

Министерство образования Республики Беларусь
Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей
Кафедра электронных вычислительных средств

К защите допустить:
Зав. кафедрой ЭВС
_____ И. С. Азаров

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к дипломному проекту
на тему

**IP-ЯДРО НЕЙРОННОЙ СЕТИ ПРЯМОГО РАСПРОСТРАНЕНИЯ ДЛЯ
РАСПОЗНОВАНИЯ РУКОПИСНЫХ ЦИФР**

БГУИР ДП 1-40 02 02 01 012 ПЗ

Студент Е. А. Кривальцевич

Руководитель М. И. Вашкевич

Консультанты:

от кафедры ЭВС М. И. Вашкевич

по экономической части И. В. Смирнов

Нормоконтролер ..

Рецензент ..

Минск 2025

Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Номер зачетной книжки_____

Преддипломная практика зачтена с оценкой

_____ (цифрой) _____ (прописью)

(подпись руководителя практики от БГУИР)

_____. _____. 2025

ОТЧЕТ

по преддипломной практике

Место прохождения практики: ЗАО «Инженерный центр ЯДРО»

Сроки прохождения практики: с 10.02.2025 по 23.03.2005

Руководитель практики от
предприятия:

П. Н. Габер

(подпись руководителя)
М.П.

Студент группы 150701

Е. А. Кривальцевич

(подпись студента)

Руководитель практики от БГУИР
Вашкевич М.И. – профессор
кафедры ЭВС

Минск 2025

Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Кафедра

Электронных вычислительных средств

УТВЕРЖДАЮ
Зав. кафедрой ЭВС

_____ И.С. Азаров
«06» марта 2025 г.

**ЗАДАНИЕ
на дипломный проект**

Обучающемуся

Кривальцевичу Егору Александровичу

(фамилия, собственное имя, отчество (если таковое имеется))

Курс 4 Учебная группа 150701
Специальность I-40 02 02 «Электронные вычислительные средства»
Тема дипломного проекта IP-ядро нейронной сети прямого распространения для
распознавания рукописных цифр

Утверждена приказом ректора

«06» марта 2025 г. № 584-с

Исходные данные к дипломному проекту

Назначение разработки: система предназначена для распознавания рукописных цифр.

Технические характеристики:

- размер изображения 28x28;
- цвет изображения: в оттенках серого;
- функция активации: softmax;
- формат представления данных: фиксированная точка;
- структура сети: многослойная.

Перечень подлежащих разработке вопросов или краткое содержание расчетно-пояснительной записи

1 Введение. 2 Обзор аналогичных разработок. 3 Анализ ТЗ. 4 Разработка структуры IP-блока нейронной сети для распознавания рукописных цифр. 5 Разработка IP-блока нейронной сети для распознавания рукописных цифр. 5.1 Представление исходных данных IP-блока нейронной сети для распознавания рукописных цифр. 5.2 Программная реализация модели нейронной сети на языке python. 5.3 Разработка электрической функциональной схемы. 6 Аппаратная реализация IP-блока нейронной сети для распознавания рукописных цифр. 7 Технико-экономическое обоснование дипломного проекта. 8 Анализ результатов проектирования. 9 Заключение. 10 Список используемых источников.

Перечень графического материала (с точным указанием обязательных чертежей и графиков

- 1 Схема электрическая структурная LST-1 преобразования – 1 лист формата А1.
- 2 Схема алгоритма работы LST-1 преобразования – 1 лист формата А1.
- 3 Схема электрическая структурная IP-блока нейронной сети – 1 лист формата А1.
- 4 Схема электрическая функциональная IP-блока нейронной сети – 1 лист формата А1.
- 5 Схема микропрограммы работы IP-блока нейронной сети – 1 лист формата А1.
- 6 Результаты проектирования – 1 лист формата А1 (плакат).

Консультанты по дипломному проекту (с указанием разделов, по которым они консультируют)

Старший преподаватель кафедры экономики Смирнов И.В., «Экономическое обоснование разработки нейронной сети для распознавания рукописных цифр»

Примерный календарный график выполнения дипломного проекта

| Наименование этапов дипломного проекта | Объём этапа в % | Срок выполнения этапа |
|--|-----------------|---|
| I этап | 40 | 24.03.25 |
| II этап | 20 | 11.04.25 |
| III этап | 20 | 05.05.25 |
| Нормоконтроль | | 12.05.25 – 21.05.25 |
| Рабочая комиссия | | 22.05.25 – 30.05.25 |
| Рецензирование | | 02.06.25 – 12.06.25 |
| Захита | | 13.06.25 – 30.06.25 (в соответствии с графиком заседаний ГЭК) |
| | | |
| | | |

Дата выдачи задания

« 6 » марта 2025 года

Срок сдачи студентом законченного дипломного проекта

«13» июня 2025 года

Руководитель дипломного проекта

_____ (подпись)

_____ (инициалы, фамилия)

Подпись обучающегося

_____ (подпись)

Дата « 6 » марта 2025 г

СОДЕРЖАНИЕ

| | |
|--|----|
| Введение | 6 |
| 1 Обзор существующих нейронных сетей для распознавания рукописных цифр | 8 |
| 1.1 Существующие нейронные сети для распознавания рукописных цифр | 8 |
| 1.2 Полносвязные нейронные сети | 9 |
| 1.2.1 Структура полносвязного слоя | 9 |
| 1.2.2 Функция активации ReLU | 10 |
| 1.2.3 Функция активации сигмоида | 11 |
| 1.2.4 Функция активации Softmax | 11 |
| 1.2.5 Функция активации гиперболический тангенс | 12 |
| 1.2.6 Обучение сети | 12 |
| 1.2.7 Анализ полносвязной нейрооной сети | 13 |
| 1.3 Сверточные нейронные сети | 13 |
| 1.3.1 Основные компоненты сверточной нейронной сети | 14 |
| 1.3.2 Операция свертки | 14 |
| 1.3.3 Слой подвыборки | 14 |
| 1.3.4 Обучение сверточной нейронной сети | 15 |
| 1.3.5 Анализ сверточной нейрооной сети | 15 |
| 1.4 Рекуррентные нейронные сети | 15 |
| 1.4.1 Основная идея рекуррентных нейронных сетей | 16 |
| 1.4.2 Проблемы стандартных рекуррентных нейронных сетей | 16 |
| 1.4.3 Применение RNN в распознавании рукописных цифр | 17 |
| 1.5 Трансформеры нейронные сети | 17 |
| 1.5.1 Архитектура трансформера | 17 |
| 1.5.2 Преимущества трансформеров | 18 |
| 1.5.3 Применение трансформеров в распознавании рукописных цифр | 18 |
| 1.6 Гибридные архитектуры | 18 |
| 1.6.1 Комбинация CNN и RNN | 18 |
| 1.6.2 Сеть CNN-Transformer | 19 |
| 1.6.3 Применение гибридных архитектур в распознавании рукописных цифр | 19 |
| 2 Анализ технического задания | 21 |
| 2.1 Анализ требований к нейронной сети | 21 |
| 2.2 Анализ требований к аппаратной реализации | 21 |
| 2.3 Выбор и обоснование метода решения задачи | 21 |
| 3 Разработка структуры IP-блока нейронной сети для распознавания рукописных цифр | 23 |
| 3.1 Функциональная спецификация системы | 23 |
| 3.2 Разбиение системы на модули | 23 |
| 3.3 Выбор соотношения между аппаратными и программными средствами | 24 |
| 3.4 Описание структурной схемы | 24 |
| 3.4.1 Принцип работы LST преобразования | 24 |
| 3.4.2 Принцип работы нейронной сети прямого распространения | 26 |
| 4 Разработка IP-блока нейронной сети для распознавания рукописных цифр | 29 |
| 4.1 Представление исходных данных | 29 |
| 4.2 Обучение нейронной сети | 30 |
| 4.3 Программная реализация эталонной модели нейронной сети на языке Python | 31 |

| | | |
|-------|---|-----------|
| 4.4 | Разработка операционной части нейронной сети | 32 |
| 4.5 | Проектирование функциональной схемы нейронной сети | 33 |
| 4.6 | Построение микропрограммы работы нейронной сети | 34 |
| 4.7 | Проектирование управляющей части | 35 |
| 4.8 | Построение графа переходов управляющего автомата | 37 |
| 5 | Аппаратная реализация IP-блока нейронной сети для распознавания рукописных цифр | 39 |
| 5.1 | Описание пакета Xilinx Vivado | 39 |
| 5.2 | Описание функциональных блоков | 40 |
| 5.3 | Описание интерфейса нейронной сети | 40 |
| 5.4 | Описание процесса создания блочной диаграммы | 40 |
| 5.5 | Результаты синтеза и симуляции | 42 |
| 6 | Технико–экономическое обоснование разработки IP–блока нейронной сети для распознавания рукописных цифр | 45 |
| 6.1 | Характеристика программного средства, разрабатываемого для собственных нужд | 45 |
| 6.2 | Расчет инвестиций в разработку программного средства для собственных нужд | 45 |
| 6.3 | Расчет экономического эффекта от использования программного средства для собственных нужд | 46 |
| 6.4 | Расчет показателей экономической эффективности разработки и использования программного средства в организации | 46 |
| 7 | Анализ результатов тестирования IP-блока нейронной сети для распознавания рукописных цифр | 48 |
| 7.1 | Описание среды тестирования | 48 |
| 7.2 | Тестирование разработанного IP-блока нейронной сети | 48 |
| 7.3 | Экспериментальное исследование IP-блока нейронной сети | 48 |
| 7.3.1 | Описание проводимого эксперимента | 48 |
| 7.3.2 | Подготовка данных для проведения эксперимента | 48 |
| 7.3.3 | Результаты проведенного эксперимента | 48 |
| 7.4 | Анализ результатов тестирования | 48 |
| | Заключение | 49 |
| | Список использованных источников | 50 |
| | Приложение А (Обязательное) Отчет о проверке на заимствование | 51 |
| | Приложение Б (Обязательное) Схема электрическая структурная LST-1 | 52 |
| | Приложение В (Обязательное) Схема алгоритма работы слоя LST-1 | 54 |
| | Приложение Г (Обязательное) Схема электрическая структурная | 56 |
| | Приложение Д (Обязательное) Python описание сети | 58 |
| | Приложение Е (Обязательное) Схема электрическая функциональная | 67 |
| | Приложение Ж (Обязательное) Схема алгоритма работы нейронной сети | 69 |
| | Приложение З (Обязательное) Verilog описание устройства | 71 |
| | Приложение И (Обязательное) Результаты работы нейронной сети | 90 |

ВВЕДЕНИЕ

Распознавание рукописных цифр является одной из ключевых задач в области обработки изображений и компьютерного зрения. Оно находит применение в различных сферах, таких как банковские системы, почтовые службы, идентификация документов и автоматизация процессов ввода данных. Современные технологии позволяют решать эту задачу с высокой точностью благодаря использованию методов машинного обучения и нейронных сетей.

В последние годы значительный интерес вызывает аппаратная реализация нейронных сетей, поскольку программные решения, работающие на центральных процессорах (CPU) и графических процессорах (GPU), не всегда обеспечивают требуемую скорость обработки и энергоэффективность. В связи с этим использование специализированных аппаратных ускорителей на базе FPGA (Field-Programmable Gate Array) или ASIC (Application-Specific Integrated Circuit) становится актуальной задачей. Аппаратная реализация нейронной сети в виде IP-ядра позволяет значительно повысить производительность и снизить задержки обработки данных, что особенно важно для встроенных систем.

При реализации нейронных сетей на базе CPU и GPU как правило используются стандартизованные типы данных (чаще всего числа с плавающей запятой одинарной точности, реже целочисленные типы). При реализации на базе FPGA появляется возможность использовать для представления параметров нейронных сетей типов данных, обеспечивающих различную точность. Причем выбор точности представления напрямую будет влиять на аппаратные затраты[1].

Целью данного дипломного проекта является разработка IP-ядра нейронной сети прямого распространения для распознавания рукописных цифр. В данном проекте используется обученное двумерное разделяемое преобразование (LST2D), которое рассматривается как новый тип слоя нейронной сети. Он следует концепции распределения веса. Проектирование IP-ядра нейронной сети прямого распространения производилось в несколько этапов, отраженных в структуре пояснительной записи данного дипломного проекта.

Первоначально был проведён анализ существующих методов и архитектур нейронных сетей, применяемых для задач распознавания рукописных цифр. Рассматривались различные подходы, включая классические многослойные персептроны, свёрточные и рекуррентные нейронные сети. Итоги данного этапа изложены в первом разделе пояснительной записи.

На следующем этапе выполнен анализ технического задания, в результате которого была выбрана архитектура нейронной сети, наиболее подходящая для аппаратной реализации. Основные критерии выбора включают компактность модели и вычислительную эффективность. Результаты этого анализа представлены во втором разделе пояснительной записи.

Далее был проведён этап проектирования IP-ядра, включающий разработку его структуры и алгоритма работы. В третьем и четвёртом разделах пояснительной записи подробно описаны процесс проектирования и обучения, особенности аппаратной реализации и взаимодействие с внешними устройствами.

Для проверки работоспособности IP-ядра была выполнена его программная модель и симуляция. Реализация и тестирование работы модели на тестовом наборе данных MNIST были выполнены в среде Python и Vivado, с последующей верификацией на FPGA. Эти этапы освещены в пятом разделе пояснительной записи.

Следующим этапом является технико-экономическое обоснование разработки IP-ядра, включающее анализ затрат на реализацию и оценку преимуществ

аппаратного ускорения по сравнению с программными методами. Результаты данного этапа приведены в шестом разделе пояснительной записи.

Завершающим этапом стала оценка производительности разработанного IP-ядра, включая анализ быстродействия, потребления ресурсов и точности распознавания. Также было проведено сравнение с существующими решениями, что позволило выявить преимущества и возможные направления оптимизации. Результаты данного анализа представлены в седьмом разделе пояснительной записи.

Отчет о проверке на заимствования представлен в приложении А.

1 ОБЗОР СУЩЕСТВУЮЩИХ НЕЙРОННЫХ СЕТЕЙ ДЛЯ РАСПОЗНАВАНИЯ РУКОПИСНЫХ ЦИФР

1.1 Существующие нейронные сети для распознавания рукописных цифр

Распознавание рукописных цифр является одной из классических задач машинного обучения, для решения которой применяется широкий спектр нейронных сетей. В зависимости от сложности задачи, требований к точности, скорости обработки и аппаратных ограничений используются различные архитектуры, включая полносвязные сети, сверточные нейронные сети и более сложные гибридные модели.

Одним из первых методов распознавания рукописных цифр стали многослойные перцептроны (MLP), относящиеся к классу полносвязных нейронных сетей (FCNN). Такие сети состоят из входного слоя, нескольких скрытых слоев и выходного слоя, где каждый нейрон соединен со всеми нейронами предыдущего и последующего слоев. Несмотря на их простоту, они способны успешно решать задачу распознавания, но требуют большого количества параметров и вычислительных ресурсов, что делает их менее эффективными по сравнению с более специализированными архитектурами.

Для обработки изображений, включая рукописные цифры, более эффективными оказались сверточные нейронные сети (CNN). Эти сети включают сверточные слои, которые извлекают характерные признаки из изображения, а также слои субдискретизации для уменьшения размерности. CNN значительно превосходят полносвязные сети в точности и скорости работы, поскольку используют пространственные зависимости в данных и требуют меньше параметров.

Хотя рекуррентные нейронные сети (RNN) и их усовершенствованные версии, такие как Long Short-Term Memory Network (LSTM) и Gated Recurrent Unit Network (GRU), чаще применяются для обработки последовательных данных, они могут использоваться и для распознавания рукописных цифр. В частности, они эффективны в случаях, когда рукописные символы анализируются в контексте строки, например, при распознавании рукописного текста.

Современные архитектуры, такие как Vision Transformer (ViT) и его модификации, предлагают альтернативный подход к обработке изображений, основанный на механизме самовнимания. Хотя традиционно трансформеры использовались в обработке естественного языка (NLP), их успешное применение в компьютерном зрении позволило достичь новых высот в распознавании символов и цифр.

Для повышения точности и эффективности могут применяться гибридные подходы, комбинирующие преимущества разных типов нейронных сетей. Например, модели, совмещающие CNN и LSTM, успешно применяются для распознавания рукописного текста, где CNN используется для выделения признаков, а LSTM — для анализа последовательностей.

На сегодняшний день существует множество архитектур нейронных сетей, способных эффективно решать задачу распознавания рукописных цифр. Выбор конкретного метода зависит от требований к точности, вычислительным ресурсам и целевой платформе. В последующих разделах будет представлен детальный анализ наиболее распространенных нейросетевых моделей, применяемых для данной задачи.

1.2 Полносвязные нейронные сети

Полносвязная нейронная сеть (Fully Connected Neural Network) — это базовая архитектура искусственных нейронных сетей, в которой каждый нейрон одного слоя соединен со всеми нейронами следующего слоя. Такой тип архитектуры используется для различных задач, включая распознавание рукописных цифр, но чаще всего применяется в качестве классификатора после сверточных или рекуррентных слоев. Количество нейронов на каждом слое может отличаться от соседних слоев.

1.2.1 Структура полносвязного слоя

Полносвязная нейронная сеть состоит из следующих основных компонентов:

1 Входной слой — принимает данные (например, изображение 28×28 пикселей в задаче распознавания рукописных цифр, развернутое в вектор размером 784).

2 Скрытые слои — выполняют нелинейные преобразования входных данных. Обычно включают несколько таких слоев.

3 Выходной слой — содержит количество нейронов, соответствующее числу классов (например, 10 для цифр 0–9), и использует функцию активации, например softmax. Функция активации — это математическая функция, которая определяет, передаст ли нейрон сигнал дальше по сети.

Общая структура полносвязных нейронных сетей показана на рисунке 1.1.

Математически данную сеть можно описать формулой для вычисления выхода нейрона в полносвязном слое:

$$h = f(Wx + b) \quad (1.1)$$

где: x — входной вектор размерности d , W — матрица весов размерности $n \times d$, b — вектор смещений размерности n , $f(\cdot)$ — функция активации (например, ReLU, сигмоида), h — выходной вектор нейронов текущего слоя.

Для многослойной архитектуры выход слоя l является входом для следующего слоя, что можно описать уравнением:

$$h^{(l+1)} = f(W^{(l)} h^{(l)} + b^{(l)}) \quad (1.2)$$

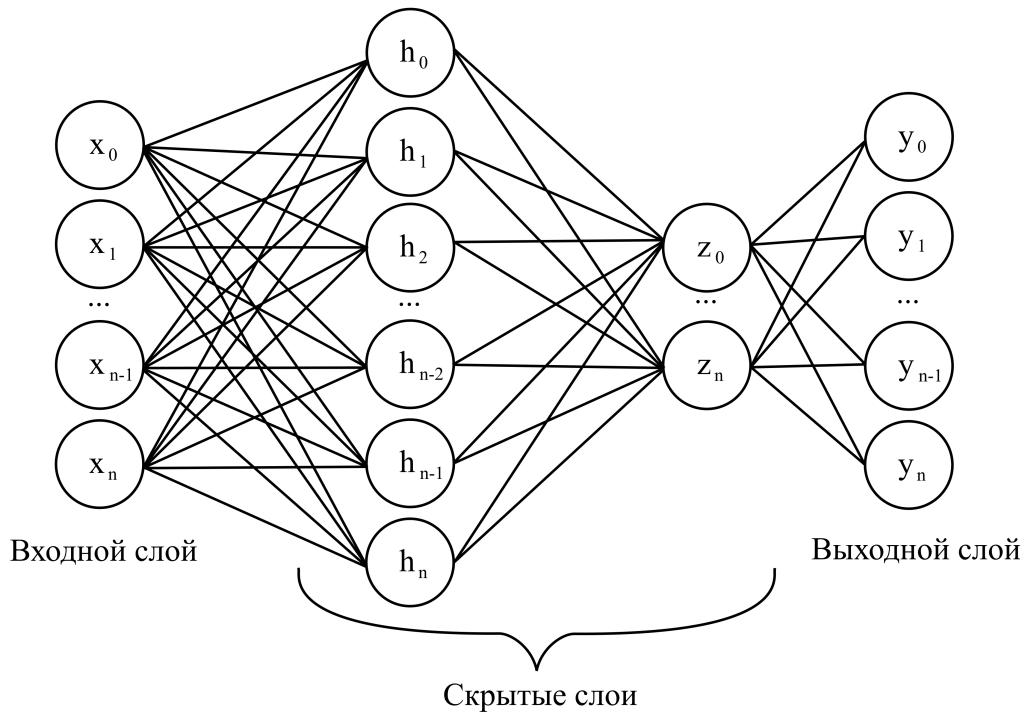


Рисунок 1.1 – Общая структура полносвязных нейронных сетей

1.2.2 Функция активации ReLU

ReLU (Rectified Linear Unit) — одна из наиболее популярных и широко используемых функций активации в нейронных сетях. Она определяется следующим образом:

$$f(x) = \max(0, x) \quad (1.3)$$

График функции ReLU представлен на рисунке 1.2.

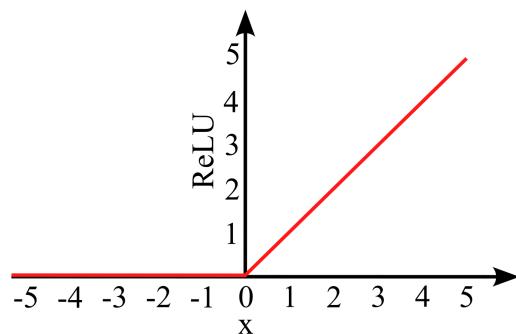


Рисунок 1.2 – График функции ReLU

Функция ReLU ускоряет обучение сети и устраняет проблему исчезающего градиента, так как имеет линейное поведение для положительных значений. Сама функция проста в реализации, но может быть чувствительна к большим значениям градиентов и в отрицательной области возвращает 0.

1.2.3 Функция активации сигмоида

Сигмоида (Logistic Function) — это одна из классических функций активации, используемая в нейронных сетях. Её математическое определение:

$$f(x) = \frac{1}{1 + e^{-x}} \quad (1.4)$$

График функции сигмоида представлен на рисунке 1.3.

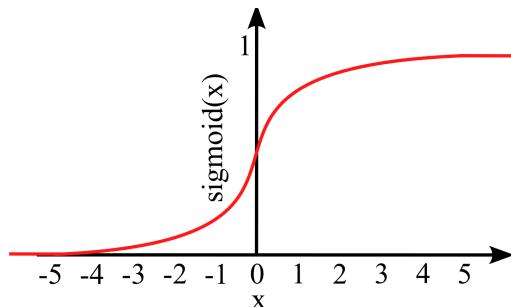


Рисунок 1.3 – График функции сигмоида

Сигмоида подходит для моделирования вероятностей, так как имеет диапазон значений (0, 1). Функция гладкая и дифференцируемая, что упрощает вычисление градиента, однако выходы не центрированы вокруг нуля, что замедляет обучение.

1.2.4 Функция активации Softmax

Функция Softmax применяется в многоклассовой классификации и преобразует входные значения в вероятностное распределение:

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}, \quad (1.5)$$

Гистограмма функции Softmax представлен на рисунке 1.4.

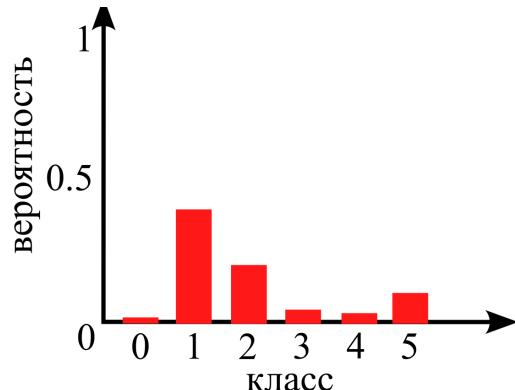


Рисунок 1.4 – Гистограмма функции Softmax

где z_i — входное значение для i -го класса, а знаменатель представляет собой сумму экспонент всех входных значений.

Softmax позволяет интерпретировать выходные значения сети как вероятности принадлежности к классам. Однако склонна к затуханию градиента при слишком больших входных значениях.

1.2.5 Функция активации гиперболический тангенс

Гиперболический тангенс (Tanh) — это сигмоидная функция, но симметричная относительно начала координат. Он определяется следующим образом:

$$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (1.6)$$

Функция принимает значения в диапазоне $(-1, 1)$, что делает её более предпочтительной по сравнению с сигмоидой в скрытых слоях нейронных сетей. Основные преимущества функции $\tanh(x)$:

1 Среднее значение её выходов ближе к нулю, что помогает ускорить сходимость сети.

2 Производная функции принимает большие значения в среднем диапазоне, что уменьшает проблему исчезающего градиента по сравнению с сигмоидой.

3 Хорошо подходит для задач, где требуется сбалансированный выход между положительными и отрицательными значениями.

График функции гиперболический тангенс представлен на рисунке 1.5.

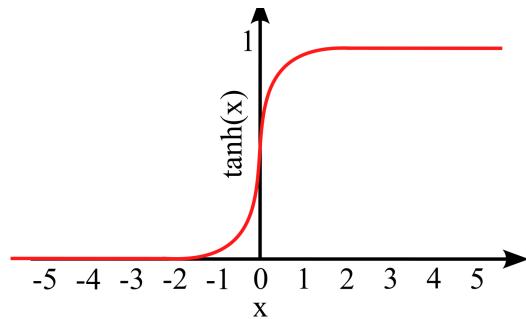


Рисунок 1.5 – График функции гиперболический тангенс

1.2.6 Обучение сети

Обучение FCNN осуществляется с использованием метода обратного распространения ошибки (Backpropagation) и оптимизационного алгоритма, например, стохастического градиентного спуска (SGD):

$$W^{(l)} \leftarrow W^{(l)} - \eta \frac{\partial L}{\partial W^{(l)}} \quad (1.7)$$

$$b^{(l)} \leftarrow b^{(l)} - \eta \frac{\partial L}{\partial b^{(l)}} \quad (1.8)$$

где η — скорость обучения, а L — функция потерь, например, кросс-энтропия для классификации:

$$L = - \sum_i y_i \log(\hat{y}_i) \quad (1.9)$$

где y_i – истинное значение, а \hat{y}_i – предсказанное значение.

Энтропия – измеряет степень неопределенности распределения вероятностей. А кросс-энтропия измеряет, насколько вероятностное распределение, предсказанное моделью, отличается от истинного распределения.

1.2.7 Анализ полно связной нейронной сети

Структура однослойной нейронной сети прямого распространения, состоящей из полно связного слоя с выходной функцией активации softmax показана на рисунке 1.6.

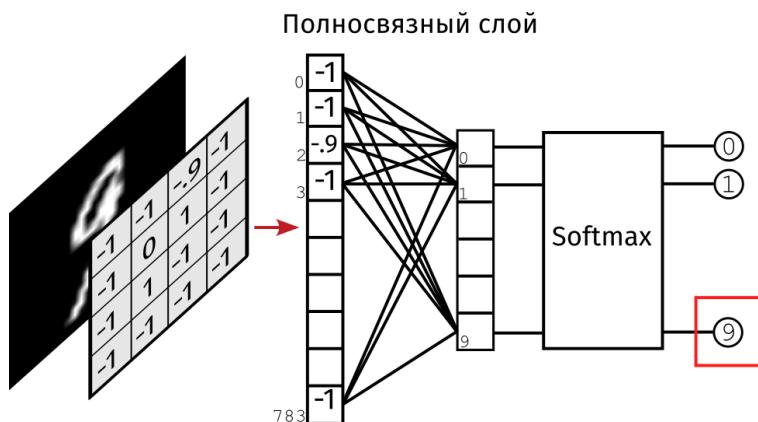


Рисунок 1.6 – Однослойной нейронной сети прямого распространения

Данная архитектура позволяет добиться точности 92,4%[2]. Нейронная сеть использует 8624 параметра, которые необходимо хранить в памяти в качестве весовых коэффициентов и смещений.

1.3 Сверточные нейронные сети

Сверточные нейронные сети (Convolutional Neural Networks) представляют собой архитектуру глубокого обучения, предназначенную для обработки данных обладающих сетчатой топологией, таких как изображения. Их основное преимущество заключается в способности эффективно извлекать пространственные и иерархические особенности входных данных с помощью операций свертки.

Принцип работы сверточных нейронных сетей показан на рисунке 1.7[3].

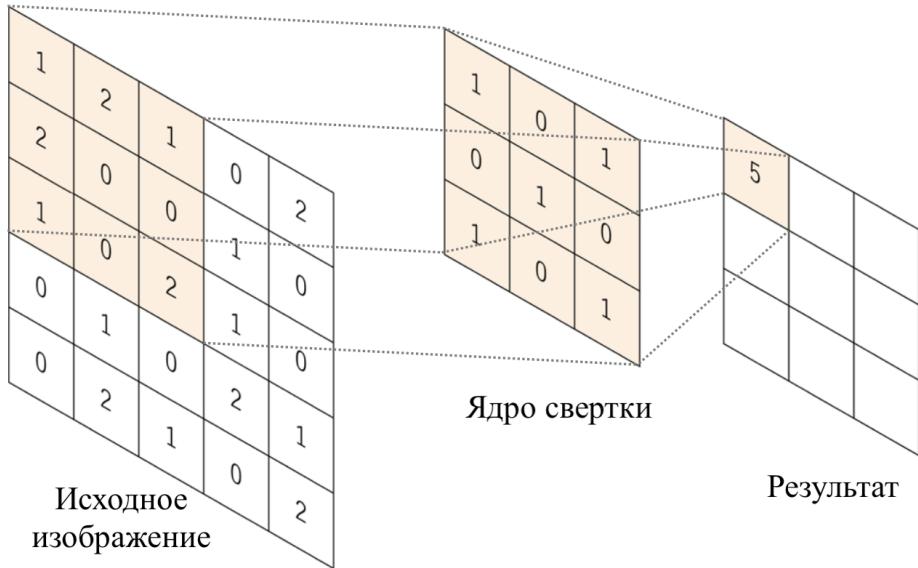


Рисунок 1.7 – Принцип работы сверточных нейронных сетей

1.3.1 Основные компоненты сверточной нейронной сети

Стандартная архитектура CNN включает в себя следующие основные слои:

- 1 Сверточные слои – выполняют операцию свертки, выделяя значимые признаки входных данных.
- 2 Функция активации – вводит нелинейность, наиболее часто используется ReLU.
- 3 Слои подвыборки (Pooling) – уменьшают размерность представления и повышают устойчивость к небольшим сдвигам.
- 4 Полносвязные слои – выполняют классификацию извлеченных признаков.
- 5 Функция потерь – используется для оценки ошибки модели, например, кросс-энтропия.

1.3.2 Операция свертки

Основная операция в сверточных нейронных сетях – это свертка, которая формально определяется следующим образом:

$$S(i, j) = \sum_m \sum_n X(i + m, j + n) \cdot K(m, n), \quad (1.10)$$

где: $X(i, j)$ – входное изображение, $K(m, n)$ – ядро свертки, $S(i, j)$ – выходное изображение после свертки.

Сверточный слой применяется к входному изображению, используя несколько фильтров, каждый из которых извлекает определенные характеристики.

1.3.3 Слои подвыборки

Для уменьшения размерности карт признаков применяется операция подвыборки. Один из наиболее распространенных методов – max pooling, который

выбирает максимальное значение в окне $k \times k$:

$$P(i, j) = \max_{(m,n) \in k \times k} S(i + m, j + n). \quad (1.11)$$

Этот метод помогает уменьшить объем вычислений и делает модель более устойчивой к сдвигам изображения.

1.3.4 Обучение сверточной нейронной сети

Обучение CNN происходит с использованием алгоритма обратного распространения ошибки и градиентного спуска. Для обновления параметров используется правило:

$$w^{(t+1)} = w^{(t)} - \eta \frac{\partial L}{\partial w}, \quad (1.12)$$

где η – коэффициент обучения, а $\frac{\partial L}{\partial w}$ – градиент функции потерь по параметру w .

1.3.5 Анализ сверточной нейронной сети

Структура сверточной нейронной сети, показатели точности которой 90% показана на рисунке 1.8[3].

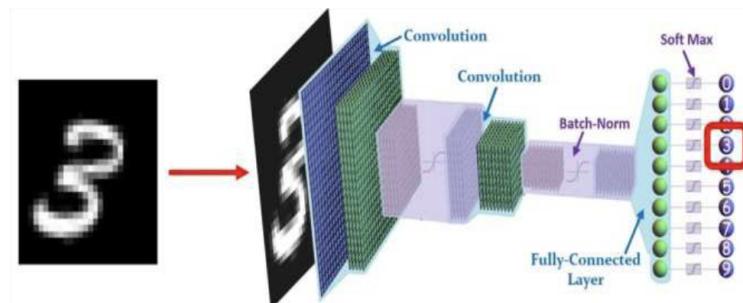


Рисунок 1.8 – Однослойной нейронной сети прямого распространения

Данная конфигурации состоит из:

- 1 Входной слой: [28×28]
- 2 Сверточный слой: 3 маски [3×3]
- 3 Слой пакетной нормализации
- 4 Слой ReLU
- 5 Полносвязанный слой
- 6 Слой Softmax
- 7 Слой классификации

Нейронная сеть использует 23520 параметра, которые необходимо хранить в памяти в качестве весовых коэффициентов и смещений.

1.4 Рекуррентные нейронные сети

Рекуррентные нейронные сети (Recurrent Neural Networks) предназначены для обработки последовательных данных, таких как текст, аудиозаписи, временные ряды и рукописные символы. В отличие от полносвязных и сверточных

сетей, рекуррентные обладают внутренним состоянием, позволяющим учитывать информацию из предыдущих шагов при обработке текущего входного сигнала.

Принцип работы рекуррентных нейронных сетей показан на рисунке 1.9.

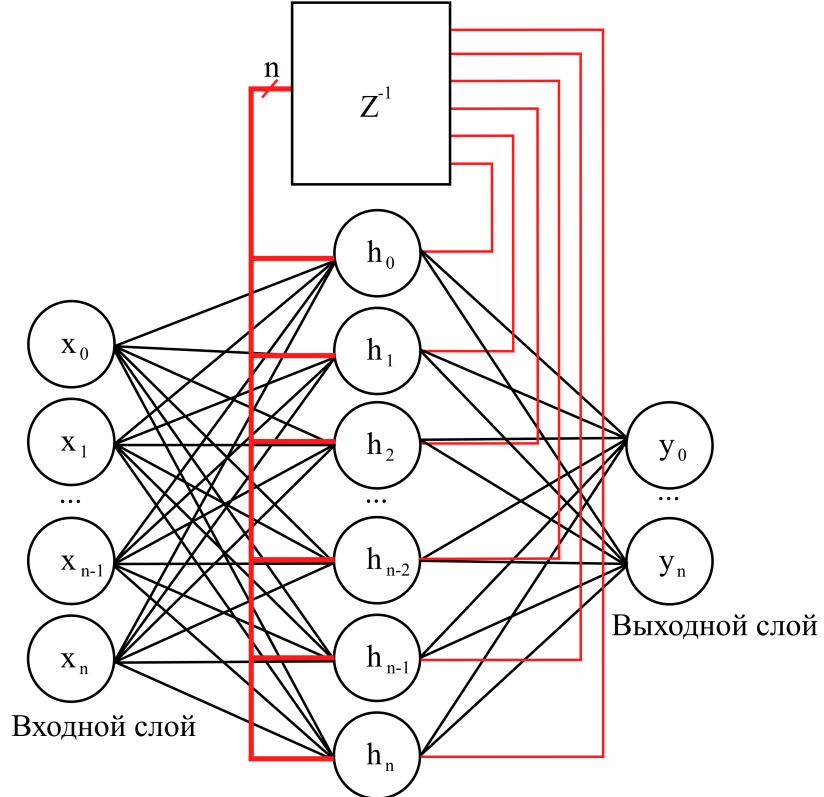


Рисунок 1.9 – Принцип работы рекуррентных нейронных сетей

1.4.1 Основная идея рекуррентных нейронных сетей

Основное отличие RNN от других типов нейронных сетей заключается в наличии обратных связей, позволяющих передавать информацию от предыдущих состояний. Для каждого временного шага t вычисляется новое состояние h_t на основе входных данных x_t и предыдущего состояния h_{t-1} :

$$h_t = f(W_x x_t + W_h h_{t-1} + b), \quad (1.13)$$

где: h_t — скрытое состояние в момент времени t ; x_t — входной вектор в момент времени t ; W_x , W_h — обучаемые весовые матрицы; b — вектор смещения; f — функция активации, например \tanh или $ReLU$.

Выход сети y_t определяется как:

$$y_t = g(W_y h_t + c), \quad (1.14)$$

где W_y — матрица весов выходного слоя, c — вектор смещения, а g — функция активации выходного слоя.

1.4.2 Проблемы стандартных рекуррентных нейронных сетей

Обычные RNN сталкиваются с проблемой затухающих и взрывающихся градиентов при обучении, что делает сложным обучение долгосрочных зависи-

мостей. Для решения этих проблем были разработаны улучшенные архитектуры, такие как: долгая краткосрочная память (Long Short-Term Memory) и управляемые рекуррентные блоки (Gated Recurrent Unit).

Эти архитектуры используют специальные механизмы управления потоками информации, такие как входные, выходные и забывающие ворота, позволяя эффективно обрабатывать длинные последовательности данных.

1.4.3 Применение RNN в распознавании рукописных цифр

Рекуррентные сети могут применяться для распознавания рукописных цифр, особенно в задачах последовательного анализа. Один из распространенных подходов — обработка строк рукописного текста, где каждый символ представлен как последовательность пикселей или векторных признаков. В таких задачах используются LSTM или GRU, способные учитывать пространственно-временные зависимости между элементами изображения. Пример архитектуры LSTM представлен на рисунке 1.10[9].

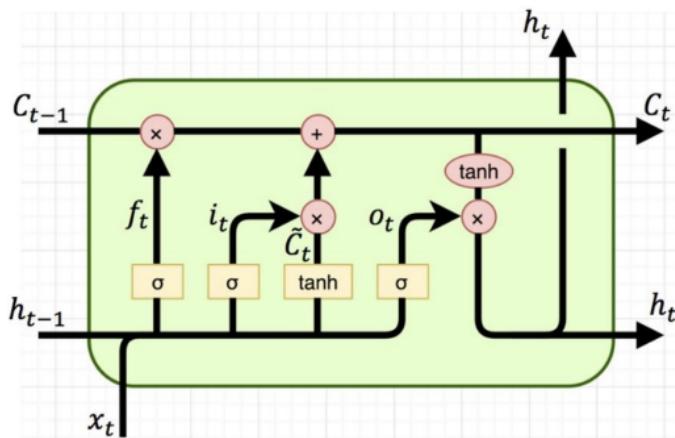


Рисунок 1.10 – Пример архитектуры LSTM

Данная архитектура позволяет решать задачу распознавания рукописного текста с точностью 94,7%.

1.5 Трансформеры нейронные сети

Трансформеры представляют собой архитектуру нейронных сетей, которая достигла значительных успехов в обработке последовательностей данных. В отличие от рекуррентных нейронных сетей, трансформеры используют механизмы самовнимания (self-attention) для обработки входной информации параллельно, что позволяет достигать высокой эффективности и точности.

1.5.1 Архитектура трансформера

Архитектура трансформера состоит из энкодера и декодера, каждый из которых содержит несколько слоев. Основные компоненты трансформера:

1 Механизм самовнимания (self-attention) — позволяет модели учитывать важность различных частей входных данных.

2 Многоголовочное внимание (multi-head attention) — улучшает способность модели учитывать разные аспекты входной информации.

3 Обратная связь (residual connections) — ускоряет обучение и предотвращает исчезновение градиента.

4 Механизм нормализации (layer normalization) — стабилизирует обучение.

5 Feed-forward сети — полносвязные слои, применяемые после механизма самовнимания.

Механизм самовнимания вычисляется следующим образом:

$$\text{Attention}(Q, K, V) = \text{softmax} \left(\frac{QK^T}{\sqrt{d_k}} \right) V, \quad (1.15)$$

где Q (query), K (key) и V (value) — матрицы, полученные линейными преобразованиями входных данных, а d_k — размерность ключей.

1.5.2 Преимущества трансформеров

В отличие от RNN, трансформеры обрабатывают входные данные одновременно, что ускоряет обучение. Механизм самовнимания позволяет эффективно учитывать контекст на больших расстояниях. Трансформеры могут применяться к различным задачам, включая обработку текста, изображений и речи.

1.5.3 Применение трансформеров в распознавании рукописных цифр

Хотя трансформеры изначально были разработаны для задач обработки естественного языка, они также успешно применяются для распознавания рукописных цифр. Основной подход заключается в преобразование изображений в последовательности пикселей и обработка их с помощью Vision Transformer (ViT).

Трансформеры обеспечивают высокую точность — 90,30% и используют большое количество параметров — 163729 при обработке изображений, особенно при наличии больших объемов обучающих данных, что делает их перспективным направлением для распознавания рукописных цифр[6].

1.6 Гибридные архитектуры

Гибридные архитектуры нейронных сетей представляют собой комбинацию различных типов нейронных сетей, объединяя их преимущества для улучшения производительности в задачах распознавания изображений, в том числе рукописных цифр. Такие модели могут включать элементы сверточных, рекуррентных, трансформерных и полносвязных сетей.

1.6.1 Комбинация CNN и RNN

Одним из наиболее распространенных гибридных подходов является использование сверточных нейронных сетей для извлечения признаков и рекуррентных нейронных сетей для обработки последовательностей.

CNN выполняет обработку изображений, извлекая пространственные признаки, которые затем передаются в RNN.

RNN, такие как LSTM или GRU, анализируют временные зависимости между последовательными фрагментами извлеченных признаков, что особенно полезно при обработке последовательностей символов или цифр.

Математически процесс можно описать следующим образом:

$$F = \text{CNN}(X) \quad (1.16)$$

$$h_t = \text{RNN}(F_t, h_{t-1}) \quad (1.17)$$

где X – входное изображение, F – извлеченные признаки, h_t – скрытое состояние RNN.

1.6.2 Сеть CNN-Transformer

Другой популярный гибридный подход – объединение CNN и трансформеров. CNN используются для локального извлечения признаков, а механизм внимания трансформера помогает учитывать глобальный контекст изображения.

CNN генерирует карту признаков изображения.

Трансформер обрабатывает полученные признаки, используя механизм самовнимания (Self-Attention).

Формально, этот процесс можно записать следующим образом:

$$F = \text{CNN}(X) \quad (1.18)$$

$$Z = \text{Transformer}(F) \quad (1.19)$$

где Z – закодированное представление входного изображения, учитывающее пространственные и контекстные зависимости.

1.6.3 Применение гибридных архитектур в распознавании рукописных цифр

Гибридные модели активно применяются для задач распознавания рукописных цифр, поскольку они позволяют достичь высокой точности за счет сочетания мощных методов обработки изображений и анализа последовательностей.

CNN-RNN используются для распознавания рукописных цифр в последовательностях, таких как почтовые индексы.

CNN-Transformer подходят для высокоточного классифицирования цифр в условиях зашумленных или искаженных изображений. Сеть, архитектура которой представлена на рисунке 1.11, позволяет добиться точности 99,89%[7].

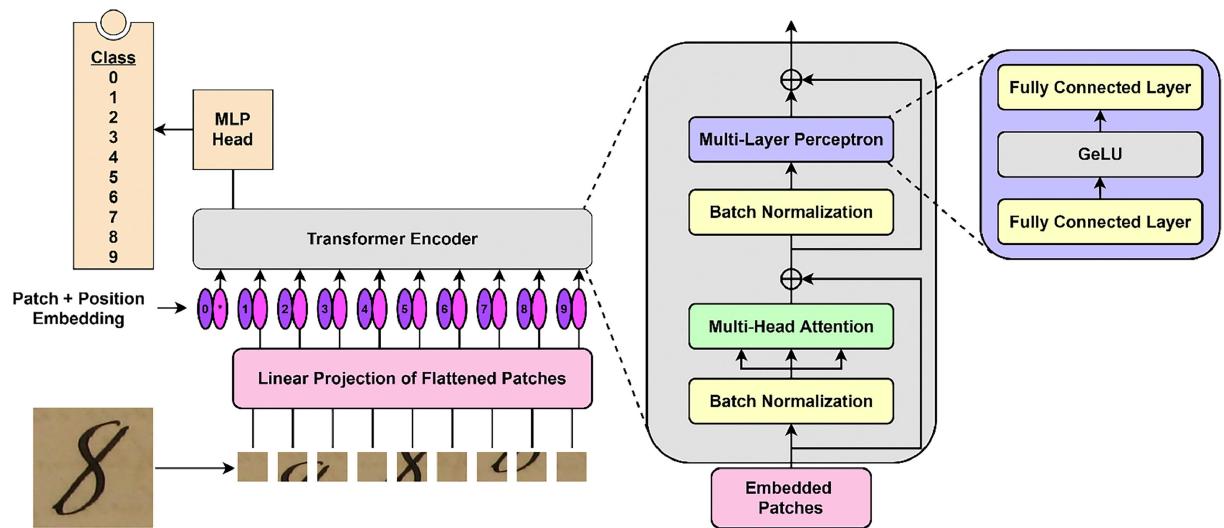


Рисунок 1.11 – Архитектура CNN-Transformer

Благодаря комбинированию разных методов гибридные архитектуры обеспечивают улучшенные результаты в задачах компьютерного зрения.

2 АНАЛИЗ ТЕХНИЧЕСКОГО ЗАДАНИЯ

2.1 Анализ требований к нейронной сети

Так как задача классификации изображений требует высокой точности и скорости работы, необходимо определить ключевые критерии, которым должна соответствовать модель:

1 Нейросеть должна демонстрировать точность классификации, сравнимую с современными методами машинного обучения. В частности, требуется достигать точности не менее 97% на тестовом наборе данных MNIST. При этом важно, чтобы сеть успешно обрабатывала различные вариации рукописных цифр, включая наклон, смещение и различную толщину линий.

2 Для реализации на аппаратном уровне необходимо минимизировать количество параметров модели, сохраняя при этом высокую точность. Это особенно важно для устройств с ограниченными ресурсами, таких как встраиваемые системы.

3 Ограниченные ресурсы памяти требуют компактного хранения параметров модели, а также эффективного использования памяти во время вычислений. Использование квантования весов или сжатых представлений поможет снизить потребление памяти.

4 Для получения вероятностного распределения классов используется функция активации softmax, позволяющая интерпретировать выходные значения сети в виде вероятностей принадлежности входного изображения к определенному классу.

2.2 Анализ требований к аппаратной реализации

Аппаратная реализация нейронной сети требует учета ряда важных факторов, которые определяют эффективность работы модели в ограниченной вычислительной среде.

Основные аппаратные требования включают:

1 Для работы на FPGA необходимо эффективно использовать аппаратные ресурсы, такие как DSP-блоки, BRAM (блоки встроенной памяти) и логические элементы.

2 Встроенные системы часто работают в условиях ограниченного энергопотребления, поэтому важно, чтобы архитектура сети была энергоэффективной.

3 Архитектура должна быть масштабируемой, чтобы можно было изменять количество слоев или нейронов в зависимости от доступных аппаратных ресурсов. Кроме того, должна быть возможность модификации модели для адаптации под новые задачи.

Основной задачей является выбор оптимальной архитектуры сети и методов вычислений, которые обеспечат баланс между производительностью и потреблением ресурсов.

2.3 Выбор и обоснование метода решения задачи

Для решения задачи распознавания рукописных цифр рассматриваются различные подходы, каждый из которых имеет свои преимущества и ограничения:

Полносвязные нейронные сети – обладают простой структурой и легко реализуются, однако требуют большого количества параметров. Это делает их

менее подходящими для аппаратной реализации, так как объем необходимых вычислений и памяти растет пропорционально числу нейронов в слоях.

Сверточные нейронные сети – обеспечивают высокую точность за счет эффективного извлечения признаков из изображений. Они используют свертки, позволяющие минимизировать количество параметров по сравнению с полно связанными сетями.

Рекуррентные нейронные сети – могут использоваться для последовательной обработки данных, однако их применение к изображениям ограничено. Такие сети лучше подходят для обработки временных рядов и речи, но могут быть адаптированы к изображениям, например, в архитектурах LSTM или GRU.

Трансформеры – показывают высокую производительность в задачах обработки естественного языка и компьютерного зрения, но требуют значительных вычислительных ресурсов. Несмотря на их преимущества, высокая сложность и потребление памяти делают их менее подходящими для работы на FPGA.

Гибридные архитектуры – могут комбинировать различные методы, такие как CNN и рекуррентные слои, что позволяет достичь оптимального баланса между точностью и вычислительными затратами.

LST2D архитектуры — основываются на архитектуре полно связанных слоев, что позволяет использовать минимальные вычислительные ресурсы.

На основании анализа требований к аппаратной реализации и доступных архитектур, предлагается использовать обучаемое двумерное разделимое преобразование (LST), которое можно рассматривать как новый тип вычислительного слоя для построения архитектуры нейронной сети[9]. Данная структура обеспечивает высокую точность классификации изображений благодаря эффективному представлению пространственных зависимостей и позволяет сократить вычислительные затраты за счет оптимизированной структуры слоев.

Основное внимание уделяется разработке IP-ядра, реализующего LST с оптимизированными вычислениями, что позволит эффективно использовать нейросеть в FPGA для задач классификации изображений.

Для реализации поставленных задач необходимо разработать структурную схему устройства. Описать алгоритм работы с учетом всех вышеперечисленных функций. По алгоритму описать модель на HDL языках.

3 РАЗРАБОТКА СТРУКТУРЫ IP-БЛОКА НЕЙРОННОЙ СЕТИ ДЛЯ РАСПОЗНАВАНИЯ РУКОПИСНЫХ ЦИФР

3.1 Функциональная спецификация системы

Для разработки системы распознавания рукописных цифр с использованием нейронной сети прямого распространения необходимо составить функциональную спецификацию, которая будет описывать требования к системе, её функциональные компоненты и интерфейсы с пользователем и окружением.

Функциональная спецификация системы распознавания рукописных цифр должна включать два основных компонента:

- 1 Список функций, выполняемых системой.
- 2 Описание интерфейса между системой и пользователем.

Система будет реализована как IP-блок нейронной сети прямого распространения, которая предварительно будет обучаться на наборе данных изображений рукописных цифр, а затем использоваться для их распознавания.

В контексте требований пользователя система должна отвечать на следующие вопросы:

- 1 Какие средства необходимо предусмотреть для ввода данных?
- 2 Как будет происходить обучение модели и её оценка?
- 3 Какие средства необходимо предусмотреть для вывода результатов?
- 4 Как будет осуществляться хранение весовых коэффициентов?
- 5 Какие средства необходимы для работы с пользователем?

Ответив на эти вопросы, можно составить функциональную спецификацию для разрабатываемой системы.

1 Данные о рукописных цифрах будут передаваться в виде последовательной передачи пикселей изображений, которые будут поступать в IP-блок по AXI4-lite интерфейсу из процессорной системы на ПЛИС.

2 Для обучения модели будет использоваться набор данных MNIST, который разделен на тренировочные и тестовые данные. Оценка качества модели будет проводиться с помощью метрик точности и потерь.

3 Результаты работы модели будут поступать в процессорную систему по интерфейсу AXI4-lite, где проходить обработку, например использовать в построении матрицы ошибок.

4 Хранение весовых коэффициентов будет осуществляться в блочной памяти (BRAM). Необходимо обеспечить поддержку трех весовых групп: первого слоя LST, второго слоя LST и выходного полно связного слоя.

5 Система будет предоставлять IP-блок, который будет подключаться по AXI4-lite интерфейсу к другим блокам системы.

3.2 Разбиение системы на модули

Система распознавания рукописных цифр на базе нейронной сети прямого распространения может быть разделена на несколько функциональных модулей:

1 Модуль памяти изображения, который хранит входное изображение и все промежуточные.

2 Модуль обработки пикселя, который передает пиксель и необходимый весовой коэффициент в модуль выполнения операции MAC.

3 Модуль MAC выполняет операцию умножения с накоплением входных данных.

4 Модули памяти весов, в которых хранятся весовые коэффициенты каждого слоя нейронной сети.

5 Модуль функции активации тангенса, который выполняет вычисление функции тангенс в LST слое.

6 Модуль функции активации softmax, который производит выбор результата распознавания.

7 Модуль управления, который осуществляет контроль над расчетом слоев нейронной сети.

3.3 Выбор соотношения между аппаратными и программными средствами

Для эффективного функционирования системы важно правильно выбрать соотношение между аппаратными и программными средствами.

Основу аппаратных средств для реализации IP-блока нейронной сети прямого распространения для распознавания рукописных цифр является ПЛИС Zynq Z7: Zynq-7000.

Программные средства, в свою очередь включают в себя все модули, описанные ранее а также фреймворк PyTorch, который используется для обучения нейронной сети и fixpoint для описания эталонной модели.

3.4 Описание структурной схемы

Для описания структуры IP-блока нейронной сети прямого распространения необходимо показать принцип работы LST-слоя.

3.4.1 Принцип работы LST преобразования

Двумерное разделимое преобразование используется в обработке изображений для уменьшения вычислительной сложности. Основная идея заключается в том, что вместо хранения полного двумерного ядра свёртки размером $n \times n$ используется его представление в виде произведения двух одномерных векторов. Это позволяет сократить число параметров с n^2 до $2n$, что снижает затраты на вычисления и память.

$$W = V \times h^T, \quad (3.1)$$

где $W \in \mathbb{R}^{n \times n}$, $v, h^T \in \mathbb{R}^{n \times 1}$

LST основан на идее совместного использования весов одного полностью связанных слоя для обработки всех строк изображения. После этого второй общий полно связанный слой используется для обработки всех столбцов представления изображения, полученных из первого слоя. Использование слоев LST в архитектуре NN значительно сокращает количество параметров модели по сравнению с моделями, которые используют стекированные полно связанные слои.

LST можно использовать для построения архитектур глубоких нейронных сетей (DNN), заменяя традиционную нейронную сеть прямого распространения. Нейронная сеть на основе LST позволяет достичь высокой точности, при этом количество параметров модели значительно меньше, чем в полно связной нейронной сети.

Предложенное обученное двумерное разделяемое преобразование (LST2D) обрабатывает изображение построчно, а не преобразует его в вектор, как это

обычно делается в полносвязных сетях.

С математической точки зрения LST2D принимает двумерное изображение X размером $d_{in} \times d_{in}$ в качестве входных данных и создает двумерное выходное изображение Y размером $d_{out} \times d_{out}$:

$$Y = LST_{d_{in} \times d_{out}}(X) = \tan(W_2 \tan(W_1 X^T)) \quad (3.2)$$

где W_1, W_2 — матрицы весов слоев FC1 и FC2 соответственно (рисунок 3.1)[9].

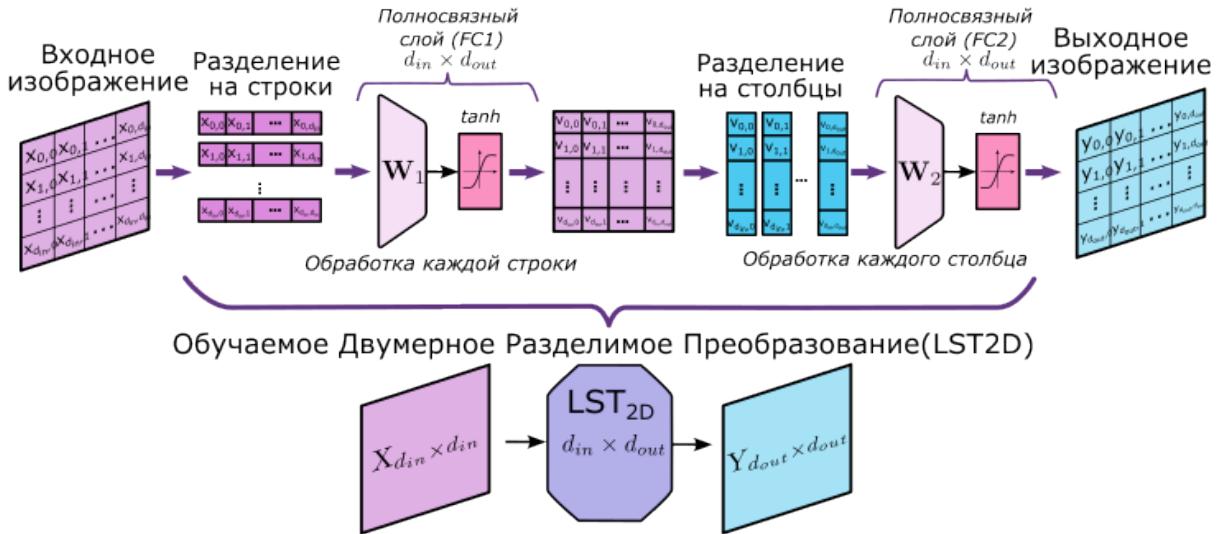


Рисунок 3.1 – Принцип работы LST2D

Здесь d_{in} и d_{out} — гиперпараметры преобразования, определяющие количество обучаемых параметров, равное $N=2 \cdot (d_{in}+1) \cdot d_{out}$.

Простейшая архитектура нейронной сети на основе LST2D для распознавания цифр MNIST требует одного блока LST2D, за которым следует полносвязный слой с функцией активации softmax. Эта архитектура представлена на рисунке 3.2[9].

Структурная схема LST-1 преобразования представлена в приложении Б. Алгоритм работы LST-1 преобразования представлен в приложении В.

Данную архитектуру можно масштабировать. Пример двухблочной архитектуры LST2D приведен на рисунке 3.3[9].

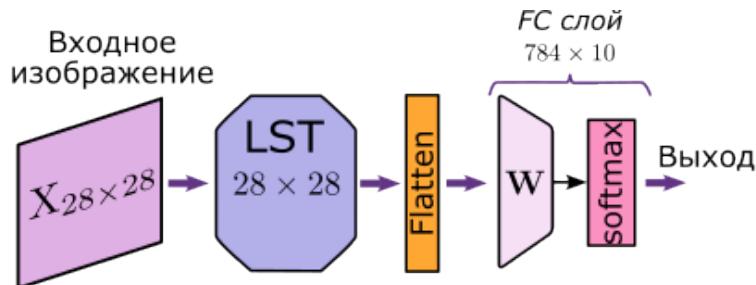


Рисунок 3.2 – Принцип работы LST-1

Модель LST-2 имеет дополнительный параметр dh , который определяет размерность $dh \times dh$ скрытого представления входного изображения.

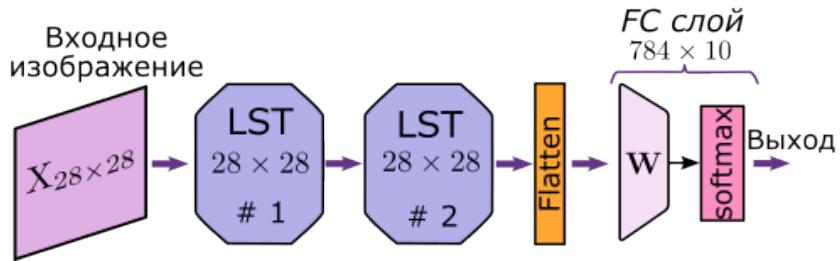


Рисунок 3.3 – Принцип работы LST-2

Пример обработки изображения нейронной сетью архитектуры LST-1 представлен на рисунке 3.4[9].

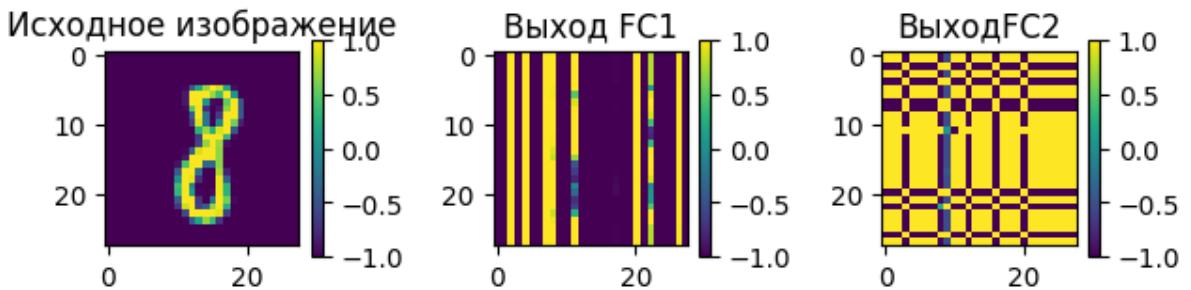


Рисунок 3.4 – Архитектура CNN-Transformer

3.4.2 Принцип работы нейронной сети прямого распространения

Принцип работы данной нейронной сети основан на комбинации обучаемого двумерного разделимого преобразования и полносвязной нейронной сети. Такая архитектура позволяет эффективно обрабатывать изображения, снижая вычислительные затраты при аппаратной реализации.

Для окончательной классификации используется полносвязный слой с функцией активации softmax. Данный слой преобразует выходные значения в вероятностное распределение по классам, позволяя определить, к какой цифре относится входное изображение.

Вычислительное ядро разработанного устройства состоит из 28 MAC (Multiply-ACcumulate) ядер, распределенных по специализированным блокам. Эти блоки организованы таким образом, чтобы обеспечить эффективное выполнение операций умножения входного вектора изображения на матрицу весов слоя нейронной сети.

Десять MAC-ядер из общего массива используются на финальном этапе вычислений и непосредственно участвуют в формировании входных данных для функции активации softmax. Эти ядра получают входные данные из трех независимых блоков памяти, что позволяет оптимизировать процесс выборки весов и повысить скорость вычислений. Остальные 18 MAC-ядер распределены по блокам, где каждая вычислительная единица получает данные из двух запоминающих устройств.

IP-блок устройства предназначен для интеграции в систему на кристалле (СнК) и взаимодействует с процессорной системой через uP-интерфейс. СнК – это интегральная схема, которая объединяет в одном чипе все основные компоненты вычислительной системы: процессор, ОЗУ (RAM) и ПЗУ (ROM),

графический процессор (GPU), контроллеры периферии. Последовательность обработки данных организована следующим образом:

1 Исходное изображение размером 28×28 пикселей поступает на вход сети через переходник интерфейсов AXI-uP и загружаются в память.

2 Каждый пиксель передается в 28 MAC-ядер, которые выполняют умножение с соответствующими весовыми коэффициентами, извлекаемыми из памяти. Таким образом реализуется два этапа LST-1 обработки.

3 Управляющее устройство синхронизирует процесс обработки, обновляя адреса памяти и координируя передачу данных между блоками.

4 По завершении вычислений слоя LST-1 результирующее изображение передается на 10 MAC-ядер, где происходит окончательное формирование выходного вектора размерностью 10, каждый элемент которого соответствует весовому классу.

5 Этот массив поступает в блок функции активации softmax, который выбирает класс с наибольшей вероятностью.

6 Индекс класса передается на выход и отправляется в процессорную систему через uP и AXI4-lite интерфейсы. uP-интерфейс – это традиционный способ взаимодействия IP-блока с процессорной системой. Данный интерфейс предоставляет простую регистровую модель для обмена командами и данными[12].

Основные сигналы uP-интерфейса:

1 ADDR – адрес регистра, к которому выполняется обращение.

2 DATA_IN – входные данные, записываемые в регистр.

3 DATA_OUT – выходные данные, считываемые из регистра.

4 RD – сигнал чтения из регистра.

5 WR – сигнал записи в регистр.

6 CLK – синхросигнал интерфейса.

7 RESET – глобальный сброс интерфейса.

8 READY – флаг готовности к передаче данных.

AXI4-Lite – это упрощённый вариант AXI4-интерфейса, предназначенный для управления и обмена небольшими порциями данных. В отличие от AXI4, он поддерживает только однослотовые транзакции без разделения на несколько пакетов[11].

Основные сигналы AXI4-Lite:

1 AWADDR – адрес записи.

2 AWVALID – флаг валидности адреса записи.

3 WDATA – записываемые данные.

4 WVALID – сигнал валидности данных.

5 BREADY – подтверждение завершения записи.

6 ARADDR – адрес чтения.

7 ARVALID – флаг валидности адреса чтения.

8 RREADY – готовность принять данные.

9 RDATA – считанные данные.

10 RVALID – сигнал валидности данных.

11 ACLK – системный тактовый сигнал.

12 ARESETN – асинхронный сброс, активный по нулю.

Оба интерфейса широко применяются в проектировании цифровых систем. uP-интерфейс удобен для работы с простыми микроконтроллерами, тогда как AXI4-Lite предоставляет стандартизованный метод взаимодействия с процессорными системами, такими как ARM-ядра на FPGA. При выборе интерфейса следует учитывать требуемую скорость обмена и сложность интеграции в

систему.

Использование распределенной памяти и многопоточной обработки позволяет значительно ускорить вычисления, минимизировать задержки при выборке данных и повысить общую эффективность системы.

Электрическая структурная схема IP-блока нейронной сети прямого распространения приведена в приложении Г.

4 РАЗРАБОТКА ИР-БЛОКА НЕЙРОННОЙ СЕТИ ДЛЯ РАСПОЗНАВАНИЯ РУКОПИСНЫХ ЦИФР

4.1 Представление исходных данных

В качестве исходных данных используется набор MNIST, содержащий 70 тысяч полутоновых изображений размера 28×28 пикселей, представляющих рукописные цифры от 0 до 9 [10]. Данный набор разделен на две части:

- Тренировочная выборка – 60 тысяч изображений, используемых для обучения модели.
- Тестовая выборка – 10 тысяч изображений, предназначенных для проверки качества работы обученной нейросети.

Пример данных из базы MNIST представлен на рисунке 4.1.

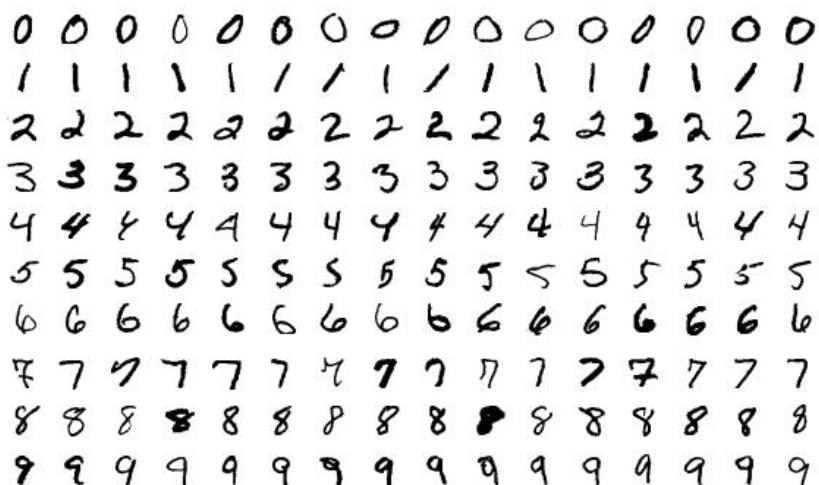


Рисунок 4.1 – Пример данных из MNIST

Перед подачей в нейросеть изображения проходят предварительную обработку, включающую нормализацию значений пикселей. Каждый пиксель исходного изображения представлен числом в диапазоне от 0 до 255 (градации серого). Для упрощения процесса обучения выполняется нормализация данных таким образом, чтобы значение каждого пикселя было в диапазоне от -1 до 1 , а стандартное отклонение (σ) составляло 0.5 . Это позволяет сделать градиентный спуск более устойчивым и ускорить процесс сходимости модели. Пример такой обработки представлен на рисунке 4.2.

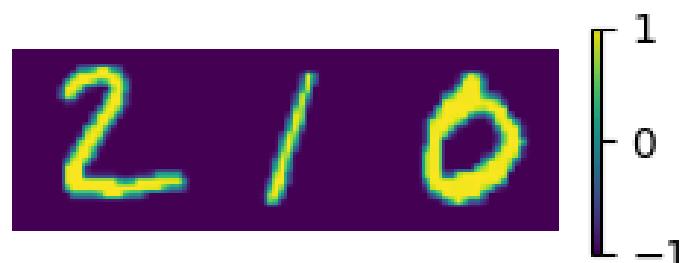


Рисунок 4.2 – Пример обработанных данных из MNIST

Для аппаратной реализации на FPGA числа представляются в формате с фиксированной запятой (Fixed-Point). В отличие от чисел с плавающей запятой (Floating-Point), данный формат обеспечивает более эффективное использование аппаратных ресурсов, таких как блоки DSP и BRAM.

В данной работе используется фиксированное представление с разрядностью $Q_{6.8}$, где:

- 6 бит отведены под целую часть числа.
- 8 бит используются для дробной части.

Таким образом, числа в этом формате могут представлять значения в диапазоне $[-32, 31.996]$ с шагом $2^{-8} = 0.00390625$.

При переводе значений из плавающей запятой в фиксированную используется метод округления к ближайшему минимальному числу (округление вниз, или Round Down). Этот метод позволяет минимизировать ошибки округления и избежать систематического смещения, которое может возникать при округлении к ближайшему числу.

Такой метод округления позволяет повысить точность вычислений и уменьшить накопление ошибок при последовательных операциях, что критично для работы нейросети на FPGA.

4.2 Обучение нейронной сети

Обучение нейронной сети LST-1 на основе 60000 тренировочных изображений из базы данных MNIST. Для обучения используется фреймворк PyTorch.

Из MNIST загружается 60000 тренировочных и 10000 тестовых изображений размером 28×28 пикселей в градациях серого. Из тренировочного набора выделяется 1000 изображений для валидации, оставшиеся 59000 используются для непосредственного обучения модели.

Нормализация изображений осуществляется в соответствии с описанными ранее требованиями.

```
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])
```

Модель L2DST состоит из двух линейных слоев LazyLinear с функцией активации \tanh и полносвязный слой Linear с функцией активации softmax для классификации.

В качестве метрики, для оценки обучаемости нейронной сети используется функция потерь кросс-энтропии – Negative Log-Likelihood Loss (NLLLoss).

Оптимизатор Adam с параметром скорости обучения 1.5×10^{-3} и шаговым снижением StepLR (с понижением на 3% каждые 10 эпох).

```
class L2DST(nn.Module):
    def __init__(self, ffn_num_hiddens, ffn_num_outputs):
        super().__init__()
        self.dense1 = nn.LazyLinear(ffn_num_hiddens)
        self.dense2 = nn.LazyLinear(ffn_num_outputs)

    def forward(self, X):
        out = torch.nn.functional.tanh(self.dense1(X))
        out1 = torch.transpose(out, -1, -2)
```

```

        out = torch.nn.functional.tanh(self.dense2(out1))
        out = torch.transpose(out, -2, -1)
        return out

criterion = nn.NLLLoss()
optimizer = optim.Adam(model.parameters(), lr=15e-4, weight_decay=0e-5)
scheduler = lr_scheduler.StepLR(optimizer, step_size=10, gamma=0.97,
                                verbose=False)

```

Обучение проводится на 370 эпохах, при этом на каждой эпохе вычисляется функция потерь NLLLoss. На рисунке 4.3 показан график функции потерь.

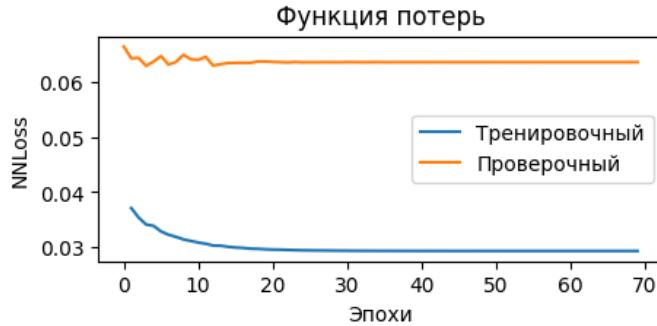


Рисунок 4.3 – Графики функции потерь на обучающем и тестовом наборе

После завершения каждой эпохи вычисляются средние ошибки на тренировочном и валидационном наборах данных.

Python описание процесса обучения нейронной сети прямого распространения приведено в приложении Д.

4.3 Программная реализация эталонной модели нейронной сети на языке Python

Программная реализация нейронной сети выполняется с использованием языка Python и специализированных библиотек машинного обучения. В качестве эталонной модели реализуется нейронная сеть на основе библиотеки fixpoint, позволяющая провести сравнение точности вычислений при использовании чисел с фиксированной запятой и чисел с плавающей запятой.

Эталонная модель представляет собой нейросеть с двумя скрытыми слоями и активационной функцией tanh.

Входные изображения размером 28×28 проходят последовательную обработку:

- Первый слой выполняет линейное преобразование входных данных с весами и смещением, после чего применяется функция активации тангенс.

- Второй слой аналогично выполняет линейное преобразование, используя параметры весов и смещений, после чего применяется функция активации тангенс.

- Выходной слой выполняет классификацию, вычисляя вероятности классов с помощью softmax.

Для проверки корректности работы алгоритма в числах с фиксированной запятой разрабатывается программная эмуляция аппаратного вычислителя. Эмуляция реализуется с использованием классов:

- 1 MAC – блок умножения и аккумуляции с фиксированной точностью.
- 2 MEM – память для хранения весов и промежуточных значений.
- 3 TANH – реализация функции гиперболического тангенса в фиксированной точности.

Обработка входных данных в модели с фиксированной запятой включает следующие этапы:

- 1 Загрузка изображения из базы данных MNIST в память.
- 2 Преобразование входных данных согласно формату Q6.8.
- 3 Выполнение первого преобразования и применение функции тангенс.
- 4 Обработка данных во втором слое с аналогичной активацией.
- 5 Вычисление выходных значений нейросети и применение softmax.

Сравнение выходных значений между моделью с фиксированной точностью и моделью с плавающей точкой представлено на рисунке 4.4.

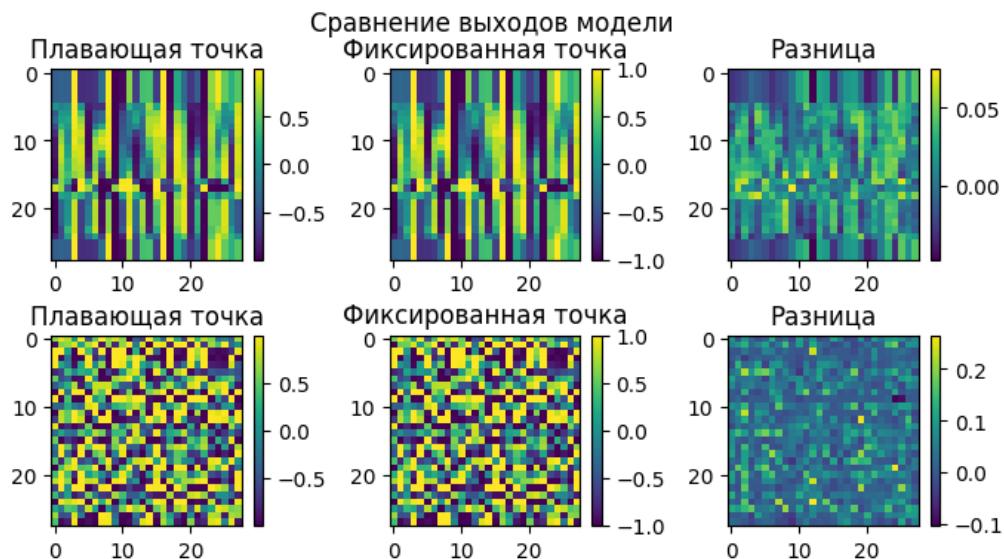


Рисунок 4.4 – Сравнение результатов вычислений

Графический анализ разности значений в обработке одного и того же изображения позволяет оценить влияние округлений и возможные потери точности.

Разработанная эталонная модель на Python позволяет проверить корректность вычислений перед аппаратной реализацией, а также провести тестирование точности чисел с фиксированной запятой.

Python описание нейронной сети с использованием библиотеки fixpoint приведено в приложении Д.

4.4 Разработка операционной части нейронной сети

Нейронная сеть есть объединение управляющего и операционного автоматов, что показано на рисунке 4.5.

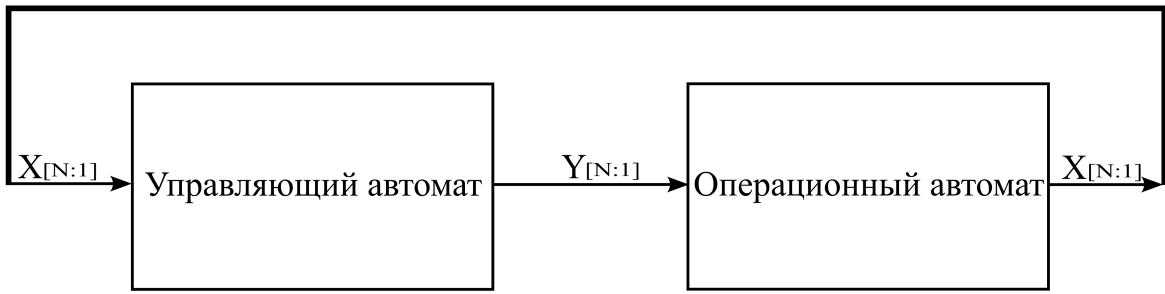


Рисунок 4.5 – Представление нейронной сети в виде управляющего и операционного автоматов

Элементарное действие, выполняемое в операционном автомате, называется микрооперацией. Пусть множество микроопераций, выполняемых под воздействием сигналов управляющего автомата, обозначается как $Y = y_1, \dots, y_N$.

Для выполнения микрооперации y_i ($i = 1, \dots, N$) необходимо появление соответствующего управляющего сигнала, принимающего значение единицы. Если в операционном автомате одновременно выполняется несколько микроопераций, они образуют микрокоманду.

Множество микрокоманд в совокупности с логическими условиями $X = x_1, \dots, x_N$, определяющими последовательность их выполнения, составляют микропрограмму. Логические условия соответствуют осведомительным сигналам, которые формируются в операционном автомате.

IP-блок нейронной сети должен быть спроектирован как независимый компонент системы, обеспечивающий модульность и возможность интеграции в различные вычислительные среды. Для этого необходимо, чтобы блок обладал четко определенным и простым интерфейсом, что позволит эффективно взаимодействовать с другими компонентами системы.

Интерфейс IP-блока должен обеспечивать передачу входных данных, управление процессом вычислений и выдачу результатов. Важно, чтобы структура взаимодействия соответствовала требованиям к гибкости и масштабируемости системы. На рисунке 4.6 представлен требуемый вид интерфейса IP-блока, демонстрирующий основные входные и выходные сигналы, а также механизмы управления вычислительным процессом.



Рисунок 4.6 – Интерфейс IP-блока нейронной сети

4.5 Проектирование функциональной схемы нейронной сети

При разработке IP-блока нейронной сети необходимо определить его функциональную структуру, обеспечивающую корректное выполнение вычислений,

взаимодействие с внешними компонентами системы и эффективное управление процессом обработки данных.

Функциональная схема IP-блока должна описывать основные модули и их взаимодействие, включая блоки обработки входных данных, вычислительное ядро нейронной сети, управляющий модуль и интерфейсы для связи с внешней средой. Важно учитывать требования к производительности, ресурсоемкости и масштабируемости, чтобы обеспечить возможность интеграции IP-блока в различные аппаратные платформы.

Также на функциональной схеме показаны микрооперации и логические условия. Данная нейронная сеть состоит из нескольких основных блоков. Существует два типа блоков обработки пикселя, которые отличаются количеством блоков памяти, которые должны обрабатываться мультиплексором и поступать в качестве входных данных на умножитель.

После вычисления полносвязного слоя для конкретной строки или столбца данные поступают в регистр обработки, состоящий из 28 чисел разрядностью 13 бит. Затем данные из этого регистра последовательно поступают в регистр данных тангенса, из которого они отправляются в блок аппроксимации тангенса гиперболического.

Также используются два пяти разрядных и один десяти разрядный счетчики, которые используются для генерации адреса в память весов и в память изображения.

Блок функции активации softmax реализован на принципе сравнения всех десяти входных классов и выбора наибольшего значения, что будет соответствует наиболее вероятному значению. В результате на выход поступает индекс максимального элемента.

Для управления работой и генерирования микроопераций на основе логических условий используется микропрограммный автомат.

Электрическая функциональная схема IP-блока нейронной сети прямого распространения приведена в приложении Е.

4.6 Построение микропрограммы работы нейронной сети

На данном этапе проектирования создается микропрограмма, определяющая последовательность выполнения операций внутри IP-блока нейронной сети. Она описывает порядок обработки входных данных, передачу информации между модулями и выполнение вычислений.

Микропрограмма строится на основе дискретных состояний, в которых выполняются конкретные микрооперации. Каждое состояние характеризуется активными управляющими сигналами, обеспечивающими выполнение определенной части вычислений.

Основные этапы работы микропрограммы включают:

1 Загрузку изображения в память – входные данные загружаются в буфер оперативной памяти, откуда они будут переданы в вычислительные блоки.

2 Выполнение операций LST-преобразования – производится начальная обработка изображения, включающая нормализацию и предварительное выделение признаков.

3 Передача данных в полносвязный слой – результаты преобразования передаются в полносвязный слой нейросети для дальнейшей обработки.

4 Выполнение финальной классификации – производится анализ данных нейросетью и определение класса входного изображения.

5 Отправка данных в систему обработки – результаты классификации передаются в систему для последующего использования.

Микропрограмма работы нейронной сети по распознаванию рукописных цифр представлена в Приложении Ж. Данная схема отображает основные микрооперации и логические условия.

4.7 Проектирование управляющей части

Проектирование управляющей части вычислительного устройства осуществляется методом структурирования конечных состояний системы на основе синхронных схем – каноническим методом структурного синтеза микропрограммного автомата (МПА). На этапе структурного синтеза принято представлять МПА в виде двух частей: памяти и комбинационной схемы, что представлено на рисунке 4.7.

Память автомата состоит из заранее выбранных элементов памяти, обеспечивающих хранение информации о текущем состоянии устройства. В качестве элементов памяти используются триггеры, которые позволяют надежно фиксировать и изменять состояние системы.

Количество элементов памяти, необходимых для хранения состояний автомата, определяется по следующей формуле:

$$N = \log_2 M, \quad (4.1)$$

где M – число состояний микропрограммного автомата.

Однако в практических реализациях часто применяется кодирование состояний с использованием кода Грея. Код Грея представляет собой двоичную систему кодирования, в которой любые два последовательных числа отличаются изменением только одного бита[13]. Это позволяет уменьшить количество переключений триггеров при переходе между состояниями, снижая вероятность ошибок, возникающих из-за временных несовпадений сигналов.

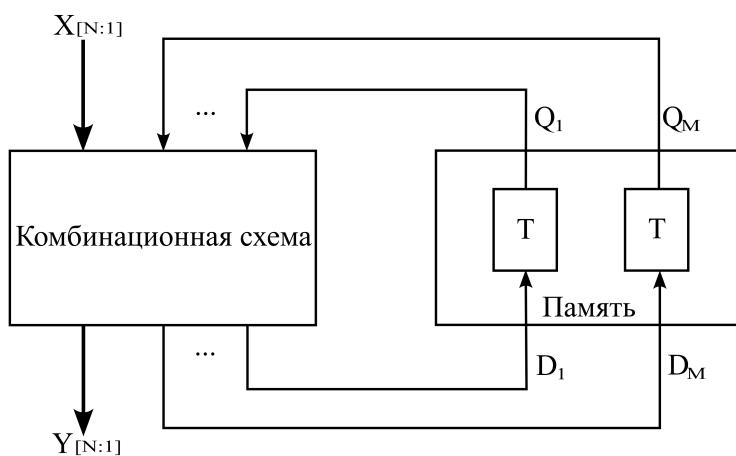


Рисунок 4.7 – Представление управляющей части нейронной сети в виде композиции памяти и комбинационной схемы

Дополнительно, в разработке автоматов управления применяется стратегия, при которой старший бит кодового слова указывает на нахождение автомата в состоянии ожидания. Это состояние используется для инициализации системы,

броса счетчиков и минимизации нежелательных переходов. Применение старшего бита в качестве указателя на режим ожидания помогает уменьшить влияние глий (glitch) на граф переходов. Глии – это кратковременные нежелательные изменения сигнала, возникающие при переключении состояний из-за различий во времени распространения сигналов через логические элементы [14]. Они могут приводить к некорректным переходам в автомате, вызывая ошибки в работе системы.

В связи с этим, традиционная формула определения числа элементов памяти модифицируется следующим образом:

$$N = \log_2 M + 1 = \log_2 11 + 1 = 5, \quad (4.2)$$

где $M = 11$ – число состояний микропрограммного автомата.

Таким образом, каждое состояние автомата описывается представлением из 5 бит, что позволяет повысить надежность работы системы, уменьшить количество переключений триггеров и снизить вероятность ошибок, связанных с глиями.

Закодированные состояния автомата представлены в таблице 1.

В соответствии со списком микроопераций (Y) и логических условий (X), а также разработанной микропрограммой составим граф-схему алгоритма, которая представлена на рисунке 4.8.

Таблица 1 – Кодирование состояний автомата

| Состояние | Код Грэя ($T_4T_3T_2T_1T_0$) |
|-----------|--------------------------------|
| a_0 | 00000 |
| a_1 | 10000 |
| a_2 | 10001 |
| a_3 | 10011 |
| a_4 | 10010 |
| a_5 | 10110 |
| a_6 | 10111 |
| a_7 | 10101 |
| a_8 | 11101 |
| a_9 | 11111 |
| a_{10} | 11110 |

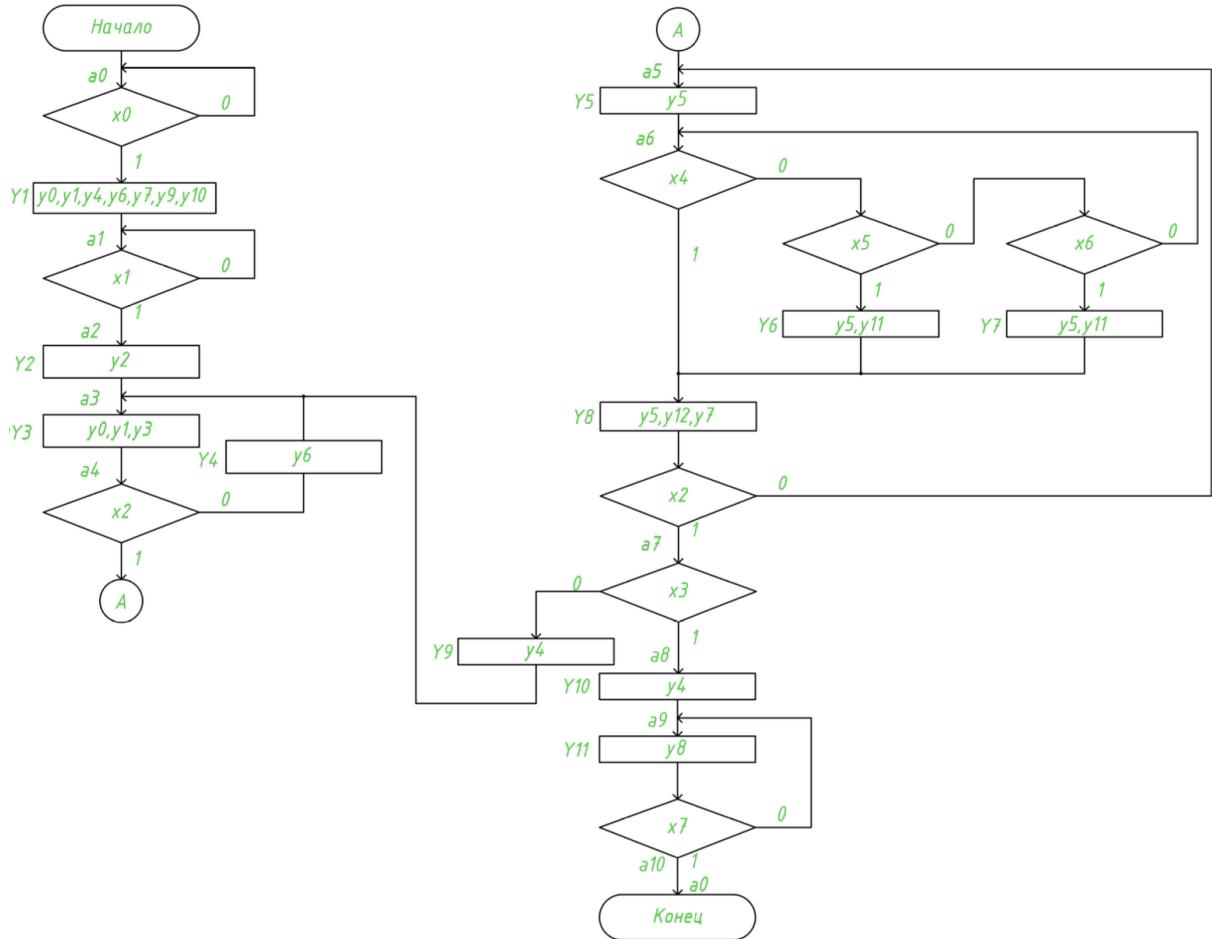


Рисунок 4.8 – Граф переходов

Важно отметить, что для упрощения понимания и реализации управляющего устройства некоторые состояния разбиты на группу из нескольких состояний. В соответствии с разработанной граф-схемой алгоритма составлена таблица 2, в которой представлены все переходы и их условия.

4.8 Построение графа переходов управляющего автомата

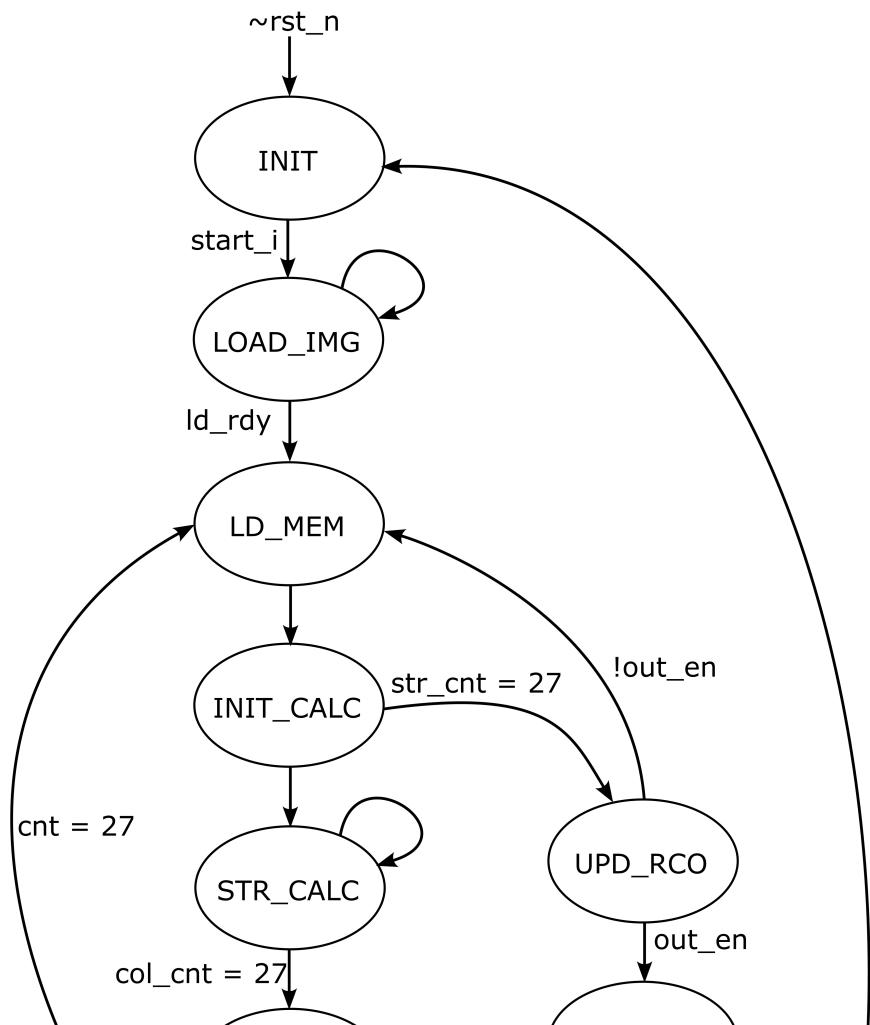
Граф переходов управляющего автомата позволяет представить граф-схему алгоритма в соответствии с условными обозначениями. Граф переходов управляющего автомата представлен на рисунке 4.9.

Состояния граф-схемы алгоритма объединены в блоки:

- 1 INIT – инициализация системы, установка начальных параметров.
- 2 LOAD – загрузка изображения.
- 3 LD_MEM – подготовка данных.
- 4 INIT_CALC – инициализация смещения.
- 5 CALC – обработка строки/столбца.
- 6 RDY – готовность результата обработки.
- 7 TG_CALC – вычисление тангенса.
- 8 UPD_RCO – смена слоя.
- 9 INIT_OUT – инициализация выходных смещений.
- 10 OUT_CALC – обработка выходного полносвязного слоя.
- 11 RDY_OUT – готовность детектирования.

Таблица 2 – Таблица переходов граф-схемы алгоритма

| h | a_m | a_s | X_h | | Y_t |
|-----|----------|----------|--|-----------------------|--|
| 1 | a_0 | a_0 | $\overline{x_0}$ | – | – |
| 2 | | a_1 | x_0 | Y_1 | $y_0, y_1, y_4, y_6, y_7, y_9, y_{10}$ |
| 3 | a_1 | a_1 | $\overline{x_1}$ | – | – |
| 4 | | a_2 | x_1 | – | – |
| 5 | a_2 | a_3 | 1 | Y_2 | y_2 |
| 6 | a_3 | a_4 | 1 | Y_3 | y_0, y_1, y_3 |
| 7 | a_4 | a_4 | $\overline{x_2}$ | Y_4 | y_6 |
| 8 | | a_5 | x_2 | – | – |
| 9 | a_5 | a_6 | 1 | Y_5 | y_5 |
| 10 | a_6 | a_6 | $x_4 \mid \overline{x_4}x_5 \mid \overline{x_4x_5}x_6$ | $- \mid Y_6 \mid Y_7$ | y_6, y_7 |
| 11 | | a_7 | x_2 | Y_8 | y_8 |
| 12 | a_7 | a_3 | $\overline{x_3}$ | Y_9 | y_4 |
| 13 | | a_8 | x_3 | Y_{10} | y_{10} |
| 14 | a_8 | a_9 | 1 | – | – |
| 15 | a_9 | a_9 | $\overline{x_7}$ | Y_{11} | y_{11} |
| 16 | | a_{10} | x_7 | – | – |
| 17 | a_{10} | a_0 | 1 | – | – |



5 АППАРАТНАЯ РЕАЛИЗАЦИЯ ИР-БЛОКА НЕЙРОННОЙ СЕТИ ДЛЯ РАСПОЗНАВАНИЯ РУКОПИСНЫХ ЦИФР

5.1 Описание пакета Xilinx Vivado

Xilinx Vivado – это современный программный пакет для проектирования цифровых схем на базе программируемых логических интегральных схем (ПЛИС, FPGA), разработанный компанией Xilinx. Данный программный комплекс является преемником Xilinx ISE и значительно превосходит его по возможностям и производительности. Vivado предназначен для автоматизированного проектирования цифровых схем, начиная с этапа описания логики на языках описания аппаратуры (HDL) и заканчивая загрузкой генерированного битового потока в устройство[15].

Одним из ключевых преимуществ Vivado является использование более производительных алгоритмов синтеза и оптимизации, что позволяет существенно ускорить процесс разработки. В отличие от традиционных средств проектирования, в Vivado реализована поддержка параллельных вычислений, что делает его особенно эффективным при работе с большими проектами.

В состав пакета Vivado входят несколько инструментов, среди которых[15]:

1 Vivado IDE – интегрированная среда разработки, предоставляющая удобный графический интерфейс для управления проектами, анализа схемотехнической логики и моделирования.

2 Vivado Synthesis – инструмент синтеза логики, который преобразует код HDL в оптимизированное представление на уровне вентилей.

3 Vivado Implementation – модуль размещения, трассировки и оптимизации логики на кристалле FPGA.

4 Vivado Simulator – встроенный инструмент для моделирования цифровых схем на уровне поведенческой, функциональной и временной верификации.

5 Vivado IP Integrator – инструмент для работы с IP-ядрами и объединения их в сложные системы на кристалле.

6 Vivado High-Level Synthesis (HLS) – средство высокоуровневого синтеза, позволяющее описывать аппаратные алгоритмы на языках C/C++ и автоматически преобразовывать их в оптимизированный RTL-код.

Основные возможности Vivado:

1 Поддержка языков описания аппаратуры VHDL и Verilog. Vivado позволяет разрабатывать схемы на двух основных языках HDL, а также поддерживает SystemVerilog для верификации цифровых проектов. Это дает инженерам широкие возможности для разработки и тестирования аппаратуры.

2 Автоматизированный синтез и размещение логики на FPGA. Vivado включает мощный механизм синтеза, который автоматически преобразует HDL-код в оптимизированную сетевую схему, выполняет ее размещение на кристалле FPGA и прокладывает соединения между логическими блоками.

3 Встроенные инструменты для моделирования и отладки.

4 Генерация IP-ядер и использование встроенных библиотек компонентов.

Для реализации нейронной сети Vivado используется на нескольких ключевых этапах разработки. Во-первых, он применяется для генерации аппаратной логики, необходимой для выполнения расчетов внутри FPGA. Во-вторых, с его помощью производится компиляция проекта и синтез аппаратной схемы. В-третьих, Vivado используется для загрузки генерированного битового потока в ПЛИС и отладки работы модели.

5.2 Описание функциональных блоков

Аппаратная реализация нейронной сети включает несколько ключевых функциональных блоков:

- 1 Блок обработки пикселя.
- 2 Блок вычисления тангенса.
- 3 Блок вычисления softmax.
- 4 Блок памяти изображения.
- 5 Блок управляющего автомата.
- 6 Блок умножения.

Каждый из этих блоков реализуется в виде аппаратного модуля на HDL.

В блоке обработки пикселя осуществляется установка параметра смещения для подготовки к вычислению слоя, а также выбирается соответствующий данному этапу весовой коэффициент из памяти.

В блоке вычисления тангенса осуществляется аппроксимированный расчет функции активации тангенс гиперболический для LST-1 слоя.

Прямая реализация вычисления тангенса будет занимать большое количество вычислительных ресурсов. Поэтому для удобства реализации используется аппроксимация в соответствии с формулой[16]:

$$\tanh(x) \approx F(x) = \begin{cases} \text{sign}(x), & \text{if } |x| > 2, \\ \left(1 + \frac{x}{4}\right) \cdot x, & \text{if } -2 < x < 0, \\ \left(1 - \frac{x}{4}\right) \cdot x, & \text{if } 0 < x < 2. \end{cases} \quad (5.1)$$

В блоке вычисления softmax происходит сравнение входных данных и определение наиболее вероятного класса.

В блоке памяти изображения храниться результат обработки на каждом этапе. От приемки входных данных, до выхода LST-1 слоя.

Блок управляющего автомата осуществляет управление и синхронизацию между всеми блоками устройства и обрабатывает входные управляющие сигналы.

Блок умножения состоит из матричного умножителя, который описан с целью анализа аппаратных затрат и точности при изменении разрядности весовых коэффициентов.

Реализация данных блоков на языке SystemVerilog представлена в приложении 3.

5.3 Описание интерфейса нейронной сети

Как было показано ранее на рисунке 4.5 нейронная сеть состоит из пяти обязательных входных и двух выходных сигналов. Взаимодействие процессорной системы и программируемой логики осуществляется через интерфейс AXI4-lite. Таким образом необходимым условием, выдвигаемым к IP-блоку будет поддержка интерфейса AXI4-lite. Для упрощения формирования IP-блока будем использовать связку AXI-uP. uP интерфейс напрямую взаимодействует с модулем верхнего уровня. Далее данные передаются в AXI-slave модуль, который и подключается к процессорной системе.

5.4 Описание процесса создания блочной диаграммы

Создадим блочную диаграмму, что показано на рисунке 5.1.

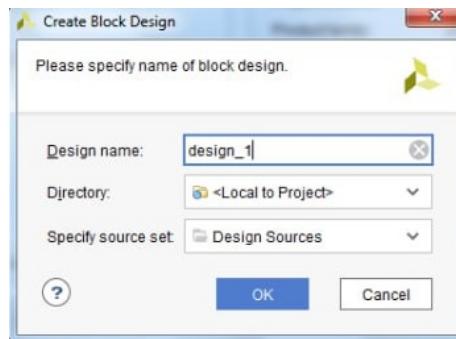


Рисунок 5.1 – Создание блочной диаграммы

Для создания блочной диаграммы необходимо создать IP-блок нейронной сети. Для этого сначала выполним добавление всех файлов в IP-блок, что представлено на рисунке 5.2

| Name | Library Name | Type | Is Include | Used In Constant | File Group Na |
|----------------------|--------------|--------------------|--------------------------|--------------------------|----------------|
| Standard | | | <input type="checkbox"/> | <input type="checkbox"/> | |
| Synthesis (13) | | | <input type="checkbox"/> | <input type="checkbox"/> | |
| src/ROM_row.sv | xil_de... | systemVerilogSo... | <input type="checkbox"/> | <input type="checkbox"/> | xilinx_anylang |
| src/ROM_w_out.sv | xil_de... | systemVerilogSo... | <input type="checkbox"/> | <input type="checkbox"/> | xilinx_anylang |
| src/mac_core.sv | xil_de... | systemVerilogSo... | <input type="checkbox"/> | <input type="checkbox"/> | xilinx_anylang |
| src/tanh_function.sv | xil_de... | systemVerilogSo... | <input type="checkbox"/> | <input type="checkbox"/> | xilinx_anylang |
| src/ROM_col.sv | xil_de... | systemVerilogSo... | <input type="checkbox"/> | <input type="checkbox"/> | xilinx_anylang |
| src/HA.vhd | xil_de... | vhdlSource | <input type="checkbox"/> | <input type="checkbox"/> | xilinx_anylang |
| src/fc.sv | xil_de... | systemVerilogSo... | <input type="checkbox"/> | <input type="checkbox"/> | xilinx_anylang |
| src/MUL_NxN.vhd | xil_de... | vhdlSource | <input type="checkbox"/> | <input type="checkbox"/> | xilinx_anylang |
| src/softmax.sv | xil_de... | systemVerilogSo... | <input type="checkbox"/> | <input type="checkbox"/> | xilinx_anylang |
| src/nn_param_pkg.sv | xil_de... | systemVerilogSo... | <input type="checkbox"/> | <input type="checkbox"/> | xilinx_anylang |
| src/ADD1.vhd | xil_de... | vhdlSource | <input type="checkbox"/> | <input type="checkbox"/> | xilinx_anylang |
| src/PE_rc.sv | xil_de... | systemVerilogSo... | <input type="checkbox"/> | <input type="checkbox"/> | xilinx_anylang |
| src/PE_rco.sv | xil_de... | systemVerilogSo... | <input type="checkbox"/> | <input type="checkbox"/> | xilinx_anylang |
| Simulation (13) | | | <input type="checkbox"/> | <input type="checkbox"/> | |

Рисунок 5.2 – Добавление файлов

Далее необходимо добавить на блочную диаграмму процессорную систему, а затем IP-блок. Также необходимыми элементами является AXI-Interconnect и системный Processor System Reset. AXI-Interconnect используется для связи процессорной системы с несколькими IP-блоками. Он выполняет маршрутизацию транзакций между мастером (процессором) и ведомыми устройствами. Блок Processor System Reset управляет сигналами сброса в системе. Он синхронизирует сброс процессора, периферийных устройств и пользовательских модулей. Полученная блочная диаграмма показана на рисунке 5.3.

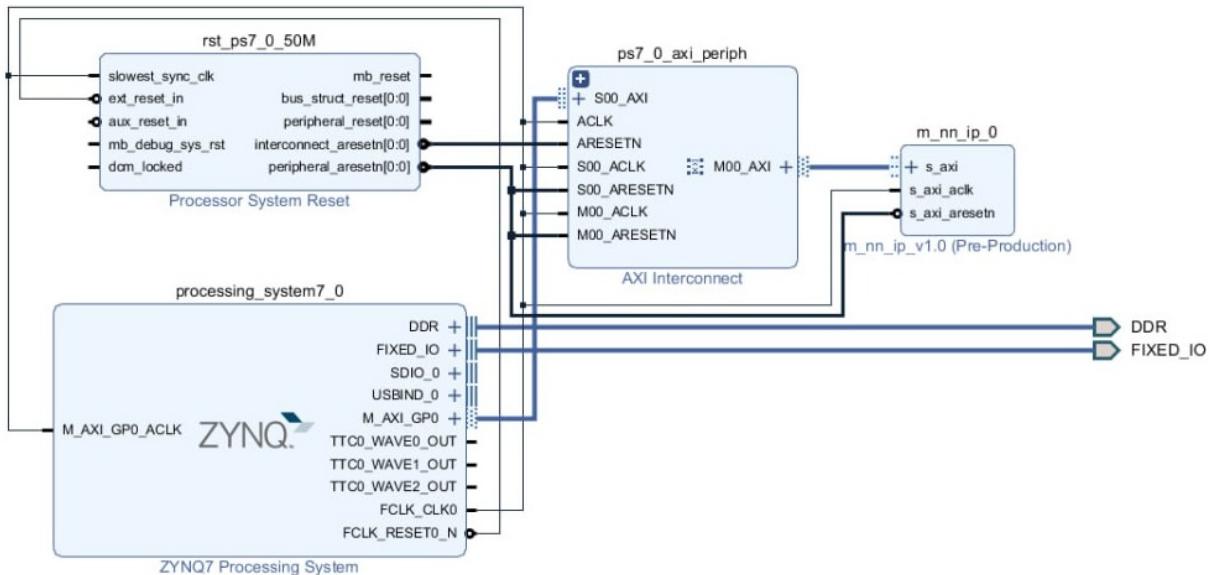


Рисунок 5.3 – Результирующий вид блочной диаграммы

5.5 Результаты синтеза и симуляции

Для проверки работоспособности написанного устройства был составлен тест, в котором формируются управляющие воздействия и изображение. Работа устройства начинается по сигналу start. Для тестирования было взято два изображения, что показано на рисунке 5.4.

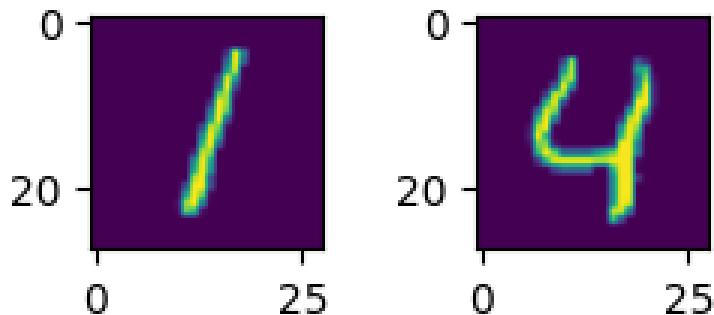


Рисунок 5.4 – Тестируемые изображения

Тест представлен в приложении З. Для проверки корректности работы нейронной сети необходимо сравнить выход результирующего полносвязанного слоя и нейронной сети. На рисунках 5.5 и 5.6 представлены временные диаграммы описанной на SystemVerilog нейронной сети и выход эталонной нейронной сети на языке Python.

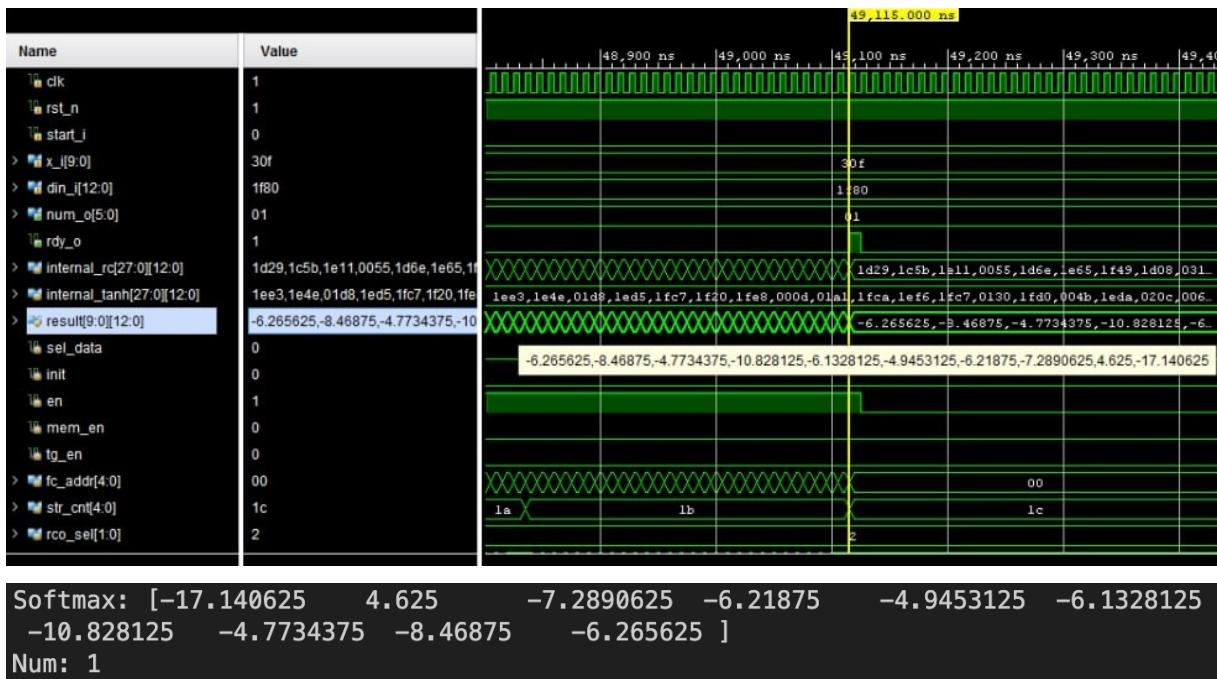


Рисунок 5.5 – Временная диаграмма нейронной сети и результат работы эталона на первом изображении

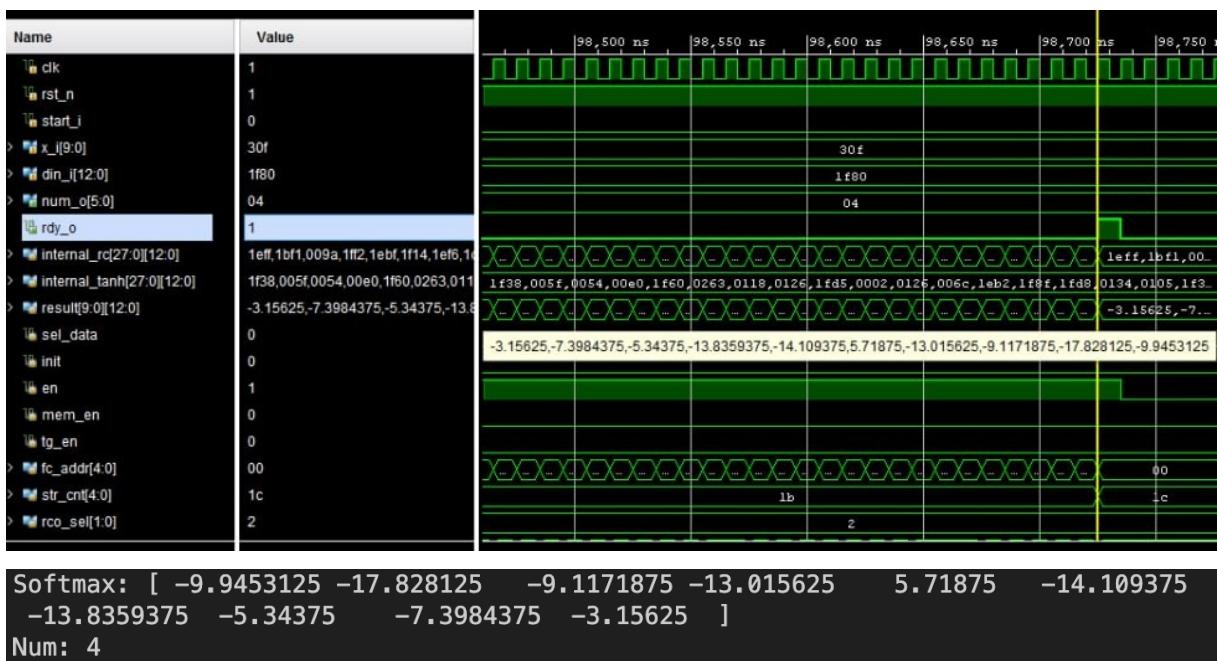


Рисунок 5.6 – Временная диаграмма нейронной сети и результат работы эталона на втором изображении

В результате мы наблюдаем полное соответствие работы эталонной и реализованной моделей. Таким образом данную реализацию можно считать корректной.

Следующим не мало важным аспектом является аппаратные затраты на реализацию данной нейронной сети. Был произведен синтез данного RTL-кода в двух вариациях:

1 В качестве умножителей используются блоки DSP48. Блок DSP48 представляет собой специализированный программируемый вычислительный блок, встроенный в FPGA, предназначенный для выполнения арифметических операций, таких как умножение, сложение и накопление сумм. Архитектура DSP48 оптимизирована для эффективного выполнения операций цифровой обработки сигналов и интенсивных вычислений.

2 Во второй вариации в качестве умножителей используются описанные матричные умножители, что удобно для оценки аппаратных затрат в зависимости от используемой разрядности данных.

Аппаратные затраты представлены на рисунке 5.7.

| Utilization | | Post-Synthesis Post-Implementation | | Utilization | | Post-Synthesis Post-Implementation | |
|-------------|------------|--------------------------------------|---------------|-------------|------------|--------------------------------------|---------------|
| | | Graph Table | | | | Graph Table | |
| Resource | Estimation | Available | Utilization % | Resource | Estimation | Available | Utilization % |
| LUT | 1538 | 17600 | 8.74 | LUT | 8111 | 17600 | 46.09 |
| LUTRAM | 169 | 6000 | 2.82 | LUTRAM | 169 | 6000 | 2.82 |
| FF | 27 | 35200 | 0.08 | FF | 27 | 35200 | 0.08 |
| BRAM | 33 | 60 | 55.00 | BRAM | 33 | 60 | 55.00 |
| DSP | 57 | 80 | 71.25 | DSP | 29 | 80 | 36.25 |
| IO | 33 | 100 | 33.00 | IO | 33 | 100 | 33.00 |
| BUFG | 2 | 32 | 6.25 | BUFG | 2 | 32 | 6.25 |

Рисунок 5.7 – Аппаратные затраты на DSP48 и матричном умножителе

6 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ IP-БЛОКА НЕЙРОННОЙ СЕТИ ДЛЯ РАСПОЗНАВАНИЯ РУКОПИСНЫХ ЦИФР

6.1 Характеристика программного средства, разрабатываемого для собственных нужд

Разрабатываемый IP-блок нейронной сети предназначен для аппаратного ускорения процесса распознавания рукописных цифр в системах обработки изображений. Основная цель разработки — создание эффективного и оптимизированного аппаратного решения, интегрируемого в FPGA-платформы, для высокоскоростного распознавания рукописных символов с целью использования в автоматизированных системах ввода рукописного текста или распознавание символов в банковских и почтовых системах. Основные задачи, решаемые IP-блоком заключаются в оптимизация вычислений за счёт аппаратной реализации и использовании специального LST преобразования.

Разработчиком IP-блока является сотрудник ИТ-отдела организации. Разработка ведётся в рамках внутренних потребностей компании для последующего внедрения в производственные и исследовательские процессы.

Необходимость в разработке IP-блока обусловлена высокой потребностью в аппаратных решениях для распознавания рукописных цифр. Встроенные программные решения не обеспечивают требуемую производительность. Разрабатываемое средство оптимизировать использование вычислительных ресурсов. Внедрение IP-блока позволит организации значительно повысить эффективность обработки изображений, что дает возможность адаптации IP-блока под специфические требования организации.

6.2 Расчет инвестиций в разработку программного средства для собственных нужд

Расчет затрат на основную заработную плату представлен в таблице 3.

Таблица 3 – Расчет затрат на основную заработную плату команды разработчиков

| Категория исполнителя | Месячный оклад, р. | Часовой оклад, р. | Трудоемкость работ, ч. | Итого, р. |
|---|--------------------|-------------------|------------------------|-----------|
| Бизнес-аналитик | 1600 | 10 | 40 | 400 |
| Системный архитектор | 4000 | 25 | 40 | 1000 |
| Программист | 2400 | 15 | 40 | 600 |
| Тестировщик | 1600 | 10 | 40 | 400 |
| Дизайнер | 1600 | 10 | 40 | 400 |
| Итого | | | | 2800 |
| Премия и иные стимулирующие выплаты (50%) | | | | 1400 |
| Всего затрат на основную заработную плату разработчиков | | | | 4200 |

Расчет инвестиций на разработку программного средства для собственных нужд представлен в таблице 4.

Таблица 4 – Расчет инвестиций на разработку программного средства для собственных нужд

| Наименование статьи затрат | Формула/таблица для расчета | Сумма, р. |
|---|---|-----------|
| 1. Основная заработка плата разработчиков | Таблица 3 | 4200 |
| 2. Дополнительная заработка плата разработчиков | $Z_d = \frac{4200 \cdot 10\%}{100}$ | 420 |
| 3. Отчисления на социальные нужды | $P_{соц} = \frac{4200 + 420 \cdot 34,6\%}{100}$ | 1598,52 |
| 4. Прочие расходы | $P_{пр} = \frac{4200 \cdot 30\%}{100}$ | 1260 |
| 5. Общая сумма инвестиций на разработку | $Z_p = 4200+420+1598,52+1260$ | 7478,52 |

6.3 Расчет экономического эффекта от использования программного средства для собственных нужд

Экономия на заработной плате и начислениях на заработную плату осуществляется в результате сокращения численности работников. В частности после внедрения программного средства сокращению подвергается 1 дизайнер.

$$\begin{aligned} \mathcal{E}_{з.п.} &= \sum_{i=1}^n \Delta \chi_i \cdot Z_i \cdot \left(1 + \frac{10}{100}\right) \cdot \left(1 + \frac{34,6}{100}\right) = \sum_{i=1}^1 \Delta \chi_i \cdot Z_i \cdot \left(1 + \frac{H_d}{100}\right) \times \\ &\times \left(1 + \frac{H_{соц}}{100}\right) = 1 \cdot 19200 \cdot 1.48 = 28427.52 \text{ р.} \end{aligned} \quad (6.1)$$

где H – норматив дополнительной заработной платы, $H_{соц}$ – ставка отчисления от заработной платы, включаемых в себестоимость.

Экономическим эффектом при использовании программного средства является прирост чистой прибыли:

$$\begin{aligned} \Delta \Pi_q &= (\mathcal{E}_{тек} - \Delta Z_{тек}^{п.с.}) \left(1 - \frac{H_p}{100}\right) = (28427,52 - 15500) \cdot \left(1 - \frac{20\%}{100}\right) = \\ &= 10342,02 \text{ р.} \end{aligned} \quad (6.2)$$

6.4 Расчет показателей экономической эффективности разработки и использования программного средства в организации

Так как разработка программного средства ведется «с нуля», то расчет показателей экономической эффективности осуществляется следующим образом:

$$ROI = \frac{\Delta \Pi_q - Z_p}{Z_p} \cdot 100\% = \frac{10342,02 - 7478,52}{7478,52} \cdot 100\% = 38\%, \quad (6.3)$$

где $\Delta\Pi_q$ – прирост чистой прибыли, полученной от использования разработанного программного средства, Z_p – затраты на разработку программного средства.

На основании полученных значений показателей эффективности инвестиций (затрат) следует сделать вывод, что разработка IP-блока нейронной сети для аппаратного ускорения распознавания рукописных цифр является экономически целесообразной. Проведенные расчеты показывают, что общие затраты на разработку составляют 7478,52 руб.

Экономический эффект от внедрения программного средства достигается за счет сокращения численности персонала (исключение 1 дизайнера), что позволяет сэкономить 28427,52 руб. на заработной плате и социальных начислениях. Прирост чистой прибыли в результате внедрения составляет 10342,02 руб.

Рассчитанный показатель ROI (рентабельность инвестиций) составляет 38%, что говорит о высокой эффективности вложений в разработку данного IP-блока. Это означает, что вложенные средства окупаются с существенной прибылью, а дальнейшее использование программного средства приведет к дополнительной экономии и росту прибыли организации.

7 АНАЛИЗ РЕЗУЛЬТАТОВ ТЕСТИРОВАНИЯ IP-БЛОКА НЕЙРОННОЙ СЕТИ ДЛЯ РАСПОЗНАВАНИЯ РУКОПИСНЫХ ЦИФР

7.1 Описание среды тестирования

Описать ZYBO и PYNQ.

7.2 Тестирование разработанного IP-блока нейронной сети

Кака осуществляется тестирование, скрипт на питоне.

7.3 Экспериментальное исследование IP-блока нейронной сети

Общие патерны, какие эксперименты проведены.

7.3.1 Описание проводимого эксперимента

Описание конкретного эксперимента

7.3.2 Подготовка данных для проведения эксперимента

Подготовка к эксперименту

7.3.3 Результаты проведенного эксперимента

Результаты, анализ матриц спутывания, графики. Можно добавить матрицы спутывания в floatpoint и fixpoint

Результаты представлены в приложении 3.

7.4 Анализ результатов тестирования

сравнить результаты.

ЗАКЛЮЧЕНИЕ

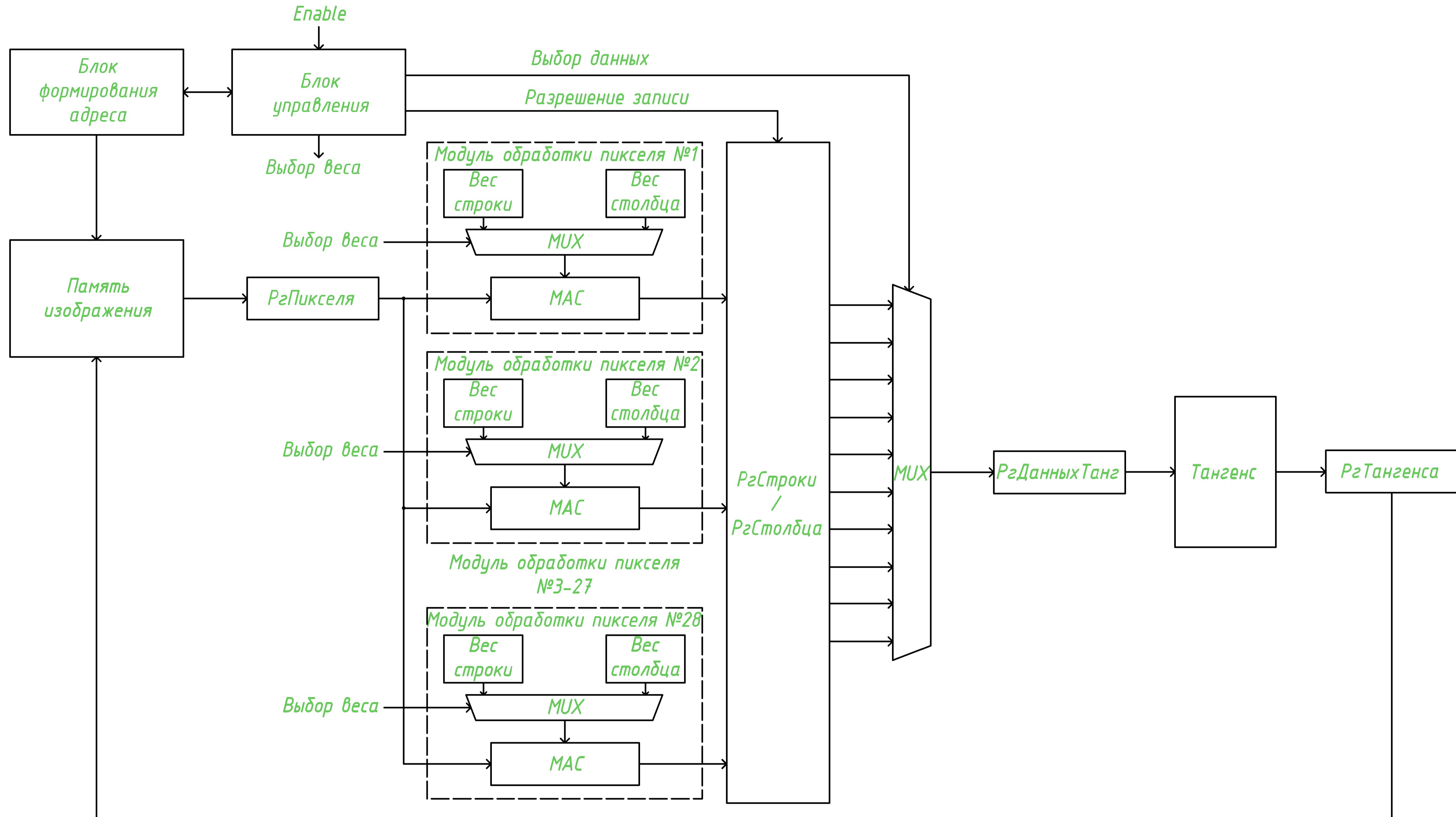
В рамках преддипломной практики была разработана структурная схема IP-блока нейронной сети прямого распространения для распознавания рукописных цифр. Проведено исследование существующих аналогичных разработок, выявлены их положительные и отрицательные стороны. Был произведен анализ технического задания, разработана функциональная спецификация системы. Также была обучена нейронная сеть, в соответствии с выбранной архитектурой. Написана эталонная модель на языке python с использованием библиотеки fixpoint, а также произведено сравнение работы модели с плавающей и фиксированной запятой.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- [1] FPGA реализация нейронной сети прямого распространения для распознавания рукописных чисел / Е. А. Кривальцевич, М. И. Ващекевич // Информационные технологии и системы 2024 (ИТС 2024) = Information Technologies and Systems 2024 (ITS 2024) : материалы международной научной конференции, Минск, 20 ноября 2024 г. / Белорусский государственный университет информатики и радиоэлектроники; редкол. : Л. Ю. Шилин [и др.]. – Минск, 2024. – С. 85–86.
- [2] Кривальцевич Е.А., Ващекевич М.И. Аппаратная реализация нейронной сети прямого распространения для распознавания рукописных цифр на базе FPGA. Доклады БГУИР. 20**; **(*)**: ***-***.
- [3] Bouvrie J. Notes on convolutional neural networks. – 2006.
- [4] Giardino D. et al. FPGA implementation of hand-written number recognition based on CNN // International Journal on Advanced Science, Engineering and Information Technology. – 2019. – Т. 9. – №. 1. – Р. 167-171.
- [5] Varsamopoulos, S. (2019). Neural Network based Decoders for the Surface Code. [Dissertation (TU Delft), Delft University of Technology]. <https://doi.org/10.4233/uuid:dc73e1ff-0496-459a-986f-de37f7f250c9>
- [6] Agrawal V., Jagtap J., Kantipudi M. V. V. P. Decoded-ViT: A Vision Transformer Framework for Handwritten Digit String Recognition //Revue d'Intelligence Artificielle. – 2024. – Т. 38. – №. 2. – С. 523.
- [7] Agrawal V. et al. Performance analysis of hybrid deep learning framework using a vision transformer and convolutional neural network for handwritten digit recognition //MethodsX. – 2024. – Т. 12. – С. 102554.
- [8] YADRO [Электронный ресурс] – Электронные данные – Режим доступа: <https://itglobal.com/ru-by/solutions/tech-partners/yadro/>
- [9] Maxim Vashkevich and Egor Krivalcevich Learned 2D separable transform: building block for designing compact and efficient neural network for image recognition.
- [10] MNIST Handwritten Numbers database [Electronic resource] – Electronic data – Access mode: <https://yann.lecun.com/exdb/mnist/Mittal>
- [11] Xilinx, AXI Reference Guide [Электронный ресурс] – Электронные данные – Режим доступа: <https://docs.amd.com/v/u/en-US/ug1037-vivado-axi-reference-guide>, 2017.
- [12] Pong P. Chu, FPGA Prototyping by Verilog Examples [Электронный ресурс] – Электронные данные – Режим доступа: <https://faculty.kfupm.edu.sa/COE/aimane/coe405/FPGA%20examples.pdf>, 2018.
- [13] Tokheim, R.L. Digital Electronics: Principles and Applications. – McGraw-Hill, 2003. – 480 p.
- [14] Katz, R.H. Contemporary Logic Design. – Benjamin-Cummings, 1994. – 700 p.
- [15] Xilinx Inc., Vivado Design Suite User Guide, UG892, 2021.
- [16] L. D. Medus, T. Iakymchuk, J. V. Frances-Villora, M. Bataller- Mompean, and A. Rosado-Munoz, “A novel systolic parallel hardware architecture for the FPGA acceleration of feedforward neural networks,” IEEE Access, vol. 7, pp. 76 084–76 103, 2019.

ПРИЛОЖЕНИЕ А
(Обязательное)
Отчет о проверке на заимствование

ПРИЛОЖЕНИЕ Б
(Обязательное)
Схема электрическая структурная LST-1



| Изм | Лист | № докум. | Подп. | Дата | Лит. | Масса | Масштаб |
|----------|------------|----------|-------|------|------|-------|---------|
| Разраб. | Криバルьевич | | | | у | | |
| Проб. | Вашкевич | | | | | | |
| Т.контр. | | | | | | | |
| Н.контр. | | | | | | | |
| Утв. | | | | | | | |

ГУИР.431289.001 З1

LST-1 преобразование

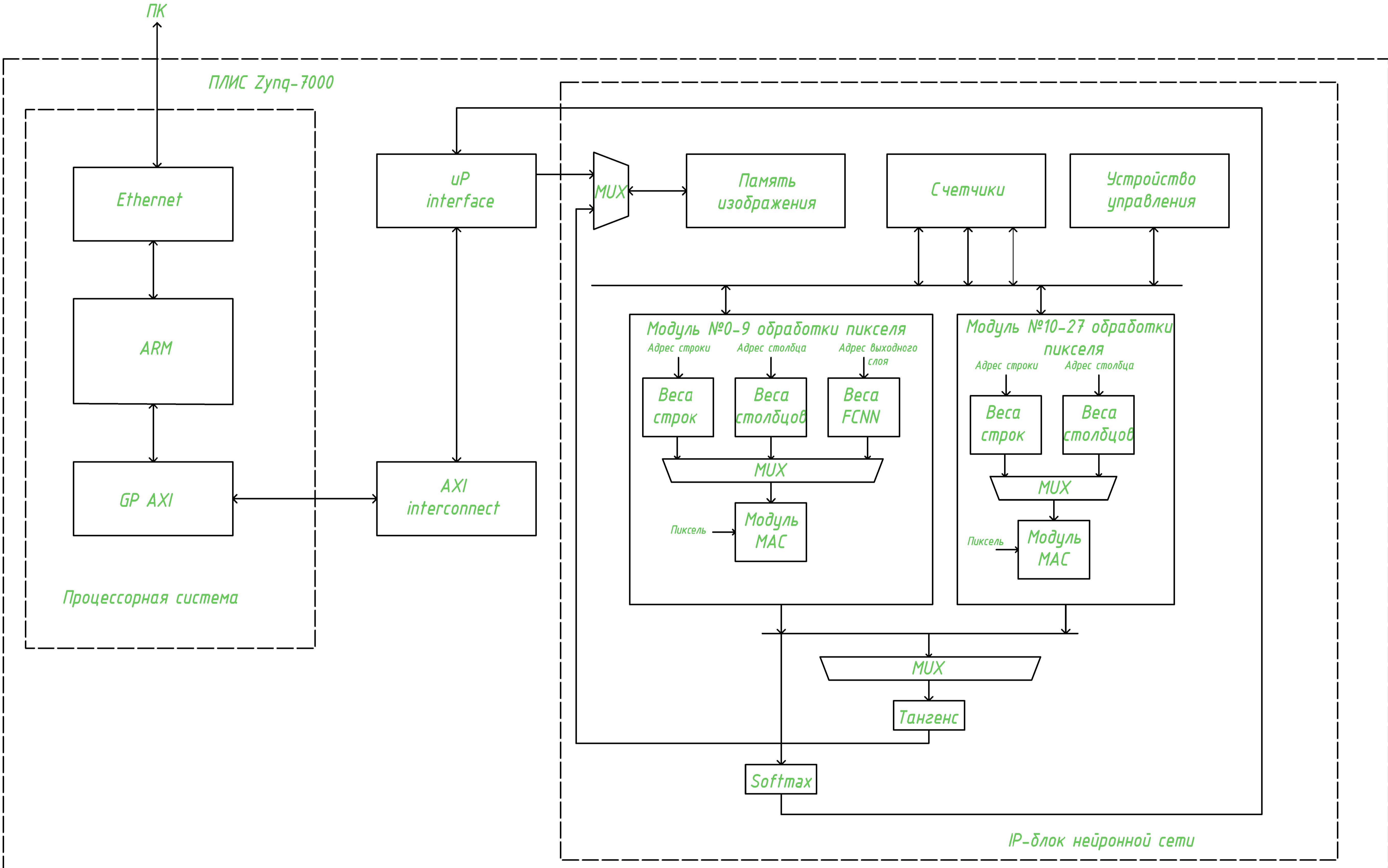
Схема электрическая структурная

Лист 1 Листов 1

БГУИР, гр.150701

ПРИЛОЖЕНИЕ В
(Обязательное)
Схема алгоритма работы слоя LST-1

ПРИЛОЖЕНИЕ Г
(Обязательное)
Схема электрическая структурная



| Изм. | Лист | № докум. | Подп. | Дата | Лит. | Масса | Масштаб |
|----------|------------|----------|-------|------|------|-------|---------|
| Разраб. | Криバルьевич | | | | у | | |
| Проб. | Вашкевич | | | | | | |
| Т.контр. | | | | | | | |
| Н.контр. | | | | | | | |
| Утв. | | | | | | | |

IP-блок нейронной сети
Схема электрическая структурная
Лист 1 Листов 1

БГУИР, гр.150701

ПРИЛОЖЕНИЕ Д

(Обязательное)

Python описание сети

Python описание обучения сети:

```
import numpy as np
import torch
import torch.nn as nn
from torch.optim import lr_scheduler

import torchvision
import matplotlib.pyplot as plt
from time import time
from torchvision import datasets, transforms
from torch import nn, optim
from tqdm import tqdm

device = torch.device('cuda' if torch.cuda.is_available()
    else 'cpu')
print(f"Working on {device}")

# Transform image to 1x784 and normalize colors
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,)))
])

# Download dataset
trainset_full = datasets.MNIST('../..../Datasets', download=
    False, train=True, transform=transform)
testset = datasets.MNIST('../..../Datasets', download=False,
    train=False, transform=transform)

val_size, train_size = 1000, 59000
val_dataset, train_dataset = torch.utils.data.random_split(
    trainset_full, [val_size, train_size], generator=torch.
    Generator().manual_seed(27))

# Create dataloaders
trainloader = torch.utils.data.DataLoader(train_dataset,
    batch_size=1000, shuffle=True)
val_loader = torch.utils.data.DataLoader(val_dataset,
    batch_size=1000, shuffle=False)
test_loader = torch.utils.data.DataLoader(testset, batch_size
    =1000, shuffle=False)

class L2DST(nn.Module):
    """The positionwise feed-forward network."""
    def __init__(self, ffn_num_hiddens, ffn_num_outputs):
        super().__init__()
        self.dense1 = nn.LazyLinear(ffn_num_hiddens)
        self.dense2 = nn.LazyLinear(ffn_num_outputs)

    def forward(self, X):
        out = torch.nn.functional.tanh(self.dense1(X))
```

```

        out1 = torch.transpose(out, -1, -2)
        out = torch.nn.functional.tanh(self.dense2(out1))
        out = torch.transpose(out, -2, -1)
        return out

    def get_embeddings(self, X):
        out = torch.nn.functional.tanh(self.dense1(X))
        e1 = out
        out = torch.transpose(e1, -1, -2)
        out = torch.nn.functional.tanh(self.dense2(out))
        e2 = out
        out = torch.transpose(e2, -2, -1)

        return e1, e2

class L2DST_11(nn.Module):
    def __init__(self, input_size, num_classes, device='cpu'):
        super(L2DST_11, self).__init__()

        self.L2DST = L2DST(input_size, input_size)

        dropout = 0.05
        self.dropout = nn.Dropout(dropout)

        # self.BN1 = nn.LayerNorm((1, input_size, input_size))

        self.W_o = nn.Linear(input_size * input_size,
                             num_classes, device=device)

        self.num_classes = num_classes

        self.log_softmax = nn.LogSoftmax(dim=1)

        nn.init.xavier_uniform_(self.W_o.weight)

    def forward(self, x):
        # Multiply input by weights and add biases

        out = self.L2DST(x)

        out = out.reshape(out.shape[0], -1)

        out = self.log_softmax(self.dropout(self.W_o(out)))

        return out

    def get_embeddings(self, x):
        e1, e2 = self.L2DST.get_embeddings(x)

        return e1, e2

    # Build the Neural Network
input_size = 28 # 28x28 images flattened
output_size = 10 # 10 classes for digits 0-9

N_epoch = 300
model = L2DST_11(input_size, output_size, device=device)

```

```

model.load_state_dict(torch.load(f'model_backup\
    L2DST_11_epoch_{N_epoch}.pth', map_location=torch.device(
        'cpu')))

# print(model)
model.to(device)

# criterion = nn.CrossEntropyLoss() # Use CrossEntropyLoss
# which includes softmax
criterion = nn.NLLLoss() # Use CrossEntropyLoss which
# includes softmax
optimizer = optim.Adam(model.parameters(), lr=15e-4,
    weight_decay=0e-5)
scheduler = lr_scheduler.StepLR(optimizer, step_size=10,
    gamma=0.97, verbose=False)

# Track loss
loss_list = []
val_loss_list = []

# Training the network
epochs = 70
time0 = time()

for epoch in range(epochs):
    running_loss = 0

    # for images, labels in tqdm(trainloader, leave=False):
    for images, labels in trainloader:
        images = images.to(device)
        labels = labels.to(device)

        images = images.squeeze(1)

        # Training pass
        optimizer.zero_grad()

        output = model(images)
        loss = criterion(output, labels)

        # This is where the model learns by back propagating
        loss.backward()

        # And optimizes its weights here
        optimizer.step()
        scheduler.step()

        running_loss += loss.item()

    # validation
    model.eval()
    with torch.no_grad():
        for images, labels in val_loader:
            images = images.to(device)
            labels = labels.to(device)
            images = images.squeeze(1)

            output = model(images)
            val_loss = criterion(output, labels)
    val_loss = val_loss / len(val_loader)

```

```

    val_loss_list.append(val_loss)

    CE_curr = running_loss / len(trainloader)
    loss_list.append(CE_curr)
    # if (epoch%10)==0:
    print(f"Epoch_{epoch} Training loss:{CE_curr:.5f}, Val
          loss:{val_loss:.5f}")

print(f"\nTraining Time (in minutes) = {(time()-time0)/60:.2f}")

# Convert lists to numpy arrays
loss_array = np.array(loss_list)

val_loss = [val.cpu().numpy() for val in val_loss_list]

val_loss_array = np.array(val_loss)

# Save the model
torch.save(model.state_dict(), f'model_backup\L2DST_11_epoch_
{N_epoch+epochs}.pth')

# Plot NLL_loss
plt.figure(figsize=(5, 2))
plt.plot(range(len(loss_array))[1:], loss_array[1:], label='Train')
plt.plot(range(len(val_loss_array)), val_loss_array, label='Validation')

plt.xlabel('Epochs')
plt.ylabel('NNLoss')
plt.title('NNLoss')
plt.legend()
plt.show()

```

Python описание сети в fixpoint:

```

import numpy as np
import torch
import os
from torchvision import datasets, transforms
import matplotlib.pyplot as plt
from fixedpoint import FixedPoint
import math

from LST_lib import LST_2d_one

input_size = 28
output_size = 10
model = LST_2d_one(input_size, output_size)

N_epoch = 300
model.load_state_dict(torch.load(f'model_backup/
L2DST_11_epoch_{N_epoch}.pth')) # , map_location=device

weights = model.state_dict()

def min_max_show(A, key):
    print(f'{key}: size = ', A[key].shape)

```

```

print(f'{key}: max value = {torch.max(A[key])}')
print(f'{key}: min value = {torch.min(A[key])}')

# Weights and biases analysis
min_max_show(weights, 'L2DST.dense1.weight')
min_max_show(weights, 'L2DST.dense1.bias')
min_max_show(weights, 'L2DST.dense2.weight')
min_max_show(weights, 'L2DST.dense2.bias')
min_max_show(weights, 'W_o.weight')
min_max_show(weights, 'W_o.bias')

transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,)))
])

valset = datasets.MNIST('c:/Docs/GitHub/Datasets/', download=False, train=False, transform=transform)

img, label = valset[4]
img_ = img.view(img.size(0), -1).squeeze().numpy()

fig = plt.figure(figsize=(1,1))
plt.imshow(img_.reshape(28,-1))
# plt.colorbar()

e1, e2 = model.get_embeddings(img)

e1 = e1.squeeze(0)
e2 = e2.squeeze(0)

class MAC():
    def __init__(self, int_bits, frac_bits):
        self.int_bits = int_bits
        self.frac_bits = frac_bits
        self.acc = FixedPoint(0, signed=True, m=self.int_bits
            , n=self.frac_bits, rounding="down")

    def initialization(self, value):
        self.acc = FixedPoint(value, signed=True, m=self.int_bits
            , n=self.frac_bits, rounding="down")

    def run(self, a, b, q_bit_to_remove):
        p = a*b
        p = FixedPoint(float(p), signed=True, m=self.int_bits
            , n=(p.n - q_bit_to_remove), rounding="down")
        self.acc = self.acc + p
        self.acc = FixedPoint(float(self.acc), signed=True, m
            =self.int_bits, n = self.acc.n, rounding="down")

class MEM():
    def __init__(self, size, int_bits, frac_bits):
        self.int_bits = int_bits
        self.frac_bits = frac_bits
        self.size = size
        self.mem = [] # FixedPoint(np.zeros((size)), signed=True, m=self.int_bits, n=self.frac_bits)

    def initialization(self, array):

```

```

        for addr in range(len(array)):
            self.mem.append(FixedPoint(array[addr], signed=True,
                m=self.int_bits, n = self.frac_bits))

    def write_value(self, value, addr):
        self.mem[addr] = FixedPoint(float(value), signed=True,
            m=self.int_bits, n = self.frac_bits)

    def numpy(self):
        np_arr = np.zeros((self.size))
        for addr in range(self.size):
            np_arr[addr] = float(self.mem[addr])

        return np_arr

    class tanh():
        def __init__(self, int_bits, frac_bits):
            self.int_bits = int_bits
            self.frac_bits = frac_bits
            self.const_q_plus_one = FixedPoint(1-2**(-frac_bits),
                signed=True, m=int_bits, n=frac_bits, str_base=16,
                rounding="down")
            self.const_q_minus_one = FixedPoint(-1, signed=True,
                m=int_bits, n=frac_bits, str_base=16, rounding="down")

        def calc(self, x):
            x_fp = FixedPoint(float(x), signed=True, m=self.int_bits,
                n = self.frac_bits, rounding="down")
            x_quater = x_fp >> 2
            if (x_fp>=2):
                mul_out = FixedPoint((1), signed=True, m=self.int_bits,
                    n = self.frac_bits, str_base=16)
            elif (x<=-2):
                mul_out = FixedPoint((-1), signed=True, m=self.int_bits,
                    n = self.frac_bits, str_base=16)
            else:
                if (x>0):
                    add_sub_out = self.const_q_plus_one -
                        x_quater
                else:
                    add_sub_out = self.const_q_plus_one +
                        x_quater
            mul_out = (add_sub_out*x_fp)
            mul_out.m = self.int_bits
            mul_out.n = self.frac_bits
            mul_out = FixedPoint(float(mul_out), signed=True,
                m=self.int_bits, n = self.frac_bits, rounding="down")
            return mul_out

        def addr_width(x):
            return math.ceil(math.log(x, 2))

    # number of MAC-cores
    N = 28
    L = 10 # output layer

    # Bit representation

```

```

int_bits, frac_bits = 6,7

# Creating MAC cores
MAC_array = []
for i in range(N):
    MAC_array.append(MAC(int_bits, frac_bits))

# Creating row-weights memory blocks
ROW_memory = []
for i in range(N):
    array_i = weights['L2DST.dense1.weight'][i].numpy()
    ROW_memory.append(MEM(len(array_i), int_bits, frac_bits))
    ROW_memory[i].initialization(array_i)

# Creating memory blocks for ROW biases
row_biases_array = weights['L2DST.dense1.bias']
ROW_biases = MEM(len(row_biases_array), int_bits, frac_bits)
ROW_biases.initialization(row_biases_array)

COL_memory = []
for i in range(N):
    array_i = weights['L2DST.dense2.weight'][i].numpy()
    COL_memory.append(MEM(len(array_i), int_bits, frac_bits))
    COL_memory[i].initialization(array_i)

# Creating memory blocks for COL biases
col_biases_array = weights['L2DST.dense2.bias']
COL_biases = MEM(len(col_biases_array), int_bits, frac_bits)
COL_biases.initialization(col_biases_array)

memory = []
for i in range(10):
    array_i = weights['W_o.weight'][i].numpy()
    memory.append(MEM(len(array_i), int_bits, frac_bits))
    memory[i].initialization(array_i)

# Creating memory blocks for COL biases
biases_array = weights['W_o.bias']
biases = MEM(len(biases_array), int_bits, frac_bits)
biases.initialization(biases_array)

# Crearing internal RAM block for image storage
RAM_bock = MEM(N*N, int_bits, frac_bits)
RAM_bock.initialization(np.zeros(N*N))

# Createing tanh function module
Tanh = tanh(int_bits, frac_bits)

# Loading image to RAM block (stage 1)
for addr in range(N*N):
    RAM_bock.write_value(float(img_[addr]), addr)

# Row processing (stage 2)
for row in range(N):
    # MAC-core initialization
    for i in range(N):
        MAC_array[i].initialization(float(ROW_biases.mem[i]))

    for col in range(N):

```

```

        # Read data from memory
        in_data = RAM_bock.mem[row*N + col]

        for i in range(N):
            MAC_array[i].run(in_data, ROW_memory[i].mem[col] ,
                q_bit_to_remove=frac_bits)

        # Write result to memory
        for col in range(N):
            tmp = Tanh.calc(MAC_array[col].acc)
            RAM_bock.write_value(float(tmp), row*N + col)

    # Copy for visualization
    FC1_out = RAM_bock.numpy()

    # Column processing (stage 3)
    for col in range(N):
        # MAC-core initialization
        for i in range(N):
            MAC_array[i].initialization(float(COL_biases.mem[i]))

        for row in range(N):
            # Read data from memory
            in_data = RAM_bock.mem[row*N + col]

            for i in range(N):
                MAC_array[i].run(in_data, COL_memory[i].mem[row] ,
                    q_bit_to_remove=frac_bits)

            # Write result to memory
            for row in range(N):
                tmp = Tanh.calc(MAC_array[row].acc)
                RAM_bock.write_value(float(tmp), row*N + col)

    # Copy for visualization
    FC2_out = RAM_bock.numpy()

    # Classification (stage 4)
    final_result = [] # Softmax output

    # Run MAC-cores
    for i in range(L):
        # MAC-core initialization
        MAC_array[i].initialization(float(biases.mem[i]))

        # 784 MAC iteration
        for j in range(N * N):
            MAC_array[i].run(RAM_bock.mem[j] , memory[i].mem[j] ,
                q_bit_to_remove=frac_bits)

        # save FC output
        final_result.append(float(MAC_array[i].acc))

    # Convert to Numpy
    final_result = np.array(final_result)

    # Output FC layer
    print("Softmax input:", final_result)

```

```

# Result image
predicted_label = np.argmax(final_result)
print("Predicted_label:", predicted_label)

FC2_out_ = FC2_out.reshape(N,-1)
FC1_out_ = FC1_out.reshape(N,-1)

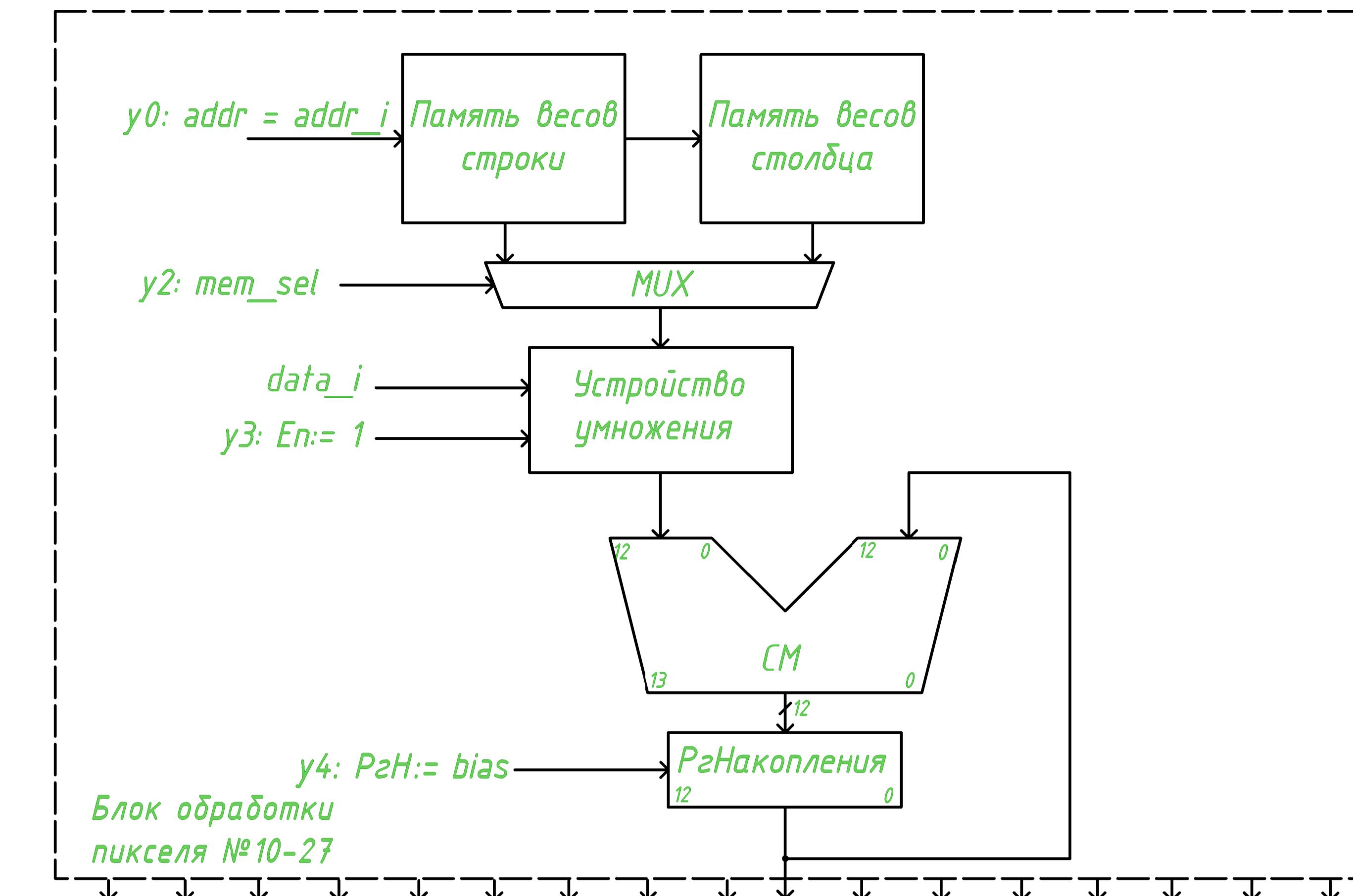
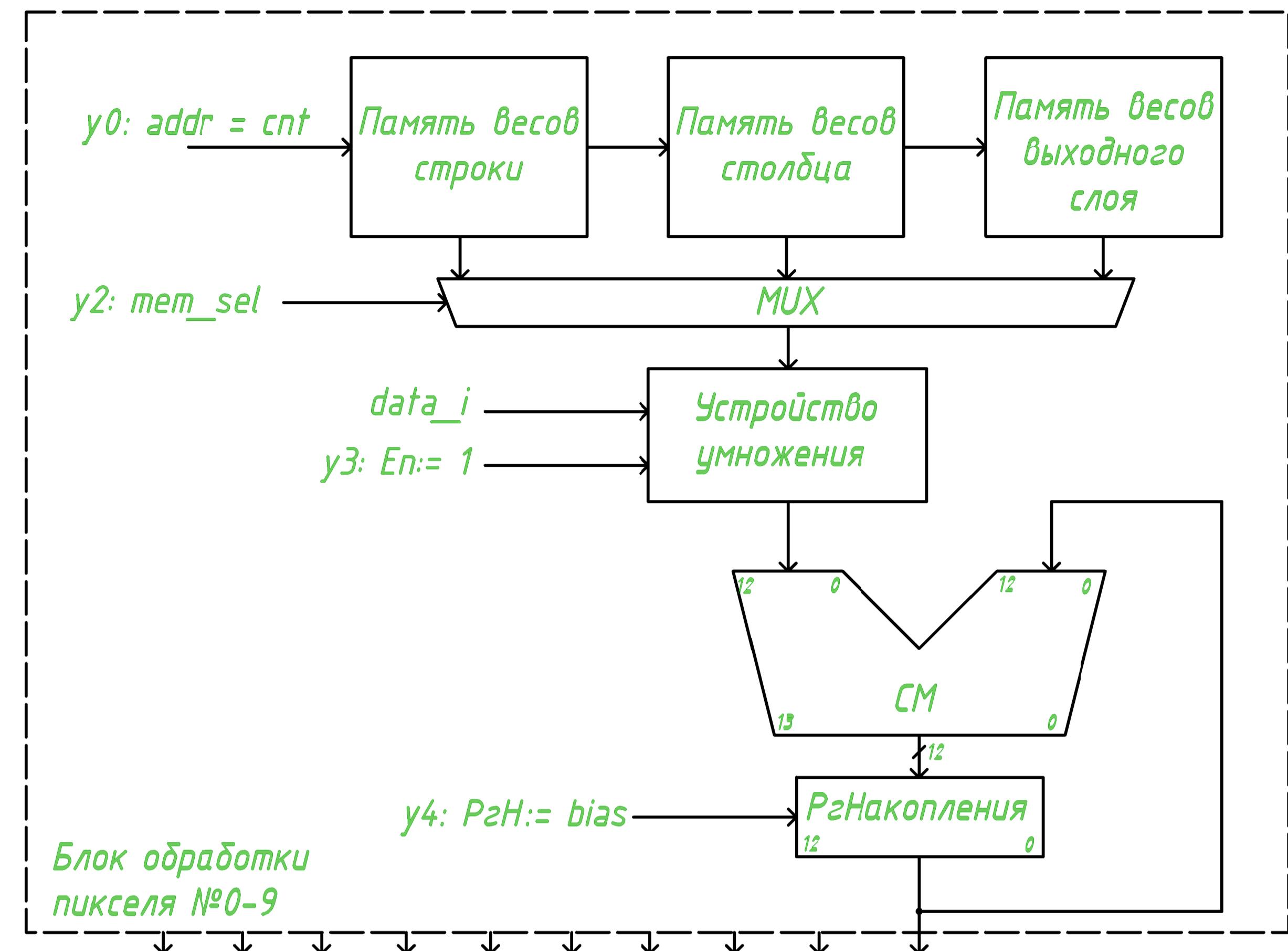
fig = plt.figure(figsize=(8,4))
plt.subplot(2,3,1)
plt.imshow(e1.detach().numpy())
plt.title('Float_precision')
plt.colorbar()
plt.subplot(2,3,2)
plt.imshow(FC1_out_)
plt.title('Fixed_point')
plt.colorbar()
plt.subplot(2,3,3)
plt.imshow(e1.detach().numpy() - FC1_out_) # , vmax=1, vmin
    =-1
plt.title('difference')
plt.colorbar()

plt.subplot(2,3,4)
plt.imshow(e2.detach().numpy())
plt.title('Float_precision')
plt.colorbar()
plt.subplot(2,3,5)
plt.imshow(FC2_out_)
plt.title('Fixed_point')
plt.colorbar()
plt.subplot(2,3,6)
plt.imshow(e2.detach().numpy() - FC2_out_) # , vmax=1, vmin
    =-1
plt.title('difference')
plt.colorbar()

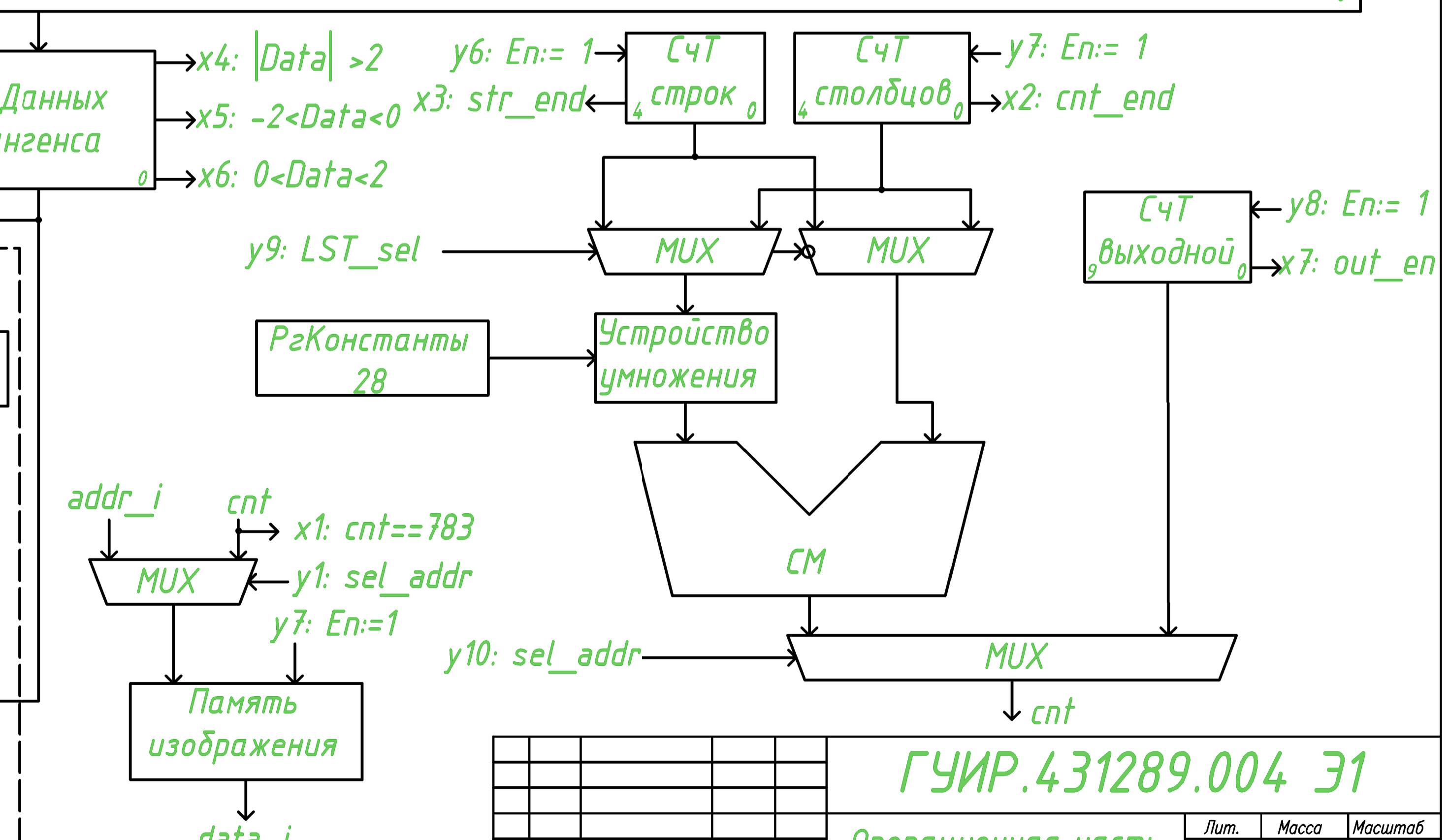
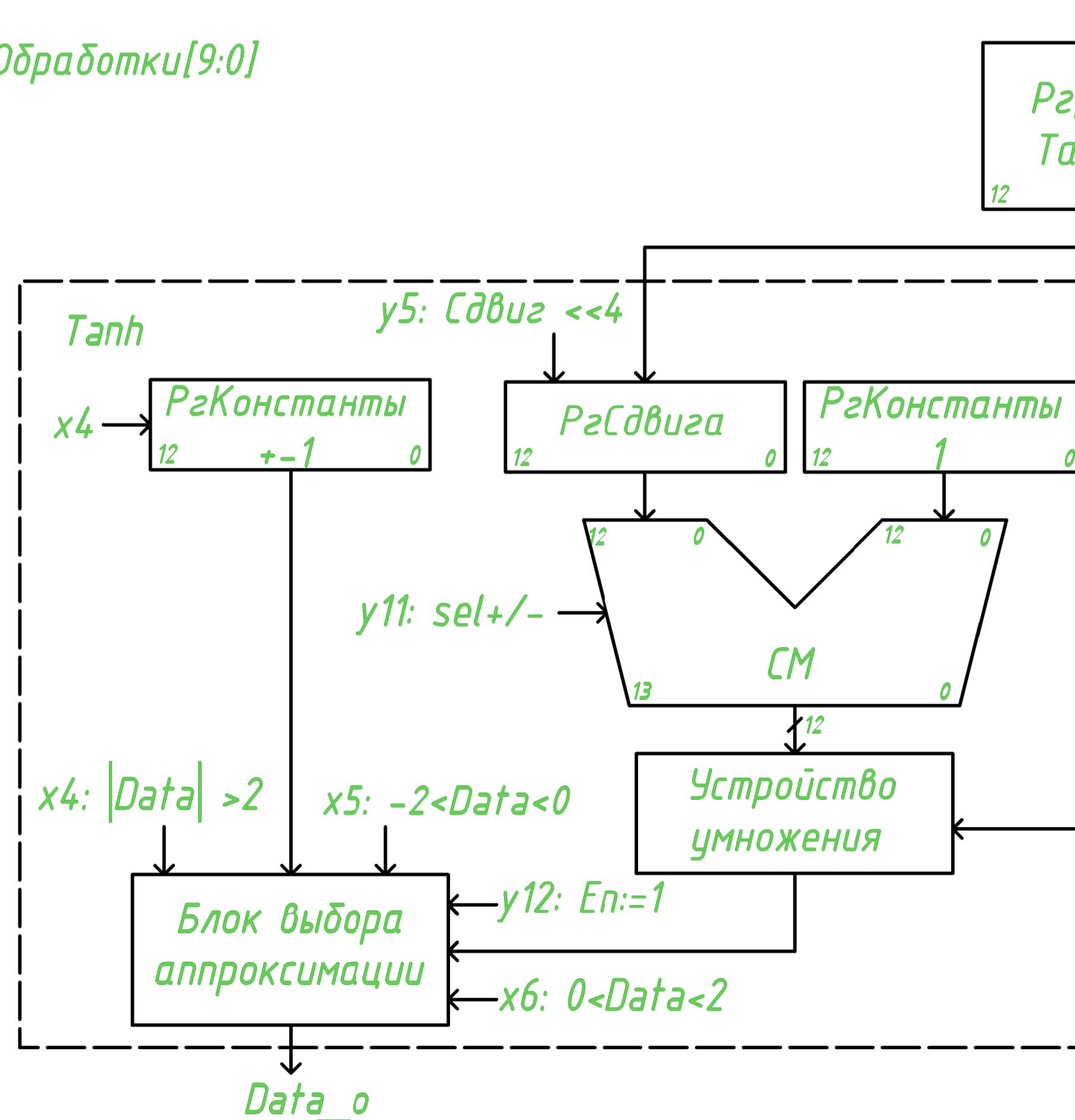
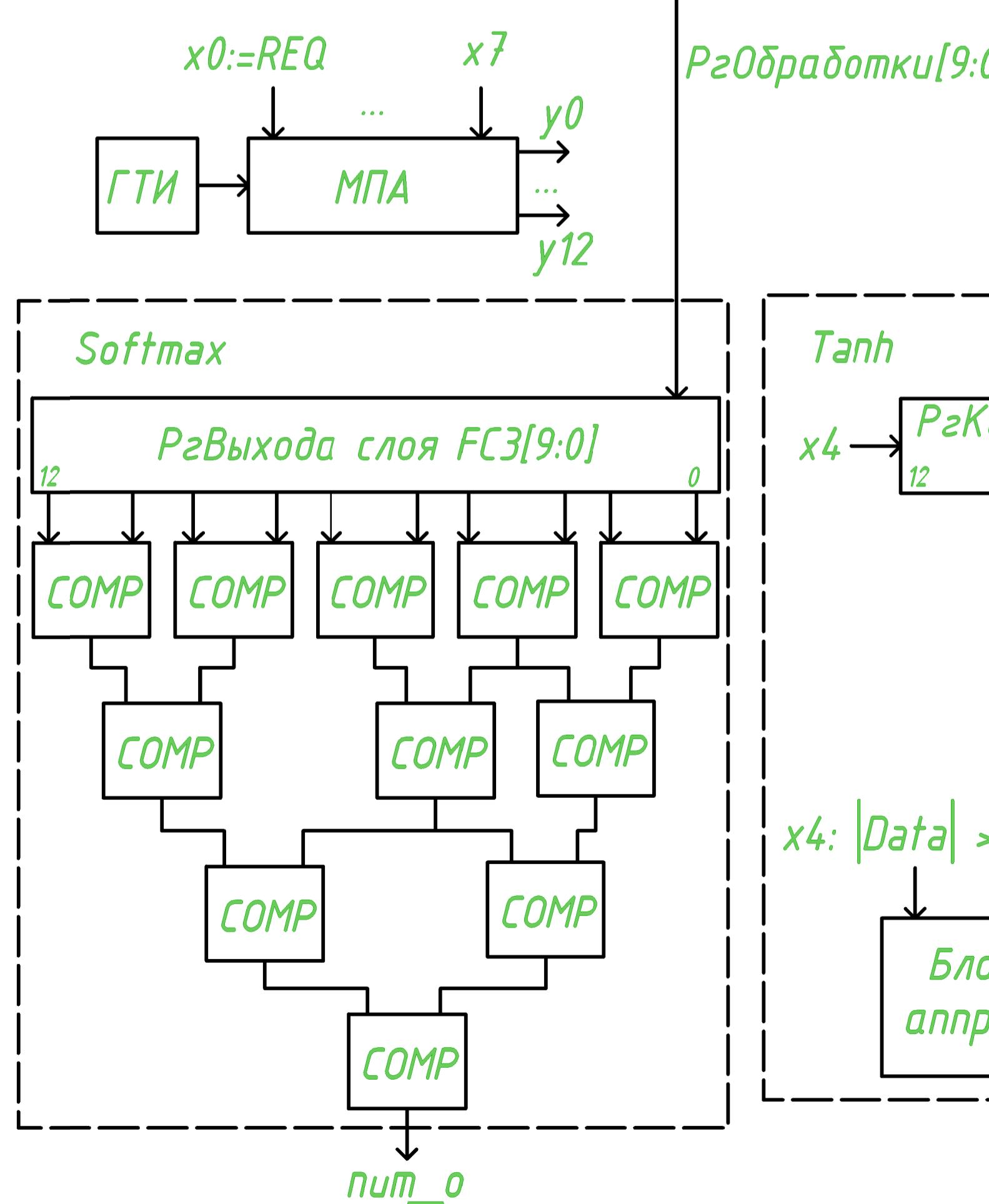
plt.subplots_adjust(wspace=0.4, hspace=0.4)

```

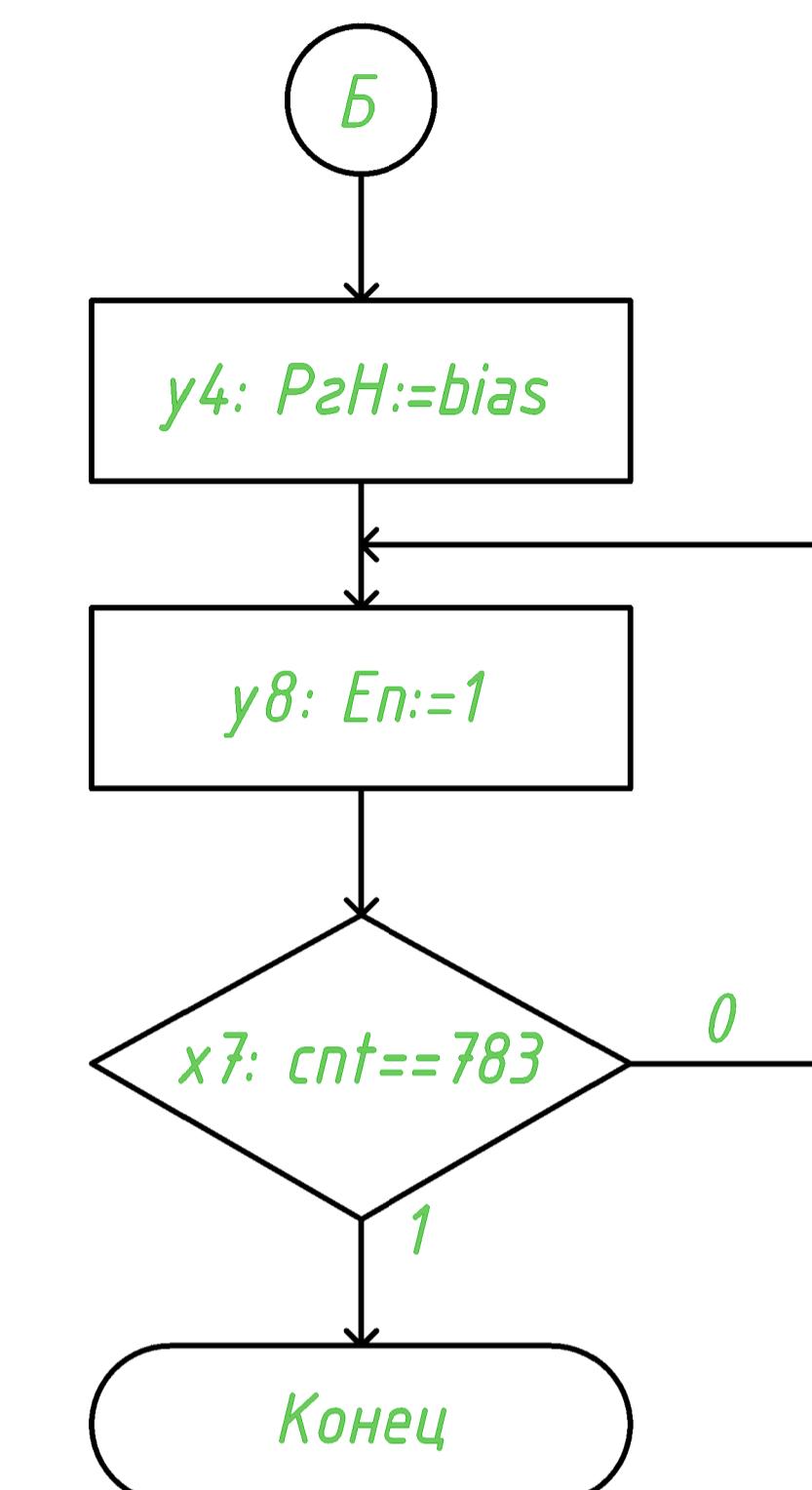
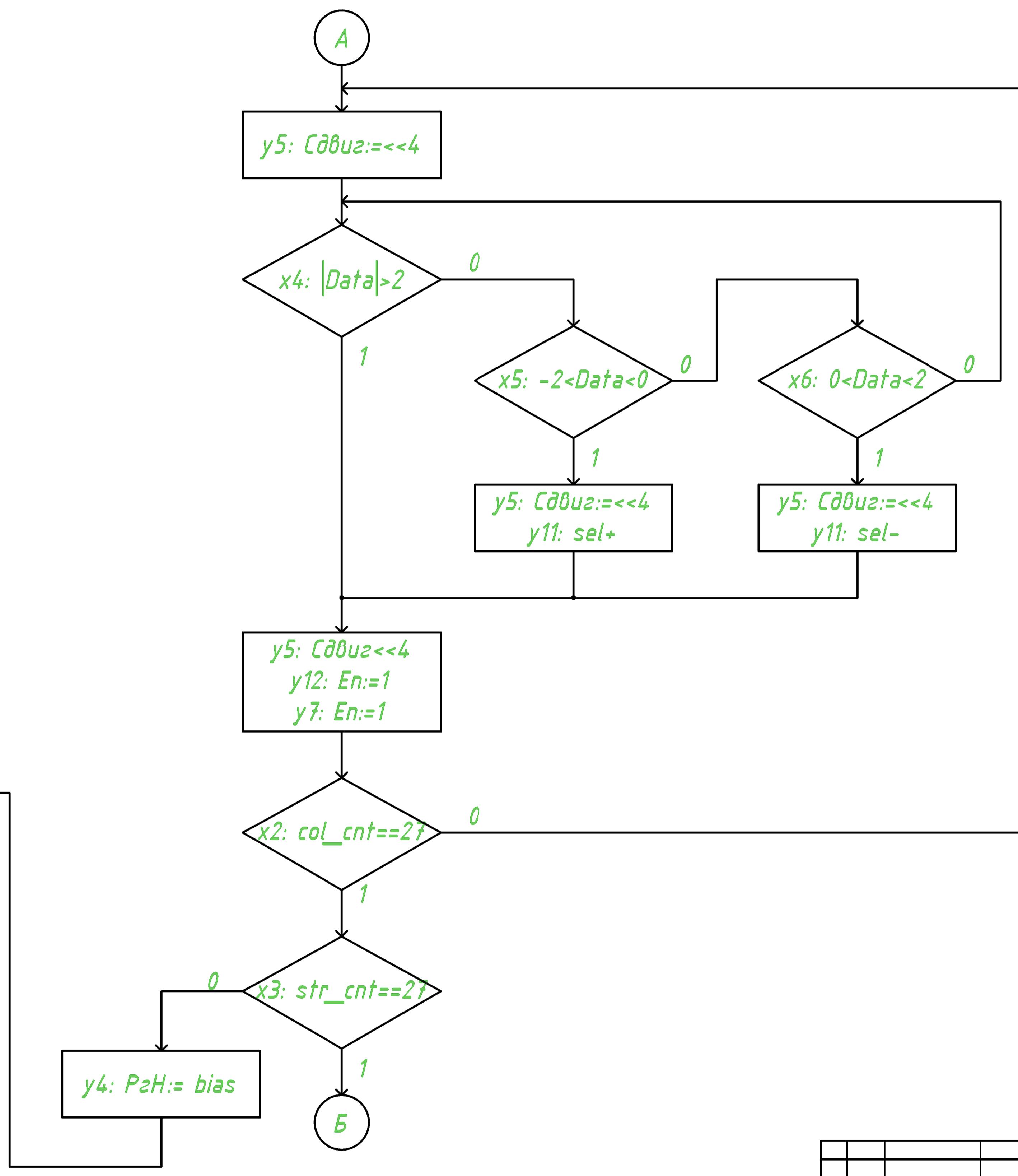
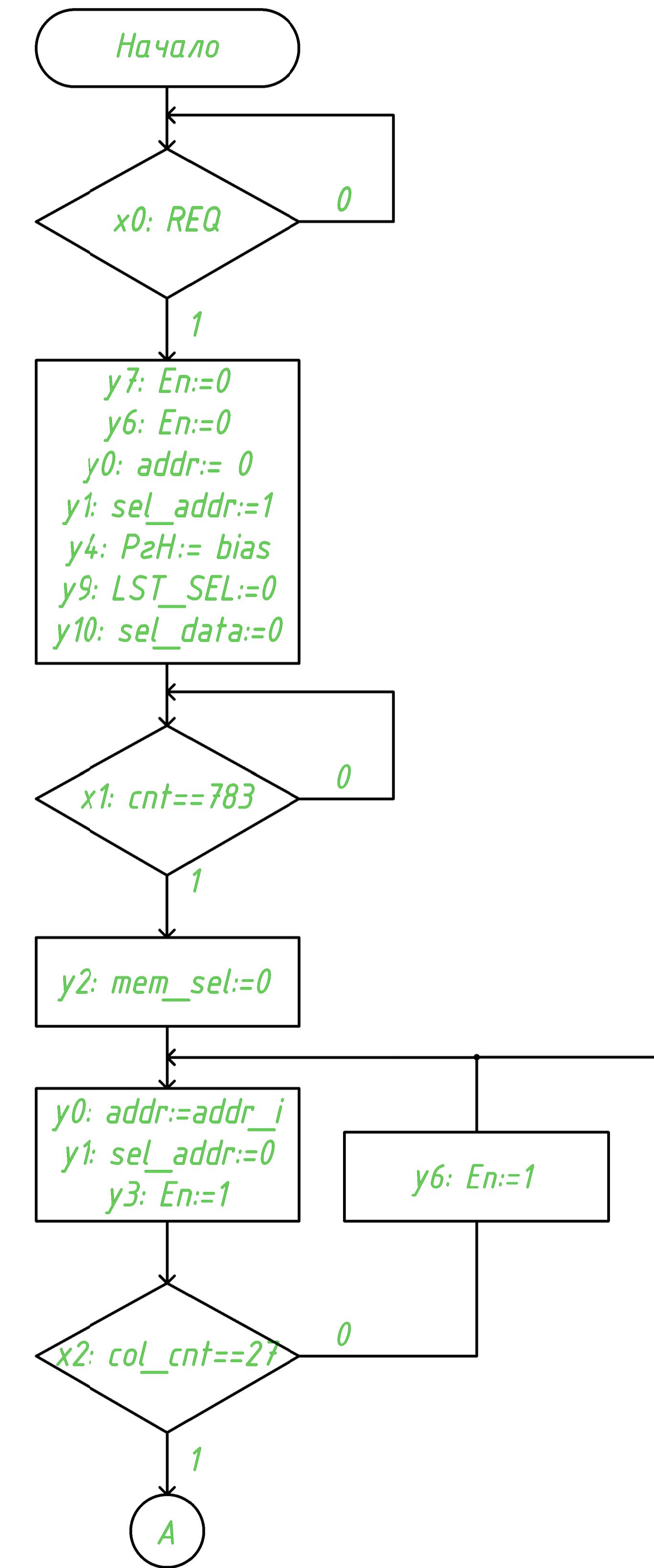
ПРИЛОЖЕНИЕ Е
(Обязательное)
Схема электрическая функциональная



РгОбработка №0-27



ПРИЛОЖЕНИЕ Ж
(Обязательное)
Схема алгоритма работы нейронной сети



ПРИЛОЖЕНИЕ З (Обязательное) Verilog описание устройства

Модуль с параметрами:

```
package nn_param_pkg;

localparam INT_BITS    = 6;
localparam FRC_BITS    = 7;
localparam BITS         = 13;

localparam PICT_SIZE   = 28;
localparam WIDTH        = 784;

localparam ADDR_RC     = 5;
localparam ADDR_OUT    = 10;

localparam RCO_TYPE    = 2;

localparam NN_FSM_BUS_WIDTH = 5;

endpackage
```

Модуль верхнего уровня:

```
(* dont_touch = "yes" *) (* keep_hierarchy = "yes" *)

module fc
    import nn_param_pkg::*;
(
    input logic                      clk,
    input logic                      rst_n,
    input logic                      start_i,
    input logic [ADDR_OUT-1:0]         x_i,
    input logic [INT_BITS+FRC_BITS-1:0] din_i,
    output logic [INT_BITS-1:0]        num_o,
    output logic                      rdy_o
);

logic [INT_BITS + FRC_BITS-1:0] memory          [PICT_SIZE
    -1:0][PICT_SIZE-1:0];
logic [INT_BITS + FRC_BITS-1:0] internal_rc      [PICT_SIZE
    -1:0];
logic [INT_BITS + FRC_BITS-1:0] internal_tanh    [PICT_SIZE
    -1:0];
logic [INT_BITS + FRC_BITS-1:0] result           [ ADDR_OUT
    -1:0];

logic sel_data;
logic init;
logic en;
logic mem_en;
logic tg_en;

(* dont_touch = "yes" *) logic [ ADDR_RC-1:0] fc_addr;
(* dont_touch = "yes" *) logic [ ADDR_RC-1:0] str_cnt;
```

```

(* dont_touch = "yes" *) logic [RCO_TYPE-1:0] rco_sel;
// selects row, column or w_out memory bloc
logic [INT_BITS + FRC_BITS-1:0] tang;

logic [INT_BITS + FRC_BITS-1:0] din;
logic [INT_BITS + FRC_BITS-1:0] dout;
logic we;
logic m_en;
(* dont_touch = "yes" *) logic [ADDR_OUT-1:0] addr;

img_mem img_mem_inst(
    .clk(clk),
    .we(we),
    .en(m_en),
    .addr(addr),
    .din(din),
    .dout(dout)
);

assign we = (sel_data)? 1 :
| (tg_en & rco_sel == 0)? 1 :
| (tg_en & rco_sel == 1)? 1 : 0
;
assign addr = ( sel_data)? x_i :
| (!sel_data & tg_en & rco_sel == 0)? PICT_SIZE*
str_cnt + fc_addr :
| (!sel_data & !tg_en & rco_sel == 0)? PICT_SIZE*
str_cnt + fc_addr-1 :
| (!sel_data & tg_en & rco_sel == 1)? PICT_SIZE*
fc_addr + str_cnt :
| (!sel_data & !tg_en & rco_sel == 1)? PICT_SIZE
*(fc_addr-1) + str_cnt :
| (!sel_data & !tg_en & rco_sel == 2)? PICT_SIZE*
str_cnt + fc_addr : 0
;
assign din = (sel_data)? din_i :
| (tg_en & rco_sel== 0)? tang :
| (tg_en & rco_sel== 1)? tang : 0
;

(* dont_touch = "yes" *) logic [ADDR_OUT-1:0]      out_addr;
logic [INT_BITS + FRC_BITS-1:0] data2PE;
assign data2PE = dout;

genvar i, j;
generate
    for (i = 0; i < 10; i = i + 1) begin : PE_rco_inst
        PE_rco#(.INT_BITS(INT_BITS), .FRC_BITS(FRC_BITS),
        .N(i)) PE_rco_0(
            .clk(clk),
            .init(init),
            .en(en),
            .address_row(fc_addr[ADDR_RC-1:0]),
            .address_col(fc_addr[ADDR_RC-1:0]),
            .address_o(out_addr),
            .rco_sel(rco_sel),
            .din(data2PE),
            .dout(internal_rc[i]));
    end
endgenerate

```

```

        end
endgenerate

generate
    for (j = 0; j < 18; j = j + 1) begin : PE_rc_inst
        PE_rc#.INT_BITS(INT_BITS), .FRC_BITS(FRC_BITS), .N(j
            +10)) PE_rc_0(
            .clk(clk),
            .init(init),
            .en(en),
            .address_row(fc_addr[ADDR_RC-1:0]),
            .address_col(fc_addr[ADDR_RC-1:0]),
            .din(data2PE),
            .dout(internal_rc[j+10]),
            .rc_sel(rco_sel[0]));
    end
endgenerate

logic [INT_BITS + FRC_BITS-1:0] data_tg;

always_comb begin
    data_tg = internal_tanh[fc_addr];
end

tanh_function #(.INT_SIZE(INT_BITS), .FRC_SIZE(FRC_BITS))
    tanh_inst(.X(data_tg), .Y(tang));

always_comb begin
    if(mem_en) begin
        internal_tanh = internal_rc;
    end
end

assign result = internal_rc[ADDR_OUT-1:0];

softmax #(.BITS(BITS), .HEIGHT(10)) softmax_inst(
    result_layer(result), .predict_num(num_o));

// -----
// CNT
// -----
logic out_en;
logic [ADDR_OUT-1:0] out_addr_next;

always_ff @(posedge clk) begin
    if(~rst_n) begin
        out_addr = ADDR_OUT'(0);
    end else if(out_en) begin
        out_addr <= out_addr_next;
    end else begin
        out_addr <= ADDR_OUT'(0);
    end
end

assign out_addr_next = out_addr + ADDR_OUT'(1);

// -----
// CNT
// -----

```

```

logic fc_en;
logic [ADDR_RC-1:0] fc_addr_next;

always_ff @(posedge clk) begin
    if(~rst_n) begin
        fc_addr = ADDR_RC'(0);
    end else if(fc_en) begin
        fc_addr <= fc_addr_next;
    end else begin
        fc_addr <= ADDR_RC'(0);
    end
end

assign fc_addr_next = fc_addr + ADDR_RC'(1);

// -----
// CNT
// -----
logic str_en;
logic rst_str_cnt;
logic [ADDR_RC-1:0] str_cnt_next;

always_ff @(posedge clk) begin
    if(~rst_n) begin
        str_cnt = ADDR_RC'(0);
    end else if(str_en) begin
        str_cnt <= str_cnt_next;
    end else begin
        if(~rst_str_cnt) begin
            str_cnt <= str_cnt;
        end else begin
            str_cnt <= ADDR_RC'(0);
        end
    end
end

assign str_cnt_next = str_cnt + ADDR_RC'(1);

// -----
// FSM
// -----
typedef enum logic [NN_FSM_BUS_WIDTH-1:0] {
    NN_INIT          = NN_FSM_BUS_WIDTH'(b00000),
    NN_LOAD_IMG     = NN_FSM_BUS_WIDTH'(b10000),
    NN_LD_MEM       = NN_FSM_BUS_WIDTH'(b10001),
    NN_INIT_CALC    = NN_FSM_BUS_WIDTH'(b10011),
    NN_STRING_CALC  = NN_FSM_BUS_WIDTH'(b10010),
    NN_STRING_RDY   = NN_FSM_BUS_WIDTH'(b10110),
    NN_TANG_STRING_CALC = NN_FSM_BUS_WIDTH'(b10111),
    NN_UPD_RCO_TYPE = NN_FSM_BUS_WIDTH'(b10101),
    NN_INIT_OUT     = NN_FSM_BUS_WIDTH'(b11101),
    NN_OUT_CALC     = NN_FSM_BUS_WIDTH'(b11111),
    NN_OUT_RDY      = NN_FSM_BUS_WIDTH'(b11110)
} nn_fc_state_t;

nn_fc_state_t fsm_state_ff, fsm_state_next;

always_ff @(posedge clk) begin

```

```

if(~rst_n) begin
    fsm_state_ff <= NN_INIT;
end else begin
    fsm_state_ff <= fsm_state_next;
end
end

logic fsm_init_tr;
logic fsm_load_img_tr;
logic fsm_ld_mem_tr;
logic fsm_init_calc_tr;
logic fsm_str_calc_tr;
logic fsm_str_rdy_tr;
logic fsm_tang_str_calc_tr;
logic fsm_upd_rc0_type_tr;
logic fsm_init_out_tr;
logic fsm_out_calc_tr;
logic fsm_out_rdy_tr;

assign fsm_state_next = (fsm_init_tr ) ? NN_INIT
:
(fsm_load_img_tr ) ?
NN_LOAD_IMG :
(fsm_ld_mem_tr ) ? NN_LD_MEM
:
(fsm_init_calc_tr ) ?
NN_INIT_CALC :
(fsm_str_calc_tr ) ?
NN_STRING_CALC :
(fsm_str_rdy_tr ) ?
NN_STRING_RDY :
(fsm_tang_str_calc_tr ) ?
NN_TANG_STRING_CALC :
(fsm_upd_rc0_type_tr ) ?
NN_UPD_RCO_TYPE :
(fsm_init_out_tr ) ?
NN_INIT_OUT :
(fsm_out_calc_tr ) ?
NN_OUT_CALC :
(fsm_out_rdy_tr ) ? NN_OUT_RDY
: fsm_state_ff;

assign rdy_o = (fsm_state_ff == NN_OUT_RDY ) ;

// -----
logic col_en;
logic rdy_en;
logic str_tg_en;

logic rc_done;

assign fsm_init_tr = (fsm_state_ff == NN_INIT
) & ~start_i
| (fsm_state_ff == NN_OUT_RDY
);

assign fsm_load_img_tr = (fsm_state_ff == NN_INIT
) & start_i;
assign fsm_ld_mem_tr = (fsm_state_ff == NN_LOAD_IMG
);

```

```

) & (x_i == WIDTH-1)
    | (fsm_state_ff ==
        NN_TANG_STRING_CALC ) & (
        fc_addr == PICT_SIZE-1)
    | (fsm_state_ff ==
        NN_UPD_RCO_TYPE ) & (
        rco_sel != 2)
;
assign fsm_init_calc_tr      = (fsm_state_ff == NN_LD_MEM
);
assign fsm_str_calc_tr       = (fsm_state_ff == NN_INIT_CALC
) & (str_cnt != PICT_SIZE);
assign fsm_str_rdy_tr        = (fsm_state_ff == NN_STRING_CALC
) & (fc_addr == PICT_SIZE);
assign fsm_tang_str_calc_tr = (fsm_state_ff == NN_STRING_RDY
) ;
assign fsm_upd_rco_type_tr  = (fsm_state_ff == NN_INIT_CALC
) & col_en;
assign fsm_init_out_tr       = (fsm_state_ff ==
    NN_UPD_RCO_TYPE ) & (rco_sel == 2);
assign fsm_out_calc_tr       = (fsm_state_ff == NN_INIT_OUT
);
assign fsm_out_rdy_tr        = (fsm_state_ff == NN_OUT_CALC
) & rdy_en;

// -----
assign m_en = (fsm_state_ff == NN_LOAD_IMG
| (fsm_state_ff == NN_STRING_CALC
| (fsm_state_ff == NN_TANG_STRING_CALC )
| (fsm_state_ff == NN_OUT_CALC
| (fsm_state_ff == NN_INIT) & start_i;

// -----
assign rdy_en     = (out_addr == WIDTH+1);
assign rst_str_cnt = fsm_upd_rco_type_tr;
assign tg_en      = (fsm_state_ff == NN_TANG_STRING_CALC );
assign rc_done    = (fc_addr == PICT_SIZE);
assign mem_en     = (fsm_state_ff == NN_STRING_RDY
);
assign str_en     = (fsm_state_ff == NN_TANG_STRING_CALC ) & (
    fc_addr == PICT_SIZE-1)
    | (fsm_state_ff == NN_OUT_CALC
        fc_addr == PICT_SIZE-1);
assign fc_en      = (fsm_state_ff == NN_INIT_CALC
| (fsm_state_ff == NN_STRING_CALC
    fc_addr != PICT_SIZE)
| (fsm_state_ff == NN_TANG_STRING_CALC )
| (fsm_state_ff == NN_OUT_CALC
    fc_addr != PICT_SIZE-1);

assign str_tg_en = (str_cnt == PICT_SIZE-1);

assign col_en     = (fc_addr == 0) & (str_cnt == PICT_SIZE)
& (rco_sel != 1) | (fc_addr == PICT_SIZE) & (str_cnt ==
PICT_SIZE) & (rco_sel == 1);

assign out_en     = (fsm_state_ff == NN_OUT_CALC
| (fsm_state_ff == NN_INIT_OUT
| (fsm_state_ff == NN_UPD_RCO_TYPE);

```

```

// -----
assign sel_data          = (fsm_state_ff == NN_LOAD_IMG) |
                         (fsm_state_ff == NN_INIT) & start_i;

// -----
logic [RCO_TYPE-1:0] rco_sel_next;

always_ff @(posedge clk) begin
    if(~rst_n) begin
        rco_sel = RCO_TYPE'(0);
    end else if(start_i) begin
        rco_sel <= RCO_TYPE'(0);
    end else if(fsm_upd_rco_type_tr) begin
        rco_sel <= rco_sel_next;
    end else begin
        rco_sel <= rco_sel;
    end
end

assign rco_sel_next = rco_sel + RCO_TYPE'(1);

assign init = (fsm_state_ff == NN_LOAD_IMG) |
              (fsm_state_ff == NN_LD_MEM) |
              (fsm_state_ff == NN_STRING_RDY) |
              (fsm_state_ff == NN_TANG_STRING_CALC) |
              (fsm_state_ff == NN_INIT_OUT) |
              (fsm_state_ff == NN_UPD_RCO_TYPE)
              ;
assign en = (fsm_state_ff == NN_STRING_CALC) |
            (fsm_state_ff == NN_OUT_CALC) |
            (fsm_state_ff == NN_OUT_RDY)
            ;
endmodule

```

Модуль памяти изображения:

```

'timescale 1ns / 1ps

module img_mem
    import nn_param_pkg::*;
(
    input logic                                clk, // clock
    input logic                                we, // clock
    input logic                                en, // clock
    input logic [ADDR_OUT-1:0] addr,
    input logic [INT_BITS + FRC_BITS-1:0] din,
    output logic [INT_BITS + FRC_BITS-1:0] dout
);

(* rom_style = "block" *) reg [INT_BITS + FRC_BITS-1:0]
    data [WIDTH-1:0];

    always @(posedge clk) begin
        if(we & en) begin
            data[addr] <= din;
        end
    end

    assign dout = (!we & en)? data[addr] : 0;

```

```
endmodule
```

Модуль обработки пикселя:

```

`timescale 1ns / 1ps
//
// Module Name: PE_rc
// Project Name: LST-1 model
//
// (* dont_touch = "yes" *) (* keep_hierarchy = "yes" *)
(* dont_touch = "yes" *) (* keep_hierarchy = "yes" *)

module PE_rc#(
    parameter int INT_BITS = 5, // integer part
    parameter int FRC_BITS = 7, // fractional part
    parameter int N = 0 // block number
) (
    input logic clk, // clock
    input logic init, // write initial value
    input logic en, // execute MAC-operation
    input logic [4:0] address_row,
    input logic [4:0] address_col,
    input logic rc_sel, // selects row, column or w_out
    memory block
    input [INT_BITS + FRC_BITS-1:0] din,
    output [INT_BITS + FRC_BITS-1:0] dout
);

// Internal signals
logic [INT_BITS + FRC_BITS-1:0] rom_row_out, rom_col_out,
    rom_w_out, mux_rom;
logic [4:0] addr_row, addr_col;
assign addr_row = (init)? 0 : address_row+1;
assign addr_col = (init)? 0 : address_col+1;

ROM_row #(.INT_BITS(INT_BITS), .FRC_BITS(FRC_BITS), .N(N))
    rom_row(.address(addr_row), .dout(rom_row_out), .clk(clk));
ROM_col #(.INT_BITS(INT_BITS), .FRC_BITS(FRC_BITS), .N(N))
    rom_col(.address(addr_col), .dout(rom_col_out), .clk(clk));

always_comb begin : MUX_ROM
    case (rc_sel)
        1'b0: mux_rom = rom_row_out;
        1'b1: mux_rom = rom_col_out;
    endcase
end

mac_core #(.INT_BITS(INT_BITS), .FRC_BITS(FRC_BITS))
    mac_inst(.din(din), .mem_in(mux_rom), .mac_out(dout),
        .clk(clk), .init(init), .en(en));

endmodule

`timescale 1ns / 1ps
//
// -----

```

```

// Module Name: PE_rco
// Project Name: LST-1 model
//
///////////////////////////////



(* dont_touch = "yes" *) (* keep_hierarchy = "yes" *)

module PE_rco#(
    parameter int INT_BITS = 5, // integer part
    parameter int FRC_BITS = 7, // fractional part
    parameter int N = 0 // block number
) (
    input logic clk, // clock
    input logic init, // write initial value
    input logic en, // execute MAC-operation
    input logic [4:0] address_row,
    input logic [4:0] address_col,
    input logic [9:0] address_o,
    input logic [1:0] rco_sel, // selects row, column or
    w_out memory block
    input [INT_BITS + FRC_BITS-1:0] din,
    output [INT_BITS + FRC_BITS-1:0] dout
);

// Internal signals
logic [INT_BITS + FRC_BITS-1:0] rom_row_out, rom_col_out,
    rom_w_out, mux_rom;
logic [4:0] addr_row, addr_col;
assign addr_row = (init)? 0 : address_row+1;
assign addr_col = (init)? 0 : address_col+1;

ROM_row #(.INT_BITS(INT_BITS), .FRC_BITS(FRC_BITS), .N(N))
    rom_row(.address(addr_row), .dout(rom_row_out), .clk(clk));
ROM_col #(.INT_BITS(INT_BITS), .FRC_BITS(FRC_BITS), .N(N))
    rom_col(.address(addr_col), .dout(rom_col_out), .clk(clk));
ROM_w_out #(.INT_BITS(INT_BITS), .FRC_BITS(FRC_BITS), .N(N))
    rom_w(.address(address_o), .dout(rom_w_out), .clk(clk));

always_comb begin : MUX_ROM
    unique case (rco_sel)
        2'b00: mux_rom = rom_row_out;
        2'b01: mux_rom = rom_col_out;
        2'b10: mux_rom = rom_w_out;
    endcase
end

mac_core #(.INT_BITS(INT_BITS), .FRC_BITS(FRC_BITS))
    mac_inst(.din(din), .mem_in(mux_rom), .mac_out(dout),
        .clk(clk), .init(init), .en(en));

endmodule

```

Модуль памяти строк:

```
'timescale 1ns / 1ps
```

11

```

// WEIGHT MEMORY (ROM)
// //////////////////////////////

(* dont_touch = "yes" *) (* keep_hierarchy = "yes" *)

module ROM_row #(
    parameter int INT_BITS = 6, // integer part
    parameter int FRC_BITS = 7, // fractional part
    parameter int N = 0 // block number
) (
    input logic clk, // clock
    input logic [4:0] address,
    output [INT_BITS + FRC_BITS-1:0] dout
);
(* rom_style = "block" *) reg [INT_BITS + FRC_BITS-1:0]
data;

generate
if (N == 0) begin
    always @ (posedge clk) begin
        case(address)
            5'b00000: data <= 13'h0000;
            ...
            default: data <= 0;
        endcase
    end
end
endgenerate

generate
if (N == 1) begin
    always @ (posedge clk) begin
        case(address)
            5'b00000: data <= 13'h000d;
            ...
            default: data <= 0;
        endcase
    end
end
endgenerate

...
generate
if (N == 27) begin
    always @ (posedge clk) begin
        case(address)
            5'b00000: data <= 13'h0027;
            ...
            default: data <= 0;
        endcase
    end
end
endgenerate

```

```

        assign dout = data;
endmodule

Модуль памяти столбцов:

'timescale 1ns / 1ps

// //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// WEIGHT MEMORY (ROM)
// //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

(* dont_touch = "yes" *) (* keep_hierarchy = "yes" *)

module ROM_col #(
    parameter int INT_BITS = 6, // integer part
    parameter int FRC_BITS = 7, // fractional part
    parameter int N = 0 // block number
) (
    input logic clk, // clock
    input logic [4:0] address,
    output [INT_BITS + FRC_BITS-1:0] dout
);

(* rom_style = "block" *) reg [INT_BITS + FRC_BITS-1:0]
data;

generate
    if (N == 0) begin
        always @ (posedge clk) begin
            case(address)
                5'b00000: data <= 13'h0030;
                ...
                default: data <= 0;
            endcase
        end
    end
endgenerate

generate
    if (N == 1) begin
        always @ (posedge clk) begin
            case(address)
                5'b00000: data <= 13'h0052;
                ...
                default: data <= 0;
            endcase
        end
    end
endgenerate

...
generate
    if (N == 27) begin
        always @ (posedge clk) begin
            case(address)

```

```

      5'b00000: data <= 13'h1fec;
      ...
      default: data <= 0;
    endcase
  end
end
endgenerate

assign dout = data;
endmodule

```

Модуль памяти выходных весов:

```

'timescale 1ns / 1ps

// ///////////////////////////////////////////////////////////////////
// WEIGHT MEMORY (ROM)
// ///////////////////////////////////////////////////////////////////

(* dont_touch = "yes" *) (* keep_hierarchy = "yes" *)

module ROM_w_out #(
  parameter int INT_BITS = 5, // integer part
  parameter int FRC_BITS = 7, // fractional part
  parameter int N = 0 // block number
) (
  input logic clk, // clock
  input logic [9:0] address,
  output [INT_BITS + FRC_BITS-1:0] dout
);

  (* rom_style = "block" *) reg [INT_BITS + FRC_BITS-1:0]
  data;

  generate
    if (N == 0) begin
      always @ (posedge clk) begin
        case(address)
          10'b0000000000: data <= 13'h1ff2;
          ...
          default: data <= 0;
        endcase
      end
    end
  endgenerate

  generate
    if (N == 1) begin
      always @ (posedge clk) begin
        case(address)
          10'b0000000000: data <= 13'h0006;
          ...
          default: data <= 0;
        endcase
      end
    end
  endgenerate

```

```

    endgenerate
    ...
generate
  if (N == 9) begin
    always @(posedge clk) begin
      case(address)
        10'b0000000000: data <= 13'h1ff2;
        ...
        default: data <= 0;
      endcase
    end
  end
endgenerate

  assign dout = data;
endmodule

```

Модуль MAC:

```

`timescale 1ns / 1ps

(* use_dsp = "yes" *) (* dont_touch = "yes" *) (*
keep_hierarchy = "yes" *)

module mac_core #(
  parameter int INT_BITS = 5, // integer part
  parameter int FRC_BITS = 7 // fractional part
) (
  input logic clk,      // clock
  input logic init,    // write initial value
  input logic en,       // execute MAC-operation
  input logic [INT_BITS + FRC_BITS-1:0] din,      // input
  data
  input logic [INT_BITS + FRC_BITS-1:0] mem_in,   // weight
  of NN
  output logic [INT_BITS + FRC_BITS-1:0] mac_out // accumulator output
);
  logic [INT_BITS + FRC_BITS-1:0] acc;           // register-
  accumulator
  logic [INT_BITS + FRC_BITS-1:0] m_o;          // multiplier output
  logic [2*(INT_BITS + FRC_BITS)-1:0] mul_out;
  logic [2*(INT_BITS + FRC_BITS)-1:0] mul_out_1;
  logic [2*(INT_BITS + FRC_BITS)-1:0] correct = 24',
  b000000000000_000001000000;
//assign mul_out = signed'(din) * signed'(mem_in);
MUL_N #(N(INT_BITS + FRC_BITS), .qA(FRC_BITS)) mul_inst(.A(
  din), .B(mem_in), .P(mul_out));

  always_ff @ (posedge clk) begin
    if (init) begin
      acc <= mem_in;
    end else begin
      if (en) begin
        acc <= m_o + acc; // MAC: A = A + B * C
      end
    end
  end
end

```

```

assign mul_out_1 = mul_out; // + correct;
assign m_o = mul_out_1[INT_BITS + 2*FRC_BITS-1 : FRC_BITS];
assign mac_out = acc;
endmodule

```

Модуль умножителя:

```

-- 
----- 
-- Company: 
-- Engineer: 
-- 
-- Create Date:      11:11:10 01/12/2008 
-- Design Name: 
-- Module Name:     ADD1 - Behavioral 
-- Project Name: 
-- Target Devices: 
-- 
----- 

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if
-- instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity ADD1 is
    Port ( A : in STD_LOGIC;
           B : in STD_LOGIC;
           Ci : in STD_LOGIC;
           S : out STD_LOGIC;
           Co : out STD_LOGIC);
end ADD1;

architecture Behavioral of ADD1 is
begin
S<= (A xor B xor Ci);
Co<=(A and B)or(A and Ci)or (B and Ci);
end Behavioral;

-- 
----- 
-- Company: 
-- Engineer: 
-- 
-- Create Date:      11:03:14 01/17/2008 
-- Design Name: 
-- Module Name:     HA - Behavioral 
-- 
----- 

```

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if
---- instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity HA is
    Port ( A : in STD_LOGIC;
           B : in STD_LOGIC;
           S : out STD_LOGIC;
           Co : out STD_LOGIC);
end HA;

architecture Behavioral of HA is

begin
S<=A xor B;
Co<=A and B;
end Behavioral;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity MUL_N is
    generic (N:natural:=16;
             qA:natural:=8); -- fractional part of A (
                           -- to remove from product)
    port ( A : in STD_LOGIC_VECTOR (N-1 downto 0);
           B : in STD_LOGIC_VECTOR (N-1 downto 0);
           P : out STD_LOGIC_VECTOR (2*N-1 downto 0) -- 
                           full multiplier
--                           P : out STD_LOGIC_VECTOR(N-1 downto
--                           0) -- half of output bits is needed
           );
end MUL_N;

architecture Behavioral of MUL_N is
type abij is array(0 to N-1) of std_logic_vector(N-1 downto
0);
component ADD1
    Port ( A : in STD_LOGIC;
           B : in STD_LOGIC;
           Ci : in STD_LOGIC;
           S : out STD_LOGIC;
           Co : out STD_LOGIC);
end component;
component HA
    Port ( A : in STD_LOGIC;
           B : in STD_LOGIC;
           S : out STD_LOGIC;
           Co : out STD_LOGIC);
end component;

```

```

signal ab,s,c: abij;
signal c_out: std_logic_vector(N-1 downto 0);
signal p_out: std_logic_vector(2*N-1 downto 0);
begin

row: for i in 0 to N-1 generate
    begin
        col: for j in 0 to N-1 generate
            begin
                n8: if ((j=N-1)and(i/=N-1))OR
                    ((i=N-1)and(j/=N-1))
                    generate
                        begin
                            ab(i)(j)<=
                                not(A(j)
                                    and B(i));
                        end generate;
                a8: if ((j/=N-1)and(i/=N-1))
                    OR((i=N-1)and(j=N-1))
                    generate
                        begin
                            ab(i)(j)<=(A(
                                j) and B(i)
                            );
                        end generate;
                end generate;
            end generate;
        s(0)<=ab(0);
        c(0)<=(others=>'0');
    mul: for i in 0 to N-2 generate
        begin
            sm: for j in 0 to N-1 generate
                begin
                    one: if (j=0) generate
                        ad: HA port map (A=>s
                            (i)(j),B=>c(i)(j),S
                            =>p_out(i),Co=>c(i
                            +1)(j));
                    end generate;
                    oth: if (j/=0) generate
                        ad: ADD1 port map (A
                            =>s(i)(j),B=>ab(i
                            +1)(j-1),Ci=>c(i)(j
                            ),S=>s(i+1)(j-1),Co
                            =>c(i+1)(j));
                    end generate;
                end generate;
                s(i+1)(N-1)<=ab(i+1)(N-1);
            end generate;
        mul_end:
            for i in 0 to N-1 generate
            begin
                fst: if (i=0) generate
                    ad: HA port map (A=>s(N-1)(i),B=>c(N
                        -1)(i),S=>p_out(N-1+i),Co=>c_out(i
                        ));
                end generate;
                one: if (i=1) generate
                    ad: ADD1 port map (A=>s(N-1)(i),B=>'1',Ci

```

```

        =>c(N-1)(i),S=>p_out(N-1+i),Co=>c_out(i
    ));
    end generate;
oth: if ((i/=0) and (i/=1)) generate
    ad: ADD1 port map (A=>s(N-1)(i),B=>c_out(
        i-1),Ci=>c(N-1)(i),S=>p_out(N-1+i),Co=>
        c_out(i));
    end generate;
end generate;
p_out(2*N-1) <= c_out(N-1);
P<=p_out;
-- full integer multiplier
--P<=p_out(2*(N-1) downto N-1);-- half of product fractional
multiplier
--P<=p_out(N-1 downto 0);-- half of product fractional
multiplier
--P<=p_out(N+(N/2)-1 downto N/2);-- half of product
--P<=p_out(N+qA-1 downto qA);-- half of product
end Behavioral;

```

Модуль тангенса:

```

'timescale 1ns / 1ps
// /////////////////////////////////
// Create Date: 29.12.2024 18:44:00
// Design Name:
// Module Name: tanh_function
// Project Name: L2DST-one-block-NN
// /////////////////////////////////

```



```

module tanh_function #(
    parameter int INT_SIZE = 3, // integer part
    parameter int FRC_SIZE = 8 // fractional part
) (
    input [(INT_SIZE+FRC_SIZE - 1):0] X,
    output logic [(INT_SIZE+FRC_SIZE - 1):0] Y
);
    localparam logic [FRC_SIZE-1:0] zero_string = '0;
    localparam logic [FRC_SIZE-1:0] ones_string = '1;

    localparam const_plus_two = {3'b010, zero_string};
    localparam const_minus_two = {3'b110, zero_string};
    localparam const_q_plus_one = {1'b0, ones_string};
    localparam const_q_minus_one = {1'b1, zero_string};

    logic [FRC_SIZE:0] add_sub_out;
    logic [INT_SIZE+FRC_SIZE-1:0] add_sub_out_ext;
//    logic [2*(INT_SIZE+FRC_SIZE)-1:0] mul_out;
    logic [(INT_SIZE+FRC_SIZE-1):0] x_quater;
    logic x_sign;

    logic [INT_SIZE + FRC_SIZE-1:0] m_o; // multiplier output
    logic [2*(INT_SIZE + FRC_SIZE)-1:0] mul_out;

```

```

logic [2*(INT_SIZE + FRC_SIZE)-1:0] mul_out_1;
logic [2*(INT_SIZE + FRC_SIZE)-1:0] correct = 26',
    b000000_0000_0000_0000_0100_0000;

assign x_sign = X[INT_SIZE+FRC_SIZE-1];

always_comb begin : Tanh_approximation
//   if (x_sign == 0) begin // (x>0) branch
    if ($signed(X) >= $signed(const_plus_two)) Y = 13',
        b00000_1000_0000;
    else if ($signed(X) <= $signed(const_minus_two)) Y =
        13'b11111_1000_0000;
    else Y = m_o;
//   end
end

assign x_quater = {x_sign, x_sign, X[INT_SIZE+FRC_SIZE
-1:2]};

always_comb begin : Add_Sub_block
if (x_sign == 0) add_sub_out = const_q_plus_one -
    x_quater[FRC_SIZE:0];
else add_sub_out = const_q_plus_one + x_quater[FRC_SIZE
:0];
end

assign add_sub_out_ext = {add_sub_out[FRC_SIZE],
    add_sub_out[FRC_SIZE], add_sub_out[FRC_SIZE], add_sub_out
[FRC_SIZE], add_sub_out};

MUL_N #(
    .N (INT_SIZE + FRC_SIZE),
    .qA(FRC_SIZE)
) mult_inst (
    .A(add_sub_out_ext),
    .B(X),
    .P(mul_out)
);

assign mul_out_1 = mul_out; // + correct;
assign m_o = mul_out_1[INT_SIZE + 2*FRC_SIZE-1 : FRC_SIZE];

```

endmodule

Модуль softmax:

```
'timescale 1ns / 1ps
```

```

// /////////////////////////////////
// SOFTMAX activation function
// This code find max element of 10 input elements,
// but don't use exp() and as a result don't use probability.
// This module just finde max elements whithout any
// transformations.
// /////////////////////////////////

```

```

//(*use_dsp = "yes")
module softmax#(
    parameter int BITS = 24, // bit depth
    parameter int HEIGHT = 10 // size of array_weight
)()
//    input logic clk, //clock
//    input logic reset, // reset
    input logic [BITS - 1 : 0] result_layer [HEIGHT-1:0], //
        input 10 elements from full connected layer
    output logic [BITS - 1 : 0] predict_num // predict number
);

// Internal signal
logic [BITS - 1 : 0] max;

always_comb begin
    max = result_layer[0];
    predict_num = 0;
    for (int i = 1; i < HEIGHT; i++) begin
        if (signed'(result_layer[i]) > signed'(max)) begin
            max = result_layer[i];
            predict_num = i;
        end
    end
end
endmodule

```

**ПРИЛОЖЕНИЕ И
(Обязательное)
Результаты работы нейронной сети**