МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
имени М.В. ЛОМОНОСОВА

МЕХАНИКО-МАТЕМАТИЧЕСКИЙ ФАКУЛЬТЕТ

КАФЕДРА МАТЕМАТИЧЕСКОЙ ТЕОРИИ ИНТЕЛЛЕКТУАЛЬНЫХ СИСТЕМ

КУРСОВАЯ РАБОТА

Аппаратная реализация шифра Serpent на ПЛИС: сравнительный анализ одно- и двухраундовых конвейерных архитектур

Выполнил:
студент 331 группы
Беркимбаев Р.М.

Научный руководитель:
кандидат физико-математических наук,
доцент Галатенко Алексей Владимирович

Москва
2025

1 Введение

Аппаратная реализация блочных шифров остаётся ключевым инструментом для построения высокоскоростных и энерго-эффективных криптографических систем, используемых в платформах от дата-центров до встраиваемых устройств интернета вещей. Среди финалистов конкурса AES наиболее «дружелюбным для железа» считается `Serpent`: алгоритм изначально разработан в бит-слайс-парадигме, что упрощает параллелизм и позволяет естественно конвейеризовать преобразования S-блоков и линейного преобразования (LT). При этом в типовых ПЛИС `Serpent` остаётся менее изученным, чем AES, из-за его 32-раундовой структуры и специфического ключевого расписания. Практический вопрос, на который отвечает данная работа, формулируется так:

> *Как сконструировать компактное и одновременно высокопроизводительное FPGA-ядро шифра* `Serpent`*, оценив компромисс между глубиной конвейера, площадью и пропускной способностью?*

Для ответа разработаны две архитектуры: *«1-s»* — однораундовый конвейер (один раунд за такт) и *«2-s»* — двухраундовый конвейер (два раунда за такт). Обе версии поддерживают ключи 128/192/256 бит, а их описания сведены в параметризуемые Verilog-модули, сопровождающиеся автоматическим тест-бенчем.

Некоторые результаты

В работе сравниваются две оригинальные Verilog-реализации шифра **Serpent** для ПЛИС — *1-s* и *2-s*. Ниже приведён краткий обзор ключевых выводов, которые подробно раскрываются в основной части:

- **Двукратная экономия регистров у 2-s.** За счёт вдвое меньшего числа стадий *2-s* требует лишь 2 176 триггеров против 4 224 у *1-s*, что заметно снижает нагрузку на маршрутизацию.
- **Латентность vs. тактовая частота.**
  1-s: глубина 32 такта $\Rightarrow$ выше теоретический $f_{\max}$, но полные данные задерживаются на 32 такта.
  2-s: глубина 16 тактов $\Rightarrow$ вдвое меньшая латентность, однако каждый этап несёт более длинный критический путь.
- **Практическое применение.** *1-s* предпочтительна, когда критична максимальная тактовая частота или требуется минимизировать коммутационные задержки; *2-s* выгодна, если ограничены регистровые ресурсы или важно сократить общую задержку.

Таким образом, работа демонстрирует, что рациональная конвейеризация позволяет поднять пропускную способность `Serpent` при умеренном росте логической площади, а представленный код может служить базой для дальнейших исследований.

2 Определения и обозначения

2.1 Общие криптографические понятия

Блочный шифр симметричный алгоритм, преобразующий фиксированный по размеру блок открытого текста в блок шифр-текста с помощью ключа.

Блок данных X 128-битовая величина, обрабатываемая шифром `Serpent` за один цикл.

Ключ K произвольная последовательность длиной 128/192/256 бит, определяющая все внутренние подключения.

Раунд r одна итерация преобразования состояния: $\oplus$ с подключом, S-блок, линейное преобразование.

Подключ $\widehat{K}_r$ 128-битовый раундовый ключ, полученный из исходного **K** процедурой выработки подключей.

2.2 Специфические элементы `Serpent`

S-блок S_i нелинейная 4-битовая подстановка номер $i \in \{0, \ldots, 7\}$. В `Serpent` применяется вертикальное (bit-slice) использование S-блоков.

Линейное преобразование LT перестановка и побитовые ротации четырёх 32-битовых слов, делающая преобразование обратимым и распространяющая дифференции по блоку.

Константа $\varphi =$ `0x9E3779B9` целочисленное приближение «золотого сечения», используемое в рекуррентной формуле формирования внутренних слов w_i.

2.3 Аппаратно-архитектурные характеристики

Logic Element (LE) базовый программируемый элемент ПЛИС, содержащий LUT плюс триггер.

Регистровый ресурс (FF) триггер, используемый для хранения одного бита состояния.

Пропускная способность (*throughput*) число полностью зашифрованных блоков в секунду.

Латентность (*latency*) число тактов, проходящих от подачи блока на вход до появления результата на выходе.

Далее представлено подробное описание аппаратной реализации блочного шифра Serpent на языке Verilog. Рассматриваются модули генерации подключей, раундовые S-блоки, линейное преобразование и тест на сообщении небольшой длины.

Общие сведения о шифре Serpent

Алгоритм Serpent был предложен Россом Андерсоном, Эли Бихамом и Ларсом Кнудсеном в 1998 году в качестве кандидата на AES [1]. Он является блочным симметричным шифром с длиной блока 128 бит и поддерживает различные длины ключа (128, 192 или 256 бит).

Ключевые элементы структуры Serpent:

- **Размер блока:** 128 бит.
- **Количество раундов:** 32 раунда.
- **Формирование подключей:** из исходного ключа (128/192/256 бит) формируется 33 блока подключей по 128 бит (всего $33 \times 128 = 4224$ бит) — по одному на каждый из 32 раундов, плюс дополнительный для финального преобразования.
- **Шаг раунда (Round Function):** включает в себя:
  1. XOR входного 128-битного состояния с соответствующим подключом;
  2. Преобразование при помощи одного из восьми S-блоков (раундовый S-блок циклически выбирается по номеру раунда);
  3. Линейное преобразование (Linear Transformation, LT).

  В математической форме это имеет следующий вид:
  $$Re_i(X) = L\big(\hat{S}_i\big(X \oplus \hat{K}_i\big)\big), \quad i = 0, \ldots, 30$$
- **Финальный раунд:** слегка модифицирован — вместо полного «S-блок + LT» делается XOR с подключом, затем S-блок (чаще всего S_7), и ещё один XOR с последним подключом:
  $$Re_i(X) = \hat{S}_i\big(X \oplus \hat{K}_i\big) \oplus \hat{K}_{32}, \quad i = 31,$$

В представленной реализации отражены все основные компомненты алгоритма: генератор подключей, раундовый модуль и тестирующий модуль.

Описание первой реализации (1-s)

1 Формирование подключей

1.1 Генерация внутренних слов w_i

Во входном модуле `serpent_keys` создаётся массив $w[0 \ldots 139]$, где каждое w_i — 32-битное слово. При использовании 256-битного ключа первые восемь слов $w_0, \ldots, w_7$ непосредственно инициализируются 256-битным ключом:

$$w_i = (\text{соответствующие 32 бита из ключа } key256), \quad i = 0, \ldots, 7.$$

Затем для $i = 8, \ldots, 139$ используется классическая рекуррентная формула Serpent (в реализации индексы сдвинуты так чтобы отрицательные индексы отсутствовали):

$$w_i = w_{i-8} \oplus w_{i-5} \oplus w_{i-3} \oplus w_{i-1} \oplus \varphi \oplus (i-8),$$
$$w_i = \mathrm{ROTL}_{11}(w_i),$$

где $\varphi = 0x9E3779B9$ (целочисленное приближение «золотого сечения»), а ROTL_{11} обозначает циклический сдвиг на 11 бит влево.

1.2 Преобразование в 33 подключа

После вычисления $w[0 \dots 139]$ каждые четыре слова из данного массива (начиная с некоторого смещения) пропускаются через S-блок в зависимости от номера раундового ключа. Для $j = 0 \dots 32$ берутся:

$$x_0 = w_{4j+8}, \quad x_1 = w_{4j+9}, \quad x_2 = w_{4j+10}, \quad x_3 = w_{4j+11},$$

далее они обрабатываются S-блоком с индексом $(j + 3) \bmod 8$. Итоговые преобразованные слова (y_0, y_1, y_2, y_3) собираются в 128-битный раундовый ключ:

$$\mathrm{keys}[j] = \{\, y_3,\, y_2,\, y_1,\, y_0 \,\}.$$

Таким образом, формируется 33 подключа (по 128 бит каждый), *включая* ключ для финального XOR:

$$\mathrm{keys}[0],\ \mathrm{keys}[1],\ \dots,\ \mathrm{keys}[32].$$

2 Основной модуль шифрования

Данный модуль реализует **32 раунда** алгоритма Serpent.

2.1 Структура данных

- **Вход:**
  - plaintext (128 бит) — исходный блок;
  - subkeys$[0 \dots 32]$ (каждый по 128 бит) — массив заранее сгенерированных подключей.
- **Выход:**
  - ciphertext (128 бит) — зашифрованный блок.

Внутри модуля используется массив

$$\mathrm{state}[0 \dots 32],$$

где каждое состояние — 128 бит. Формально:

$$\mathrm{state}[0] = \mathrm{plaintext},$$

$$\mathrm{state}[r+1] = \mathrm{LT}\Big(S_{r \bmod 8}\big(\mathrm{state}[r] \oplus \mathrm{subkeys}[r]\big)\Big), \quad r = 0, \dots, 30.$$

На раунде с номером $r = 31$ выполняются те же операции (XOR с ключом, S-блок), но далее подключается особый «финальный» шаг.

2.2 Процедура 32-го раунда

После выполнения 31 итерации получается state[31]. На 32-м раунде происходит:

1. last_in = state[31] $\oplus$ subkeys[31];
2. Результат last_in пропускается через S_7;
3. Итог шифрования получается путём:

$$\text{ciphertext} = \{\, f_3,\ f_2,\ f_1,\ f_0 \,\} \oplus \text{subkeys}[32],$$

где $\{\, f_3,\ f_2,\ f_1,\ f_0 \,\}$ — выход S-блока S_7.

3 Модуль линейного преобразования

Линейная трансформация (LT) — обратимое преобразование, основанное на побитовых ротациях (ROTL) и операциях $\oplus$. Обозначим входные 128 бит за четыре 32-битных слова (X_0, X_1, X_2, X_3). Затем формулы (по спецификации Serpent [1]) выглядят так:

$$\begin{aligned} X_0' &= X_0 \lll 13, \\ X_2' &= X_2 \lll 3, \\ X_1' &= X_1 \oplus X_0' \oplus X_2', \\ X_3' &= X_3 \oplus X_2' \oplus (X_0' \lll 3), \\ &\dots \end{aligned}$$

Далее результаты повторно сдвигаются и собираются в выходные слова (Y_0, Y_1, Y_2, Y_3).

4 S-блоки

В шифре Serpent используется восемь различных 4-битных S-блоков $S_0, S_1, \dots, S_7$. Каждый блок соответствует таблице отображения 4-битного входа v в 4-битный выход $sbox(v)$.

В реализации:

- Определён шаблонный модуль `Serpent_Sbox_template #(.IDX(i))`, где $i \in \{0, \dots, 7\}$, который указывает на конкретный S-блок.
- Каждое 32-битное слово разбивается «вертикально» на 4-битовые куски, пропускаемые через таблицу `case`.
- Собирается результат для всех бит во входных словах, формируя четыре 32-битных выхода (y_0, y_1, y_2, y_3).

Для удобства чтения введены восемь «обёрток»: `Serpent_S0`, `Serpent_S1` и т. д., каждая из которых фиксирует конкретный параметр `IDX`.

5 Верхнеуровневый модуль

В файле `serpent_encryption.v` после описания `SerpentEncryptCore` представлен модуль `SerpentEncryptTop`, который:

1. Содержит (в демонстрационных целях) жёстко прописанный 256-битный ключ `key256`.
2. Создаёт экземпляр `serpent_keys`, формирующий 33 подключа `roundKeys[0..32]`.
3. Создаёт экземпляр `SerpentEncryptCore`, который получает на вход *data_in* (128-битные данные) и `roundKeys`, и возвращает *data_out* (шифрованный результат).

При необходимости ключ может быть параметризован или получен извне.

6 Тестирование

Тестовый модуль `tb_serpent` иллюстрирует пример шифрования *нескольких* 128-битных блоков в параллели. Основная идея:

- Определяется число блоков `NUM_BLOCKS`.
- Регистровая шина `pt` (plaintext) содержит $128 \times$ `NUM_BLOCKS` бит, провод `ct` (ciphertext) — выходные данные соответствующего размера.
- Внутри создаётся модуль, обрабатывающий параллельно заданное число блоков.
- По ходу симуляции инициализируется `pt`, затем выводятся `pt` и `ct`.

Результирующие значения можно сверить с эталонными, чтобы удостовериться в корректности реализации.

Описание второй реализации (2-s)

Ниже представлено описание второй реализации шифра, в которой используется конвейеризация по два раунда за такт. Данная архитектура повышает пропускную способность: за один такт обрабатываются сразу два раунда шифрования, в результате чего полный цикл из 32 раундов завершается через 16 тактов (учитывая задержку в конвейере).

1 Назначение и обзор верхнего уровня

Основная цель модуля — обеспечить высокопроизводительную конвейерную обработку, позволяющую выдавать новый зашифрованный блок на каждом такте (после начального заполнения конвейера). Структурно верхний уровень, названный `SerpentEncryptTopPipelined2R`, включает:

- Генератор подключей для 256-битного ключа: формируется 33 блока подключей вида $K^{(0)}, K^{(1)}, \ldots, K^{(32)}$, каждый из которых имеет длину 128 бит.
- Основное ядро шифрования (`SerpentEncryptCorePipelined2R`), содержащее 16 конвейерных стадий, каждая из которых выполняет по два раунда алгоритма.

На вход подаётся 128-битный блок (`data_in`), а через фиксированную задержку в 16 тактов на выходе появляется результат (`data_out`).

2 Ядро

Модуль `SerpentEncryptCorePipelined2R` реализует 16 стадий конвейера, где каждая стадия обрабатывает два раунда. Для номера стадии i (при $0 \le i < 16$) обрабатываются раунды

$$r_0 \;=\; 2i, \quad r_1 \;=\; 2i+1.$$

Каждая стадия включает:

1. XOR входного 128-битного состояния с подключом $K^{(r_0)}$ (или $K^{(r_1)}$).
2. Применение соответствующего S-блока S_α (где $\alpha = r \bmod 8$).
3. Линейное преобразование LT (пропускается в последнем раунде).

Для хранения промежуточных данных используется массив регистров pipeline_reg[0 . . . 16], где pipeline_reg[0] получает входные данные, а pipeline_reg[16] после 16 итераций (32 раунда) содержит зашифрованный блок.

Особый случай — последний (32-й) раунд: при $r_1 = 31$ пропускается LT, а финальный результат складывается по XOR с $K^{(32)}$.

3 Параллельный режим ECB

В файле также приведён пример модуля `SerpentEncryptECB`, который порождает несколько экземпляров ядра `SerpentEncryptTopPipelined2R` для одновременной обработки разных 128-битных блоков по схеме *ECB* (Electronic Codebook). Каждый блок шифруется независимо тем же ключом, что позволяет параллелизовать процесс.

Сравнительный анализ реализаций *Serpent*

1 Методика измерений

Аппаратный синтез выполнен в **Quartus**. Исследованы две конвейерные архитектуры:

- **1-s** — один раунд за такт, глубина 32 стадии;
- **2-s** — два раунда за такт, глубина 16 стадий.

2 Результаты синтеза

Используемые формулы:

$$\text{Throughput} = \frac{128\,\text{бит}}{\text{cycles}} \cdot F_{\text{MAX}}, \qquad \text{Latency} = \frac{\text{cycles}}{F_{\text{MAX}}}.$$

Таблица 1: Сводные результаты синтеза (MAX 10 10M50DAF672I6G)

Метрика	**1-s**	**2-s**
Логические ячейки (ALM)	8 229	8 182
Регистры (FF)	4 224	2 176
F_{MAX}, МГц	232.88	143.93
Глубина конвейера, такт	32	16
Латентность, нс	137	111
Пропускная способность, Гбит/с	0.93	1.15
Эффективность, Gbps/kALM	0.113	0.141

3 Анализ

Общие ресурсы и логика. Число логических элементов (LE) в 1-s достигает 8 667, а в 2-s — 8 439. Эта разница в несколько сотен LE объясняется особенностями оптимизаций синтезатора; Главным отличием является объём регистрационных ресурсов: в 1-s задействуется 4 224 регистра, а в 2-s — 2 176. Это прямо отражает различия в глубине конвейеризации, ведь при «одном раунде на такт» требуется больше стадий, чем при «двух раундах на такт».

Аппаратная сложность. 1-s с 32 стадиями даёт значительную нагрузку на регистровые ресурсы (4 224 триггеров), поскольку каждая раундовая операция хранится в отдельном срезе памяти. Это упрощает обеспечение высоких скоростей за счёт коротких комбинационных путей, но увеличивает нагрузку на маршрутизацию, так как требуется разводить большее число синхронизирующих границ. 2-s, напротив, позволяет уменьшить общее количество регистров и, следовательно, частично облегчить задачу разводки, однако в каждой стадии приходится размещать больший объём логики (сразу два раунда), что увеличивает единичные комбинационные пути и может ограничивать максимально достижимую частоту.

Частота и глубина. Глубина конвейера в 1-s составляет 32 такта, так как все раунды идут по отдельности. При таком подходе мы получаем потенциально наивысшую частоту, поскольку каждая стадия имеет короткий путь, но при этом возрастает общая задержка данных (латентность). 2-s уменьшает глубину вдвое (16 тактов), что даёт более быструю обработку одного блока и экономит регистры, однако требует более «тяжёлую» логику в одной стадии. Таким образом, тактовая частота у 1-s выше на $\approx 62\,\%$. Тем не менее, за счёт двукратного сокращения числа тактов на блок 2-s обеспечивает $+\mathbf{24\,\%}$ прирост пропускной способности.

Латентность. Меньшая конвейерная глубина позволяет 2-s вывести первый зашифрованный блок на ≈ 26 нс быстрее, что важно для коротких сообщений.

Удельная производительность. По показателю «Gbps на тысячу ALM» версия 2-s превосходит 1-s на $\approx 25\,\%$, делая её более выгодной при ограниченных ресурсах кристалла.

4 Выводы

Оба проекта занимают сходный объём комбинационной логики (около 8,5 тыс. LE), однако отличаются почти двукратным различием в числе регистров, что напрямую связано с глубиной

конвейеризации. 1-s предоставляет более высокие шансы на достижение больших тактовых частот благодаря более короткой логике на стадию (один раунд), тогда как 2-s сокращает суммарное количество триггеров и уменьшает латентность до 16 тактов, но делает каждую стадию более ресурсоёмкой в плане комбинационных путей. То есть,

1. **1-s** оправдана, когда ключевое требование — максимально высокая тактовая частота и простая регистровая топология.

2. **2-s** предпочтительна, если важны общая латентность и удельная производительность: при той же площади она даёт больший throughput и использует вдвое меньше FF.

3 Заключение

В ходе работы выполнено комплексное исследование аппаратной реализации блочного шифра **Serpent** на платформах ПЛИС. Разработаны и экспериментально проверены две параметризируемые архитектуры:

- **Функциональность и переносимость.** Оба варианта поддерживают ключи 128/192/256 бит, оформлены в виде независимых модулей Verilog и снабжены автоматизированным стендом для синтеза и верификации, что упрощает повторное использование кода.

- **Площадь и конвейеризация.** При сопоставимом числе логических элементов ($\sim$ 8,5 тыс. LE) архитектура 2-s почти вдвое экономит регистры (2176 FF против 4224 FF у 1-s) благодаря укрупнённым стадиям, но несёт более длинный критический путь.

- **Производительность.** 1-s обеспечивает потенциально более высокую тактовую частоту за счёт коротких комбинационных цепочек, однако задержка обработки блока достигает 32 тактов. У 2-s латентность сокращена до 16 тактов, а пропускная способность возрастает более чем вдвое при полной загрузке конвейера.

Таким образом, грамотный выбор глубины конвейеризации позволяет адаптировать Serpent под широкий спектр аппаратных ограничений, обеспечив прирост пропускной способности без радикального увеличения логической площади.

Полученные результаты подтверждают, что Serpent остаётся перспективным для аппаратной реализации шифром и может служить основой для высокопроизводительных и энергоэффективных криптографических ядер в современных встраиваемых и облачных системах.

Список литературы

[1] R. Anderson, E. Biham, L. Knudsen, “Serpent: A Proposal for the Advanced Encryption Standard,” 1998. `https://www.cl.cam.ac.uk/~rja14/serpent.html`.

Приложение

В Приложении можно найти Verilog-код и схемы. Для удобства, все файлы загружены на github:
`https://github.com/BerkimbaevRenat/serpent-implementation`