ВЕСТНИК ТОМСКОГО ГОСУДАРСТВЕННОГО УНИВЕРСИТЕТА
2025 Управление, вычислительная техника и информатика № 71
Tomsk State University Journal of Control and Computer Science

# ИНФОРМАТИКА И ПРОГРАММИРОВАНИЕ

# INFORMATICS AND PROGRAMMING

Научная статья
УДК 621.382
doi: 10.17223/19988605/71/12

## Реализация на ПЛИС и сравнительный анализ вычислителей сигмоида, работающих с полным диапазоном аргумента с учетом симметрии

**Инна Владимировна Ушенина**

*Пензенский государственный технологический университет, Пенза, Россия, ivl23@yandex.ru*

**Аннотация.** Реализованы на ПЛИС и проанализированы две схемы вычислителей функции сигмоида, одна из которых использует симметрию функции, а вторая работает с полным диапазоном аргумента. Каждая из схем реализована при разрядности от 7 до 10 бит. Показано, что по сравнению со второй схемой первая расходует меньшее количество ресурсов. Однако в первой схеме используются вспомогательные блоки с длинными цепями переноса, которые существенно удлиняют критический путь распространения сигнала и таким образом влияют на время вычислений. По сравнению с автоматическим размещением на кристалле элементов вспомогательных блоков, принадлежащих критическому пути, их ручное размещение строго по порядку сокращает время работы блоков на величину до 25%, а время работы первой схемы в целом – до 17%. Но в любом случае вторая схема будет работать как минимум вдвое быстрее первой.

**Ключевые слова:** сигмоид; нейронная сеть; ПЛИС; метод поразрядного отображения; симметрия; критический путь.

***Благодарности*:** Исследование выполнено за счет гранта Российского научного фонда и Пензенской области № 24-21-20100, https://rscf.ru/project/24-21-20100/

***Для цитирования*:** Ушенина И.В. Реализация на ПЛИС и сравнительный анализ вычислителей сигмоида, работающих с полным диапазоном аргумента с учетом симметрии // Вестник Томского государственного университета. Управление, вычислительная техника и информатика. 2025. № 71. С. 120–129. doi: 10.17223/19988605/71/12

Original article
doi: 10.17223/19988605/71/12

## FPGA implementation and comparative analysis of sigmoid calculators processing the full argument range in view of symmetry

**Inna V. Ushenina**

*Penza State Technological University, Penza, Russian Federation, ivl23@yandex.ru*

**Abstract**. This paper implements on FPGA and analyses two sigmoid calculation circuits: the first circuit uses the function's symmetry, and the second one processes the full argument range. Each circuit is implemented with a bit width of 7 to 10 bits. It is shown that, compared to the second circuit, the first one consumes a smaller amount of resources. But the first circuit uses auxiliary blocks containing long carry chains. sigmoid function; neural network; FPGA; bit-level mapping method; symmetry; critical path.

**Keywords:** sigmoid function; neural network; FPGA; bit-level mapping method; symmetry; critical path.

***Acknowledgments*:** The research was carried out with the financial support of the Russian Science Foundation and Penza region under Grant № 24-21-20100 https://rscf.ru/project/24-21-20100/

***For citation*:** Ushenina, I.V. (2025) FPGA implementation and comparative analysis of sigmoid calculators processing the full argument range in view of symmetry. *Vestnik Tomskogo gosudarstvennogo universiteta. Upravlenie, vychislitelnaja tehnika i informatika – Tomsk State University Journal of Control and Computer Science*. 71. pp. 120–129. doi: 10.17223/19988605/71/12

## Введение

Аппаратная реализация нейронных сетей необходима в тех случаях, когда они должны работать в реальном времени, а также когда требуется ускорение их работы или обучения. Среди аппаратных платформ, подходящих для реализации нейронных сетей – графических процессоров, заказных микросхем, ПЛИС – последние могут проигрывать другим платформам в производительности и энергоэффективности, но являются наиболее доступным вариантом и имеют возможность перепрограммирования [1–3].

Нейронные сети, обучающиеся по алгоритму обратного распространения ошибки, используют в качестве функций активации сигмоид

$$s(x) = \frac{1}{1+e^{-x}}. \quad (1)$$

или гиперболический тангенс, которые имеют подходящую форму, непрерывно дифференцируемы и требуют небольших дополнительных аппаратных затрат на вычисление производных в ходе обучения.

Современные ПЛИС не позволяют напрямую вычислять частное и экспоненту. Для вычисления на ПЛИС сигмоида и гиперболического тангенса к настоящему моменту предложен ряд методов, среди которых – метод поразрядного отображения [4]. Он заключается в том, что каждый разряд двоичного представления значения функции отделяется от других и вычисляется как булева функция от разрядов двоичного представления аргумента. Это позволяет выполнять вычисления относительно быстро, без дополнительной погрешности и использовать для вычислений блоки программируемой логики, имеющиеся на ПЛИС в большом количестве [5].

Дополнительным преимуществом функций сигмоида и гиперболического тангенса, актуальным при их аппаратной реализации, является симметрия. Так, $s(x)$ при отрицательных аргументах благодаря симметрии можно вычислить по формуле

$$s(x) = 1 - s(|x|). \quad (2)$$

В [4], где предложен метод поразрядного отображения, и в [5–9], использующих этот метод для вычисления функций сигмоида или гиперболического тангенса, описаны вычисления только при неотрицательных аргументах. В [4, 6], где вычисляется $s(x)$, отрицательные аргументы предлагается обрабатывать согласно (2). При этом в [4] схема вычислений по формуле (2) не описывается, а в [6] представлена только функциональная схема, без описания ее реализации.

Дополнение вычислителей отдельных разрядов значений $s(x)$, работающих только с неотрицательными аргументами, вычислениями по (2) может привести к существенному снижению скорости вычислений. Это связано с тем, что аппаратная реализация сложения и вычитания требует переноса битов переполнения. Перенос выполняют цепочки комбинационной логики, длина которых соответствует разрядности обрабатываемых чисел. Вместе с вычислителем младшего разряда значения $s(|x|)$ цепь переноса будет образовывать критический путь, время прохождения сигнала по которому будет влиять на скорость вычислителя $s(x)$ в целом. Также, поскольку отрицательные аргументы представлены в дополнительном коде, их представления необходимо перевести в прямой код, что требует установки еще одного сумматора и приводит к удлинению критического пути еще одной цепью переноса.

Существует возможность выполнения вычислений по формуле (2) на сумматорах блоков цифровой обработки сигналов (ЦОС) ПЛИС, которые работают значительно быстрее блоков программируемой логики. Однако блоки ЦОС обычно выполняют линейную часть вычислений в нейронах, а именно суммирование произведений входных сигналов нейронов и масштабирующих коэффициентов. То есть

придется вводить временное разделение использования блоков ЦОС и усложнять схему управления нейронной сетью. Кроме того, в слоях нейронных сетей, реализуемых аппаратно, часто есть параллелизм на уровне нейронов, но его нет на уровне синапсов. Это означает, что каждый нейрон в слое представлен собственным блоком ЦОС, на который поступают по очереди сигналы от нейронов предыдущего слоя [10]. В таких случаях вычислитель функции активации устанавливается общим для всех нейронов слоя, и он отделен от блоков ЦОС регистром сдвига [11] или мультиплексором [12].

Способом избежать удлинения критического пути может стать отказ от использования симметрии функции активации и вычислений по (2) и работа методом поразрядного отображения с полным диапазоном аргумента.

Настоящая работа является продолжением [5], где метод поразрядного отображения использован для реализации на блоках программируемой логики ПЛИС схем вычислителей отдельных разрядов значений $s(x)$. Эти схемы работают с аргументами в диапазоне [0; 8) и предполагают использование симметрии сигмоида при обработке отрицательных аргументов. В настоящей работе представлены и проанализированы две схемы вычислителей значений $s(x)$, использующие схемы вычислителей их отдельных разрядов, полученные в [5]. Первая схема использует симметрию функции и имеет длинный критический путь, образованный цепями переноса. Вторая схема не использует симметрию и содержит удвоенный набор вычислителей отдельных разрядов значений $s(x)$: для работы с неотрицательными и отрицательными аргументами.

Представленные схемы работают с числами в формате с фиксированной запятой с разрядностью от 7 до 10 бит. Разрядность 7–8 бит достаточна при развертывании обученной нейронной сети, а разрядность 9–10 бит – при обучении нейронной сети [13]. Разрядности представлений аргументов и значений функции сигмоида во всех случаях равны. При этом аргумент находится в диапазоне [–8; 8), и из всех представляющих его разрядов один является знаковым, три отводятся на представление целой части, а все остальные разряды – на представление дробной части. При указании разрядности аргумента знаковый разряд не учитывается. Значение $s(x)$ находится в диапазоне (0; 1), и все представляющие его разряды относятся к дробной части. Обе схемы используют только блоки программируемой логики ПЛИС, оставляя ресурсы блоков ЦОС для линейной части вычислений в нейронах. Схемы описаны на языке VHDL и реализованы на ПЛИС GW2AR от Gowin Semiconductor [14]. Разработка, реализация и анализ схем выполнены в среде Gowin EDA.

## 1. Схемы вычислителей функции сигмоида

Метод поразрядного отображения предполагает представление отдельных разрядов значений $s(x)$ в виде булевых функций разрядов ее аргумента. Для их получения сначала формируется полный список аргументов и соответствующих им значений $s(x)$, представленных в двоичном коде с заданной разрядностью. На основе этого списка для каждого разряда $s(x)$ формируются таблицы истинности. Вычислители разрядов $s(x)$ можно реализовать непосредственно по данным таблицам. В таком случае вычислители должны отличать каждую из комбинаций разрядов $x$, расходуя на это максимальное количество ресурсов.

По таблицам истинности для расчета разрядов $s(x)$ можно получить булевы функции, представленные в совершенной дизъюнктивной нормальной форме, а затем минимизировать их. В [5] и в настоящей работе минимальные дизъюнктивные нормальные формы (МДНФ) булевых функций для расчета разрядов $s(x)$ получены с использованием алгоритма Куайна–МакКласки.

Значения старших разрядов $s(x)$ часто совпадают для целых групп соседних значений $x$; особенно это выражено для $s(x)$, по модулю близких к нулю и единице. За счет этого свойства представление булевых функций для расчета старших разрядов $s(x)$ в МДНФ сводится к сумме нескольких простых импликант, вычисление которой требует значительно меньшего количества ресурсов, чем работа с таблицей истинности.

По мере продвижения к младшим разрядам $s(x)$ булевы функции в МДНФ содержат все большее количество импликант и литералов в каждой из них, что в конечном итоге сравнивает сложность их вычисления с вычислениями по таблицам истинности [5]. В схемах, представленных в данной работе,

от двух до четырех младших разрядов значений $s(x)$ вычисляются по таблицам истинности, остальные разряды – по булевым функциям, представленным в МДНФ.

### *1.1. Схема вычислителя функции сигмоида, использующая ее симметрию (схема 1)*

Структурная схема вычислителя функции сигмоида, использующего ее симметрию, представлена на рис. 1. Схема содержит, помимо набора описанных в [5] вычислителей отдельных разрядов значений функции сигмоида $s_i(|x|)$, два вспомогательных блока, на которые поступает знаковый разряд аргумента *sign*. Первый блок принимает от линейной части нейрона аргумент сигмоида, представленный в дополнительном коде $x_{доп}$ и формирует представление $|x|$: если аргумент неотрицательный, он остается без изменений; если отрицательный – он переводится в прямой код, т.е. инвертируется и складывается с единицей в младшем разряде. Второй блок при $x \geq 0$ приравнивает $s(x)$ к $s(|x|)$, а при $x < 0$ вычисляет $s(x)$ согласно (2): переводит $–s(|x|)$ в дополнительный код и складывает результат с единицей.

Сумматоры, необходимые во вспомогательных блоках, можно реализовать на базе программируемого примитива арифметико-логического устройства ALU, предлагаемого в [15]. Однако ALU содержит многочисленные настройки, которые не будут использоваться, но займут ресурсы и удлинят критический путь.

С учетом сказанного схемы вспомогательных блоков были построены на базе табличных преобразователей (look-up tables, LUTs) в строгом соответствии с решаемой задачей. LUT работают по таблицам истинности, составленным с учетом особенностей выполняемых операций (диапазоны аргумента и значения $s(x)$, обработка только одного неизвестного слагаемого при сложении, необходимость операции инверсии). Таким образом, функции, требуемые от вспомогательных блоков, экономно реализованы на небольшом количестве LUT.


Рис. 1. Структурная схема вычислителя функции сигмоида, использующего ее симметрию
Fig. 1. Block diagram of the sigmoid function calculator that uses the function's symmetry

Вспомогательные блоки представляют собой последовательно соединенные звенья с одним или двумя LUT, у которых может быть два или три входа (LUT2, LUT3). В звеньях с двумя LUT каждый из них принимает один и тот же набор сигналов. Схема 8-разрядного вычислителя $s(x)$ с вспомогательными блоками на базе LUT представлена на рис. 2.

Во вспомогательном блоке 1 (на рис. 2 слева) каждое звено выполняет при *sign* = 1 инверсию одного из разрядов $x_{доп}$, формируя из него один из разрядов представления модуля $x$ в прямом коде – $x_{i\ пр}$. Нижнее звено, обрабатывающее младший разряд аргумента, $x_{0\ доп}$, при *sign* = 1 складывает его с единицей и выдает $x_{0\ пр}$ и бит переполнения $carry_1$ на следующее звено. Остальные звенья при *sign* = 1 обрабатывают бит переполнения $carry_i$, полученный от предыдущего звена, и $x_{i\ доп}$, формируя бит переполнения для следующего звена $carry_{i+1}$ и $x_{i\ пр}$. Если *sign* = 0, $x_{i\ доп} = x_{i\ пр}$ и $carry_{i+1} = 0$. Нижнее и промежуточные звенья содержат по два LUT, один из которых формирует $x_{i\ пр}$, а другой – $carry_{i+1}$. Верхнее звено содержит один LUT3 и формирует $x_{7\ пр}$.

Полученные разряды $x_{7\ пр} \ldots x_{0\ пр}$ поступают на вычислители $s_i(|x|)$, описанные в [5]. Как и в [5], $s_8(|x|)$ соответствует младшему разряду $s(|x|)$. Также $s_8(x)$ соответствует младшему разряду $s(x)$.

Знаковый разряд аргумента *sign* и $s_i(|x|)$ поступают на звенья вспомогательного блока 2 – вычислителя $s(x)$ по формуле (2), изображенного на рис. 2 справа. Звено, формирующее старший разряд $s_1(x)$, работает как инвертор знакового разряда *sign*, так как при отрицательном аргументе $0 < s(x) < 0{,}5$, а при неотрицательном – $0{,}5 \leq s(x) < 1$. При *sign* = 0 остальные разряды $s(x)$ приравниваются соответствующим разрядам $s(|x|)$, а все биты переноса – нулю. При *sign* = 1 $–s(|x|)$ переводится в дополнительный код

и складывается с единицей, согласно (2). Звено, формирующее разряд $s_2(x)$, не выдает бита переноса и состоит из одного LUT3. Остальные звенья содержат пару LUT, один из которых формирует бит переноса $carry_{i-1}$, а другой – разряд $s_i(x)$. Все звенья, кроме формирующего $s_1(x)$, связаны между собой цепями переноса $carry_2$–$carry_7$.

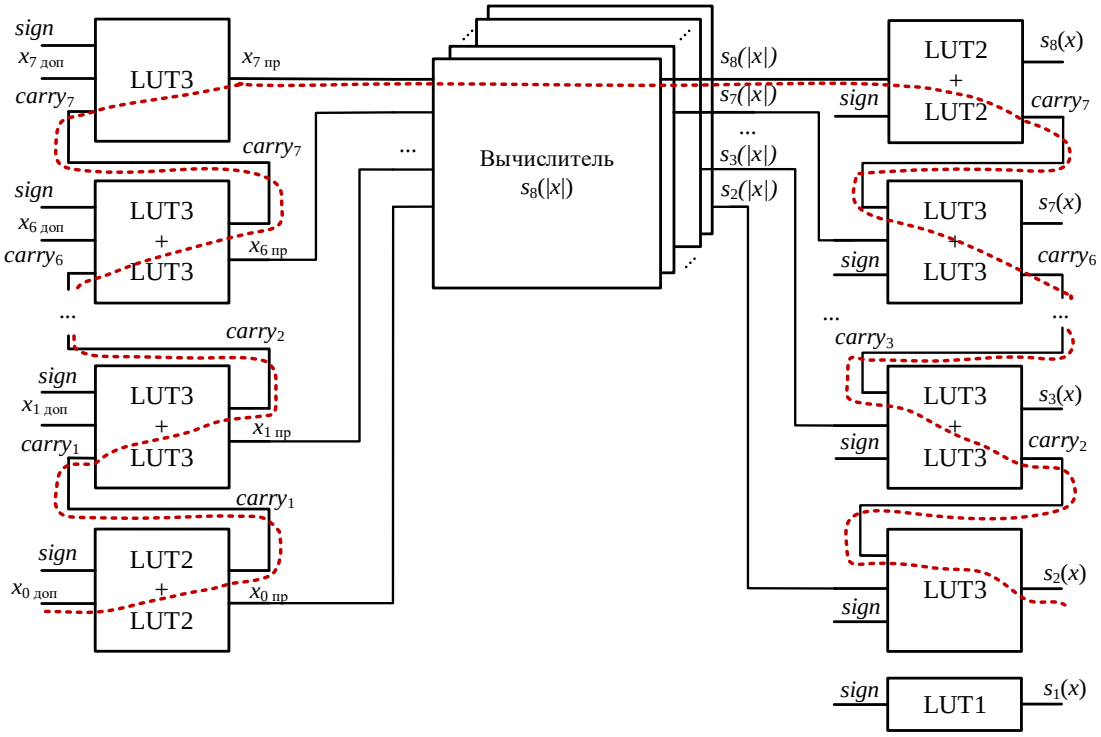


Рис. 2. Схема 1. 8-разрядный вычислитель функции сигмоида, использующий ее симметрию. Вспомогательные блоки выполнены на табличных преобразователях. Критический путь показан штриховой линией

Fig. 2. Circuit 1. 8-bit calculator of the sigmoid function that uses the function's symmetry. The auxiliary blocks are implemented on the LUTs. The critical path is shown by the dashed line

В обоих вспомогательных блоках LUT, формирующие $x_{i\,пр}$ и $s_i(x)$, реализуют одинаковые таблицы истинности. То же касается LUT, формирующих биты переноса в обоих вспомогательных блоках. Таблицы истинности, реализуемые LUT во вспомогательных блоках 1 и 2, представлены в табл. 1, 2.

На рис. 2 показано, что цепи переноса вспомогательных блоков вместе с вычислителем разряда $s_8(|x|)$ образуют критический путь. При других разрядностях $x$ и $s(x)$ схема 1 строится аналогично, отличия будут только в количествах вычислителей $s_i(|x|)$ и звеньев вспомогательных блоков. Таким образом, время вычисления $s(x)$ в схеме 1 определяется критическим путем, содержащим две цепи переноса, количество звеньев в которых соответствует разрядности $x$ и $s(x)$, и вычислитель младшего разряда значения $s(|x|)$.

Таблица 1

**Таблицы истинности LUT2 и LUT3 вспомогательного блока 1**

| Входы | | | Выходы | Выходы | Выход | Выход |
|---|---|---|---|---|---|---|
| $carry_i$ (только LUT3) | *sign* | $x_{i\,доп}$ | $carry_{i+1}$ (LUT3) | $x_{i\,пр}$ (LUT3) | $carry_1$ (LUT2) | $x_{0\,пр}$ (LUT2) |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 0 | – | – |
| 1 | 0 | 1 | 0 | 1 | – | – |
| 1 | 1 | 0 | 1 | 0 | – | – |
| 1 | 1 | 1 | 0 | 1 | – | – |

Таблица 2

**Таблицы истинности LUT2 и LUT3 вспомогательного блока 2**

| Входы | | | Выходы $carry_{i-1}$ (LUT3) | Выходы $s_i(x)$ (LUT3) | Выход $carry_7$ (LUT2) | Выход $s_8(x)$ (LUT2) |
|---|---|---|---|---|---|---|
| $carry_i$ (только LUT3) | *sign* | $s_i(\|x\|)$ | | | | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 0 | – | – |
| 1 | 0 | 1 | 0 | 1 | – | – |
| 1 | 1 | 0 | 1 | 0 | – | – |
| 1 | 1 | 1 | 0 | 1 | – | – |

### *1.2. Схема вычислителя функции сигмоида с двумя наборами вычислителей $s_i(x)$ (схема 2)*

В схеме, представленной на рис. 3 (схема 2), аргумент обрабатывается одновременно двумя наборами вычислителей отдельных разрядов значений функции сигмоида. Первый набор, тот же что и в схеме 1, работает, предполагая $x \geq 0$; второй – предполагая $x < 0$. При этом вычислители первого набора работают с $x$, представленным в прямом коде, а вычислители второго набора – с $x$, представленным в дополнительном коде. То есть преобразователь представления $x$ из дополнительного кода в прямой в данном случае не нужен. Также упраздняются и вычисления по формуле (2), вместо которых в схеме 2 устанавливается многоразрядный мультиплексор 2-1. На его входы подаются значения функции сигмоида $sp(x)$ и $sm(x)$, рассчитанные наборами вычислителей, предполагающими соответственно $x \geq 0$ и $x < 0$. Нужное значение выбирается знаковым разрядом аргумента *sign*, поступающим на управляющий вход мультиплексора.

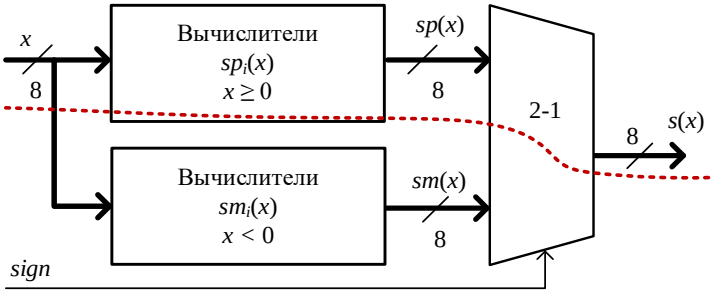


Рис. 3. Схема 2. 8-разрядный вычислитель функции сигмоида с двумя наборами вычислителей отдельных разрядов ее значений. Критический путь показан штриховой линией
Fig. 3. Circuit 2. 8-bit calculator of the sigmoid function with two sets of output bit calculators. The critical path is shown by the dashed line

Таким образом, в схеме 2 критический путь составлен вычислителем одного из разрядов значения $s(x)$ и мультиплексором, – например, как показано на рис. 3.

## 2. Результаты реализации схем

Основные результаты реализации схем 1 и 2 при их разрядности 7–10 бит показаны в табл. 3–5. В табл. 3 представлена ресурсоемкость схем, выраженная в требуемом количестве основных ресурсов ПЛИС – 4-входовых табличных преобразователей (LUT4) и содержащих их блоков программируемой логики (configurable function unit, CFU). Видно, что схема 2 требует на 27–86% большего количества LUT4 по сравнению со схемой 1. Возрастание разницы в ресурсоемкости двух схем с повышением их разрядности связано с тем, что бóльшая часть задействованных в схемах ресурсов используется вычислителями $s_i(|x|)$, $sm_i(x)$ и $sp_i(x)$. Булевы функции, реализуемые ими, с повышением разрядности усложняются и требуют все большего количества ресурсов.

Таблица 3

**Ресурсоемкость схем 1 и 2**

| Разрядность, бит | Схема 1 (рис. 2) | | Схема 2 (рис. 3) | |
|---|---|---|---|---|
| | CFU | LUT4 | CFU | LUT4 |
| 7 | 8 | 59 | 10 | 75 |
| 8 | 13 | 98 | 19 | 148 |
| 9 | 24 | 189 | 41 | 326 |
| 10 | 52 | 414 | 97 | 771 |

Таблица 4

**Временны́е характеристики схем 1 и 2**

| Разрядность, бит | Схема 1 (рис. 2) | | Схема 2 (рис. 3) | |
|---|---|---|---|---|
| | $F_{max}$, МГц | $t_{кр}$, нс | $F_{max}$, МГц | $t_{кр}$, нс |
| 7 | 80 | 10,005 | 155 | 2,893 |
| 8 | 69 | 11,03 | 130 | 3,723 |
| 9 | 66 | 12,948 | 108 | 4,748 |
| 10 | 58 | 15,171 | 80 | 6,924 |

Таблица 5

**Время, затрачиваемое вспомогательным блоком 1 схемы 1 на вычисление $|x|$**

| Разрядность, бит | Автоматическое размещение | | | Ручное размещение (LUT цепей переноса организованы в столбцы и расположены строго по порядку) | | |
|---|---|---|---|---|---|---|
| | Элементы, нс | Соед., нс | Сумма, нс | Элементы, нс | Соед., нс | Сумма, нс |
| 7 | 4,444 | 1,168 | 5,612 | 4,152 | 0,033 | 4,186 |
| 8 | 4,999 | 0,959 | 5,958 | 4,893 | 0,036 | 4,929 |
| 9 | 5,564 | 0,454 | 6,018 | 5,368 | 0,261 | 5,629 |
| 10 | 5,842 | 2,091 | 7,933 | 5,584 | 0,267 | 5,851 |

В табл. 4 для схем 1 и 2 показаны максимальная тактовая частота $F_{max}$ и время $t_{кр}$, затрачиваемое на вычисление $s(x)$. Время $t_{кр}$ включает в себя время работы всех находящихся на критическом пути логических элементов и задержки распространения сигналов по соединениям между ними. Из табл. 4 видно, что при разрядностях 7 и 8 бит переход от схемы 1 к схеме 2 повышает $F_{max}$ почти в два раза, но при большей разрядности этот выигрыш уже не так велик. Это объясняется тем, что при разрядностях 7 и 8 бит вычислители младших разрядов значений $s(|x|)$ в схеме 1 относительно просты, и их вклад в $t_{кр}$ невелик по сравнению с вкладом вспомогательных блоков. Например, у 7-разрядной схемы 1 время вычисления $s_7(|x|)$ составляет 1,775 нс, а время работы вспомогательных блоков 1 и 2 – как минимум 4,186 и 3,42 нс соответственно. С повышением разрядности схем 1 и 2 вычислители $s_i(|x|)$, $sm_i(x)$ и $sp_i(x)$ усложняются, и время вычислений в них приближается к времени работы вспомогательных блоков. Тем не менее даже при разрядности 10 бит $t_{кр}$ в большей степени определяется вспомогательными блоками, и переход от схемы 1 к схеме 2 дает выигрыш по $F_{max}$ 37%.

Из табл. 4 следует, что переход от схемы 1 к схеме 2 дает больший выигрыш по $t_{кр}$, чем по $F_{max}$, и с повышением разрядности схем это все более заметно. Это объясняется тем, что для расчета $F_{max}$ входные и выходные сигналы схем 1 и 2 синхронизируются. Пути распространения сигналов, в том числе используемые в расчетах критические пути, при этом дополняются соединениями схем с входными и выходными синхронизирующими регистрами. В случае перехода от схемы 1 к схеме 2 площадь, занимаемая схемой, увеличивается до 86%, поэтому соединения регистров с элементами схемы 2 являются более длинными.

Проектируемое устройство может быть размещено на ПЛИС автоматически, т.е. средой проектирования, или вручную. При ручном размещении проектируемого устройства место каждого его элемента выбирает разработчик. Ручное размещение может быть частичным, например выполняться только для критических путей. В табл. 4 для схемы 1 приведены параметры, полученные при ручном

размещении элементов (LUT) ее вспомогательных блоков (см. рис. 2). При автоматическом размещении, выполняемом средой проектирования, LUT располагаются хоть и недалеко друг от друга, но в произвольном порядке, что удлиняет критический путь. При ручном размещении LUT цепей переноса, образующие критический путь, были организованы в столбцы и расположены строго по порядку, что позволило минимизировать длину соединений между LUT и сократить критический путь.

В табл. 5 для примера приведены составляющие времени вычисления $|x|$ вспомогательным блоком 1 схемы 1 при автоматическом и ручном размещении элементов (LUT) критического пути, а именно: время работы элементов (Элементы, нс), задержка распространения сигналов по соединениям между элементами (Соед., нс) и сумма этих составляющих (Сумма, нс). Видно, что за счет ручного размещения сокращение времени вычисления $|x|$ может составить до 25%. Использование ручного размещения в обоих вспомогательных блоках схемы 1 позволяет сократить ее $t_{кр}$ на 5–17% по сравнению с результатами автоматического размещения. При этом ручное размещение вычислителей $s_i(|x|)$ не выполнялось: они не имеют единственного пути распространения сигналов значительно длиннее остальных, который следовало бы оптимизировать. Несколько путей могут иметь небольшие отличия во времени распространения, и укорочение одного пути при оптимизации может дать удлинение другого на примерно ту же величину.

## Заключение

ПЛИС используются для реализации нейронных сетей при необходимости их ускорения или работы в реальном времени, и сохранение высокой скорости вычислений является в этих случаях актуальной задачей. Поэтому, если нет дефицита ресурсов, для вычисления сигмоидной функции активации следует выбирать схему 2, работающую методом поразрядного отображения с полным диапазоном аргумента функции. Схема 1, использующая симметрию функции сигмоида, проигрывает в скорости схеме 2 из-за двух цепей переноса в составе вспомогательных блоков, образующих совместно с вычислителем младшего разряда двоичного представления значения функции сигмоида длинный критический путь. Ручное размещение на кристалле элементов вспомогательных блоков, принадлежащих критическому пути, строго по порядку позволяет сократить время работы вспомогательных блоков максимум на 25% по сравнению результатом их автоматического размещения; при этом общее время вычисления сигмоида схемой 1 снижается на 5–17%. Однако даже при использовании ручного размещения элементов вспомогательных блоков схемы 1 она расходует на вычисления как минимум в два раза больше времени по сравнению со схемой 2.

### Список источников

1. Nurvitadhi E., Venkatesh G., Sim J., Marr D., Huang R., Ong Gee Hock J., Boudoukh G. Can FPGAs beat GPUs in accelerating next-generation deep neural networks? // Proc. of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'17), 22–24 February, Monterey, CA, USA. 2017. P. 5–14.
2. Шашев Д.В., Шатравин В.В. Реализация сигмоидной функции активации с помощью концепции перестраиваемых вычислительных сред // Вестник Томского государственного университета. Управление, вычислительная техника и информатика. 2022. № 61. С. 117–127. doi: 10.17223/19988605/61/12
3. Шипицин С.П., Ямаев М.И. Развитие аппаратно-ориентированных нейронных сетей на FPGA и ASIC // Вестник Пермского национального исследовательского политехнического университета. Электротехника, информационные технологии, системы управления. 2019. № 31. С. 177–192.
4. Tommiska M.T. Efficient digital implementation of the sigmoid function for reprogrammable logic // IEE Proceedings-Computers and Digital Techniques. 2003. V. 150 (6). P. 403–411. doi: 10.1049/ip-cdt:20030965
5. Ушенина И.В. Реализация на современных ПЛИС вычислителя сигмоидной функции активации нейронных сетей табличным методом // Вестник Томского государственного университета. Управление, вычислительная техника и информатика. 2024. № 69. С. 124–133. doi: 10.17223/19988605/69/13
6. Li X.J., Li L. IP core based hardware implementation of multi-layer perceptrons on FPGAs: a parallel approach // Advanced Materials Research. 2012. V. 433. P. 5647–5653. doi: 10.4028/www.scientific.net/AMR.433-440.56471

7. Yang T., Wei Y., Tu Z., Zeng H., Kinsy M.A., Zheng N., Ren P. Design Space Exploration of Neural Network Activation Function Circuits // *I*EEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. 2019. V. 38 (10). P. 1974–1978. doi: 10.1109/TCAD.2018.2871198
8. Rajput G., Raut G., Chandra M., Vishvakarma S.K. VLSI implementation of transcendental function hyperbolic tangent for deep neural network accelerators // Microprocessors and Microsystems. 2021. V. 84. Art. 104270. doi: 10.1016/j.micpro.2021.104270
9. Saranya S., Elango B. Implementation of PWL and LUT based approximation for hyperbolic tangent activation function in VLSI // Proc. of 2014 International Conference on Communication and Signal Processing, 03–05 April, Melmaruvathur, India. 2014. P. 1778–1782.
10. Omondi A.R., Rajapakse J.C. FPGA implementations of neural networks. New York : Springer, 2006.
11. Thasnimol V.S., George M.A. Hardware Accelerator Implementation of Multilayer Perceptron // Proc. of Congress on Intelligent Systems (CIS 2020). 2021. V. 1. P. 107–119.
12. Vu H.M., Thang H.V. A Customized Hardware Architecture for Multi-layer Artificial Neural Networks on FPGA // Proc. of Fourth International Conference on Information Systems Design and Intelligent Applications, India. 2018. P. 637–644.
13. Holt J.L., Hwang J.N. Finite precision error analysis of neural network hardware implementations // IEEE Transactions on Computers. 1993. V. 42 (3). P. 281–290. doi: 10.1109/12.210171
14. GW2AR series of FPGA Products : Data Sheet. URL: https://cdn.gowinsemi.com.cn/DS226E.pdf (accessed: 10.09.2024).
15. Configurable Function Unit (CFU) : User Guide. URL: https://www.gowinsemi.com/upload/database_doc/1777/document/62e351486e153.pdf (accessed: 10.09.2024).

**References**

1. Nurvitadhi, E., Venkatesh, G., Sim, J., Marr, D., Huang, R., Ong Gee Hock, J. & Boudoukh, G. (2017) Can FPGAs beat GPUs in accelerating next-generation deep neural networks? *Proceedings of the 2017 ACM/SIGDA international symposium on field-programmable gate arrays (FPGA'17).* 22–24 February. Monterey California. USA. pp. 5–14.
2. Shashev, D.V. & Shatravin, V.V. (2022) Implementation of the sigmoid activation function using the reconfigurable computing environments. *Vestnik Tomskogo gosudarstvennogo universiteta. Upravlenie, vychislitel'naya tekhnika i informatika – Tomsk State University Journal of Control and Computer Science*. 61. pp. 117–127. DOI: 10.17223/19988605/61/12
3. Shipitsin, S.P. & Iamaev, M.I. (2019) Hardware neural networks progress on FPGA and ASIC. *Vestnik Permskogo natsional'nogo issledovatel'skogo politekhnicheskogo universiteta. Elektrotekhnika, informatsionnye tekhnologii, sistemy upravleniya*. 31. pp. 177–192.
4. Tommiska, M.T. (2003) Efficient digital implementation of the sigmoid function for reprogrammable logic. *IEE Proceedings-Computers and Digital Techniques*. 150(6). pp. 403–411. DOI: 10.1049/ip-cdt:20030965
5. Ushenina, I.V. (2024) Implementation of the sigmoid activation function for neural networks on current FPGAs by the table-driven method. *Vestnik Tomskogo gosudarstvennogo universiteta. Upravlenie, vychislitel'naya tekhnika i informatika – Tomsk State University Journal of Control and Computer Science*. 69. pp. 124–133. DOI: 10.17223/19988605/69/13
6. Li, X.J. & Li, L. (2012) IP core-based hardware implementation of multi-layer perceptrons on FPGAs: a parallel approach. *Advanced Materials Research*. 433. pp. 5647–5653. DOI: 10.4028/www.scientific.net/AMR.433-440.5647
7. Yang, T., Wei, Y., Tu, Z., Zeng, H., Kinsy, M.A., Zheng, N. & Ren, P. (2019) Design Space Exploration of Neural Network Activation Function Circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*. 38(10). pp. 1974–1978. DOI: 10.1109/TCAD.2018.2871198.
8. Rajput, G., Raut, G., Chandra, M., & Vishvakarma, S.K. (2021) VLSI implementation of transcendental function hyperbolic tangent for deep neural network accelerators. *Microprocessors and Microsystems*. 84. pp. 104270. DOI: 10.1016/j.micpro.2021.104270
9. Saranya S. & Elango B. (2014) Implementation of PWL and LUT based approximation for hyperbolic tangent activation function in VLSI. *Proceedings of 2014 International Conference on Communication and Signal Processing*. 03-05 April. Melmaruvathur. India. pp. 1778–1782.
10. Omondi, A.R. & Rajapakse, J.C. (2006) *FPGA implementations of neural networks*. New York: Springer.
11. Thasnimol, V.S. & George, M. (2021) A Hardware Accelerator Implementation of Multilayer Perceptron. *Proceedings of Congress on Intelligent Systems (CIS 2020).* 1. pp. 107–119.
12. Vu, H.M. & Thang, H.V. (2018). A Customized Hardware Architecture for Multi-layer Artificial Neural Networks on FPGA. *Proceedings of Fourth International Conference on Information Systems Design and Intelligent Applications.* India. pp. 637–644.
13. Holt, J.L. & Hwang, J.N. (1993) Finite precision error analysis of neural network hardware implementations. *IEEE Transactions on Computers*. 42(3). pp. 281–290. DOI: 10.1109/12.210171
14. *GW2AR series of FPGA Products*. [Online] Available from: https://cdn.gowinsemi.com.cn/DS226E.pdf. (Accessed: 10th September 2024).
15. *Configurable Function Unit (CFU) User Guide*. [Online] Available from: https://www.gowinsemi.com/upload/database_doc/1777/document/62e351486e153.pdf. (Accessed: 10th September 2024).

***Информация об авторе:***
**Ушенина Инна Владимировна** – доцент, кандидат технических наук, доцент кафедры «Программирование» Пензенского государственного технологического университета (Пенза, Россия). E-mail: ivl23@yandex.ru

***Автор заявляет об отсутствии конфликта интересов.***

***Information about the author:***
**Ushenina Inna V.** (Candidate of Technical Sciences, Associate Professor, Penza State Technological University, Penza, Russian Federation). E-mail: ivl23@yandex.ru.

***The author declares no conflicts of interests.***

*Поступила в редакцию 07.10.2024; принята к публикации 02.06.2025*

*Received 07.10.2024; accepted for publication 02.06.2025*