В данной лабораторной работе вам предложено разработать аппаратный ускоритель шифрования **ГОСТ 34.12-2015 "Кузнечик"**.

Ниже дано описание алгоритма шифрования. С более полным описанием, содержащим сопутствующую матчасть можно ознакомиться здесь. Отмечу только, что я постарался выкинуть всю сложную математику, чтобы не грузить лишней информацией, которая не потребуется при выполнении лабораторной работы. Помимо прочего, можно ознакомиться с самим ГОСТом, в конце которого приведены тестовые варианты шифрования, которые можно использовать для проверки корректности работы своего модуля.

Все связанные с безопасностью операции внутри страны требуется защищать посредством данного криптоалгоритма.

Процесс шифрования данных состоит из 10 раундов девять из которых можно разделить на три этапа:

- Этап наложения ключа на шифруемые данные — это пример древнейшего способа шифровать данные — всякие алгоритмы Цезаря (немного не то, но примерно из той же оперы). Проблема в том, что если остановиться на этом этапе, то при условии владения зашифрованным и расшифрованным сообщением, можно с лёгкостью получить ключ шифрования (поскольку этап наложения ключа, это операция XOR с входными данными, чтобы получить ключ достаточно сделать XOR шифрованных и расшифрованных данных).
- Этап нелинейного преобразования. Нелинейное преобразование, в отличии от предыдущего этапа необратимо — если сделать (a xor b) xor b (повторить наложение ключа на зашифрованное предыдущим этапом сообщение), мы получим исходное сообщение (именно поэтому, (a xor b) xor a и даст ключ шифрования). Повторение нелинейного преобразования не даст сообщение с предыдущего этапа. Именно поэтому способ нелинейного преобразования — самая важная часть криптоалгоритма (и именно поэтому у зарубежных коллег есть определенные вопросы к разработчикам Кузнечика).
- Этап линейного преобразования. Если честно, моих знаний не хватает, чтобы объяснить на пальцах зачем оно, в моем понимании это дальнейшее закрепление нелинейного преобразования путём полного перемешивания байт, что затруднит линейный анализ зашифрованных данных.

Десятый раунд состоит только из этапа наложения ключа. Каждый раунд принимает полученные данные из предыдущего в качестве входных и берет новый ключ.

Представим, что необходимо обеспечить полную безопасность системы и шифровать весь входящий и исходящий трафик. Говорят, что на 64-битной машине под линуксом, программное шифрование на си может достичь скорости шифрования в 130 МБайт/с. Не знаю на каком железе под какой загрузкой ЦП обеспечена эта скорость, зато могу привести "на вскидку" минимальное количество инструкций процессора, требующееся для выполнения шифрования одного 16-байтного блока по алгоритму, реализованному в скрипте, лежащем рядом:

- 1 такт на XOR
- 1 такт на вычисление условия выхода из цикла раундов
- 1 такт на нелинейное преобразование через таблицу.

13 умножений + 16 xor-ов, + 1 сдвиг = 30 операций за одну стадию 16-стадийного линейного преобразования.

Умножаем полученные операции на 9 полных раундов: 33*9 = 300.

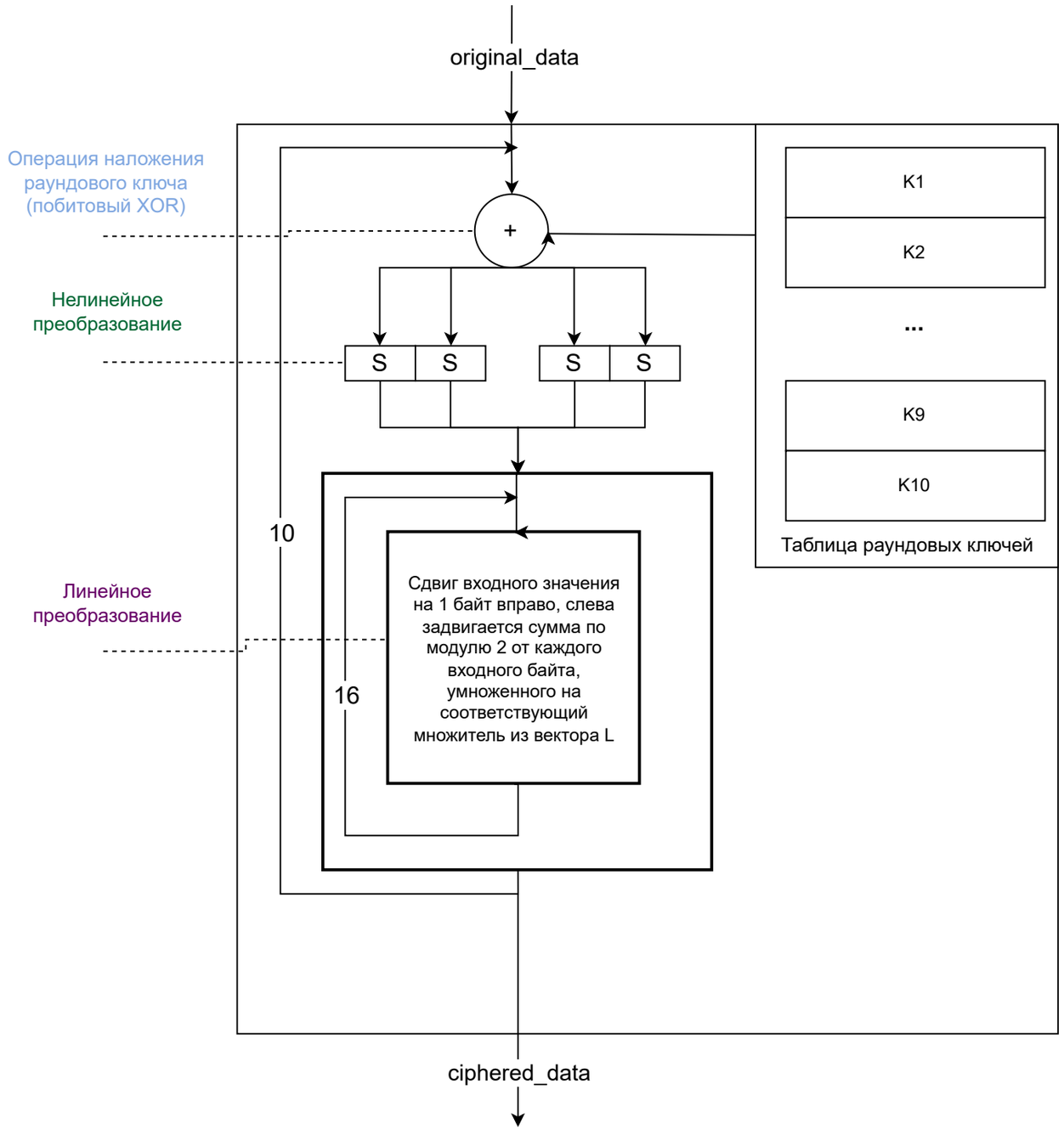
На самом деле, поскольку мы работаем со 128-битными данными, даже для 64-битного процессора без поддержки векторных операций придётся тратить гораздо больше тактов на выполнение этих операций. Кроме того я опустил несколько потенциальных условий, проигнорировал любую работу с оперативной памятью (а её тут очень много), поэтому этот результат можно смело умножать на 4 и все равно дать крайне заниженную оценку на количество этих инструкций для шифрования.

Устройство, которое вы реализуете в рамках этой лабораторной работы будет требовать:

- 1 такт на XOR
- 1 такт на нелинейное преобразование
- 16 тактов на линейное преобразование

Итого: 18*9 = 162 такта. Сюда надо конечно добавить несколько тактов на загрузку и выгрузку данных. Однако, стоит отметить, что ценой дополнительных ресурсов (в частности нескольких блоков BRAM, представленных на ПЛИС) можно выполнять линейное преобразование за 1 такт, на порядки ускорив работу устройства (дальнейшая его конвейеризация сделает шифрование практически мгновенным, сделав те самые такты на загрузку и выгрузку данных из памяти, которыми я пренебрёг в оценке скорости, бутылочным горлышком системы).

В целом, любой аппаратный ускоритель решает именно такую задачу: колоссально разгружает ресурсы центрального процессора на какую-то специфичную задачу. Кроме того, такой аппаратный блок можно установить на сетевой интерфейс. В таком случае, система будет работать с нешифрованными данными, а обмен с внешним миром будет происходить исключительно в зашифрованном на сетевом интерфейсе виде. (На самом деле такие блоки есть на любых сетевых интерфейсах, потому что большая часть информации передаётся именно в зашифрованном виде).



В «Кузнечике» выполняется девять полных раундов, каждый из которых включает в себя три последовательные операции:

* Операция наложения раундового ключа (Ki) или побитовый XOR ключа и входного блока данных;
* Нелинейное преобразование, которое представляет собой простую замену каждого байта на другой в соответствии с таблицей S
  (Каждый байт меняет свое значение на значение из таблицы, взятое по индексу, равному этому самому байту);
* Линейное преобразование. Повторяется 16 раз за раунд.
  В чем оно заключается:

  Каждый байт умножается в поле Галуа на один из коэффициентов ряда:
  (148, 32, 133, 16, 194, 192, 1, 251, 1, 192, 194, 16, 133, 32, 148, 1)
  в зависимости от порядкового номера байта.

  Пусть на вход пришло число 1. В порядке big-endian записывается как:
  0x00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_01.
  0 умножается на 148, 0 умножается на 32, ..., 1 умножается на 1.

  Полученные произведения складываются между собой по модулю 2. Все 16 байт блока сдвигаются на один байт вправо, а полученная ранее сумма произведений записывается на место старшего байта. Операция повторяется 16 раз, полностью заменяя все байты пришедшего к этой операции числа.

  Операцию умножения над полем Галуа следует производить следующим образом:

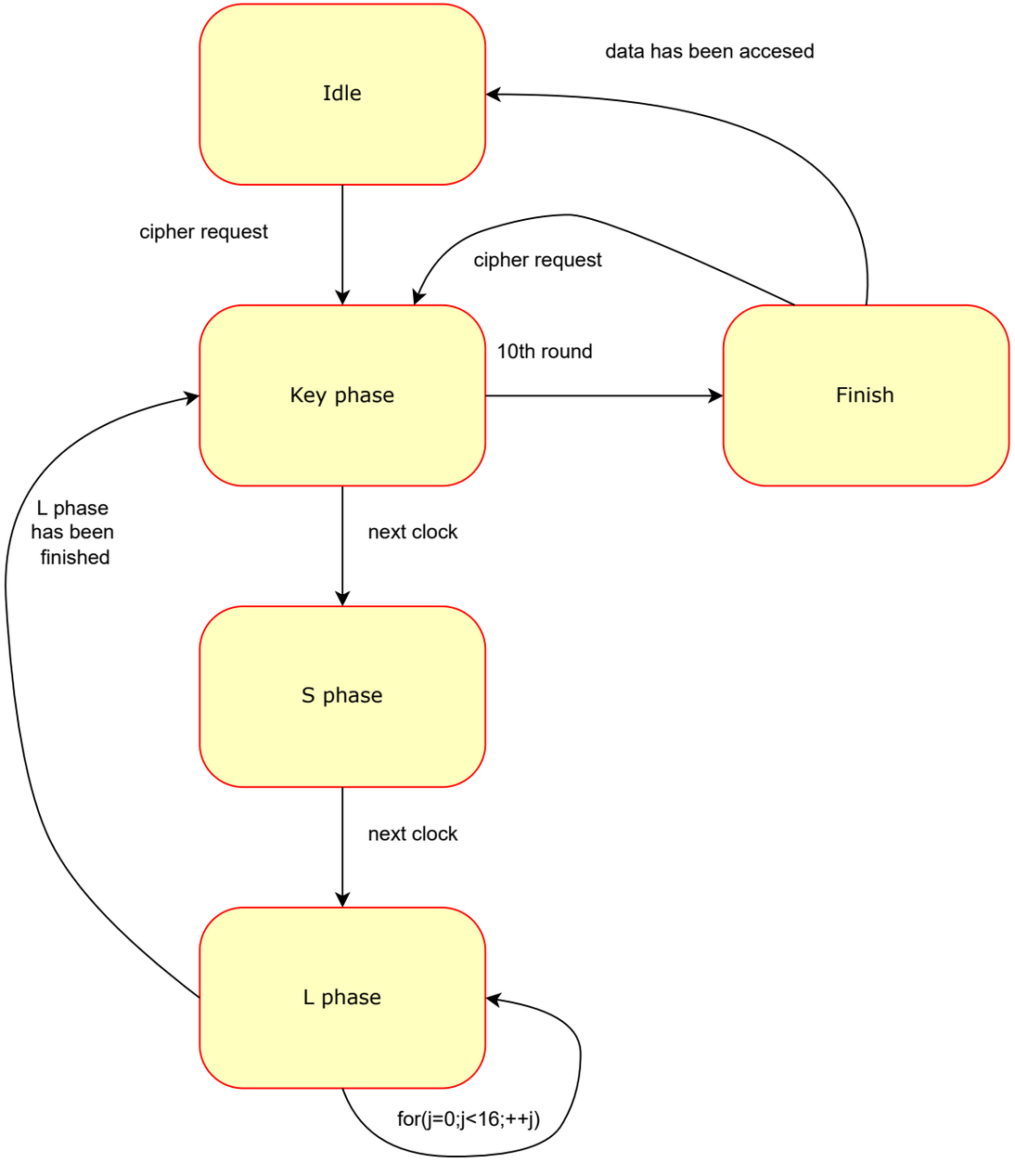
  Рассмотрим коэффициенты ряда:
  (148, 32, 133, 16, 194, 192, 1, 251, 1, 192, 194, 16, 133, 32, 148, 1).

  Отсортируем его и удалим из него дубликаты:
  (1, 16, 32, 133, 148, 192, 194, 251). Поскольку умножение на 1 аналогично обычному умножению и вернет второй множитель, исключим единицу из этого ряда: (16, 32, 133, 148, 192, 194, 251).

  Для каждого элемента получившегося ряда создается таблица умножения из 256 элементов.

  Таким образом, если мы хотим умножить число 243 на 16, мы возьмем 243-е значение первой заготовленной таблицы умножения
  (Всего таблиц семь, первая для элемента 16, вторая для элемента 32 и т.д.).

Последний десятый раунд не полный, он включает в себя только первую операцию XOR.



Работу модуля можно описать в виде конечного автомата, представленного слева.
Автомат содержит 5 состояний: Idle, Key phase, S phase, L phase, Finish.
* В состояние Idle автомат переходит после сброса, а так же после передачи результата предыдущего шифрования.
  В этом состоянии не происходит каких-либо значимых вычислений.
* В состояние Key phase автомат попадает либо по приходу сигнала request,
  пока автомат находится в состоянии Idle или Finish, либо после завершения L-фазы.
  В этом состоянии происходит наложение очередного ключа на шифруемые данные посредством операции XOR
* В состояние S phase автомат переходит на следующий такт после перехода в состояние Key phase.
  Это фаза нелинейного преобразования, когда в значении, полученном с предыдущей фазы каждый байт меняет свое значение посредством таблицы S_box. Если байт равен 0x03, то он получит значение, хранящееся в таблице по третьему индексу.
* В состояние L phase автомат переходит либо на следующий такт, после перехода в состояние S phase, либо на следующий такт после перехода в L phase до тех пор, пока счетчик не отсчитает 16 повторов.
  В этом состоянии на каждом повторе, модуль умножает каждый байт пришедшего на эту фазу значения с соответствующим байтом из ряда:
148, 32, 133, 16, 194, 192, 1, 251, 1, 192, 194, 16, 133, 32, 148, 1.
Старший байт умножается на 148, младший байт умножается на 1. Для каждого множителя ряда (кроме единицы) создается таблица умножения. Для умножения на какой-либо из множителей ряда достаточно взять значение из таблицы по индексу, равному второму множителю. Поскольку умножение на единицу над полем Галуа аналогично обычному умножению на единицу, для нее нет смысла создавать таблицу умножения. Кроме того, поскольку множители из ряда повторяются, достаточно создать семь таблиц вместо 16. Полученые произведения складываются между собой по модулю два. Эта сумма задвигается справа.
  После чего, процесс повторяется вплоть до 16 раз.
* В состояние Finish автомат переходит из состояния Key phase в случае прохождения 10ой стадии шифрования.