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

Факультет компьютерных систем и сетей
Кафедра электронных вычислительных машин
Дисциплина: Схемотехника

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
на тему

ВЫЧИСЛИТЕЛЬ КВАДРАТНОГО КОРНЯ НА ОСНОВЕ ПЛИС

БГУИР КП 1-40 02 01 311 ПЗ

Студент: гр. 050503 Деруго Д. В.

Руководитель: ассистент каф. ЭВМ
Жук Д.С.

Минск 2022

Оглавление

ВВЕДЕНИЕ

Вычислительная техника используется повсеместно уже не первый десяток лет, от элементарных детских игрушек и компьютеров с процессорами общего назначения до станков с ЧПУ и узкоспециализированных схем для самых требовательных к вычислительной мощности задач. Возникает все больше проблем, требующих обработки информации, и в то время как старые задачи нашли приемлемое на данный момент решение, нынешние тенденции развития науки и техники бросают вызов не только программному, но и аппаратному обеспечению. На слуху у инженеров вычислительной техники масса инновационных решений в сфере «железа», например, нейропроцессоры и продвинутые модули цифровой обработки сигналов.

Алгоритмы обработки данных продвинулись далеко вперед, но могут ли они обеспечить достаточный результат на неэффективном аппаратном обеспечении? Безусловно, можно улучшить старые методы, выпустив новую линейку процессоров, но можно пойти другим путем, например, в сторону параллельных вычислений. Однако заказывать отдельные схемы для задач узкого спектра было бы крайне накладно. Для этих пограничных задач применяются схемы программируемой логики.

ПЛИС — программируемая логическая интегральная схема — является в достаточной мере эффективным средством решения задач специализированных вычислений. Ее структуру мы изучаем в текущем курсе схемотехники. Помимо самого чипа с программируемой логикой и массы электронных компонентов, служащих для корректной работы чипа, отладочная плата содержит индикаторы и множество портов различного назначения, некоторые из которых будут использоваться в данном курсовом проекте.

Проект представляет собой вычислитель квадратного корня. Загрузка и получение значений будет производиться через устаревший, но очень подходящий для учебных целей протокол RS-232, а обработка данных — в разработанном блоке, который и является самой сутью курсового проекта. В ходе работы будут применены знания как нынешнего курса схемотехники, так и навыки, приобретенные на других дисциплинах.

1 ОБЗОР ЛИТЕРАТУРЫ

1.1 Используемые алгоритмы

В этом подразделе рассмотрим несколько алгоритмов вычисления квадратного корня, подходящие для реализации на аппаратной основе. Безусловно, все вычисления так или иначе проходят через соответствующую аппаратуру, однако проводить расчеты с помощью методов, основанных на экспоненциальной функции или ряде Тейлора, намного проще на процессоре, чем на специальной схеме.

Многие итерационные алгоритмы требуют начального значения. Очевидно, это значение должно быть больше единицы для целого числа и как можно ближе к значению самого корня числа, чтобы для его вычисления на основе оценки потребовалось меньше итераций. Поэтому выгодно иметь первоначальную оценку, что ускоряет сходимость ряда, выстраивающегося в ходе вычислений.

Так, для аппаратного воплощения был выбран итеративный вавилонский метод, или же метод Герона, с предшествующим ему вычислением двоичной оценки.

Двоичная оценка имеет достаточную точность и дает сбой только с малыми целыми и дробными значениями, например, корень числа 0.5 такой метод оценивает в 0.5, что мало чем полезно для последующих вычислений. Пусть S — число, корень которого требуется. Представим его в двоичном виде так:

$$S = m \cdot 2^{2n}. \tag{1.1}$$

Тогда корень числа будет выглядеть так:

$$\sqrt{S} \approx (0.5 + 0.5 \cdot m) \cdot 2^{n}. \tag{1.2}$$

Значения, вычисленные на этом этапе, перейдут во второй подблок вычислителя, то есть в блок вавилонского метода.

Формула для вавилонского метода выглядит следующим образом:

$$x_{n+1} = 0.5 \cdot \left(x_n + \frac{S}{x_n}\right). \tag{1.3}$$

Здесь x_n означают значение корня, приближающегося к своему реальному значению с каждой итерацией.

Как видно из формул, реализация алгоритмов потребует проектирования логики с использованием сдвиговых регистров, делителей и умножителей. Также требуется решить, на каком принципе будет построена логика итеративных операций.

1.2 Внутреннее устройство ПЛИС

ПЛИС — это программируемая логическая интегральная схема.

Строение схем с программируемой логикой в достаточной мере сложное, поэтому будет рассмотрена только требуемая основа.

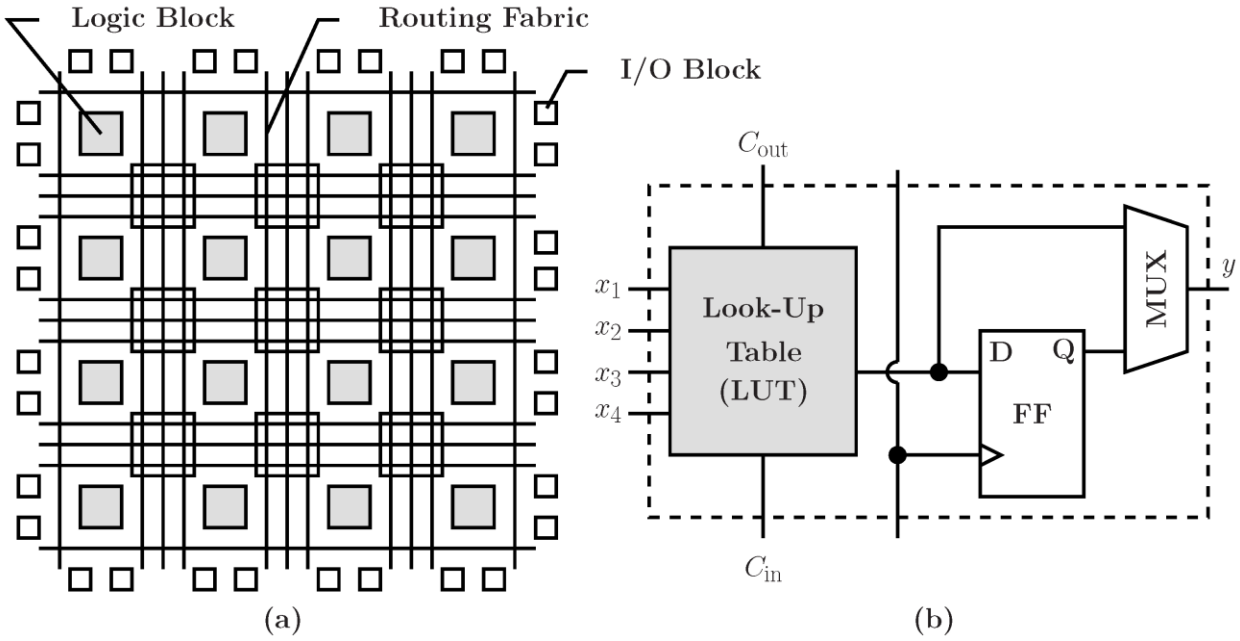

Рисунок 1.2.1 — устройство логической ячейки и массива ЛЯ

На рисунке 1.2.1 в упрощенном виде представлено внутреннее строение ППВМ. Помимо блоков ввода-вывода присутствуют программируемые логические ячейки и линии связи (коммутационное поле), которые их соединяют в нужном виде. В отличие от программируемых логических матриц или матриц логики, ПЛМ или ПМЛ, которые содержат массив элементарных логических элементов И-НЕ или ИЛИ-НЕ, в базисе которых можно отобразить любую логическую функцию, в ППВМ ячейки реализованы на небольшом объеме памяти, которая содержит таблицу истинности для требуемой функции.

Поскольку мало какой проект в наше время являет собой чисто комбинационную схему, в блок включен D-триггер, на основе которых строятся, например, сдвиговые регистры. Помимо этого, крупные схемы содержат блоки RAM, которые используются по решению разработчика или в ходе оптимизации схемы синтезатором кода, чтобы не расходовать на память ценные программируемые ячейки.

Память LUT строится на SRAM, то есть она энергозависима. Линии связи также основаны на программируемой логике. По этой причине прошивка стирается при выключении, но она может быть загружена во внешнюю энергонезависимую память, откуда будет загружаться обратно при запуске.

В зависимости от сложности своего исполнения, ПЛИС содержат некоторое количество готовых сдвиговых регистров или блоков умножения-суммирования, широко применяющихся в задачах цифровой обработки сигналов.

Такая гибкая архитектура позволяет создать требуемую схему без трудоемкого процесса разработки заказного чипа и еще более затратного его изготовления. ПЛИС стали известны обществу с набором популярности майнинга криптовалют, так как нужный алгоритм требовал серьезного распараллеливания вычислений, которое не могли предоставить даже графические процессоры GPU.

1.3 Протокол RS-232

Существует немало протоколов передачи данных. Многие из них используются в наши дни, многие уже устарели и их порты давно не предусматриваются в электронной технике. Тем не менее, RS-232, известный как COM-порт, оставил достаточный след в индустрии, чтобы все еще быть востребованным, хоть и активно вытесняемым современными интерфейсами. Этот протокол прост в аппаратной реализации, потому для данного курсового проекта был выбран именно он.

Протокол RS-232 является последовательным интерфейсом, то есть биты данных в нем передаются друг за другом. Самая элементарная посылка, без дополнительного кодирования, представлена на рисунке 1.3.1.

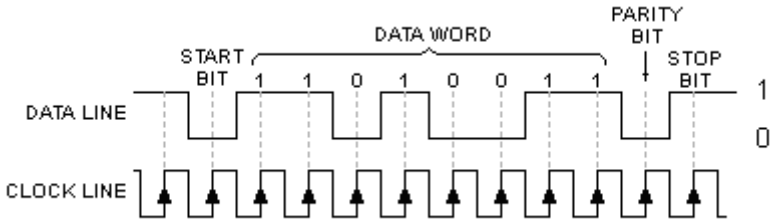

Рисунок 1.3.1 — посылка протокола RS-232

При отсутствии передачи линия содержит значение логической единицы. Начало передачи сигнализируется старт-битом в виде логического нуля. Стандарт предусматривает длину передаваемого слова в 5-9 бит без учета бита четности.

К недостаткам интерфейса можно отнести его низкую скорость передачи, так как в силу своего физического устройства скорость существенно падает с ростом расстояния между подключенными устройствами. Также наличие старт- и стоп-битов на каждый символ из 8 бит дает 20% лишней информации. К тому же коммутация осуществляется только между двумя устройствами.

Взвесив все за и против с учетом возможностей и требований к этому курсовому проекту можно сделать вывод, что протокол RS-232 будет самым удачным выбором.

2 РАЗРАБОТКА СТРУКТУРНОЙ СХЕМЫ

В этом разделе будет разработана структурная схема проекта, на основе которой будут составлены функциональная и принципиальная схемы. Необходимо выделить функции устройства и распределить по блокам схемы.

Структурная схема представлена в приложении А.

Структура проекта следующая:

- устройство приема и передачи, запрашивающее обработку данных
- аппаратура переходника интерфейсов USB-RS-232
- реализованная на ПЛИС логика

Реализованная на ПЛИС логика, входящая в структуру:

- модули приема и передачи данных по интерфейсу RS-232
- схема предварительной оценки
- схема точного расчета

2.1 Устройство приема и передачи

Для обработки данных на ПЛИС будет использоваться персональный компьютер с программой, осуществляющей передачу изначальных данных и прием обработанных значений. От ПК будет питаться и сама плата программируемой логики.

2.2 Аппаратура переходника интерфейсов USB-RS-232

Аппаратура интерфейсов представлена в виде USB-разъема ПК, кабеля приема-передачи со встроенным переходником USB-RS-232, поскольку современные ПК не имеют разъема для COM-порта, и портом интерфейса на самой отладочной плате ПЛИС.

2.3 Реализованные на ПЛИС модули

Поскольку процесс расчета и сборки схемы с ПЛИС достаточно трудоемкий и чреватый порчей аппаратуры, в проекте будет использоваться отладочная плата, содержащая на себе не только саму плату ПЛИС и схему ее питания, но и нужный в этой курсовой работе порт RS-232. Для загрузки прошивки будет задействован программатор JTAG.

2.3.1 Модули приема и передачи данных по интерфейсу RS-232

Значения для последующей обработки будут приниматься модулем приема и передаваться на схемы расчета. После получения значений, модуль передачи отправляет их обратно во внешнюю систему по тому же физическому каналу.

2.3.2 Схема предварительной оценки

В данной схеме происходит первичная обработка значений, переданных из модуля приема. В соответствии с модифицированным алгоритмом, более подробно рассмотренным в разделе 3, над числом с плавающей точкой в стандарте IEEE 754 производятся манипуляции, дающие довольно точную оценку квадратного корня.

2.3.3 Схема точного расчета

После предварительного расчета данные направляются в этот блок, организующий работу вавилонского метода. Метод предусматривает неоднократное исполнение одних и тех же действий, что реализуется последовательным соединением одинаковых блоков или использованием одного блока несколько раз. На выходе схемы формируется число в стандарте IEEE 754, которое передается в модуль передачи данных интерфейса RS-232.

3 РАЗРАБОТКА ФУНКЦИОНАЛЬНОЙ СХЕМЫ

Этот раздел посвящен разработке функциональных блоков данного курсового проекта. Необходимо организовать работу блоков, рассмотренных в структурной схеме, а именно разработать программу на языке программирования для организации приема-передачи с компьютера и спроектировать цифровые электронные схемы, реализующие требуемый функционал.

Начнем с рассмотрение с ключевых функциональных схем проекта. При этом будет приведен весь путь разработки до финальной версии.

Функциональная схема проекта представлена в приложении Б.

3.1 Выбор формата представления чисел

Первоначальной задумкой было организовать работу устройства на целых числах, занимающих 2 байта. Протокол RS-232 передавал бы эти числа за две посылки, по принятию которых сформированное число направлялось бы на входы схемы предварительной оценки. Кажущаяся простота такого представления оказалась менее выгодной, чем более сложно организованные числа с плавающей запятой.

Предполагалось, что числа не будут иметь бита знака за ненадобностью, поскольку единственная функция устройства — расчет квадратного корня. Таким образом число занимало бы 2 байта без знака, в программировании это был бы `unsigned short int`. Двух байт хватило бы для демонстрации функционала устройства, так как 128 значений в одном байте слишком мало, а в четырех слишком много.

Рисунок 3.1.1 — изначальная задумка формата чисел

Также была разработана оптимизация приема таких чисел. Поскольку каждое число занимало бы 2 байта независимо от своего значения, числа от 0 до 255 всегда имели бы второй байт, заполненный исключительно нулями. Протокол RS-232 предусматривает до 9 бит информации в посылке, который можно было бы использовать в качестве информации о последующем за пришедшим числе. Если число в диапазоне от 0 до 255, то этот бит сигнализировал бы о том, что далее идет не ненужный «нулевой» байт, а начало следующего числа. Для оставшихся чисел бит не играл роли, так как следующий байт все же нужен для правильного представления числа. Однако от этой идеи было

решено отказаться по следующей причине: каждый байт содержал бы лишний бит информации, а выигрыша времени в перспективе не было. Допустим, что частота появления чисел возможного диапазона равномерна. В таком случае 256 чисел «выигрывали» бы 9 бит информации, но оставшиеся 65279 чисел содержали бы такой бит без надобности. 2304 бита «выигрыша» явно не шли в сравнение с потерями. Безусловно, можно было бы организовать использование этого бита с этими же целями другим образом, но это приводило бы к усложнению устройства управления модуля приемника уже на самой ПЛИС.

Тем не менее все еще вставал веский вопрос об остатке вычисленного корня, который наверняка у него будет, и если оставить формат в таком виде, то пришлось бы остаток не учитывать. Это серьезно сказывалось на точности расчета устройства, вследствие чего было решено разделить два байта на байт целой и байт дробной части, то есть организовать число с фиксированной точкой.

Из этого вытекала другая проблема: как именно организовать переход от одного вида числа к другому, учитывая тот факт, что последующая обработка будет вестись с применением значения степени числа. Схема предварительной оценки требовала значения экспоненты для корректной работы, а в таком случае пришлось бы рассчитывать степень числа в самой схеме. Неизвестно, сколько бы тактов пришлось бы потратить на такой расчет. Также такая схема имела бы немалое количество исключений, что потребовало бы траты логических ресурсов на усложнение управления, создавая угрозу появления новых ошибок.

На этой почве стандарт IEEE 754 смотрелся все более предпочтительным, а именно формат half-precision в 16 бит. Хотя он несколько проигрывал в точности представления целых чисел и имел лишний в этом случае бит для знака, такой формат сразу имел степень числа в своей записи и имел возможность работать с дробями.

Формат IEEE 754 half-precision состоит из 16 бит, где старший бит является битом знака, 5 после него кодируют степень числа со смещением 15, последние 10 содержат мантиссу с мнимой единицей в старшем бите, то есть мантисса состоит из одиннадцати бит. Логика должна будет предусматривать обработку исключительных значений: $\pm\infty$, NaN, денормализованные числа.

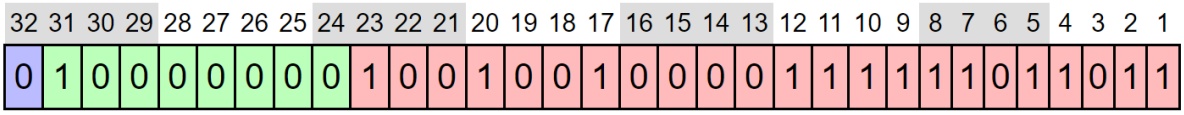

Рисунок 3.1.2 — формат чисел IEEE 754 single-precision. Число π

Тем не менее, в процессе разработки устройства было принято решение перейти на формат single-precision, рисунок 3.1.2. Полноценный float используется гораздо чаще, чем short float, который наоборот является редким исключением в программировании. К тому же формат большей точности даст более

наглядные результаты работы устройства. Single-precision отличается только размером поля экспоненты в 8 бит со смещением в 127 и поля мантиссы в 23 бита.

3.2 Разработка схемы предварительной оценки

Рассмотрим требуемый к реализации алгоритм подробнее.

Пусть S — число, корень которого требуется. Представим его в двоичном виде так:

$$S = m \cdot 2^{2n}\,. \tag{3.1}$$

Тогда корень числа будет выглядеть так:

$$\sqrt{S} \approx (0.5 + 0.5 \cdot m) \cdot 2^{n}. \tag{3.2}$$

В двоичном виде умножение на 0.5, то есть деление на 2, легко реализовать сдвигом числа в сторону младших разрядов, а 0.5 для суммирования в этом случае представляются лишь одним разрядом двоичного числа. При смещении вправо на место старшего разряда мантиссы встает всегда, в нормальном представлении числа, присутствующая единица, которая тут же суммируется с 0.1 и переходит обратно на мнимый разряд. Таким образом, изменение мантиссы состоит только в том, чтобы сдвинуть ее на один разряд вправо.

Также для реализации алгоритма требуется сместить значение экспоненты на один бит вправо с учетом того, что при смещении старшей единицы значение степени поменяет свой знак. Таким образом, смещение нужно проводить только для четырех младших бит, что приведет к желаемому результату, но только для положительных значений степени. При смещении отрицательного значения степень станет еще меньше, чем была до этого. То есть для положительных значений второй по старшинству бит должен принимать значение 0, а для отрицательных — 1.

Пример: 10000110 ⇒ 10000011, 01110101 ⇒ 01111010. В первом случае значение степени 7 стало 4, во втором -10 перешло в -5. Но есть исключение для степени со значением $2^{1}{}_{10}$, рассмотренное далее.

Существенным минусом этого алгоритма стал тот факт, что при смещении нечетных значений степени результат автоматически округляется в большую сторону, что видно на примере выше. Степень 7 должна была стать степенью 3.5, но приняла значение 4, что существенно влияет на результат оценки всей схемы. При четных значениях степени отклонение результата обычно в пределах 2-3% от истинного числа, при нечетных доходит до 40 и более процентов. Очевидно, что такая погрешность сильно затормозит последующие вычисления, требуя большее количество итераций. Но эту проблему удалось решить в ходе разработки блока.

Итак, при сдвиге нечетной степени всегда случается завышение значения оценки корня. В попытках как-либо усовершенствовать алгоритм оказалось, что очень действенным способом решения проблемы будет уменьшение значения степени на единицу и увеличение значения мантиссы на 0.5. Особенно радует тот факт, что оценка с такой модификацией так же точна, что и

оценка при четном значении экспоненты, и они обе зачастую вносят погрешность лишь в третий знак после запятой в десятичной форме. Это преобразование можно обосновать математически:

$$2^{2n+1} \cdot m \rightarrow 2^{n+\frac{1}{2}} \cdot \left(\frac{m}{2} + \frac{1}{2}\right) = 2^{n-\frac{1}{2}} \cdot (m+1), \quad (3.3)$$

$$2^{2n+1} \cdot m \rightarrow 2^{n+1} \cdot \left(\frac{m}{2} + \frac{1}{2}\right) = 2^{n} \cdot (m+1), \quad (3.4)$$

$$2^{2n+1} \cdot m \rightarrow 2^{n+1-1} \cdot \left(\frac{m}{2} + \frac{1}{2} + \frac{1}{2}\right) = 2^{n-1} \cdot (m+2), \quad (3.5)$$

где *2n + 1* означает нечетное значение степени, *m* — мантиссу, значение которой всегда лежит в диапазоне от 1 до 2, так как ее уменьшение или увеличение влечет изменение значения степени.

Формула (3.3) является формулой самого алгоритма, это то выражение, к которому должна стремиться реализация. Формула (3.4) — это то, что получается без модификации для нечетных степеней, как видно итоговое значение степени всегда завышается на 0.5, внося большую погрешность. Формула (3.5) есть модифицированная формула (3.4), где отнимается единица от степени и прибавляется 0.5 к мантиссе.

Следует отметить, что такая манипуляция с мантиссой является не единственным вариантом из возможных. В ходе разработки были опробованы разные варианты, например, инверсия всех битов кроме старшего не мнимого. Но эти формулы не показали той же эффективности, как представленная. Также можно вычислить лучший вариант модификации мантиссы, если приравнять (3.3) и (3.5), где *m + 2* заменить на *m + 1 + x,* и искомое значение *x* будет лучшим слагаемым к мантиссе. Расчет произведен, формула *x* приведена ниже:

$$x = \left(\sqrt{2} - 1\right)(m+1). \quad (3.6)$$

Как видно из формулы, искомое зависит от значения мантиссы и требует умножения на константу, недостижимого с помощью сдвига степени. Было принято решение отказаться от реализации такой функции, несмотря на то, что она в точности позволяла повторить оригинальный алгоритм. В приложении Г представлен график трех функций, наглядно демонстрирующий эффективность приведенного способа модификации.

На языке Verilog данную схему можно реализовать как приведено ниже. Для четных значений степени:

```
sqrt_sign <= in_sign;
sqrt_exponent <= {in_exponent[4], ~in_exponent[4],
                  in_exponent[2:0] >> 1};
sqrt_mantissa <= {in_mantissa[9:0] >> 1};
```

Для нечетных значений:

```
sqrt_sign <= in_sign;
sqrt_exponent <= {in_exponent[4], ~in_exponent[4],
                  {in_exponent[2:0] >> 1} - 1'b1};
trunc_mantissa[9:0] = in_mantissa[9:0] >> 1;
sqrt_mantissa <= {1'b1, trunc_mantissa[8:0]};
```

Реализация нечетных значений потребовала создания промежуточного провода для корректного смещения мантиссы. Декларации `in_*` — входные сигналы, `sqrt_*` — выходы схемы предварительного расчета типа `reg`.

Обработка некорректных значений выполняется следующим образом: на регистр ошибки `incorrect` выставляется единица, в регистры выхода пришедшие значения записываются без изменений.

Присутствуют два исключения для нормализованного числа. Если степень числа $2^1{}_{10}$, то есть равна 10000000, то степень преобразуется в 01111111, иначе оценка будет ошибочна. Помимо этого, число 0_{10} также обрабатывается отдельно, несмотря на то, что логика обработки нормализованного числа сработает корректно, но внесет погрешность.

3.3 Разработка схемы точного расчета

В этом подразделе рассматривается схема итеративного метода, улучшающая точность расчета.

Рассмотрим упомянутый ранее вавилонский метод.

$$x_{n+1} = 0.5 \cdot \left(x_n + \frac{S}{x_n}\right). \quad (3.7)$$

Приведенная формула является частным случаем метода Ньютона, а именно случаем $f(x) = x^2 - S$, где S — число, корень которого требуется найти. В англоязычных источниках этот метод обычно называют методом Ньютона-Рафсона.

Умножение на 0.5 есть деление на 2, то есть простой сдвиг числа в сторону менее значимых битов, а в случае с IEEE 754 уменьшение значения поля степени на 1.

Суммирование чисел с плавающей запятой осуществляется гораздо сложнее суммирования чисел с фиксированной запятой, поскольку усложняется логика работы с нормализацией чисел, появляется задача обработки поля экспоненты. Однако и эта формула случай исключительный: число S делится на приближение корня самого себя, притом это приближение достаточно точно, чтобы можно было утверждать, что x_n и S/x_n будут числами одного порядка. По сути эта сумма является умножением на 2, после которого тут же происходит деление на 2. Исходя из этого можно немного сэкономить на логике работы со степенью числа, так как ее значение будет сохраняться с момента вычисления приблизительной оценки, то есть с выхода первой схемы. Тем не менее логика нормализации остается, но и она существенно упрощается: мантиссы суммируются и результат сдвигается вправо на один бит.

Операцию деления было решено реализовывать через умножение на обратное число, для чего требуется реализовать его поиск, что не является тривиальной задачей. Во всей изученной в процессе разработки литературе по теме аппаратных вычислений рекомендуется использовать единственный метод — уже упомянутый метод Ньютона. Разница состоит в том, что функция

уже другая, $f(x) = \frac{1}{x} - D$, где D — число, обратное значение которого требуется найти. Сама же итерационная формула имеет следующий вид:

$$x_{n+1} = x_n \cdot (2 - D \cdot x_n). \quad (3.8)$$

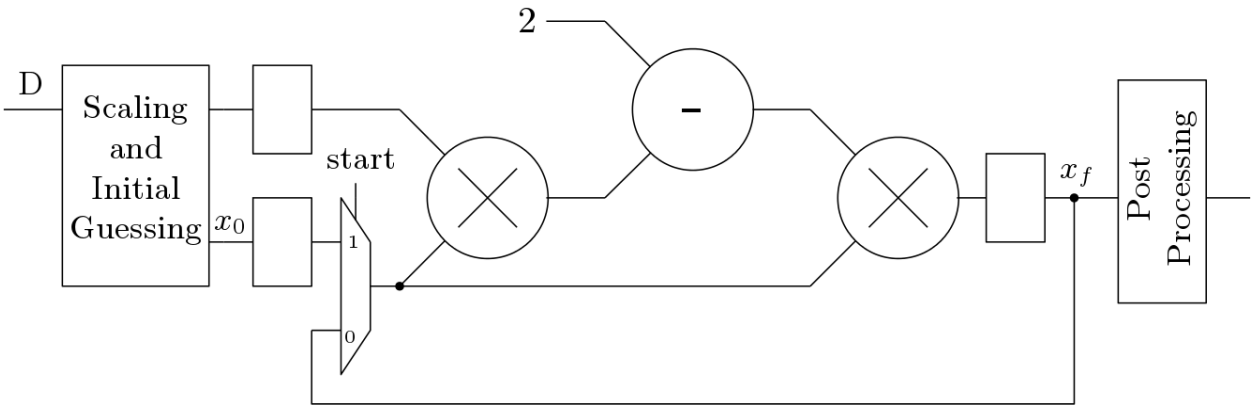

Рисунок 3.3.1 — схема расчета обратного числа

Как и для формулы вавилонского метода, этот метод требует первоначальной оценки для ускорения поиска наиболее верного приближенного значения. Проблема заключается в том, что найти аппроксимирующую функцию для всего диапазона функции *1/x* невозможно в силу ее нелинейности, однако можно найти подходящую функцию в определенном интервале. Литературные источники рекомендуют использовать следующую формулу:

$$x_0 = \frac{48}{17} - \frac{32}{17} \cdot D, \quad (3.9)$$

при этом число *D* должно лежать в интервале [0.5;1], так как именно для этого интервала аппроксимация возымеет наилучшую результативность. В приложении Г приведен график истинной функции и ее линейной аппроксимации.

Итак, требуется привести число к такому интервалу. Очевидно, что значение в нужном интервале [0.5;1] любое число с плавающей точкой будет иметь только со значением экспоненты 2^{-1}, а результат поиска обратного числа, соответственно, будет всегда иметь степень 2^0, то есть само число будет лежать в диапазоне [1;2]. При этом нужно учесть значение, «занятое» у приведенного числа, так как чем степень числа больше, тем меньше его обратное значение. В последствии «занятую» степень нужно будет отнять от результата поиска обратного числа и отнять еще единицу.

Пример: пусть число равно 50, обратное значение 0.02. При изменении степени до 2^{-1} значение 50 станет 0.78125, при этом запомним старое значение степени 2^5. После первого приближения и нескольких итераций метода Ньютона получим обратное для 0.78125 число 1.28. Отняв от степени этого числа 5 + 1 = 6, получим искомое 0.02.

Также была обнаружена полезная закономерность. Старый алгоритм искал разность степени числа и степени 2^{-1}, после чего отнимал от значения степени 2^0 разницу. Такие действия требовали задействования двух сумматоров, оба из которых искали разность, то есть требовалось еще два преобразователя в дополнительный код. Оптимизация заключается в том, что требуется найти только разность первоначальной степени и 2^{-1}, а итоговая степень числа будет результатом проведения операции исключающего ИЛИ, XOR, над результатом разности и 2^0. Следует привести пример.

Пусть некоторое число имеет степень 9, в двоичном виде со смещением 127 это 10001000. Разность со степенью -1, 01111110, будет составлять 10, то есть 00001010. Старый алгоритм требовал перевода 10 в дополнительный код и суммирования результата со значением степени 0, то есть 01111111. Теперь требуется только применить XOR: 00001010 $\oplus$ 01111111 = 01110101, и результат равен -10.

Еще одна оптимизация: сумматор с 48/17 не обязательно делать полноценным сумматором чисел с плавающей запятой. Число D всегда лежит в интервале [0.5;1], а значит известен интервал выходных значений формулы (3.9). Степень выходного числа может быть либо 2^{-1}, либо 2^0, что можно реализовать на мультиплексоре, выбирающем значение в зависимости от какой-то функции. Степень зависит от того, будет результат формулы (3.9) больше или меньше единицы, а это зависит от числа D, конкретно от четырех старших бит. При D = 0.96875 четыре старших бита мантиссы равны 1, остальные 0. Если все 4 бита единицы, то выход функции меньше или равен 1, степень 2^{-1}, исключение только само значение 1 со степенью 2^0, в любом другом случае выход больше единицы и степень 2^0.

Насчет логических оптимизаций есть следующий вопрос: будут ли они полезны при логическом синтезе для программируемой логики типа FPGA? Та же оптимизация с использованием XOR вместо сумматора определенно потребует меньшее количество логических элементов малой степени интеграции или сэкономит ячейки в CPLD. Но программное обеспечение синтезатора для FPGA в теории может просто иначе задать функцию в LUT, то есть просто изменить записанные в память значения, само количество использованных ячеек от этого может не измениться вообще. Конкретно в случае с этой оптимизацией можно утверждать, что изменится как минимум требуемое на исполнение операции время, так как реализующим операцию XOR блокам не нужно дожидаться окончания работы предыдущего для получения информации о переносе.

В ходе разработки было решено использовать 2 итерации метода Ньютона для поиска обратного числа с предварительной оценкой, поскольку формат представления числа в 32 бита, single-precision, зачастую не может предоставить достаточную точность для записи результата трех и более итераций метода. Также оценка x_0 достаточно точная, чтобы получить ответ с хорошей точностью уже через 2 итерации.

Равно как и для схемы поиска обратного числа, для схемы точного расчета было выбрано 2 повторения алгоритма. Первичная оценка точная, а также точности представления не хватает, чтобы оправдать траты логики на большее количество итераций алгоритма.

3.4 Разработка схем приемника и передатчика RS-232

Описанные ранее схемы должны каким-либо образом получить значения для обработки. Для демонстрации работы схемы хватило бы записанных в RAM ПЛИС значений, однако такая система не имела бы никакого практического применения. По этой причине было принято решение организовать работу протокола передачи данных, а именно модули приема и передачи протокола RS-232.

Следует определиться со скоростью передачи. В программах работы с портами по умолчанию зачастую стоит значение в 9600 бод, то есть 9600 бит в секунду. Было решено оставить это значение, хотя даже при максимальной скорости в 115200 бод длина одного бита составляет примерно 1 микросекунду, что достаточно большой период для генератора тактовой частоты ПЛИС в 50 мегагерц, чтобы не доставить проблем с задержками при обработке. Для определения количества тактов генератора для ожидания передачи или приема одного бита требуется разделить 50 миллионов на 9600. Полученное значение в 5208.33 есть количество тактов в длине одного бита, именно до этого числа должен будет считать счетчик тактов. Дробное значение делителя не сможет повлиять на работу устройства, поскольку ошибка не успеет накопиться за время приемопередачи кадра.

Зачастую в ходе разработки цифровых устройств очень удобно использовать конечные автоматы. Конечный автомат — математическая абстракция, используемая для описания модели поведения дискретного устройства. Автомат конечный, поскольку имеет конечное число собственных состояний. Значения состояний хранятся в триггерах, состояния меняются в зависимости от входных сигналов и текущего состояния.

Конечный автомат как приемника, так и передатчика будет иметь 4 состояния: ожидание данных, старт-бит, прием или передача данных, стоп-бит. Несмотря на то, что в схемах задействованы использующие память счетчики для подсчета пришедших бит, оставшихся для передачи бит и тактов синхронизирующего сигнала, в состояниях конечного автомата их значения не учитываются, так как это сильно усложняло бы разработку автомата и модуля на его основе. Оптимизацию кодов автомата было решено не проводить по причине малого количества состояний, достаточно малой частоты работы устройства, чтобы это создавало проблемы с гонками данных, и в связи с тем фактом, что триггеры автомата непосредственно не принимают аналоговый сигнал, который может отправить триггер в метастабильное состояние. Диаграммы состояний автоматов приведены в приложении В.

Рассмотрим алгоритм передатчика. В состоянии ожидания линия передачи Tx в значении логической единицы. По пришествию сигнала о разрешении пересылки данных от другого модуля, Tx переключается в состояние нуля, отсылая старт-бит. После этого независимо от внешних сигналов управления автоматом схема побитово считывает данные из предоставленного ей регистра. Затем передатчик отправляет стоп-бит, по окончанию передачи которого снова переходит в состояние ожидания. Счетчики переданных битов и тактов синхросигнала обнуляются.

Алгоритм приемника немного отличается от алгоритма передатчика. По стандарту запись пришедшего бита следует осуществлять по его середине, так как в этом месте уровень напряжения на линии наиболее стабилен. Переход в состояние ожидания старт-бита происходит по получению нуля на линии, однако подтверждается он только через половину такта. Если к этому времени на линии сохранился ноль, то еще через половину произойдет переход к чтению данных, в ином случае будет выставлен бит ошибки. Также проводится проверка на корректный прием кадра: если значение уровня на линии к моменту считывания стоп-бита не равно логической единице, то считается, что кадр принят некорректно.

4 РАЗРАБОТКА ПРИНЦИПИАЛЬНОЙ СХЕМЫ

В этом разделе рассматривается разработка принципиальной схемы, а именно описывается элементная база, на основе которой выполнялось устройство.

В проекте используется микросхема преобразования интерфейса USB в интерфейс UART. Конструктивно она располагается в пластмассовом корпусе штекера COM-порта. Схема CH340G имеет частоту работы в 12 Мгц, требует 5 вольт питания, которые получает от USB, имеет поддержку дуплексного режима передачи. Существенным, но не влияющим на работу проекта, минусом является то, что амплитуда сигнала всего 5 вольт, от 0 до 5, в то время как реальный RS232 имеет уровни -12 и +12 вольт. Отладочная плата не требует таких уровней и распознает данные даже с такой амплитудой.

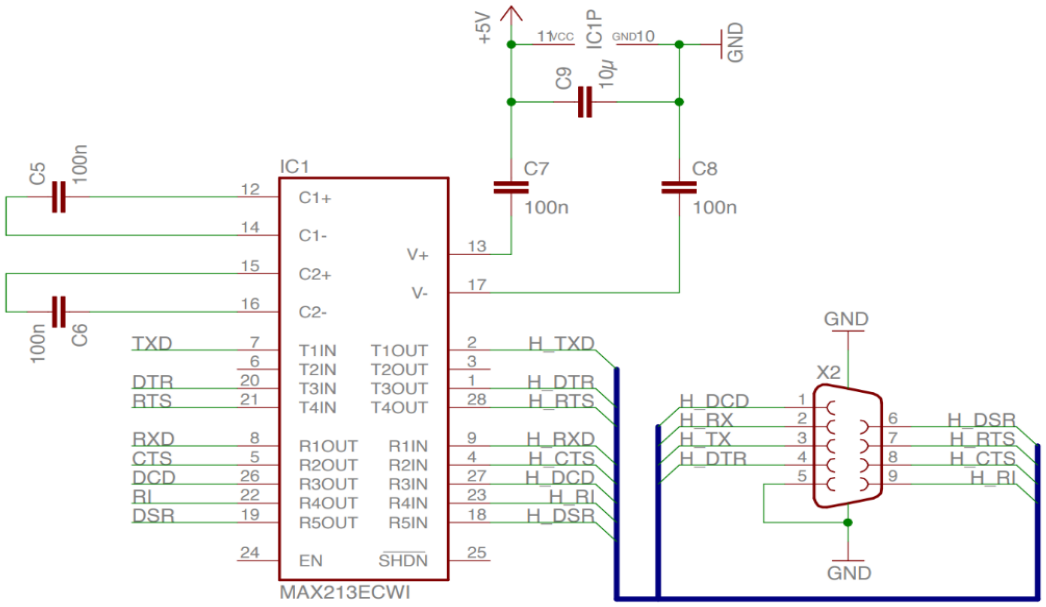

Рисунок 4.1 — схема подключения микросхемы CH340G

В проекте используется микросхема программируемой логики от производителя Altera, а именно Cyclone IV EP4CE6N на отладочной плате Altera-Cyclone-IV-board-V3.0. Число 6 означает количество логических ячеек в 6 тысяч штук. Схема имеет встроенные умножители 18 на 18 разрядов, которые можно использовать вместо затрачивания логики. LUT имеют 4 входа, в каждой ячейке есть один D-триггер, входы и выходы коммутации с другими ячейками и возможность конфигурирования прошивкой в арифметический режим для более эффективного использования одной LUT.

Таблица 4.1 — характеристики Cyclone IV EP4CE6N

Символ	Параметр	Min	Max	Единицы
V_{CCINT}	Напряжение питания ядра	-0.5	1.8	В
V_{CCIO}	Напряжение блоков ввода-вывода	-0.5	3.75	В
I_{OUT}	Выходной ток на пин	-25	40	мА
t_{RISE}, t_{FALL}	Крутизна фронтов	500	500	пс

Для прошивки платы использовался интерфейс JTAG, разъем для которого присутствует на отладочной плате. В операционной системе Linux с ним возникла проблема: вместо USB-Blaster программатор распознавался как USB-Blaster varient [2-1] и не прошивал схему. Проблема была решена записью инициализирующей информации в конфигурационный файл работы с USB-устройствами, а именно в файл /etc/udev/rules.d/51-altera-usb-blaster.rules.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсового проекта было разработано устройство вычисления квадратного корня на основе программируемой логики. Отличительной особенность схемы является то, что память данных схемы используется минимально. Большинство ресурсов рекомендовали использовать таблицу аппроксимаций функций как для оценки корня, так и обратного числа. В этом проекте удалось заменить использование внушительного для таких систем размеров памяти на несложные алгоритмы вычисления. Однако это можно отнести и к минусам, так как имеющаяся память не используется, замененная логикой. Неплохо было бы соблюдать определенный баланс между логикой и задействованием внутренней памяти. К плюсам также можно отнести работу с плавающей запятой, что является достаточно специфичной задачей.

Проекту определенно есть куда расти, например, распараллелить вычисление насколько это возможно, поработать над временными характеристиками схемы и ее разводкой на плате, чтобы можно было получить максимальные результаты.

Проектирование этого устройства дало полезный опыт разработки на всех этапах работы с программируемой логикой с точки зрения программиста ПЛИС.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1]. Вычислительные машины, системы и сети: дипломное проектирование (методическое пособие) [Электронный ресурс]: Минск БГУИР 2019. — Электронные данные. — Режим доступа: https://www.bsuir.by/m/12_100229_1_136308.pdf — Дата доступа: 30.11.2022

[2]. Краткий курс HDL-Verilog [Электронный ресурс]. — Электронные данные. — Режим доступа: http://iosifk.narod.ru/hdl_coding/verilog.htm – Дата доступа: 5.11.2022

[3]. Документация отладочной платы Altera-Cyclone-IV V3.0 [Электронный ресурс]. — Электронные данные. — Режим доступа: https://github.com/jvitkauskas/Altera-Cyclone-IV-board-V3.0 — Дата доступа: 30.11.2022

[4]. Методы вычисления квадратных корней [Электронный ресурс]. — Электронные данные. — Режим доступа: https://wikipredia.net/ru/Methods_of_computing_square_roots#Initial_estimate — Дата доступа: 16.10.2022

[5]. Fast inverse square root [Электронный ресурс]. — Электронные данные. — Режим доступа: https://wikipredia.net/ru/Methods_of_computing_square_roots#Initial_estimate — Дата доступа: 5.11.2022

[6]. 754-2019 - IEEE Standard for Floating-Point Arithmetic [Электронный ресурс]. — Электронные данные. — Режим доступа: https://ieeexplore.ieee.org/document/8766229 — Дата доступа: 29.10.2022

[7]. Synopsys Design Constraints [Электронный ресурс]. — Электронные данные. — Режим доступа: https://people.ece.ubc.ca/edc/7660.jan2018/lec9.pdf — Дата доступа: 29.10.2022

[8]. TimeQuest для чайников. [Электронный ресурс]. — Электронные данные. — Режим доступа: https://caxapa.ru/thumbs/442268/TimeQuest_for_dummies.pdf — Дата доступа: 10.11.2022

[9]. Quartus II Handbook, Volume 1. Design & Synthesis. [Электронный ресурс]. — Электронные данные. — Режим доступа: http://class.ece.iastate.edu/arun/Cpre381_Sp06/lab/labw01a/QuartusII_Handbook.pdf — Дата доступа: 16.10.2022

[10]. Метод Ньютона. [Электронный ресурс]. — Электронные данные. — Режим доступа: https://ru.wikipedia.org/wiki/Метод_Ньютона — Дата доступа: 16.11.2022

[11]. Division algorithm. [Электронный ресурс]. — Электронные данные. — Режим доступа: https://en.wikipedia.org/wiki/Division_algorithm#Newton–Raphson_division — Дата доступа: 16.11.2022

[12]. Харрис, Д. М. Цифровая схемотехника и архитектура компьютера: RISC-V / Д. М. Харрис, С. Л. Харрис. — СПб.: ДМК Пресс, 2022. — 809 с.

[13]. Цифровой синтез: практический курс / под общ. ред. А. Ю. Романова, Ю. В. Панчула. — М.: ДМК Пресс, 2020. — 556 с.

[14]. Угрюмов, Е. П. Цифровая схемотехника: учеб. пособие для вузов. — 3-е изд., переработанное и дополненное / Е. П. Угрюмов. — СПб.: БХВ-Петербург, 2010. — 816 с.

[15]. Томас, Д. Логическое проектирование и верификация систем на SystemVerilog / пер. с анг. А. А. Слинкина, А. С. Камкина, М. М. Чупилко; науч. ред. А. С. Камкин, М. М. Чупилко. — М.: ДМК Пресс, 2019. — 284 с.

ПРИЛОЖЕНИЕ А

(обязательное)

Схема электрическая структурная

ПРИЛОЖЕНИЕ Б

(обязательное)

Схема электрическая функциональная

ПРИЛОЖЕНИЕ В

Диаграмма состояний автоматов передатчика и приемника

ПРИЛОЖЕНИЕ Г

Графики функций оценок обратного числа и квадратного корня

ПРИЛОЖЕНИЕ Д

(обязательное)

Перечень элементов

ПРИЛОЖЕНИЕ Е

(обязательное)

Ведомость документов