DC",
x"E8", x"28", x"50", x"4E", x"33", x"0A", x"4A",
      x"A7", x"97", x"60", x"73", x"1E", x"00", x"62", x"44", x"1A",
x"B8", x"38", x"82", x"64", x"9F", x"26", x"41",
      x"AD", x"45", x"46", x"92", x"27", x"5E", x"55", x"2F", x"8C",
x"A3", x"A5", x"7D", x"69", x"D5", x"95", x"3B",
      x"07", x"58", x"B3", x"40", x"86", x"AC", x"1D", x"F7", x"30",
x"37", x"6B", x"E4", x"88", x"D9", x"E7", x"89",
      x"E1", x"1B", x"83", x"49", x"4C", x"3F", x"F8", x"FE", x"8D",
x"53", x"AA", x"90", x"CA", x"D8", x"85", x"61",
      x"20", x"71", x"67", x"A4", x"2D", x"2B", x"09", x"5B", x"CB",
x"9B", x"25", x"D0", x"BE", x"E5", x"6C", x"52",
      x"59", x"A6", x"74", x"D2", x"E6", x"F4", x"B4", x"C0", x"D1",
x"66", x"AF", x"C2", x"39", x"4B", x"63", x"B6"
      );

type mass_64x7_type is array (0 to 63) of std_logic_vector(7 downto
0);
signal data_in_int : mass_64x7_type;
signal data_out_ff : mass_64x7_type;
type mass_64x1_type is array (0 to 63) of std_logic;
signal valid_i : mass_64x1_type := (others => '0');

begin

fill_inputs_cycle_j: for in_j in 0 to 63 generate
      data_in_int(in_j)     <= lps_arg((7+in_j*8) downto in_j*8);
end generate fill_inputs_cycle_j;

fill_outputs_cycle_j: for in_j in 0 to 63 generate
      after_s((7+in_j*8) downto in_j*8) <= data_out_ff(in_j);
end generate fill_outputs_cycle_j;

      valid_out <= '1' when valid_i(0 to 63) = x"FFFFFFFFFFFFFFFF";

g_cycle: for g1 in 0 to 63 generate
      process( clk )
      begin
            if( rising_edge(clk) ) then
                        if( clken = '1' ) then
                        if( data_in_int(g1) /= "UUUUUUUU" and
data_in_int(g1) /= "ZZZZZZZZ" and data_in_int(g1) /= "XXXXXXXX" ) then
                              data_out_ff(g1) <=
sbox_table(conv_integer(data_in_int(g1)));
                              valid_i(g1) <= '1';
                        end if;
                  else
                        valid_i(g1) <= '0';
                  end if;
            end if;
```

```
      end process;
      end generate;

end arch;
```

ПРИЛОЖЕНИЕ В

(обязательное)

Результаты проверки в системе «Антиплагиат»

ПРОТОКОЛ

проверки ВКР на оригинальность в системе «Антиплагиат» выпускной квалификационной работы на тему «Узел обнаружения искажения данных в памяти компьютера» студента 6 курса Н. С. Никулина специальности 100502 «Информационная безопасность телекоммуникационных систем».

Результаты прохождения ВКР в системе «Антиплагиат» представлены на рисунке В1.

Руководитель ВКР: к.т.н., инженер I категории АО «ПНИЭИ» Кулагин О.В.

Информация о документе:

Имя исходного файла:	ВКР_11ПТ1_Никулин_НС.docx
Имя компании:	Пензенский гос. Университет
Тип документа:	Прочее
Имя документа:	Группа 11ПТ1 ВКР Никулин НС
Дата проверки:	11.01.2017 16:16
Модули поиска:	Интернет (Антиплагиат), Диссертации и авторефераты РГБ, Модуль поиска ЭБС БиблиоРоссика, Модуль поиска ЭБС "Лань", Университетская библиотека онлайн, Кольцо вузов, Пензенский гос. Университет

Текстовые статистики:

Индекс читаемости:	сложный
Неизвестные слова:	в пределах нормы
Макс. длина слова:	в пределах нормы
Большие слова:	в пределах нормы

Оригинальные блоки: 70,06%

Заимствованные блоки: 29,94%

Заимствование из "белых" источников: 0%

Итоговая оценка оригинальности: 70,06%

Рисунок В1 – Результаты прохождения ВКР в системе «Антиплагиат»

«12» 01 2017 г. _____ О. В. Кулагин

АКЦИОНЕРНОЕ ОБЩЕСТВО ГОРЕНЗА
«Пензенский научно-исследовательский электротехнический институт»
ОТДЕЛ КАДРОВ
АО «ПНИЭИ»