МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ» (НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)

Факультет информационных технологий

Кафедра информационно-измерительных систем

Направление подготовки: 230100 ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА

Магистерская программа: «Информационно-измерительные системы»

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

Высокопроизводительная обработка данных электромагнитного каротажа

Персов Егор Михайлович

Тема диссертации утверждена распоряжением по НГУ № 536 от «14» декабря 2012г.

«К защите допущена»

Заведующий кафедрой,

д.т.н., профессор

Потатуркин О.И./……..

(фамилия , И., О.) / (подпись, МП)

«……»……20…г.

Научный руководитель

н.с. ИАиЭ СО РАН

к.т.н.

Лысаков К.Ф./……...

(фамилия , И., О.) / (подпись, МП)

«……»……20…г.

Дата защиты: «19» июня 2014г.

Автор Персов Е.М./……...

(фамилия , И., О.) / (подпись)

Новосибирск, 2014г.

МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ» (НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)

Факультет информационных технологий

Кафедра информационно-измерительных систем

Направление подготовки: 230100 ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА

Магистерская программа: «Информационно-измерительные системы»

УТВЕРЖДАЮ

Зав. Кафедрой Потатуркин О.И.
(фамилия, И., О.)
…….
(подпись, МП)
«……»……20…г.

ЗАДАНИЕ

НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ

МАГИСТЕРСКУЮ ДИССЕРТАЦИЮ

Студенту Персову Егору Михайловичу

Тема «Высокопроизводительная обработка данных электромагнитного каротажа»

Исходные данные (или цель работы):

Для ускорения обработки данных, полученных путём электромагнитных каротажных исследований, было предложено создать высокопроизводительный программно-аппаратный комплекс для решения нейросетевого аналога прямой задачи ВИКИЗ.

Структурные части работы:

Основными этапами работы являются: анализ алгоритма и выдвигаемых к комплексу требований, выбор аппаратной платформы комплекса, представления алгоритма в терминах выбранной платформы, разработка программной и аппаратной частей комплекса.

Научный руководитель
н.с. ИАиЭ СО РАН
к.т.н.
Лысаков К.Ф./……..
(фамилия , И., О.) / (подпись)
«…»……20…г.

Задание принял к исполнению
Персов Е.М./……..
(ФИО студента) / (подпись)
«…»……20…г.

Содержание

Введение

Одной из основных задач современных геофизических исследований является интерпретация экспериментальных данных. Однако сложность решения данной задачи породила множество различных подходов к её решению. Многие алгоритмы базируются на минимизации отклонений синтетических сигналов от измеренных величин в пространстве модельных параметров. В геоэлектрике процесс минимизации является целенаправленным итерационным перебором сигналов при различных значениях геоэлектрических и геометрических характеристик среды, при этом эффективность и скорость инверсии в первую очередь обусловлены ресурсными характеристиками (быстродействие, используемая память) программ решения соответствующих прямых задач. Поэтому проблема ускорения при вычислении синтетических каротажных диаграмм является предметом усилий многих исследователей.

Одним из перспективных способов приближенного моделирования является метод нейронных сетей. Данная работа является развитием идеи подмены прямой задачи ВИКИЗ её нейросетевым аналогом при классическом процессе минимизации. Применимость и эффективность нейросетевых методов в данной области были показаны в работах Соболева А.Ю., а также им был построен нейросетевой аналог прямой задачи ВИКИЗ. Разработанный алгоритм открыл новые перспективы по созданию специализированного вычислителя, позволяющего производить часть работ по интерпретации экспериментальных данных непосредственно в момент их сбора на базе каротажного комплекса.

Целью данной работы является создание высокопроизводительного программно-аппаратного комплекса для решения прямой нейросетевой задачи ВИКИЗ.

Для достижения поставленной цели необходимо решить следующие задачи:

- Выбор аппаратной платформы на основании анализа алгоритма и условий эксплуатации разрабатываемого устройства;
- Представление алгоритма в терминах выбранной аппаратной платформы;
- Формирование аппаратной базы комплекса;
- Разработка программного обеспечения, необходимого для использования вычислителя;
- Исследование созданного программно-аппаратного комплекса, оценка полученных результатов.

Данная работа позволит оценить целесообразность дальнейшего развития метода подмены прямой задачи ВИКИЗ нейросетевым аналогом в направлении создания специализированного вычислителя.

Работа представлена в трёх главах: первая глава посвящена выбору аппаратной платформы, вторая – представлению алгоритма в терминах выбранной платформы, третья – разработке самого вычислителя и сопутствующего ПО.

Глава 1

1.1 Выбор аппаратной платформы

Данная работа направлена на изучение потенциальных возможностей алгоритма, разработанного А. Ю. Соболевым, исследование его применимости на базе каротажного комплекса в реальном времени, выбор направления развития аппаратной части реализации с дальнейшей целью создания магнитно-каротажного комплекса с интегрированной реализацией алгоритма. При разработке программно-аппаратного комплекса со стороны специалистов-геофизиков были выдвинуты следующие назначения работ и требования:

Назначение 1: Разрабатываемая реализация алгоритма решения прямой задачи высокочастотного электромагнитного каротажа должна быть направлена на ускорение решения прямой задачи.

Назначение 2: Созданный проект технического задания на проведение ОКР по теме: «Разработка и создание серийного образца вычислительного комплекса по решению прямой задачи высокочастотного электромагнитного каротажа» должен быть направлен на создание малогабаритного вычислительного комплекса обработки каротажных данных, который можно поместить внутри корпуса каротажного прибора.

Требование 1: При проведении экспериментальных работ по реализации алгоритма решения прямой задачи высокочастотного электромагнитного каротажа должно достигаться ускорение более чем в 10 раз по сравнению с последовательной версией.

Требование 2: Реализация нейросетевого аналога решения прямой задачи высокочастотного электромагнитного каротажа должна обеспечивать

моделирование каротажных данных в реальном времени, т.е. время решения не должно превосходить 4 секунды.

Без учёта выдвинутых к реализации требований, нейросетевой алгоритм решения прямой задачи ВИКИЗ достаточно прост, но включает в себя достаточно большой объём вычислений. Технически, почти весь спектр возможных решений, представленных на современном рынке вычислительных систем, удовлетворяет минимальным требованиям, необходимым для реализации алгоритма. Однако выдвинутые требования по производительности существенно сужают круг потенциально возможных архитектур. Немногие решения способны дать необходимый выигрыш по производительности в сравнении с современными ПК.

Различные серверные и кластерные решения, бесспорно, обеспечивают необходимую производительность и обладают завидной лёгкостью реализации алгоритма, однако по физическим характеристикам такие системы не подходят для назначения об интеграции в магнитно-каротажный комплекс. Создание малогабаритного устройства, помещающегося в корпус каротажного прибора, на базе кластерной вычислительной системы не представляется выполнимой в обозримом будущем задачей. Затронув аспект серверных решений, нельзя не упомянуть возможность передачи входных данных на внешний сервер. Такой подход вызывает массу специфических трудностей, в основном связанных с передачей нужного объёма данных на значительное расстояние, однако решающим фактором в данной ситуации является требование о том, что разрабатываемая реализация должна обеспечивать решение задачи в реальном времени. Если даже распределённую реализацию и можно приблизить к понятию реального времени, то необходимость передачи больших объёмов данных в условиях каротажных исследований гарантирует невыполнение требований о работе комплекса в интерактивном режиме.

Другим возможным решением является реализация с использованием графических ускорителей. Стоит отметить, что в подобном направлении существует множество исследований, однако все они преследуют немного чуждые данной работе цели, все они нацелены на ускорение вычислений, однако исследований, направленных на создание малогабаритных устройств, обеспечивающих решение прямой нейросетевой задачи в реальном времени, найти не удалось. Использование графических ускорителей даёт хорошую производительность при минимальной стоимости, это неоспоримый факт современного рынка вычислительных систем. Однако важно помнить, что большинство существующих на сегодняшний день графических карт рассчитаны на работу под управлением центрального процессора. Эта особенность графических карт сильно затрудняет их использование при создании малогабаритного устройства. Также необходимость иметь промежуточный управляющий модуль с динамическим распределением ресурсов в виде центрального процессора общего назначения катастрофически сказывается на гарантированном времени решения.

Решения на базе FPGA удовлетворяют всем выдвинутым требованиям: вычислительной мощности современных кристаллов достаточно для обеспечения обозначенной производительности. В то же время задача алгоритмически проста, в ней отсутствуют условные переходы, она состоит лишь из арифметических и тригонометрических операций. Для задач подобного характера использование программируемых матриц позволяет создать наиболее эффективное специализированное вычислительное устройство [1].

1.2 Общая схема программно–аппаратного комплекса

Программно–аппаратный комплекс, необходимый для решения прямой задачи на базе FPGA состоит из трёх основных компонентов:

- аппаратное устройство
- программное обеспечение
- схемотехническая реализация

Устройство – это та аппаратная база, необходимая для возможности применения FPGA технологии. Устройство является физическим носителем FPGA матрицы, а так же обеспечивает матрицу необходимыми для её нормального функционирования ресурсами, такими как генератор тактовой частоты, программирующий модуль, интерфейс для подключения в вычислительную систему и т.д. В данной работе не затрагиваются проблемы разработки такого аппаратного устройства, в качестве целевого устройства рассматривается аппаратное решение SLEDv7 (ЗАО «СофтЛаб-НСК», г. Новосибирск). Краткое описание устройства приведено в приложении 1.

Необходимое программное обеспечение включает в себя драйвер устройства и программный интерфейс, скрывающий внутри себя технические подробности обращения к устройству. В ходе данной работы был создан драйвер с поддержкой технологии DMA для устройства SLEDv7 для семейства операционных систем MS Windows. Процесс разработки сопутствующего программного обеспечения в тексте данной работы освещён не будет ввиду его низкой научной ценности, само же созданное обеспечение вкратце описано в разделе 3.4.

Схемотехническая реализация описывает конфигурацию FPGA матрицы. Другими словами это есть представление некоего виртуального специализированного вычислительного устройства для решения нейросетевого аналога прямой задачи ВИКИЗ. Проектированию

схемотехнической реализации посвящена глава 2, непосредственной реализации на языке описания аппаратуры VHDL – большая часть главы 3.

Место каждой из перечисленных выше трёх компонент в программно-аппаратном комплексе относительно друг друга продемонстрировано на рисунке 1:

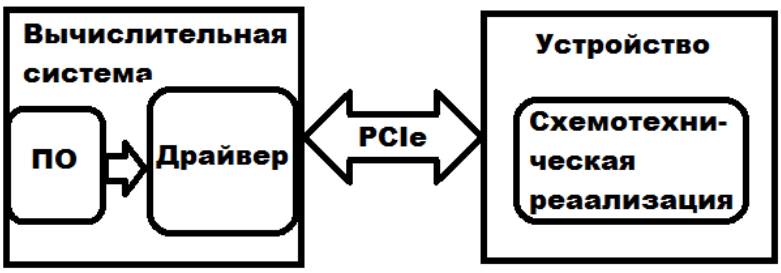

Рис. 1 Общая схема программно–аппаратного комплекса.

Устройство устанавливается в систему на шину PCIE. Драйвер устройства обеспечивает его представление в системе, отвечает за обращения к устройству и предоставляет пользовательскому ПО программный интерфейс для использования комплекса. FPGA матрица устройства сконфигурирована с помощью схемотехнической реализации и, фактически, является основным вычислителем комплекса. Драйвер отслеживает состояние устройства и передаёт команды запуска/остановки решения посредством CSR регистров устройства, а большие объёмы данных (входные данные, выходные данные, предварительная запись таблиц коэффициентов) передаются по DMA.

Глава 2

2.1 Анализ алгоритма

В основе прямой задачи ВИКИЗ лежит идея подбора характеристик среды, при которых электромагнитное зондирование дало бы результаты, схожие с результатами, полученными на практике. Разработанный Соболевым А.Ю. одномерный нейросетевой алгоритм обрабатывает четырёхмерную сетку следующих возможных значений среды:

- 0.5 < RoP < 200.0 - сопротивление пласта, Омм
- 1.0 < RoZp < 200.0 - сопротивление зоны проникновения, Омм
- 0.05 < AP < 2.0 - радиус зоны проникновения, м
- 0.02 < RoSkv < 10 - сопротивление бурового раствора, Омм

Сначала каждая четвёрка этих значений логарифмируется:

$$A = (\ln(RoP), \ln(RoZp), \ln(AP), \ln(RoSkv))$$

Затем от вектора *A* вычисляется следующая сумма синусов для каждого из 9 каротажных зондов:

$$S_z = \sum_{i=1}^{1000} \sin\left(c_{i0} + \sum_{j=1}^{4} a_j \cdot c_{ij} \right)$$

каждому из 9 зондов соответствует своя таблица коэффициентов c_{ij}. На последнем этапе в зависимости от логарифмированности зонда вычисляется либо $\varphi = \exp(c_0 + c_1 \cdot S)$, либо $\varphi = (c_0 + c_1 \cdot S)^2$. Результатом работы алгоритма для одной четвёрки входных значений является вектор из 9 значений ϕ – по одному на каждый каротажный зонд. В дальнейшем,

обработку алгоритмом одной четвёрки входных значений будем называть итерацией. Про выходные данные априори известно, что $0.2° < \varphi < 90°$.

Также необходимо указать на то, что алгоритм предполагается запускать над сеткой из множества (не менее 100 тыс.) четвёрок входных значений, то есть, между итерациями нет зависимости по данным – можно производить вычисления параллельно над несколькими узлами сетки входных данных [2][3].

2.2 Структура схемотехнической реализации

Благодаря независимости итераций разработанного Соболевым А.Ю. алгоритма и алгоритмической простоте самой итерации, данная задача хорошо представима в терминах FPGA. Отсутствие условных переходов делает реализацию одной итерации алгоритма в виде вычислительного конвейера наиболее привлекательной среди возможных подходов. Независимость между итерациями позволяет расположить на одном кристалле несколько вычислительных конвейеров, при наличии такой возможности. Учитывая объём ресурсов используемого в данной работе кристалла XC6VSX315T и характер вычисления одной итерации, можно с уверенностью сказать, что такая возможность имеет место быть.

Таким образом, мы приходим к общей схеме прошивки матрицы, решающей поставленную задачу:

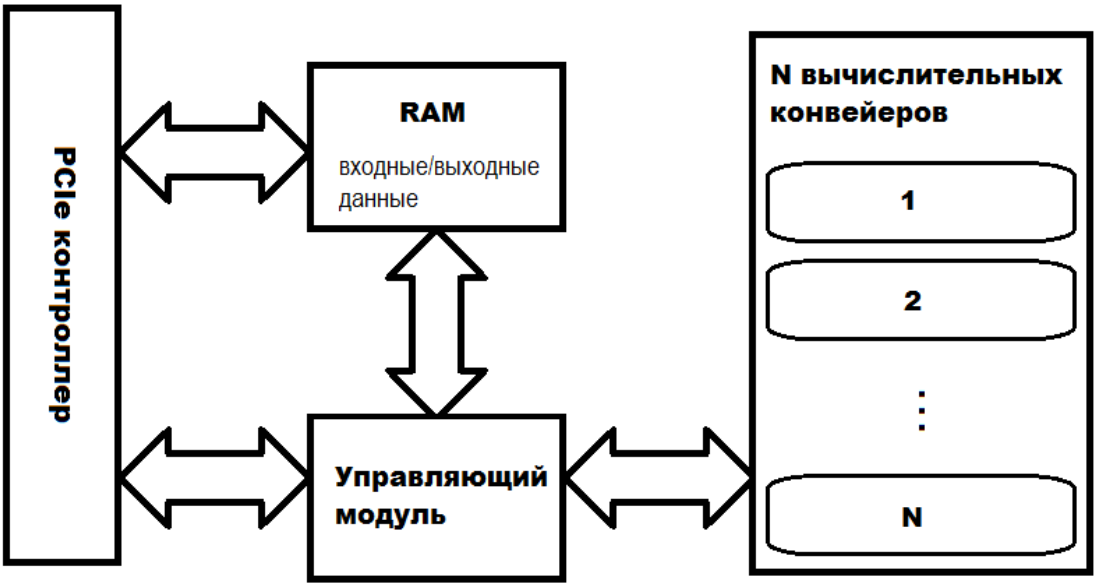

Рис.2 Структура схемотехнической реализации.

В аппаратном решении SLEDv7 сигналы шины соединены напрямую с входами и выходами кристалла XC6VSX315T. Таким образом, для работы устройства в составе программно-аппаратного комплекса необходимо иметь в схемотехнической реализации модуль контроллера шины. Также для обеспечения функциональности аппаратного решения в схемотехнической реализации присутствуют модуль передачи данных с помощью технологии DMA и модуль, обеспечивающий поддержку CSR регистров устройства, через которые программное обеспечение вычислительной системы следит за состоянием устройства (готовность к запуску, передаче входных/выходных данных) и посылает команды управления. Все эти модули не разрабатывались в ходе данной работы, а использовались уже существующие их реализации. Для удобства в дальнейшем будем называть объединения этих трёх функциональных блоков, обеспечивающих коммуникацию между аппаратной частью комплекса и вычислительной системой, PCIe-контроллером.

Следует отметить, что хотя на устройстве и имеется несколько гигабайт внешней оперативной памяти, поток данных между устройством и вычислительной системой незначителен. Это позволяет обходиться имеющимися в кристалле RAM блоками по 4KB для буферизации входных и выходных данных, не используя динамическую память. Отсутствие необходимости использовать динамическую память – довольно важный момент, так как иначе был бы нужен контроллер памяти, который пришлось так же реализовывать за счёт ресурсов кристалла. Таким образом, использование RAM блоков при передаче данных позволяет сэкономить немного FPGA логики.

Управляющий модуль следит за состоянием CSR регистров и вычислительных конвейеров, при необходимости снабжает вычислительные конвейеры входными данными из памяти, обеспечивает сбор и запись в память выходных данных. В общих чертах управляющий модуль представляет собой машину состояний, подробности реализации управляющего модуля по большей части зависят от конкретной реализации вычислительных конвейеров.

Вычислительные конвейеры – это модули ответственные за численные операции необходимые для решения задачи. Из-за специфики реализуемого алгоритма эта часть является самой важной, и именно вычислительные конвейеры будут занимать большую часть ресурсов кристалла. В идеале, вычислительный конвейер должен представлять собой последовательно соединённые конвейеризированные арифметические устройства – каждый такт результат каждого вычислителя подаётся на вход следующему за ним, без каких-либо управляющих сигналов. На данном этапе ключевой задачей работы является проектирование вычислительного конвейера максимизирующего суммарную производительность всех N конвейеров, которые можно разместить на кристалле.

2.3 Проектирование вычислительного конвейера

Возвращаясь к анализу алгоритма, его можно разбить на три принципиальных этапа:

1. Логарифмирование вектора входных значений. Вычислительная емкость данного этапа – взятие четырёх натуральных логарифмов;
2. Вычисление девяти сумм S_z: 45000 операций сложения, 36000 операций умножения, 9000 вычислений синуса;
3. Вычисление вектора ϕ: 9 сложений, 14 умножений, 4 вычисления экспоненты.

Объём вычислений на первом и третьем этапах незначителен по сравнению с вычислением сумм S_z. Специфика технологии FPGA и подхода, основанного на вычислительных конвейерах, требует наличия отдельных вычислителей для каждой операции. Таким образом, для логарифмирования входного вектора на первом этапе, вычислительный конвейер должен содержать хотя бы один логарифмирующий модуль. Но после окончания первого этапа этот модуль будет долгое время бездействовать. Учитывая ресурсоёмкость цифровой реализации натурального логарифма (как и взятия экспоненты), это крайне неэффективное использование ресурсов кристалла. Эту проблему можно решить двумя способами:

Первый способ заключается в вынесении вычислений первого и третьего этапов из вычислительных конвейеров в управляющий модуль. На первый взгляд такое может показаться грубым нарушением принципа разделения ответственностей – управляющий модуль должен заниматься обработкой управляющих сигналов, а не тяжёлыми математическими вычислениями. Но если взглянуть на ситуацию с другой стороны, то эти два этапа алгоритма можно рассматривать как вспомогательные, как этапы формирования входных и выходных данных. Основная выгода в вычислении

этих этапов в управляющем модуле кроется в возможности использования одного логарифмирующего элемента (и соответственно одного сложения, двух умножений и одной экспоненты) для всех N вычислительных конвейеров без потери производительности. Несмотря ни на какие ухищрения в вычислительных конвейерах, входные данные для них управляющий модуль берёт из одного RAM блока, получая их по одной шине данных последовательно, по одному элементу входного вектора за такт. Если на эту шину поставить логарифмирующий модуль, то входные данные управляющий модуль будет получать уже логарифмированными, только с задержкой, вносимой логарифмированием. Но в условиях конвейеризации всего процесса (пока устройство обрабатывает один пакет данных, хост передаёт следующий), эта задержка абсолютно никак не скажется на производительности комплекса. Аналогичные рассуждения можно провести для третьего этапа алгоритма. Чтобы записать выходные данные в RAM блок, управляющий блок должен рано или поздно их последовательно упорядочить во времени. В этом же месте можно и разместить конвейеризированный блок вычисления третьего этапа, нужно лишь учитывать поправку на вносимую задержку при передаче адреса в выходной RAM блок.

Второй способ борьбы с простоями блоков – это вовсе перенести вычисление первого и третьего этапов в программное обеспечение, исполняющееся на центральном процессоре. Нужно помнить, что не для любой вычислительной системы, выступающей в роли хоста, этот способ будет применим. Но так как в рамках данной работы в роли хоста выступает ПК, то такой подход стоит рассматривать.

Вынеся вычисления первого и третьего этапа за пределы вычислительного конвейера, мы приходим к новым, более узким, требованиям к функциональности конвейера. Каждый вычислительный

конвейер принимает на вход уже логарифмированный вектор A и возвращает девять сумм S_z.

Перейдём к более подробной организации вычислительного конвейера – на уровне вычисляющих суммы S блоков. Рассмотрим два основных подхода к организации конвейера. Первый подход, который возьмём за основу для сравнения, заключается в компоновке вычислительного конвейера из девяти независимых блоков вычисления суммы S, каждый из которых использует свою таблицу коэффициентов C. Таким образом, при запуске конвейера все девять блоков начинают параллельно вычислять суммы S для девяти зондов, и по окончанию работы конвейер выдаёт девять численных результатов.

Альтернативой такому подходу будет размещение в вычислительном конвейере только одного блока вычисления суммы S и последовательное вычисление на нём девяти сумм с разными таблицами коэффициентов C. Такой конвейер требует в девять раз меньше вычислительных ресурсов, но и работает в девять раз медленнее, для обработки одного вектора входных значений блок вычисления суммы S должен отработать девять раз. Таким образом, соотношение производительность/вычислительные ресурсы остаётся прежней: девять таких конвейеров будут занимать столько же места на кристалле, как и один большой, и будут выдавать такую же производительность. Однако, при таком подходе единица масштабирования алгоритма, один вычислительный конвейер, становится существенно меньше. Это даёт схемотехнической реализации дополнительную гибкость, например, возможность использования маленького кристалла, на котором невозможно разместить вычислительный конвейер, состоящий из девяти блоков суммы S.

С другой стороны, такой подход требует больше RAM блоков для хранения таблиц коэффициентов C. Девяти одноблочным конвейерам нужно в девять раз больше RAM блоков, чем одному, состоящему из девяти блоков,

так как каждый блок вычисления суммы S обрабатывает не один и тот же зонд, а разные зонды в разные моменты времени. Подводя итоги, получается, что выбор одного из описанных выше подходов зависит от ресурсов используемого кристалла и конкретной используемой реализации блоков суммы S. Также при необходимости можно применять гибридный подход, например, использовать в одном вычислительном конвейере три блока суммы S с тремя разными таблицами коэффициентов у каждого. Выбор количества блоков вычисления суммы S в вычислительном конвейере должен проводиться непосредственно в применении к конкретному кристаллу FPGA. Забегая вперёд, можно отметить, что для используемого в данной работе XC6VSX315T хорошо подходит вариант с девятью блоками S.

Перейдём к внутреннему устройству блоков вычисления суммы S. В формуле

$$S_z = \sum_{i=1}^{1000} \sin\left(c_{i0} + \sum_{j=1}^{4} a_j \cdot c_{ij} \right)$$

можно логически выделить три принципиальных с точки зрения проектирования FPGA–схем этапа:

1. Вычисление аргумента синуса $\alpha_i = c_{i0} + \sum_{j=1}^{4} a_j \cdot c_{ij}$;
2. Вычисление $\sin \alpha_i$;
3. Суммирование синусов.

На первых двух этапах нет зависимости по данным по индексу i, следовательно, вычисление суммы S можно ускорить в m раз, если внутри одного блока поставить m вычислителей для первого и второго этапа. Однако достаточная степень параллелизма уже достигнута установкой множества

блоков S, а такой подход не только не улучшит соотношение производительность/ресурсы кристалла, а даже немного ухудшит его: для конвейеризованного суммирования данных, поступающих каждый такт из m входов, потребуется $\log_2 m + 1$ сумматоров. Таким образом, идею ускорения работы блоков вычисления суммы S за счёт внутреннего распараллеливания можно смело считать бесперспективной. Предпочтительнее будет максимально упростить внутреннюю логическую организацию и минимизировать необходимое количество ресурсов кристалла.

Первоначально был предложен наиболее естественный для такой суммы синусов конвейер с древовидным суммированием аргумента:

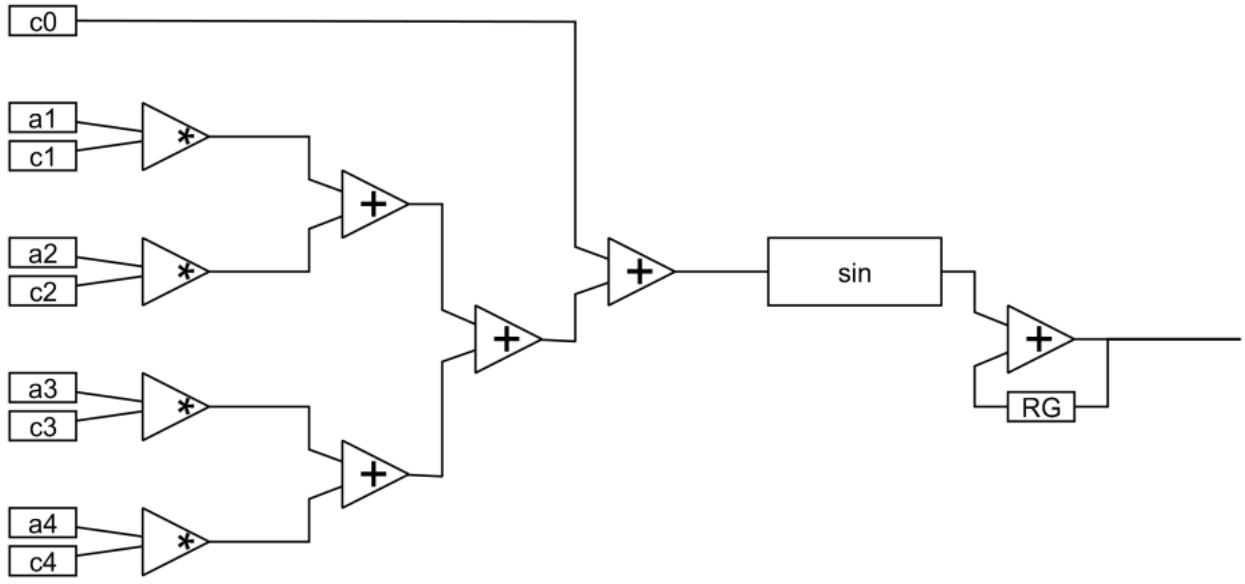

Рис.3 Древовидное суммирование аргумента

Такая структура конвейера интуитивно понятна. Параллельно считаются 4 произведения $a_j \cdot c_{ij}$, затем попарно складываются друг с другом и с коэффициентом c_{i0}, и, наконец, результат подаётся на вход блока вычисления синуса. Результаты взятия синусов суммируются, накапливаясь в специальном регистре. Так как все вычислители конвейеризованы, то каждый такт начинается вычисление над следующим C_i вектором. Таким образом,

одна сумма S будет посчитана за (1000+d) тактов, где d – суммарная задержка вычислителей. При вычислении нескольких сумм подряд (чем блок и будет загружен бОльшую часть времени) благодаря конвейеризации задержка d становится незначительной.

Такая структура обладает одним недостатком, который можно исправить реорганизацией: коэффициенты c0..c5 изменяются каждый такт, соответственно такому конвейеру нужно подавать 5 слов данных из памяти каждый такт, для чего конвейер должен иметь не менее 5 блоков памяти. Количество потребляемых конвейером данных можно уменьшить, если использовать аккумулятор вместо древовидного суммирования:

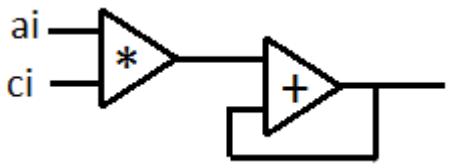

Рис.4 Аккумулятор аргумента синуса

Такому конвейеру необходимо только 1.25 слов данных из памяти на каждый такт (помимо ci, каждый 4й такт накопленное значение нужно перезаписывать на c0). Но тогда на подсчёт одного аргумента требуется 4 такта, и 75% времени блок синуса будет простаивать. Сложение, умножение и используемая в данной работе реализация синуса имеют схожие временные характеристики, поэтому считать аргумент и синус с разной тактовой частотой невыгодно. В качестве решения проблемы простоя блока синуса было предложено ввести межпроходный параллелизм уже на уровне конвейера, объединив 4 аккумулятора аргумента, считающие аргументы для одного зонда (одна таблица коэффициентов c), но для разных входных векторов A.

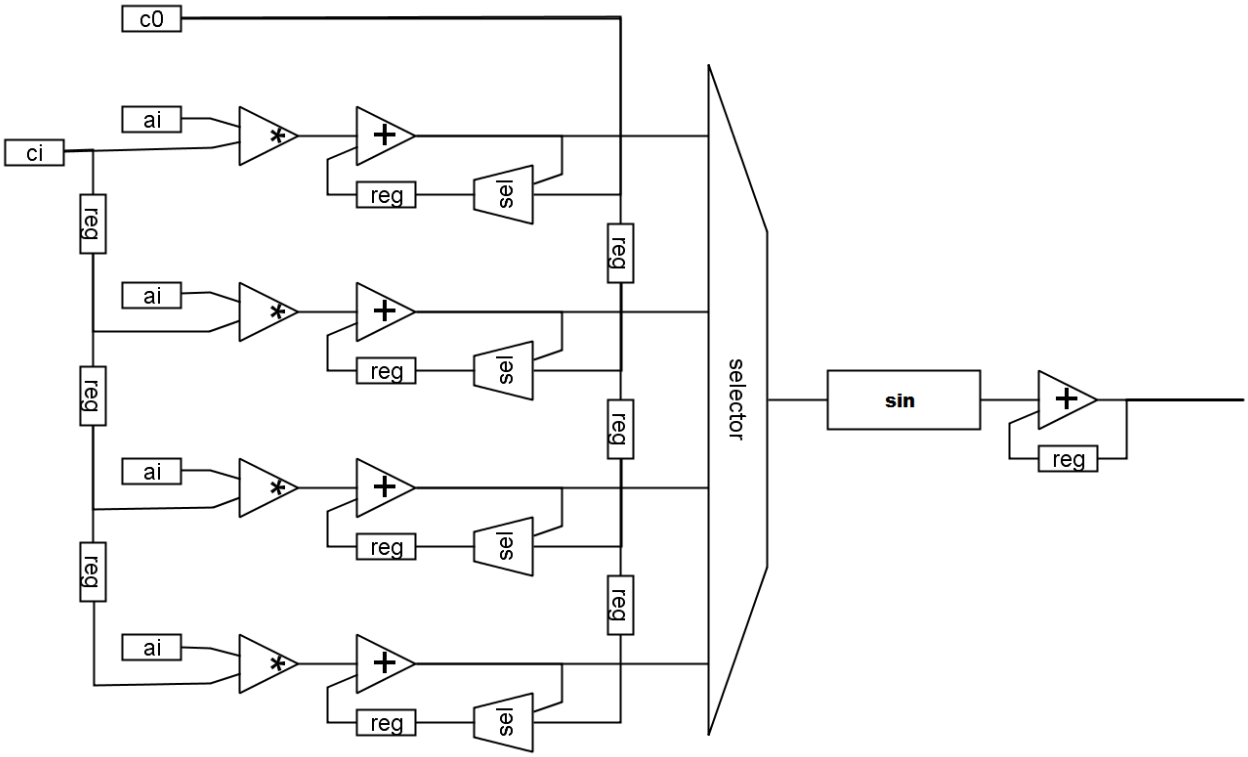

Рис.5. Блок вычисления суммы S с 4 аккумуляторами аргументов

Такой конвейер параллельно рассчитывает 4 независимых прохода, но в 4 раза медленнее, в итоге обладая такой же производительностью, и требует всего 2 слова данных из памяти на каждый такт. Число арифметических блоков, как видно из схемы, остаётся то же, но появляются дополнительные расходы ресурсов на создание десяти регистров, ставших необходимыми из–за логического усложнения конвейера.

Целесообразность такой замены зависит от целевого кристалла FPGA. При использовании кристалла XC6VSX315T она не даёт выигрыша по количеству необходимых RAM блоков. В данном кристалле используются RAM блоки размером по 4 килобайта каждый, следовательно, даже при типе данных коэффициентов float, и в том, и в другом случае потребуется 5 RAM блоков. При наличии RAM блоков по 16 килобайт, такой приём сократил бы их необходимое количество с пяти до двух, но в рамках данной работы

остаётся придерживаться первого варианта структурной организации блока вычисления суммы S – с древовидным суммированием аргумента синуса.

2.4 Используемые форматы данных

Этот раздел посвящён технологическим аспектам схемотехнической реализации, в частности, выбору двух тесно связанных параметров – используемых форматов данных и конфигураций арифметических блоков.

Самый простой способ – использовать стандартные вещественные числа одиночной или двойной точности. Однако средства FPGA позволяют создавать специализированные вычислительные устройства с любыми форматами данных, поэтому был подробно рассмотрен вариант использования чисел с нужной фиксированной точностью.

Все коэффициенты зондов c_{ij} по модулю не превосходят 20, логарифмированные входные значения компонент A лежат в еще более узком диапазоне. Основываясь на максимальной разрядности RAM блоков кристалла XC6VSX315T, был выбран начальный формат данных, принимаемых аппаратной частью на вход – восемь бит для целой части и 24 для дробной. Отрицательные числа записываются в дополнительном коде. В дальнейшем будем коротко называть такой формат 8p24.

Использование данных с фиксированной точностью позволяет использовать целочисленные сумматоры и умножители, которые требуют меньше ресурсов, чем дробные. На основании результатов работы синтезатора схем среды разработки Xilinx ISE14 была составлена следующая таблица, демонстрирующая необходимое количество ресурсов кристалла для реализации древовидного суммирования аргумента синуса, описанного в предыдущем разделе, в зависимости от используемого формата входных данных:

Формат данных	DSP48E	LUT	FF
Single precision floating point	20	~4000	~1600
8p24	20	0	0
Double precision floating point	56	~4400	~5200

Из данной таблицы явно следуют следующие заключения:

- Если использование 32-битных форматов данных будет обеспечивать необходимую точность, то использование 64-битного формата данных не оправданно;
- Использование данных с фиксированной точностью предпочтительнее с точки зрения экономии ресурсов кристалла.

Также следует указать, что приведённые в таблице данные получены из расчёта на максимальное использование блоков DSP48E. В том случае, если при увеличении количества вычислительных конвейеров блоки DSP48E закончатся намного раньше, чем логика кристалла, то конвейер можно всегда сбалансировать под оптимальное соотношение используемых DSP/логики, заменив часть блоков DSP48E на логические вычисления.

В качестве умножителей в конвейере используются состоящие из четырёх DSP48E блоки умножения двух целых чисел по 32 бита [4]. Подавая на вход данные в формате 8p24, на выходе получается число в формате 16p48. Учитывая диапазон возможных входных значений, у результатов умножений отбрасываются старшие и младшие восемь бит, получается число в формате 8p40. Это нужно потому, что максимальная разрядность DSP48E в режиме сложения 48 бит, и именно они используются в качестве сумматоров

в дерева подсчёта аргумента синуса. В блок, вычисляющий синус, также подаётся число в формате 8p40.

Для вычисления синуса была использована реализация алгоритма поворота единичного вектора CORDIC. Данная реализация принимает на вход данные в формате 3pN, в диапазоне $[-\pi;\ \pi]$. Поэтому предварительно вычисляется остаток от деления аргумента на π. Остаток от деления вычисляется путём двух умножений на константу: сначала аргумент умножается на $\frac{1}{\pi}$, затем дробная часть результата – на π.

Вычислять экспоненту так же можно с помощью алгоритма CORDIC. Предоставленная в среде Xilinx ISE14 готовая реализация алгоритма позволяет вычислять гиперболические функции sinh и cosh от аргумента, лежащего в интервале $\left[-\frac{\pi}{4};\frac{\pi}{4}\right]$, причем возможно параллельное вычисление обеих функций в одном блоке. Из диапазона допустимых выходных значений можно сделать вывод о том, что при допустимых входных значениях аргумент экспоненты всегда будет лежать в интервале (-1.7 ; 4.5). Тогда $-\frac{\pi}{4} < \frac{x-1.4}{4} < \frac{\pi}{4}$. Следующая формула выражает метод, использовавшийся для вычисления экспоненты с помощью данной реализации алгоритма поворота единичного вектора CORDIC:

$$e^x = \left(e^{\left(\frac{x-1.4}{4}\right)}\right)^4 \cdot e^{1.4} = \left(\sinh^{\left(\frac{x-1.4}{4}\right)} + \cosh^{\left(\frac{x-1.4}{4}\right)}\right)^4 \cdot e^{1.4}$$

Стоит отметить, что возведение в степень, являющуюся степенью числа 2, тривиально реализовывается последовательным умножением, а константу $e^{1.4}$ можно вычислить заранее. Таким образом, для включения

этапа вычисления экспоненты в схемотехническую реализацию потребуется: один блок, реализующий алгоритма CORDIC, три сумматора, четыре умножителя и один RAM-блок для хранения коэффициентов c0 и c1.

Такой вычислительный конвейер был реализован на языке описания аппаратуры VHDL и при тестировании с помощью симулятора Xilinx ISim показал погрешность 10^{-5} относительно реализации алгоритма с помощью стандартных средств языка C++ при использовании вещественных чисел с плавающей точкой двойной точности (double). Таким образом, точность вычислений при использовании данных фиксированной точности можно считать достаточной. Учитывая принятые ранее решения относительно структурной организации программно-аппаратного комплекса, было решено приводить данные к формату с фиксированной точностью на стадии подготовки данных к передаче в устройство из программного обеспечения, исполняющегося на ПК. В результате, в устройство будут приходить данные уже в формате 8p24, нет необходимости реализовывать конвертер данных в рамках схемотехнической реализации. В случае использования устройства в условиях невозможности предварительных вычислений в хосте, конвертер можно объединить с вычислением логарифма входных данных, так, как это было описано в предыдущем разделе. Так можно обойтись одним блоком приведения типов независимо от количества вычислительных конвейеров.

Глава 3

3.1 Управляющий модуль

Управляющий модуль в первую очередь служит связующим звеном между PCI-контроллером, архитектурное устройство которого в данной работе не рассматривается, и вычислительным модулем, который в свою очередь уже состоит из непосредственно вычислительных конвейеров. Управляющий модуль должен организовывать поступающие из контроллера данные для конвейера и компоновать результаты работы конвейера для отправки контроллером в хост через шину PCIe. Также модуль должен осуществлять синхронизацию между вычислительной частью схемотехнической реализации и контроллером шины, так как они работают на разной тактовой частоте [11]. Таким образом, архитектура управляющего модуля зависит только от интерфейсов и семантики использования этих двух компонент схемотехнической реализации.

Ответственная за передачу данных часть интерфейса PCI-контроллера в терминах VHDL выглядит следующим образом:

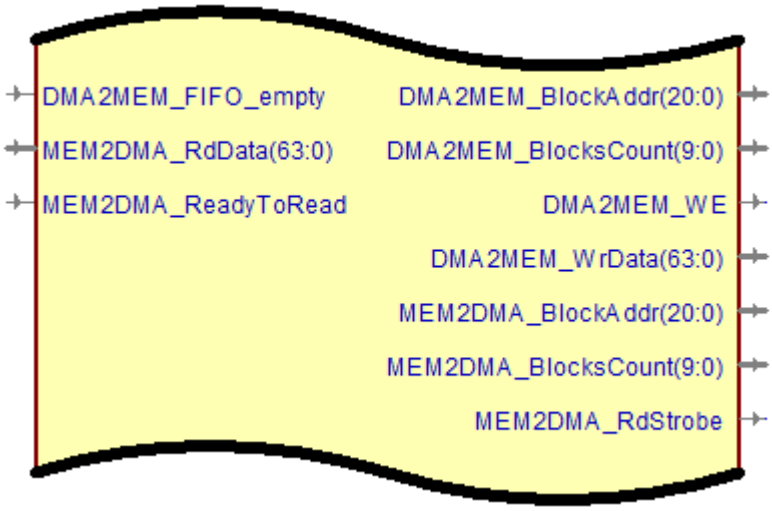

Рис.6.

- DMA2MEM_WrData – это шина данных для записи в устройство;

- MEM2DMA_RdData – шина для чтения данных из устройства;
- DMA2MEM_WE и MEM2DMA_RdStrobe – стробирующие сигналы записи и чтения соответственно;
- DMA2MEM_FIFO_empty и MEM2DMA_ReadyToRead – входящие сигналы, говорящие о готовности управляющего модуля (в нашем случае) принимать или отдавать данные.

Сигналы X_BlockAddr и X_BlocksCount нужны для адресации динамической памяти устройства, но так как было решено сделать аппаратную часть комплекса работающей в режиме потоковой обработки данных, то эти сигналы, как и сама динамическая память, в рамках данной работы остаются невостребованными. И чтение, и запись контроллером производятся блоками по 4096 байт.

Интерфейс вычислительного модуля в терминах VHDL таков:

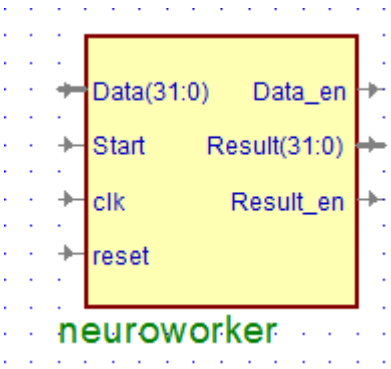

Рис.7. Модуль NeuroWorker.

Помимо сигналов у интерфейса модуля есть два параметра времени компиляции pipeline_count и pass_length, они определяют количество вычислительных конвейеров в устройстве и размерность матрицы коэффициентов C при вычислении суммы S_z соответственно.

- Data и Result – шины входных и выходных данных;
- Data_en и Result_en – соответствующие стробирующие сигналы;

- Start – это сигнал запуска конвейеров, при единице на этом входе, вычислительный модуль последовательно забирает $4 \cdot pipeline\_count$ 32-битных слова из шины данных (один вектор A) и передаёт их непосредственно в вычислительные конвейеры.

В вычислительном модуле не предусмотрено никаких средств для приостановки конвейеров, поэтому, подав единицу на вход Start, управляющий модуль должен быть готов не только предоставить вектор входных значений A, но и принять $9 \cdot pipeline\_count$ слов результатов вычислений.

Как видно из приведённых выше описаний интерфейсов PCIE контроллера и вычислительной части аппаратной базы комплекса, для их коммутации необходима буферизация, как входных данных, так и результатов вычислений. Простейший способ буферизовать передачу – использовать один RAM-блок для входных значений и один RAM-блок для результатов. Сначала контроллер записывает во входной буфер один блок данных (4096 байт). Затем вычислительный модуль обрабатывает этот блок, записывая результаты в выходной буфер. После этого PCIE контроллер передаёт содержимое выходного буфера в вычислительную систему.

Очевидный минус такого подхода – невозможность параллельного исполнения задач контроллера и вычислителя. Пока идёт передача данных между устройством и системой, вычислитель вынужден простаивать, и, наоборот, во время вычисления алгоритма оба буфера недоступны для контроллера.

Эти простои можно нивелировать, используя для буферизации четыре RAM-блока, по два блока для каждого буфера. Обозначим RAM-блоки, использующиеся в качестве входного буфера, как A и B. Пока контроллер записывает блок входных данных в RAM-блок A, вычислитель обрабатывает данные, ранее записанные контроллером в блок B. Когда и запись в блок A, и обработка данных из блока B закончены, RAM-блоки переключаются:

контроллеру для записи входных данных предоставляется блок B, а вычислитель начинает обработку данных из RAM-блока A. Буфер выходных данных устроен аналогичным образом, обозначим использующиеся в нём RAM-блоки как C и D. Тогда изложенный подход можно проиллюстрировать следующей диаграммой:

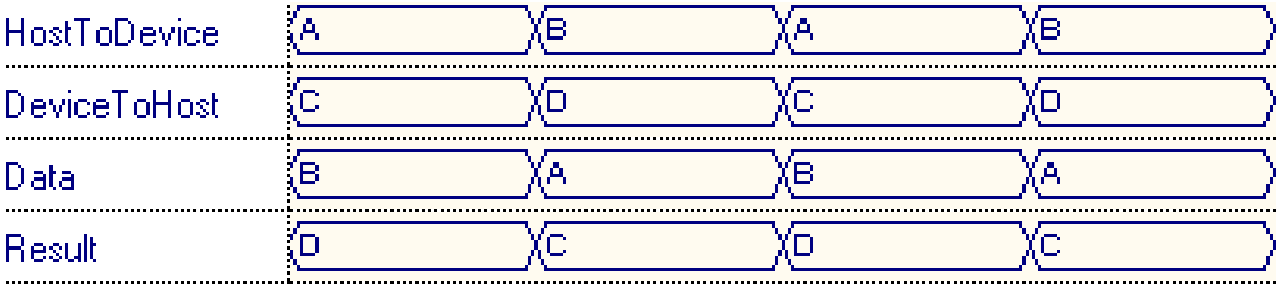

Рис.8 Буферизация входных и выходных данных.

Именно такой подход был использован при реализации буферизации входных и выходных данных в ходе данной работы. Переключение входных и выходных RAM-блоков было решено сделать независимым, это позволяет избежать небольшого возможного простоя контроллера в промежутке между получением вычислителем последнего вектора входных данных из блока и записью результатов обработки этого вектора в RAM-блок выходного буфера. Принимая во внимание это решение, и тот факт, что вычислитель и PCIE контроллер работают на разных тактовых частотах, для организации управляющего модуля были использованы четыре одинаковых по структуре конечных автомата.

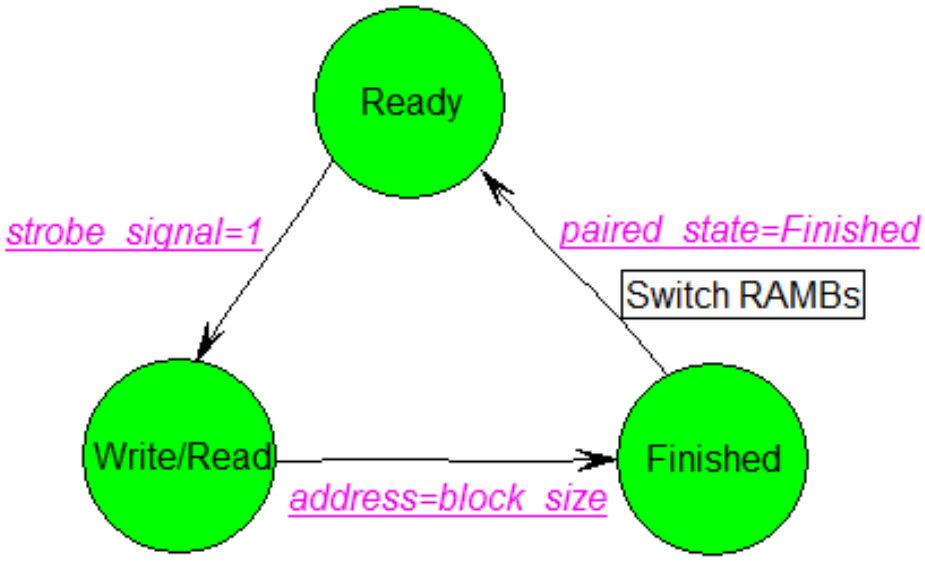

Рис.9. Конечные автоматы управляющего модуля

Четыре таких конечных автомата описывают состояния следующих процессов в управляющем модуле: запись PCIE контроллером входных данных в буфер, чтение вычислителем входных данных из буфера, запись вычислителем результатов в буфер и чтение контроллером результатов из буфера. Использованные на рисунке обозначения:

- strobe_signal – это соответствующий процессу стробирующий сигнал;
- paired_state – состояние второго процесса, работающего с тем же буфером.

Рассмотрим рабочий цикл автомата подробнее. Изначально автомат находится в состоянии готовности S_Ready. При этом PCIE контроллеру посредством соответствующего сигнала сообщается о том, что управляющий модуль готов принимать данные. Когда система инициирует передачу данных в устройство, контроллер начинает передавать эти данные управляющему модулю, параллельно посылая стробирующий импульс. Получив этот сигнал, автомат переходит в состояние приёма данных. В этом состоянии управляющий модуль записывает принимаемые данные в активный на текущий момент RAM-блок. Записав 4096 байт, конечный автомат переходит в состояние ожидания S_WaitSwitch, при этом сигнализируя контроллеру о неготовности принимать данные далее. Как

только конечный автомат, описывающий чтение вычислителем входных данных, тоже оказывается в состоянии ожидания S_WaitSwitch, оба автомата переходят в состояние S_Ready. При этом переходе происходит переключение RAM-блоков.

Парный вышеописанному конечный автомат, описывающий процесс чтения входных данных вычислителем, отличается тремя принципиальными моментами. Во первых, он работает на тактовом сигнале вычислителя. Во вторых, его начальное состояние не S_Ready, а состояние ожидания S_WaitSwitch – вычислитель не может сразу приниматься за работу, сначала он должен дождаться первого блока входных данных от контроллера. И, наконец, третье отличие – в состояние ожидания конечный автомат переходит по прочтении не 4096 байт, а 1808 байт. Последнее отличие обуславливается следующим: так как квант обмена данными между устройством и системой – это блок размером 4096 байт, то за одну передачу можно передать до $\left\lfloor \frac{4096}{4 \cdot 9} \right\rfloor$ векторов выходных значений. Следовательно, и векторов входных данных передавать нужно столько же. А это и есть 1808 байт. Аналогичным образом устроена и пара конечных автоматов, отвечающая за передачу результатов вычислений от вычислителя до PCIE контроллера.

В процессе разработки схемотехнической реализации также рассматривался вариант использования большего кванта обмена данными между устройством и системой. При обмене данными по 4096 байт, больше половины блока входных данных в вычислениях не используются, этого можно избежать, если устройство будет принимать входные данные пакетами по четыре блока и возвращать результат пакетами по девять блоков. Однако задача чтения и выбора результата из девяти RAM-блоков за один такт по переднему фронту стробирующего сигнала PCI-e контроллера оказалась неразрешимой, без внесения изменений в реализацию контроллера

шины, а нагрузка на шину PCI-e при работе комплекса на базе кристалла XC6VSX315T была оценена как незначительная. Поэтому было решено использовать квант передачи данных равный одному блоку по 4096 байт.

3.2 Вычислительный модуль

Множество вычислительных конвейеров объединено внутри вычислительного модуля NeuroWorker, интерфейс которого был приведен в предыдущем разделе. Конечный автомат, описывающий семантику работы вычислительного модуля, приведён на следующем рисунке:

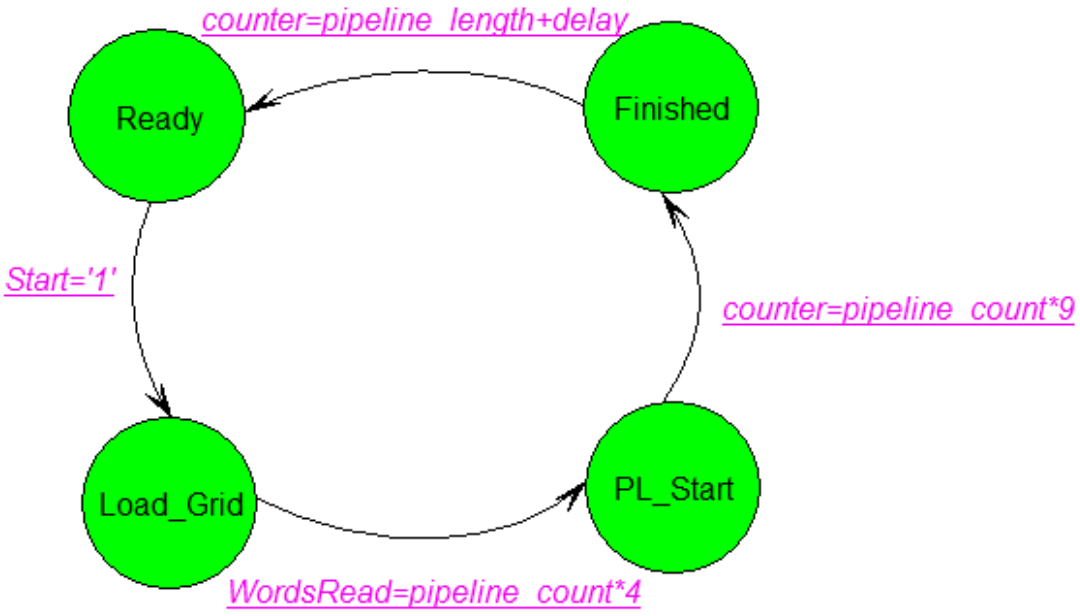

Рис.10 Конечный автомат модуля NeuroWorker.

В начальном состоянии S_Ready вычислительный модуль ждёт, пока управляющий модуль сообщит о наличии входных данных и готовности принимать результат, подав высокий логический сигнал на вход Start. Получив этот сигнал, вычислительный модуль переходит в состояние S_Load_Grid, в котором он вычитывает вектора входных значений и распределяет их между вычислительными конвейерами. Передав каждому конвейеру по одному вектору входных значений, модуль переходит в

состояние S_PL_Start, в котором он запускает конвейеры по одному и с промежутком в девять тактов. Это нужно затем, чтобы упорядочить по времени результаты вычислений конвейеров. Запустив все вычислительные конвейеры, модуль выжидает время, гарантирующее готовность конвейеров принимать следующую партию входных данных, и снова переходит в состояние S_Ready.

Интерфейс вычислительного конвейера в терминах VHDL выглядит следующим образом:

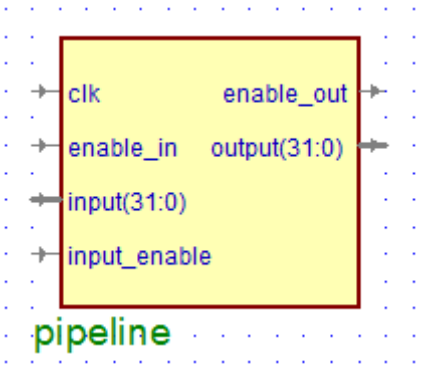

Рис.11 Интерфейс модуля pipeline.

Параметры времени компиляции pass_length и sonde_count описывают размерность матрицы C и количество зондов соответственно. Входной сигнал input нужен для последовательной передачи в конвейер вектора входных значений, внутри модуля pipeline он соединён со сдвиговым регистром, управляющим сигналом которого служит сигнал input_enable. В итоге, во время работы конвейера в регистре data_bus находится четырёхмерный вектор входных значений, который конвейер в данный момент обрабатывает, и именно из этого регистра блоки вычисления сумм S_z каждый такт берут входные данные.

Сигнал enable_in – это сигнал запуска конвейера. Для того, чтобы запускать блоки вычисления сумм S_z не одновременно, а с временной задержкой на такт, так же используется сдвиговый регистр.

Сигнал pl_enable_in(i) в дальнейшем используется, как сигнал сброса внутреннего счётчика блока вычисления суммы S_i. Сигнал output_enable – это логическое ИЛИ аналогичных сигналов всех девяти блоков S_z. Сигнал output - это корень дерева из мультиплексоров от соответствующих сигналов блоков S_z, управляющими сигналами этих мультиплексоров служат сигналы output_enable блоков S_z. Сам интерфейс блока вычисления суммы S_z таков:

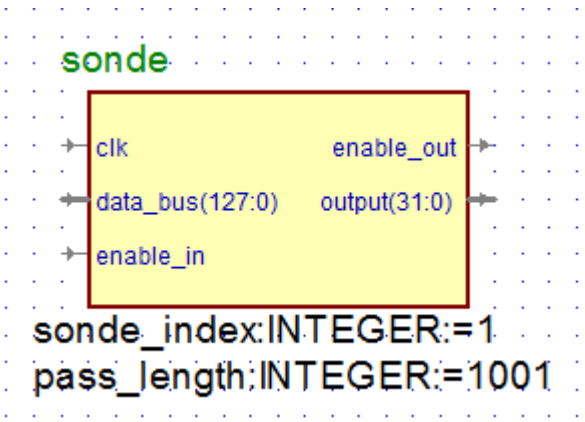

Рис.12. Интерфейс модуля sonde.

Константа sonde_index определяет, какая таблица коэффициентов C будет синтезирована в данном блоке, константа pass_length – размер используемой части этой таблицы. Data_bus – это порт, на который подаётся вектор входных значений, output – это порт выходных данных.

Enable_in играет роль сигнала запуска вычислителя, когда на него подаётся высокий уровень сигнала, сбрасывается и запускается внутренний счётчик блока counter, инкрементирующийся каждый такт. На основании значения этого счётчика определяется внутреннее состояние вычислительного блока и готовность результата. Когда значение счётчика достигает величины равной pass_length+sonde_delay, где sonde_delay – это суммарная задержка арифметических блоков вычислителя, сигнал output_enable устанавливается высоким, а на порт output подаётся результат вычисления суммы S_i от данного вектора входных значений.

Архитектурно, компонент sonde представляет собой потоковый полностью конвейеризованный вычислитель, логическое устройство которого было представлено на рис. 3.

Первый этап вычислений – это четыре умножения компонент входного вектора на соответствующие коэффициенты C_{ij}. Компоненты вектора берутся непосредственно из входных регистров Data_bus, значения C_{ij} берутся из специальных RAM-блоков. Эти RAM-блоки представляют собой экземпляры компонента CoefTable:

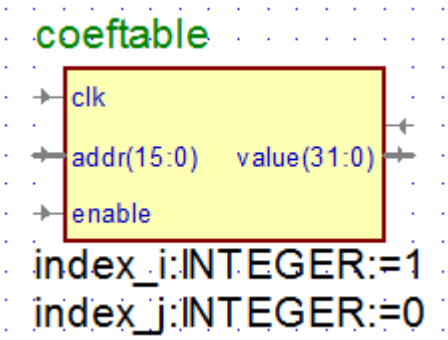

Рис.12. Интерфейс модуля CoefTable.

Внутри этот компонент содержит RAM-блок, содержимое которого определяется на стадии синтеза схемы на основании значений index_i и index_j. Индекс i определяет, какому зонду будут соответствовать коэффициенты в RAM-блоке, а индекс j – это номер столбца коэффициентов. Текст программы модуля CoefTable вместе с инициализирующими данными RAM-блоков достигает в размере 1 MB, поэтому он был сгенерирован по таблицам коэффициентов в программе, специально созданной для этого на языке C++.

В реализации, использующей данные с фиксированной точностью, для умножения двух чисел в формате 8p24 используются конвейеризованные целочисленные знаковые умножители, состоящие из четырёх блоков DSP48E каждый. Задержка этих умножителей составляет шесть тактов. Таким образом, этап умножения реализован четырьмя блоками, представленными на следующей блок-схеме:

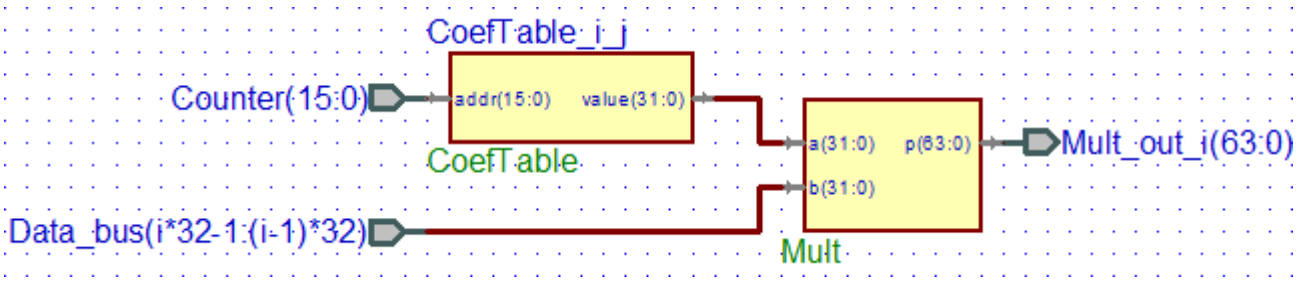

Рис. 13. Блок-схема одного умножения.

Затем старшие 48 бит результатов умножений суммируются друг с другом и с соответствующим коэффициентом C_{i0}. Используемые целочисленные знаковые 48-битные сумматоры реализуются либо на DSP48E, либо на логике кристалла, в зависимости от параметров синтеза схемы. Процесс суммирования представлен на следующей схеме:

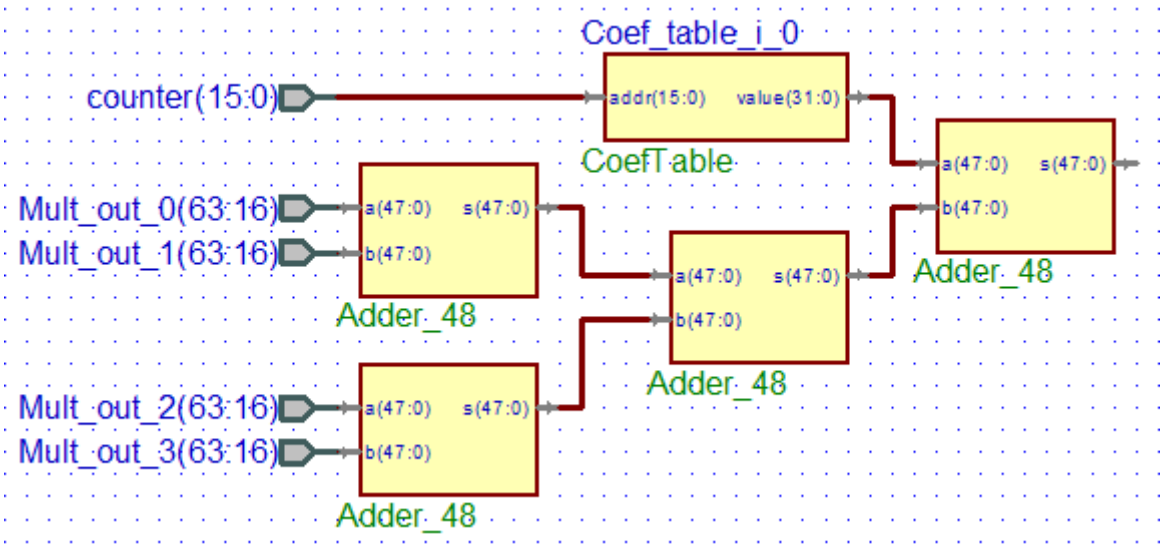

Рис. 14. Блок-схема суммирования произведений.

В реализации, использующей формат данных с плавающей точкой одиночной точности, два этапа вычислений, описанные выше, отличаются используемыми арифметическими блоками и шириной промежуточных сигналов. Для умножения и сложения используются реализации соответствующих операций с плавающей точкой, входящие в среду разработки Xilinx ISE, ширина всех промежуточных сигналов соответствует

размеру используемого формата данных – 32 бита. Результат суммирования произведений затем подаётся на модуль, приводящий число с плавающей запятой к формату с фиксированной точностью. Это необходимо из-за того, что в обоих случаях используется одна и та же реализация вычисления синуса, принимающая аргумент в формате с фиксированной точностью. Поэтому последующие этапы вычислительного конвейера для схемотехнических реализаций с плавающей и фиксированной точностью абсолютно идентичны.

Умозрительно следуя за вычислениями конвейера, в текущий момент мы имеем вычисленный в формате с фиксированной точностью результат выражения $P_i = C_{i0} + \sum_{j=1}^{4} C_{ij} \cdot a_j$. Следующий шаг в вычислении суммы S_z – вычисление синуса. В качестве арифметического блока для взятия синуса была выбрана реализация алгоритма поворота единичного вектора, распространяющаяся вместе с пакетом разработки Xilinx ISE. Допустимый диапазон входных значений этой реализации: $\varphi \in (-\pi;\pi)$, следовательно, необходимо привести имеющийся результат к данному диапазону. Для этого использовались два последовательных целочисленных знаковых умножения на константу. Сначала, P_i умножается на $\frac{1}{\pi}$, результат разбивается на целую и дробную часть:

$$\frac{P_i}{\pi} = z + f \text{ , где } z \in Z,\ f \in (-1;1)$$

Затем на π умножается либо f , если z – чётное, либо $-f$, если z – нечётное:

$$\varphi = (-1)^z \cdot \pi \cdot f$$

Таким образом, $\varphi \in (-\pi;\pi)$ и $\sin\varphi = \sin P_i$, этот аргумент и подаётся на вход блока CORDIC.

Последний элемент модуля sonde – это аккумулятор значений синуса. Он представляет собой реализованный на логике кристалла целочисленный сумматор и регистр. На один из входов сумматора подаётся выходное значение блока синуса, а на второй – значение, лежащее в этом регистре. Если значение счётчика counter равно sonde_delay, то значение регистра сбрасывается на ноль, иначе, по переднему фронту тактового сигнала в регистр записывается выходное значение сумматора. Когда значение счётчика counter превышает pass_length – конвейер готов принимать новый вектор входных значений *A*, когда оно равно pass_length+sonde_delay – значение в аккумуляторе равно сумме S_z.

Результаты вычислений девяти модулей sonde, принадлежащих одному вычислительному модулю проходят через двухуровневое дерево мультеплексоров с тремя входами [7]. Так как все модули sonde запускаются последовательно один за другим, а время работы конвейеров детерминировано с точностью до такта, то на входы этого дерева никогда не будут поданы два результата одновременно.

3.3 Модули логарифмирования и взятия экспоненты

В разработанном в ходе данной работы программно-аппаратном комплексе логарифмирование входных данных и вычисление последнего этапа нейросетевого алгоритма исполняются на стороне ПК, так как реализации необходимых блоков достаточно затратны с точки зрения ресурсов кристалла, а доли этих вычислений в общем объёме операций достаточно малы. К тому же, большая часть времени работы комплекса приходится на вычисление сумм S_z, во время которых ПК простаивает, дожидаясь окончания работы устройства. В рамках данной работы также была исследована возможность переноса этих вычислений на устройство, и были разработаны прототипы блоков, реализующих эти вычисления.

Существующие конвейеризуемые реализации натурального логарифма требуют намного больше ресурсов кристалла, нежели последовательные. В ходе работы рассматривалась возможность отказа от конвейеризации логарифма, за счёт утилизации потенциального времени простоя блока. Для этого было предложено реализовать логарифмирование в виде самостоятельного модуля, размещающегося между управляющим и вычислительным модулями, и имеющим подобную управляющему модулю семантику передачи входных и выходных данных.

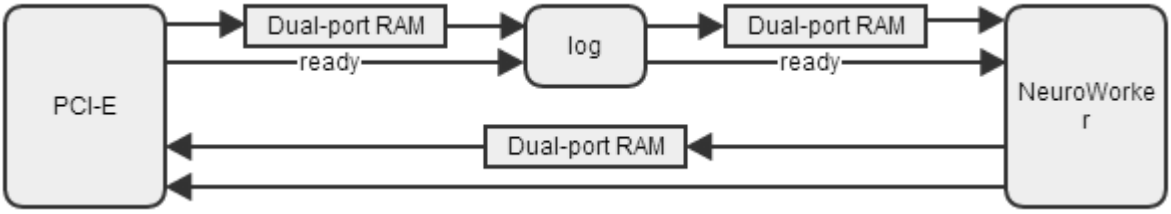

Рис.15 Место логарифма в схемотехнической реализации.

Принятый от контроллера шины блок входных данных передаётся не вычислительному модулю, а логарифмирующему, который постепенно логарифмирует весь блок данных. Пока NeuroWorker обрабатывает один блок входных данных, логарифмирующий модуль подготавливает следующий, при условии, что он уже был передан в устройство. Таким образом, если логарифмирование одного блока будет происходить быстрее, чем обработка в модуле NeuroWorker, такой логарифмирующий модуль будет лишь вносить дополнительную задержку при обработке одного пакета данных, что никак не скажется на производительности комплекса в режиме потоковой обработки данных. Обработка одного блока данных занимает у модуля NeuroWorker $\frac{1000 \cdot 113}{n}$, где n – это количество вычислительных конвейеров. Ресурсов используемого в данной работе кристалла XC6VSX315T хватило для размещения четырёх вычислительных конвейеров,

следовательно, для логарифмирования одного числа есть $\left\lfloor \frac{1000 \cdot 113}{4 \cdot 4 \cdot 113} \right\rfloor = 62$ такта, которых более чем достаточно, для большинства распространённых алгоритмов [8][9]. При использовании кристалла большего объёма, однако, необходимо изучать используемый алгоритм логарифмирования. Возможно, один блок последовательного логарифмирования не будет успевать подготавливать логарифмированные входные данные, и подход с использованием одного конвейеризованного блока окажется выгоднее, с точки зрения требуемых ресурсов кристалла.

Для вычисления экспоненты был разработан и протестирован схемотехнический модуль, реализующий приведённую в разделе 2.4 формулу вычисления экспоненты с помощью алгоритма поворота единичного вектора CORDIC. Этот конвейеризованный модуль обрабатывает данные в потоковом режиме, его место в схемотехнической реализации – внутри модуля NeuroWorker, между деревом мультиплексоров, выделяющим эффективный результат из множества выходов вычислительных конвейеров, и непосредственным выходом модуля NeuroWorker.

3.4 Драйвер устройства

Для использования устройства SLEDv7 необходимо некоторое программное обеспечение, отвечающее за идентификацию устройства в системе и обращение к устройству. Поэтому в ходе данной работы возникла необходимость создания драйвера устройства SLEDv7 для семейства операционных систем Microsoft Windows. К разрабатываемому драйверу были выдвинуты следующие функциональные требования:

- Драйвер должен исполняться на уровне ядра
- Идентификация устройства по hardware ID
- Чтение и запись CSR регистров устройства
- Обработка прерываний устройства

- Передача данных по DMA

Для реализации драйвера использовалась среда разработки Windows Driver Foundation. Идентификация и работа с регистрами и прерываниями устройства была реализована стандартными средствами WDF в соответствии с общепринятыми при разработке драйверов уровня ядра подходами [10].

Реализация передачи данных по DMA основывалась на том, как передача данных организована со стороны устройства. Для записи данных устройство предоставляет два 32-битных регистра: управляющий регистр и адрес страницы дескрипторов. Управляющий регистр состоит из трёх полей: бит запуска/конца операции, 10-битное количество страниц для копирования и 21-битный номер страницы в устройстве, куда будет производиться запись. При записи драйвером 1 в бит запуска операции, передача данных начинается. По завершению передачи данных бит запуска сбрасывается устройством, и устройство также поднимает флаг прерывания. Передача данных происходит по страницам размером 4 килобайта, поэтому объём данных для передачи и адрес в адресном пространстве устройства указываются в страницах, а не в байтах.

Отдельного рассмотрения требует регистр адреса страницы дескрипторов. В этот регистр драйвер должен записать адрес 4-килобайтной страницы, размещённой в системном адресном пространстве. Эта страница должна быть заполнена 1024 адресами страниц, также лежащими в системном адресном пространстве. При запуске операции устройство копирует содержимое первых n из этих страниц в своё адресное пространство, где n – это количество страниц для копирования, указанное в управляющем регистре.

Чтение происходит точно так же, только копирование идёт из памяти устройства в страницы, перечисленные в странице дескрипторов.

Передача данных по DMA в драйвере была реализована следующим образом. При инициализации нового устройства драйвер запрашивает в

системном адресном пространстве буфер размером $\left(\frac{1024}{2^n}+1\right)\cdot 4096$ байт. Сначала n=0, если система отказывает драйверу, то n инкрементируется до тех пор, пока драйвер наконец не получит запрашиваемую память. В конце концов, драйвер имеет буфер на $\left(\frac{1024}{2^n}+1\right)$ страниц размером по 4 килобайта. Первая страница заполняется физическими адресами следующих за ней $\frac{1024}{2^n}$ страниц, и её собственный адрес записывается в регистр адреса страницы дескрипторов устройства. Когда приложение запрашивает у драйвера запись данных в устройство по DMA, драйвер копирует предоставленные данные в этот буфер, начиная со второй страницы, указывает в управляющем регистре адрес и размер передачи и запускает операцию.

Чтение из устройства реализовано аналогично, для записи и чтения заводятся два разных буфера, чтобы обеспечить возможность одновременной записи и чтения. DMA операции сделаны блокирующими, инициировав передачу драйвер не возвращает управление приложению, а засыпает на объекте ядра. Этот объект переводится в активное состояние обработчиком прерываний драйвера при завершении соответствующей операции. Также за счёт внутренней синхронизации драйвер является thread-safe и прозрачным с точки зрения многопоточности. Запросив передачу данных в случае, когда передача такого типа уже инициирована другим потоком, приложение не получает сообщения об ошибке, а блокируется внутри драйвера до тех пор, пока текущая (и уже запланированные) операции не закончатся. Копирование данных из пользовательского адресного пространства в системное, сделано с помощью перехода в пользовательский контекст исполнения средствами WDF.

Помимо функций общего назначения, в созданный драйвер лабораторного стенда SLEDv7 была добавлена специализированная функция для использования разработанной схемотехнической реализации. Эта

функция принимает массив входных векторов в формате double*, объём входных данных и указатель на выделенный пользователем буфер для выходных данных. Эта функция логарифмирует входные вектора, приводит их к формату 8p24, используемому в текущей реализации комплекса, производит финальный этап вычислений алгоритма и приводит выходные данные из формата 8p24 в формат с плавающей точкой двойной точности. Приведения типов и арифметические операции производятся параллельно с вычислениями на устройстве.

Также для отладки, как драйвера, так и устройства в драйвере был реализован сбор статистики о количестве прерываний, объёме переданных данных и т.д. Для отладки устройства было создано программное обеспечение, позволяющее в интерактивном режиме изменять содержимое CSR регистров устройства, писать и читать данные из памяти устройства. Описание данного программного обеспечения приведено в приложении 2.

Заключение

В результате данной работы был создан программно-аппаратный комплекс, удовлетворяющий выдвинутым в главе 1 требованиям. В основе разработанного комплекса лежит схемотехническая реализация вычислительного конвейера, описанного в главе 2. Этот вычислительный конвейер является единицей масштабирования комплекса, и именно его характеристиками определяется быстродействие комплекса на базе какого-либо FPGA кристалла. В следующей таблице приведены занимаемые одним конвейером ресурсы кристалла и производительность конвейера, полученные при синтезировании схемотехнической реализации с одним конвейером на кристалле XC6VSX315T на частоте 125MHz:

Slice registers	41135
LUTs	40296
36k RAMBs	45
DSP48E1	279
Средняя производительность, точек/с	120 000

Таблица 1. Характеристики вычислительного конвейера

Ресурсы кристалла XC6VSX315T, используемого в устройстве SLEDv7, на использование которого в свою очередь ориентирована данная работа, позволили параллельно разместить четыре таких вычислительных конвейера. Таким образом, в ходе данной работы был получен программно-аппаратный комплекс, способный решать нейросетевой аналог прямой задачи ВИКИЗ со средней скоростью 480 тыс. векторов входных значений в секунду. В качестве дальнейшего развития работы была рассмотрена возможность переноса аппаратной части комплекса на кристалл XC7VX690T. Без каких-

либо изменений и адаптаций в схемотехнической реализации этот кристалл может вместить в себя двенадцать вычислительных конвейеров. При запуске на тактовой частоте 125 MHz такая аппаратная база поднимет среднюю производительность комплекса до 1,5 млн. точек в секунду. Также при переходе на этот кристалл прогнозируется возможность поднятия тактовой частоты до 200 MHz. Напомним, что изначально перед разрабатываемым комплексом ставилось минимальное требование по производительности – обработка не менее 25 000 точек в секунду, а измеренная в ходе работ производительность последовательной реализации алгоритма на процессоре Intel Core i7-3770 @ 3.40GHz составила 6400 точек в секунду.

Логическая организация программно-аппаратного комплекса такова, что при 100% загрузке аппаратной части, необходимый объём передачи данных между устройством и ПК составляет около N*4 МБ/с, где N – количество вычислительных конвейеров в устройстве. Даже при использовании кристалла XC7VX690T и 12 вычислительных конвейеров, комплекс не загрузит шину PCIe даже на треть от максимальной пропускной способности шины.

Также из качественных характеристик разработанного комплекса стоит отметить следующее: время вычисления алгоритма на FPGA кристалле строго определено с точностью до наносекунд, а любые отклонения в реальном времени решения задачи обусловлены задержками операционной системы при передаче входных и выходных данных между устройством и пользовательским контекстом процесса. Таким образом, перед нейросетевым аналогом прямой задачи ВИКИЗ, разработанным А.Ю. Соболевым, открывается еще одна перспектива – если использовать аппаратную базу созданного комплекса не в составе ПК, вычислительной системы общего назначения, а в специализированной вычислительной системе реального времени, то такой программно-аппаратный комплекс будет давать гарантированное время работы.

Литература

1. Лысаков К.Ф. Применение программно-аппаратных комплексов на базе FPGA для реализации трудоемких распараллеливаемых алгоритмов / Лысаков К.Ф., Шадрин М.Ю.: Труды III Международной конференции «Инфокоммуникационные и вычислительные технологии и системы». Улан-Удэ: Издательство Бурятского госуниверситета, 2010. С. 192-195.

2. Ельцов И. Н. Нейросетевое моделирование сигналов ВИКИЗ / Ельцов И. Н., Охонин В. А., Симонов К. В., Соболев А.Ю., Эпов М. И.: Электрические и электромагнитные методы исследования в нефтегазовых скважинах. – Новосибирск: Изд-во СО РАН, НИЦ ОИГГМ. – 1999. – С. 79-85.

3. Ельцов И. Н. Нейросетевое моделирование сигналов ВИКИЗ / Ельцов И. Н., Симонов К. В., Соболев А. Ю.: Каротажник. – 2006. - № 9. – С. 136-152.

4. K. Chapman. Fast integer multipliers fit in FPGAs – EDN Magazine, 1994, – 14 p.

5. J. H. Anderson, B. Teng. Latch-based performance optimization for FPGAs – Toronto, ON, Canada, University of Toronto, Dept. of ECE, 2009. – 6 p.

6. S. Golson. State machine design techniques for Verilog and VHDL - Mountain View, CA, USA, Synopsys, 1994 – 48 p.

7. K. Chapman. Multiplexer selection – San Jose, CA, USA: Xilinx Inc., 2008 – 14 p.

8. H. Ding, J. Hu, Yu. Shang, Ch. Wang, Sh. Wang An FPGA Implementation of the Natural Logarithm Based on CORDIC Algorithm - Research Journal of Applied Sciences, Engineering and Technology 6(1) – Maxwell Scientific Organization, 2013, p. 119-122

9. A. Hermanek, J, Kadlec, M. Licko, R. Matousek. FPGA implementation of logarithmic unit – Prague, Czech Republic, Czech Technical University, Dept. of telecommunication engineering, 2002. – 7 p.

10. М. Руссинович Внутреннее устройство Microsoft Windows / М. Руссинович, Д. Соломон: Спб, Издательство Питер, 2008. – 969 с.

11. N. Lashkarian Advanced FPGA design techniques: multiple clock domains – San Jose State University, Electrical Engineering Department, 2008. – 19 p.

Приложение 1

Учебно-лабораторный стенд SLEDv7 представляет собой аппаратное решение, предоставляющее возможность использования ПЛИС XC6VSX315T. Помимо самого кристалла ПЛИС, устройство оснащено программирующим модулем, генератором тактовой частоты, оперативной памятью, различными интерфейсами передачи данных.

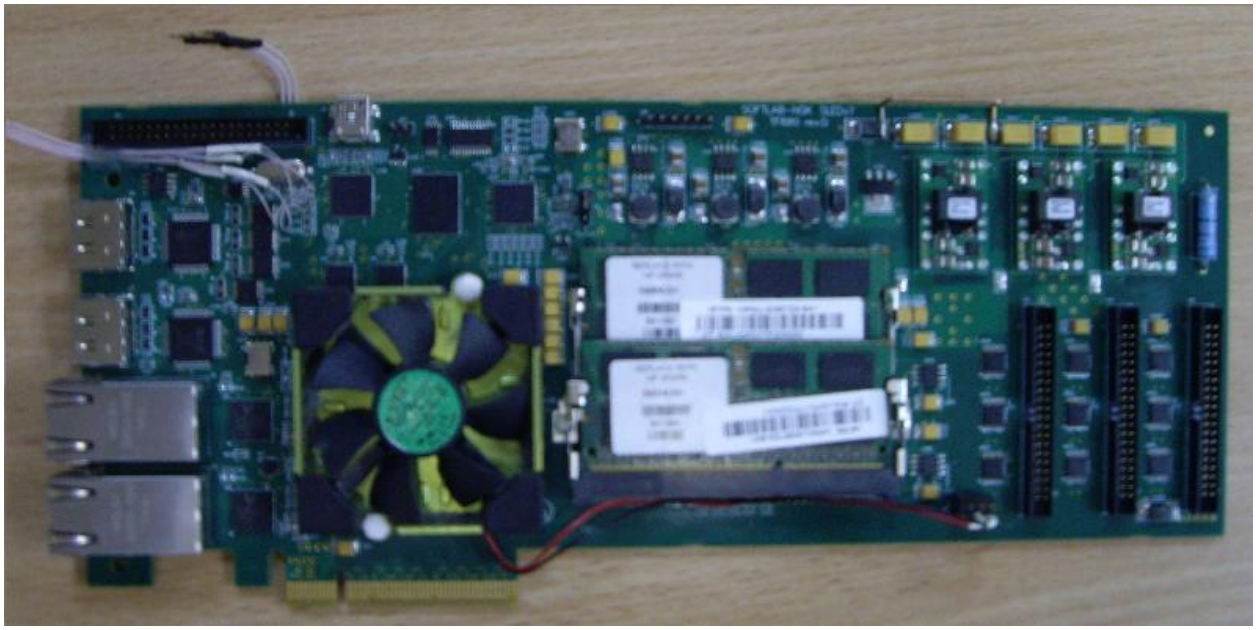

Рис.16. Учебно-лабораторный стенд SLEDv7

Учебно-лабораторный стенд SLEDv7 на базе ПЛИС обеспечивает следующую функциональность:

- работа в составе ПК в качестве спецпроцессора для обеспечения; высокой производительности обработки данных;
- работа без участия ПК в качестве процессорного средства обработки информации поступающей по внешним интерфейсам.

Учебно-лабораторный стенд SLEDv7 имеет следующие интерфейсы:

- Ввод мультимедиа-данных по интерфейсу HDMI/DVI;
- Вывод мультимедиа-данных по интерфейсу HDMI/DVI;
- 2 интерфейса 1Gb Ethernet для приема и передачи данных;
- USB для подключения внешних устройств;
- 9 интерфейсов SpaceWire для приема или передачи данных;
- PCI-E x8.

Приложение 2

Программа DeviceTest посредством графического интерфейса предоставляет пользователю возможность читать и писать состояния CSR-регистров устройства, постранично читать и писать содержимое оперативной памяти устройства, производить автоматизированные тесты памяти и регистров устройства, собирать статистику о работе драйвера устройства (количество DMA передач, количество вызовов ISR, средняя скорость передачи данных и т. д.). Доступны автоматизированные тесты памяти и регистров следующих типов:

- Запись константы в каждую ячейку;
- Поочерёдная запись двух констант;
- Бегущая единица/бегущий ноль;
- Запись в ячейку её адреса;
- Последовательная запись случайных значений;
- Запись случайных значений по случайным адресам.

Тесты запускаются в отдельных потоках, следовательно, их исполнение не блокирует работы программы; параллельно с запуском теста имеется возможность работы с устройством, остановки идущего теста. Любой из типов тестов также доступен в режиме одиночной записи.

Программа написана на языке программирования C++ с использованием библиотеки MFC.