# Процессор хэш-функции Salsa 20/8

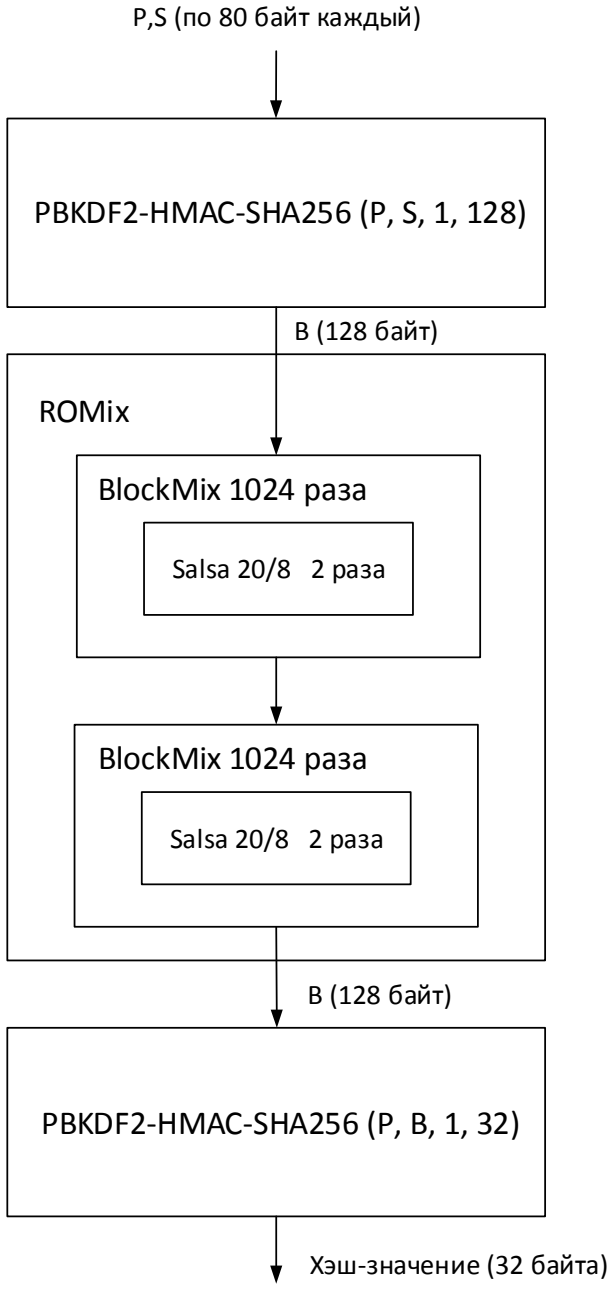
Станкевич А.В., Петровский Н.А., Качинский М.В.
БГУИР

Salsa 20 является криптосистемой для поточного шифрования. Хэш-функция Salsa 20 используется в указанной криптосистеме для получения псевдослучайных чисел, которые потом поразрядно складываются по модулю два с открытым текстом.

В настоящей работе рассматривается аппаратная реализация хэш-функции Salsa 20/8 применительно к системе майнинга криптовалюты Litcoin.

Одним из главных отличий криптовалюты Litecoin от ранее появившейся Bitcoin является использование функции Scrypt [RFC7914. The scrypt Password-Based Key Derivation Function] вместо SHA-256 для доказательства выполненной работы.

Функция Salsa20/8 основывается на преобразовании quaterraund над четырьмя 32-разрядными словами.




# Графическое представление алгоритма вычисления значения функции Scrypt

Общее количество тактов, необходимых для вычисления функции Scrypt при аппаратной реализации, будет существенно зависеть от числа тактов, требуемых для вычисления функции Salsa20/8, поскольку эта функция многократно используется во внутренних циклах алгоритма. Поэтому требуется выбрать вариант архитектурного решения, обеспечивающий максимально возможную производительность при минимально возможных аппаратных затратах с учетом того, что функция ScryptROMix функции Scrypt может вычисляться только итеративно (организовать параллельное выполнение двух групп циклов функции ScryptBlockMix нельзя без очень больших аппаратных затрат).

# Алгоритм Salsa20/8, описанный на языке C [RFC7914]

```
#define R(a,b) (((a) << (b)) | ((a) >> (32 - (b))))
void salsa20_word_specification(uint32 out[16],uint32 in[16])
{
int i;
uint32 x[16];
for (i = 0;i < 16;++i) x[i] = in[i];
for (i = 8;i > 0;i -= 2) {
x[ 4] ^= R(x[ 0]+x[12], 7); x[ 8] ^= R(x[ 4]+x[ 0], 9);
x[12] ^= R(x[ 8]+x[ 4],13); x[ 0] ^= R(x[12]+x[ 8],18);
x[ 9] ^= R(x[ 5]+x[ 1], 7); x[13] ^= R(x[ 9]+x[ 5], 9);
x[ 1] ^= R(x[13]+x[ 9],13); x[ 5] ^= R(x[ 1]+x[13],18);
x[14] ^= R(x[10]+x[ 6], 7); x[ 2] ^= R(x[14]+x[10], 9);
x[ 6] ^= R(x[ 2]+x[14],13); x[10] ^= R(x[ 6]+x[ 2],18);
x[ 3] ^= R(x[15]+x[11], 7); x[ 7] ^= R(x[ 3]+x[15], 9);
x[11] ^= R(x[ 7]+x[ 3],13); x[15] ^= R(x[11]+x[ 7],18);

x[ 1] ^= R(x[ 0]+x[ 3], 7); x[ 2] ^= R(x[ 1]+x[ 0], 9);
x[ 3] ^= R(x[ 2]+x[ 1],13); x[ 0] ^= R(x[ 3]+x[ 2],18);
x[ 6] ^= R(x[ 5]+x[ 4], 7); x[ 7] ^= R(x[ 6]+x[ 5], 9);
x[ 4] ^= R(x[ 7]+x[ 6],13); x[ 5] ^= R(x[ 4]+x[ 7],18);
x[11] ^= R(x[10]+x[ 9], 7); x[ 8] ^= R(x[11]+x[10], 9);
x[ 9] ^= R(x[ 8]+x[11],13); x[10] ^= R(x[ 9]+x[ 8],18);
x[12] ^= R(x[15]+x[14], 7); x[13] ^= R(x[12]+x[15], 9);
x[14] ^= R(x[13]+x[12],13); x[15] ^= R(x[14]+x[13],18);
}
for (i = 0;i < 16;++i) out[i] = x[i] + in[i];
}
```

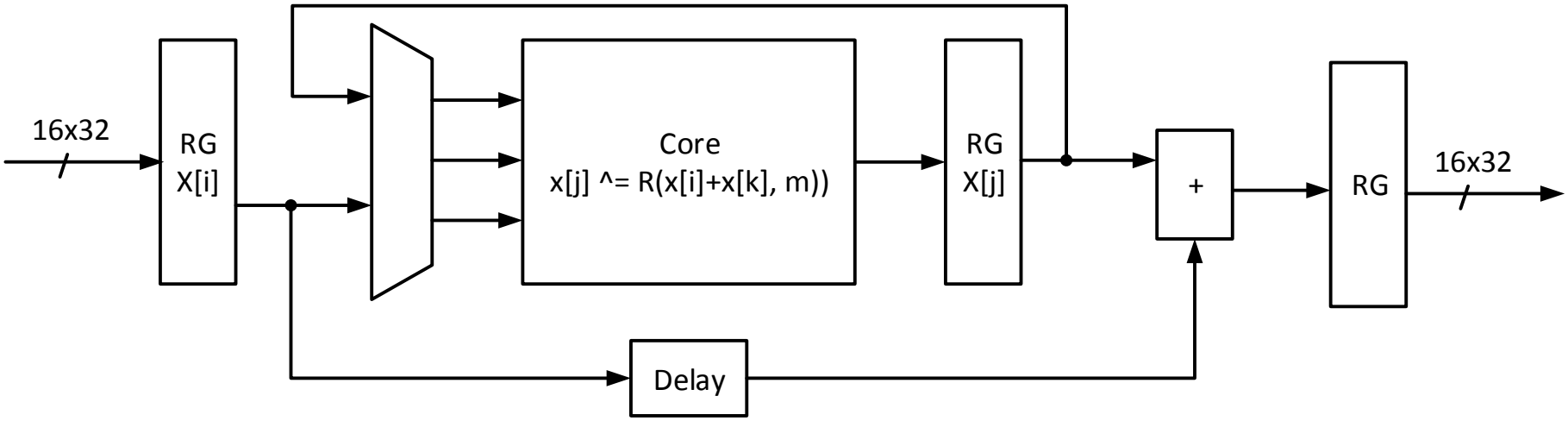
Две соседние строки в приведенном алгоритме описывают quaterraund.

Внутренний вычислительный цикл образует doubleround, состоящий из восьми quaterraund.

Анализ алгоритма показывает, что каждый quaterraund над четырьмя 32-разрядными словами может выполняться независимо от других, следовательно, вычислительный процесс можно распараллелить. Следует учесть, что вторая половина выражений внутреннего цикла (16 последних выражений вида `x[j] ^= R(x[i]+x[k], m))` может вычисляться только после завершения вычислений первой половины.

Рассмотрим варианты возможных архитектурных решений аппаратной реализации функции Salsa20/8.

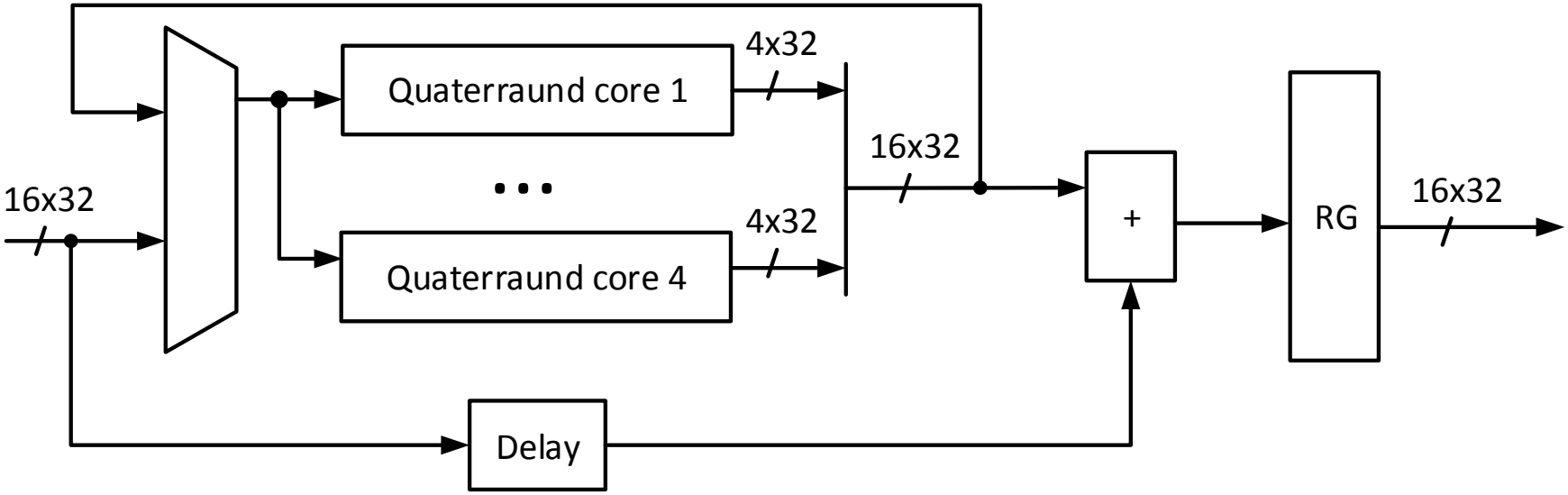
# 1. Итеративная архитектура на уровне вычислений выражения вида `x[j] ^= R(x[i]+x[k], m)`



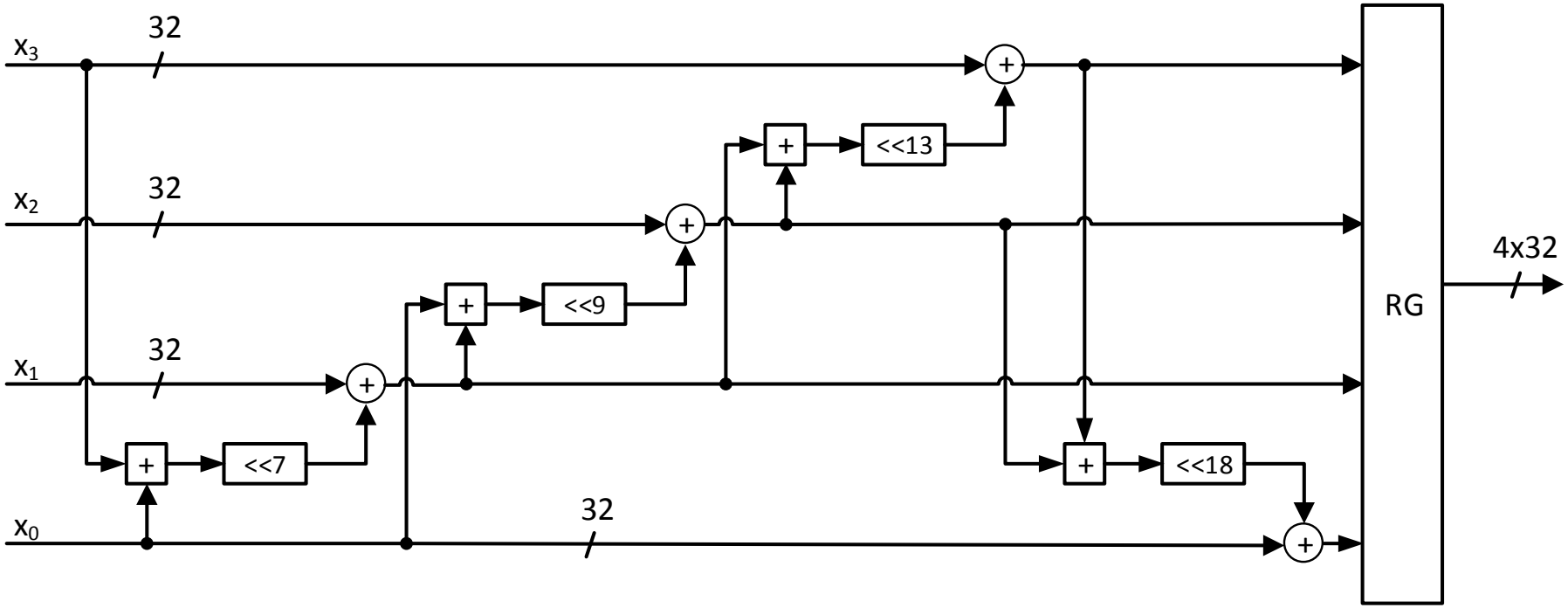

Если вычисление приведенного выражения выполнить за один такт, то вычислительный процесс займет 32 такта на один цикл (из четырех) алгоритма или всего 32*4 = 128 тактов. После этого надо выполнить завершающее сложение (еще один такт). Весь вычислительный процесс займет 129 тактов.

2. Итеративная архитектура на уровне quaterraund. Если вычисления quaterraund реализовать за один такт, то такое решение без учета завершающего сложения сократит количество повторений по сравнению с предыдущим решением в 4 раза. Длительность полученного такта должна быть меньше, чем длительность четырех тактов для первого варианта за счет отсутствия дополнительной задержки на регистре результата (вместо четырехкратного запоминания результата будет однократное). Общее количество тактов с учетом завершающего сложения 32+1=33.

3. Параллельно-итеративная архитектура на уровне четырех quaterraund. В этом случае будет выигрыш по числу тактов без учета завершающего сложения по сравнению с вариантом 2 в четыре раза. Общее количество тактов 9.



# Ядро quaterraund



4. Конвейерная реализация. На одной ступени конвейера можно реализовать вычисления выражения

```
x[j] ^= R(x[i]+x[k], m)
```

либо quaterraund. В первом случае будет меньше длительность такта, во втором случае будет меньше ступеней конвейера.

5. Параллельно-конвейерная реализация. В этом случае на одной ступени конвейера можно реализовать процессор quaterraund и параллельно будут работать четыре таких конвейерных вычислителя. По сравнению с 3 вариантом затраты будут в несколько раз выше (менее, чем в 8 раз за счет входного мультиплексора в итеративной архитектуре), но за счет отсутствия входного мультиплексора будет некоторый выигрыш по уменьшению длительности такта.

6. Итеративная параллельно-конвейерная реализация. Содержит четыре параллельных вычислителя на уровне quaterraund с реализацией внутри quaterraund четырех ступеней конвейера для вычислений выражения вида `x[j] ^= R(x[i]+x[k], m)`.

С учетом того, что функция ScryptROMix будет вычисляться итеративно, первые ступени всех конвейерных реализаций будут простаивать до тех пор, пока на выходе конвейера не будет получен результат преобразования. В таком решении будет значительное увеличение аппаратных затрат, пропорциональное числу ступеней конвейера, однако за счет удаления входного мультиплексора будет некоторый выигрыш по сравнению с итеративной архитектурой по длительности такта.

Проведенный анализ позволяет выбрать для реализации функции Salsa20/8 параллельно-итеративный вариант архитектуры 3 как обеспечивающий достаточно высокую производительность по сравнению с другими итеративными архитектурами при относительно небольших аппаратных затратах по сравнению с конвейерными вариантами. Предлагается вычислять quaterraund за один такт, поскольку длительность одного такта будет меньше суммы длительностей четырех тактов (вычисления выражения вида `x[j] ^= R(x[i]+x[k], m)` реализуются последовательно) на величину трех задержек переключения регистров для хранения результатов плюс дополнительно при реализации на FPGA добавится задержка на трассировочных ресурсах кристалла.

Проведено прототипирование процессора Salsa20/8 на кристалле XC7Z020-2CLG484 для сравнения с результатами реализации [Implementation and Analysis of Scrypt Algorithm in FPGA. – [Электронный ресурс]. – Режим доступа: https://alpha-t.net/wp-content/uploads/2013/11/Alpha-Technology-Scrypt-Analysis-on-FPGA-proof-of-concept.pdf]. Характеристики предлагаемой реализации после процедуры синтеза с использованием средств ISE 14.7 приведены ниже. Число тактов для получения результата – 9. Оценка частоты после процедуры синтеза – 163 МГц.

| Device Utilization Summary (estimated values) | | | [-] |
|---|---|---|---|
| **Logic Utilization** | **Used** | **Available** | **Utilization** |
| Number of Slice Registers | 1552 | 106400 | 1% |
| Number of Slice LUTs | 2321 | 53200 | 4% |
| Number of fully used LUT-FF pairs | 1424 | 2449 | 58% |
| Number of bonded IOBs | 1028 | 200 | 514% |
| Number of BUFG/BUFGCTRLs | 1 | 32 | 3% |

В реализации [Implementation and Analysis of Scrypt Algorithm in FPGA] указывается, что общее число тактов для вычисления Salsa20/8 равно 34. Полная реализация Scrypt [Implementation and Analysis of Scrypt Algorithm in FPGA] имеет рабочую тактовую частоту 120 МГц. Предлагаемая реализация при сравнимой тактовой частоте требует для получения результата Salsa20/8 почти в 4 раза меньше тактов (9 тактов вместо 34).