ПРОЦЕССОР ХЭШ-ФУНКЦИИ SALSA20/8

Станкевич А. В., Петровский Н. А., Качинский М. В.
Кафедра электроннных вычислительных средств,
Белорусский государственный университет информатики и радиоэлектороники
Минск, Республика Беларусь
E-mail: {stankevich, nick.petrovsky, kachinsky}@bsuir.by

Salsa20 является криптосистемой для поточного шифрования [1]. Хэш-функция Salsa20 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 Salsa20/8 применительно к системе майнинга криптовалюты Litcoin.

Введение

Одним из главных отличий криптовалюты Litecoin от ранее появившейся Bitcoin является использование функции Scrypt [2] вместо SHA-256 для доказательства выполненной работы. Основной целью использования Scrypt является усложнение реализации майнинга криптовалюты за счет увеличения вычислительной сложности процесса майнинга и необходимости использования в этом процессе памяти. Благодаря таким особенностям алгоритма предполагалось усложнить разработку и увеличить стоимость ASIC (application-specific integrated circuit) для процесса майнинга криптовалюты.

I. Реализация хэш-функции

Функция Salsa20/8 основывается на преобразовании `quaterraund` над четырьмя 32-разрядными словами. Алгоритм Salsa20/8, описанный на языке C, приведен ниже [2]:

Листинг 1 – Описание алгоритма Salsa20/8

```
#define R(a,b) ((a << b)|(a >> (32 - b)))
#define C(y,a,b,c,d) y[a]^= R(y[b]+y[c],d);
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) {
  C(x, 4, 0,12, 7); C(x, 8, 4, 0, 9);
  C(x,12, 8, 4,13); C(x, 0,12, 8,18);
  C(x, 9, 5, 1, 7); C(x,13, 9, 5, 9);
  C(x, 1,13, 9,13); C(x, 5, 1,13,18);
  C(x,14,10, 6, 7); C(x, 2,14,10, 9);
  C(x, 6, 2,14,13); C(x,10, 6, 2,18);
  C(x, 3,15,11, 7); C(x, 7, 3,15, 9);
  C(x,11, 7, 3,13); C(x,15,11, 7,18);
  C(x, 1, 0, 3, 7); C(x, 2, 1, 0, 9);
  C(x, 3, 2, 1,13); C(x, 0, 3, 2,18);
  C(x, 6, 5, 4, 7); C(x, 7, 6, 5, 9);
  C(x, 4, 7, 6,13); C(x, 5, 4, 7,18);
  C(x,11,10, 9, 7); C(x, 8,11,10, 9);
  C(x, 9, 8,11,13); C(x,10, 9, 8,18);
  C(x,12,15,14, 7); C(x,13,12,15, 9);
  C(x,14,13,12,13); C(x,15,14,13,18);  }
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`. 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 [2] нельзя без очень больших аппаратных затрат).

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)`. В этом случае все вычисления будут проводиться на одном процессорном ядре, на вход которого поочередно мультиплексируются соответствующие входные данные (три операнда) (рисунок 1). Если вычисление приведенного выражения выполнить за один такт, то вычислительный процесс займет 32 такта на один цикл (из четырех) алгоритма или всего $32 \times 4 = 128$ тактов. После этого надо выполнить завершающее сложение (еще один такт). Весь вычислительный процесс займет 129 тактов.

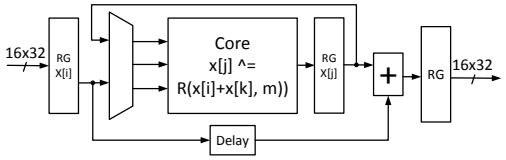

Рис. 1 – Итеративная архитектура

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` (рисунки 2, 3). В этом случае будет выигрыш по числу тактов без учета завершающего сложения по сравнению с вариантом 2 в четыре раза. Общее количество тактов 9.

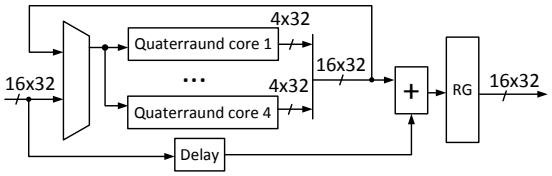

Рис. 2 – Параллельно-итеративная архитектура

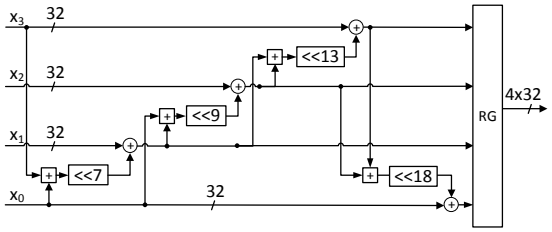

Рис. 3 – Ядро quaterraund

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 [3]. Характеристики предлагаемой реализации после процедуры синтеза с использованием средств ISE 14.7 приведены на рисунке 4.

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%

Рис. 4 – Аппаратные затраты процессора Salsa20/8

Число тактов для получения результата – 9. Оценка частоты после процедуры синтеза – 163 МГц.

В реализации [3] указывается, что общее число тактов для вычисления Salsa20/8 равно 34. Полная реализация Scrypt [3] 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).

1. Daniel J. Bernstein. Salsa20 specification. – [Электронный ресурс]. – Режим доступа: `http://cr.yp.to/snuffle/spec.pdf`. Date of access: 09.10.2019.
2. RFC7914. The scrypt Password-Based Key Derivation Function. – [Электронный ресурс]. – Режим доступа: `https://tools.ietf.org/html/rfc7914`. Date of access: 09.10.2019.
3. 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`. Дата доступа 09.10.2019.