УДК 621.396.41

Н.П. Борисенко, В.Л. Нгуен

Алгоритмы минимизации числа логических элементов при реализации линейных отображений

Рассматривается эффективный алгоритм минимизации числа логических элементов при аппаратной реализации линейных отображений большой размерности, представленных множеством линейных булевых функций (ЛБФ). Сущность алгоритма заключается в использовании структурированных данных описания двоичного дерева, построенного при определении общих логических элементов, реализующих пары ЛБФ. На основе полученного двоичного дерева строится алгоритм синтеза логических схем линейных отображений большой размерности.
Ключевые слова: линейное отображение, булева функция, минимизация, реализация.

Основные положения. Известно, что линейное отображение представляет собой один из важнейших компонентов, обеспечивающих свойство рассеивания в симметричных шифрах [4]. С одной стороны, применимое на практике линейное отображение должно удовлетворять ряду криптографических свойств, таких как свойство распространения [4], свойство неподвижных точек [10]. А с другой стороны, оно должно быть легко реализуемым, программными и аппаратными средствами на различных платформах, чтобы при незначительном расходе ресурсов обеспечить необходимое быстродействие. С практической точки зрения это задача актуальна, поскольку линейное отображение есть самый медленный компонент в симметричных шифрах. В настоящее время идет процесс создания отечественного алгоритма шифрования данных нового поколения, в котором проблема построения линейного отображения *большой размерности*, удовлетворяющего всем вышеприведённым свойствам и критериям, оказывается самой сложной. В данной статье рассматриваются вопросы аппаратной реализации заданного линейного отображения, представленного множеством ЛБФ.

Рассмотрим алгоритм минимизации числа логических элементов при реализации линейных отображений на основе структурированных данных, выраженных двоичным деревом [8], и алгоритм синтеза логических схем по полученному дереву. Построенный здесь алгоритм минимизации является усовершенствованным вариантом алгоритма, разработанного Н.П. Борисенко и Хоанг Дык Тхо в [6]. Разница между этими алгоритмами будет представлена ниже. В различных источниках описываются программы и алгоритмы, позволяющие минимизировать системы булевых функций. Типичная программа Logic Friday [9] позволяет строить логическую схему с минимальным количеством логических элементов. Но при числе булевых функций больше 16 программа уже не работает.

Линейное отображение, которое предстоит минимизировать, – это отображение в блочном шифре «Кузнечик» [7]. Это отображение $L:V_{128}\mapsto V_{128}$ строится на основе регистра сдвига с линейной обратной связью [2, 4] (РСЛОС – рис. 1) Фибоначчи в композиционном поле [3] $GF((2^8)^{16})$ с внутренним примитивным полиномом $f(x)=x^8\oplus x^7\oplus x^6\oplus x\oplus 1$ и внешним неприводимым полиномом

$$h(y)=y^{16}+148y^{15}+32y^{14}+133y^{13}+16y^{12}+194y^{11}+192y^{10}+y^9+ \\ +251y^8+y^7+192y^6+194y^5+16y^4+133y^3+32y^2+148y+1. \quad (1)$$

Под прямым линейным отображением $L:V_{128}\mapsto V_{128}$ исходного вектора данных $Q=\left(q_{15}\ q_{14}\cdots q_2\ q_1\ q_0\right)$, $q_i\in GF\left(2^8\right)$, $0\le i\le 15$ будем понимать 16 тактов работы РСЛОС (рис. 1). Обратная операция $L^{-1}:V_{128}\mapsto V_{128}$ будет выполняться аналогичным образом. При этом регистр работает в противоположном направлении.

Любое линейное отображение может быть представлено в виде бинарной матрицы. При этом каждая ее строка соответствует одной ЛБФ. В [5] был введен способ представления линейного отображения L в виде множества ЛБФ при рассмотрении влияния каждого бита входного вектора на биты выходного вектора. В результате чего получили n = 128 ЛБФ $f_0,...,f_{n-1}$.

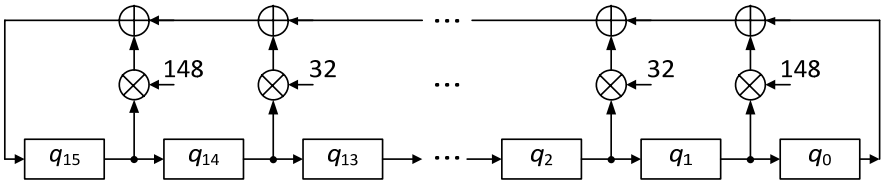


Рис. 1. РСЛОС Фибоначчи для прямого отображения *L*

Известно, что каждая линейная булева функция (ЛБФ) *f* может быть представлена единственным образом в виде полинома с коэффициентами из $GF(2)$ (полином Жегалкина или в алгебраической нормальной форме – (АНФ)) [1].

$$f(x)=a_0\oplus\sum_{i=1}^{n-1}a_i x_i\ , \tag{2}$$

где все коэффициенты a_i лежат в поле $GF(2)$ и $x_i\in\{0,1\}$, $\forall i=0,\ldots,n-1$. Обозначим набор коэффициентов функции в (2) через вектор $A=(a_0,\ a_1,\ldots,a_{n-1})$. Следовательно, вектор A является представлением *линейной* булевой функции (2) в виде полинома Жегалкина. Все булевы функции имеют *n* переменных. Но за счет их линейности удобно было их представить через *n*-мерный вектор. Вес Хемминга вектора *A*, обозначенный через $wt(A)$, равен количеству ненулевых его координат. Определим следующую функцию:

$$Nxor(A)=\begin{cases}wt(A)-1, & \text{если } wt(A)>1,\\ 0, & \text{если } wt(A)\le 1.\end{cases} \tag{3}$$

При этом количество требуемых операций сложения по модулю два (далее – «⊕») для реализации *i*-й ЛБФ равно

$$N_{A_i}=Nxor\left(A_i\right). \tag{4}$$

Если аппаратная реализация линейного отображения осуществляется путем независимой реализации его множества $D_0=\{A_0,\ A_1,\ldots,A_{n-1}\}$ векторов, каждый вектор соответствует одной ЛБФ, то количество операций «⊕» определяется как

$$N_{D_0}=\sum_{\mathrm{i}=0}^{n-1}N_{A_i}\ . \tag{5}$$

Однако более экономичной является совместная реализация векторов $\{A_0,\ A_1,\ldots,A_{n-1}\}$, составляющих линейное отображение *L*. Для уменьшения общего числа логических элементов, необходимых для реализации системы из *n* ЛБФ, т.е. для всего линейного отображения, необходимо учесть их повторяемость. Главная идея заключатся в разбиении исходного множества векторов D_0 на два подмножества D_1 и D_2, где подмножество D_1 содержит общие координаты соответствующих пар векторов из D_0, а D_2 содержит оставшиеся векторы, не содержащие ни одной пары с общими компонентами. После этого взять D_1 в качестве исходного множества ЛБФ и продолжить процесс разбиения на пары до тех пор, пока не останется общих компонентов. В результате получится бинарное дерево, в котором сохраняется информация о минимизации. Подробно об этом будет сказано ниже.

На каждом шаге выполнения алгоритма минимизации необходимо вычислить вектор, состоящий из общих координат текущей пары векторов и соответствующее им количество операций «⊕». Для этого рассмотрим пару векторов из коэффициентов ЛБФ:

$$A_i=(a_0,a_1,\ldots,\ a_{n-1}), \tag{6}$$

$$A_j=(b_0,b_1,\ldots,\ b_{n-1}). \tag{7}$$

Тогда, если $a_k=b_k$, $k=0,\ldots,n-1$, для некоторых *k,* то говорят, что векторы A_i,A_j имеют общий вектор, или, по-другому, две ЛБФ f_i,f_j имеют общие слагаемые. Этот вектор определяется так:

$$\nu=(A_i\ \&\ A_j)=(a_0\ \&\ b_0,a_1\ \&\ b_1,\ldots,a_{n-1}\ \&\ b_{n-1}), \tag{8}$$

где & – логическое умножение соответствующих коэффициентов векторов A_i и A_j. Назовем CO – количество общих операций «⊕» в паре $\{A_i, A_j\}$, CO определяется формулой

$$CO = Nxor(\nu). \tag{9}$$

При этом количество $N_{i,j}$ операций «⊕» для реализации пары A_i и A_j, вычисляется по формуле

$$N_{i,j} = N_{A_i} + N_{A_j} - CO \tag{10}$$

или

$$N_{i,j} = Nxor(A_i \oplus \nu) + Nxor(A_j \oplus \nu) + CO, \tag{11}$$

где $N_{A_i} = Nxor(A_i)$, $N_{A_j} = Nxor(A_j)$. Следовательно, схема реализации пары $\{A_i, A_j\}$ состоит из схем:

– оставшихся элементов в A_i (т.е. $A_i \oplus \nu$),

– оставшихся элементов в A_j (т.е. $A_j \oplus \nu$),

– общих элементов в векторе ν.

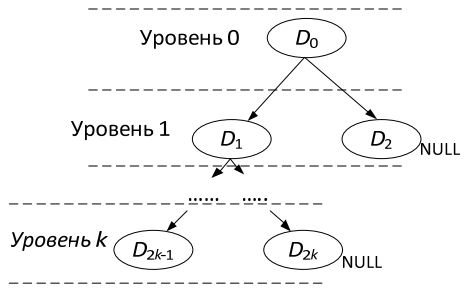


Рис. 2. Двоичное дерево в минимизации

Алгоритм минимизации логических элементов при аппаратной реализации линейных отображений. Схема полученного дерева изображена на рис. 2. Для его построения учтены следующие правила и обозначения.

Общие правила построения двоичного дерева: Дерево строится по уровням «сверху вниз» и по правилу «слева направо»:

- На уровне 0 сохраняется только исходное множество векторов ЛБФ D_0.
- На уровне 1 сохраняются два множества векторов: D_1 – левые наследники – множество из соответствующих векторов из D_0, содержащее общие координаты, и D_2 – множество оставшихся векторов после «удаления» из них общих координат (метод их получения рассмотрен ниже).
- На уровне k также сохраняются два множества векторов: D_{2k-1} – «левые наследники» множества общих векторов из подходящих векторов из $D_{2(k-1)-1}$ и D_{2k} – «правые наследники» множества оставшихся векторов.
- Все множества D_{2k}, $k \geq 1$ не имеют наследников.
- Множества, у которых имеются наследники, называются родителями.

Краткое обозначение векторов во всех множествах имеет вид $A_{w,i}$, а полное обозначение – $A_{w, z_i, r_{0,i}, r_{1,i}, i}$, где:

- w – указатель вектора, т.е. значение $w = 0$, $2k-1$ и $2k$ соответствует i-му вектору в множестве D_0 (уровень 0), D_{2k-1} (уровень k, левый наследник) и D_{2k} (уровень k, правый наследник) соответственно;
- z_i – название i-го вектора;
- $r_{0,i}$, $r_{1,i}$ – номера пары векторов родителей для i-го вектора;
- i – номер вектора в множестве.

Под описанием $z_{i,\ D}$, или $r_{0,i,D}, r_{1,i,D}$ понимается название или номера пары родителей i-го вектора в множестве D. Количество векторов в множестве D обозначают через N_D.

Значения $r_{0,i} = 0$, $r_{1,i} = 0$, $0 \leq i \leq n-1$ для всех векторов в D_0, так как у них нет пары векторов родителей. Названия векторов только в D_0 равны их номерам, т.е. $z_i = i$, а в других множествах они не равны. Значение N_{D_0} равно n.

Алгоритм минимизации количества логических элементов линейного отображения состоит из следующих шагов:

ВХОД: n полученных векторов полиномов Жегалкина множества ЛБФ $D_0 = \{A_0, A_1,\ldots, A_{n-1}\}$ заданного линейного отображения.

ВЫХОД: число общих логических элементов; множества векторов полного двоичного дерева (рис. 2).

Шаг 1 (блок 2, здесь и далее ссылка на рис. 3):

– вычислить число операций «⊕» для всех $(A_{0,i})$, i=0, …, n–1 по формулам (5), (т.е. вычислить общее количество операций «⊕» $Nxor_{D_0}$, необходимых при реализации L по отдельным векторам A_i);

– инициализация значения числа общих операций «⊕» $CO = 0$;

– инициализация уровня двоичного дерева $k = 1$;

– инициализация значения индекса $w = 0$ для векторов в множестве D_0.

Шаг 2 (блок 3):

– инициализация двух пустых множеств $D_{2k-1} = \varnothing$ и $D_{2k} = \varnothing$, которые находятся на уровне k как левые и правые наследники соответственно (см. рис. 2);

– инициализация значения $t = 0$ – индекс векторов в множестве D_{2k-1}.

Шаг 3 (блок 4):

– определить пару $\left(A_{w,z_i,r_{0,i},r_{1,i},i}, A_{w,z_j,r_{0,j},r_{1,j},j}\right)$, где $0 \le i, j \le n-1$ и $i \ne j$, такую, что значение $m = Nxor\left(A_{2k-1,z_t,r_{0,t},r_{1,t},t} = \left(A_{w,z_i,r_{0,i},r_{1,i},i} \& A_{w,z_j,r_{0,j},r_{1,j},j}\right)\right)$ принимает максимальное значение.

Шаг 4 (блок 5–7): проверяют условие $m \ge 2$:

– Если условие выполнено, то вычисленный вектор $A_{2k-1,z_t,r_{0,t},r_{1,t},t}$ является общим вектором в соответствии с выражением (8), в нем находятся общие координаты пары векторов $\left(A_{w,z_i,r_{0,i},r_{1,i},i}, A_{w,z_j,r_{0,j},r_{1,j},j}\right)$. В этом случае выполняются следующие действия:

- сохранить $A_{2k-1,z_t,r_{0,t},r_{1,t},t}$ как t-й вектор в множестве D_{2k-1} (блок 6);
- обновить значения пары векторов в множестве $D_w = \left\{A_{w,0}, A_{w,1},\ldots, A_{w,i},\ldots, A_{w,j},\ldots, A_{w,N_{D_w}-1}\right\}$, где $A_{w,i} = A_{w,i} \oplus A_{2k-1,t}$ и $A_{w,j} = A_{w,j} \oplus A_{2k-1,t}$ (Блок 6);
- вычислить название общего вектора в множестве D_{2k-1}: $z_{t,D_{2k-1}} = z_{N_{D_w}-1} + N_{D_w} + t + 1$ (блок 7);
- определить номера пары векторов, породившие данный общий вектор: $r_{0,\,t,\,D_{2k-1}} = z_{i,\,D_w}$, $r_{1,\,t,\,D_{2k-1}} = z_{j,\,D_w}$ (блок 7);
- вычислить общее количество «⊕» для данного общего вектора: $CO = CO + m - 1$ (блок 7);
- увеличить значение $t = t + 1$ для определения номера следующего общего вектора (блок 7);
- для другой пары векторов в D_w определить следующее максимальное значение $m = Nxor\left(A_{2k-1,z_t,r_{0,t},r_{1,t},t} = \left(A_{w,z_i,r_{0,i},r_{1,i},i} \& A_{w,z_j,r_{0,j},r_{1,j},j}\right)\right)$ (блок 7).

Процесс вычислений в шаге 4 продолжается до тех пор, пока пары векторов в D_w имеют общие координаты (т.е. условие $m \ge 2$ выполнено). В противном случае переходят к шагу 5 для определения других значений следующего уровня двоичного дерева.

Замечание: Для реального заданного линейного отображения $L: V_n \to V_n$, среднее число нулевых коэффициентов в его множестве ЛБФ (множество D_0) равно $\sum_{i=0}^{N_{D_0}-1} wt\left(A_{0,i}\right) \approx n \cdot (n/2) = n^2/2$, где n – количество ЛБФ или количество векторов, а $n/2$ – среднее количество переменных в каждой ЛБФ. После выполнения шага 4, если значение $m \ge 2$, то вес Хемминга векторов $A_{0,i}$ и $A_{0,j}$

равен $wt(A_{0,i})-m$ и $wt(A_{0,j})-m$ соответственно. Значит, после l выполнений шага 4, если все $m\geq 2$, среднее количество ненулевых коэффициентов в D_0 равно $(n^2/2)-2m\cdot l$. Значение l – число возможных пар, имеющих общие координаты ($m\geq 2$). Так как

$$(n^2/2)-2m\cdot l\geq 0\Leftrightarrow l\leq\frac{n^2}{4m}. \qquad (12)$$

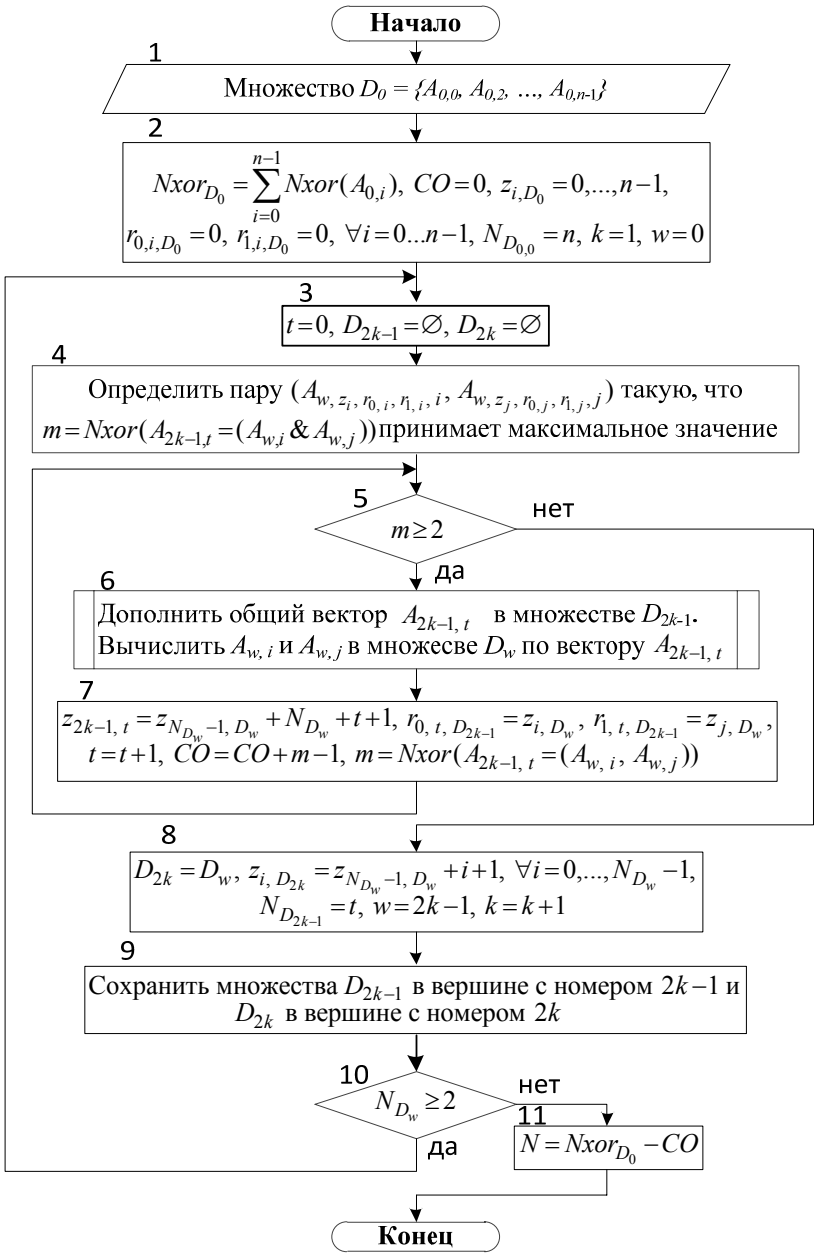


Рис. 3. Блок-схема алгоритма вычисления числа операций «⊕»
и определения данных для двоичного дерева

Максимальное значение $l_{\max}\leq\frac{n^2}{8}$ получается при $m=2$. Это приводит к тому, что число общих векторов (множество D_1), полученных в шаге 4, не превышает $l_{\max}=n^2/8$ (т.е. $0\leq t\leq(n^2/8)-1$. Следовательно, множество D_{2k-1} всегда определится на каждом уровне двоичного дерева.

Шаг 5 (блок 8):

– вычислить число векторов в множестве D_{2k-1}: $N_{D_{2k-1}} = t$;

– определить множество $D_{2k} = D_w$. Множество D_{2k} содержит оставшиеся векторы, у которых нет общих коэффициентов;

– присвоить новые номера для всех векторов в D_{2k} по формуле: $z_{l,D_{2k}} = z_{N_{D_w}-1,\,D_w} + l + 1$, $\forall l = 0,...,N_{D_w} - 1$;

– вычислить значение $w = 2k - 1$;

– увеличить значение на единицу: $k = k + 1$, т.е. переходят на следующий уровень в двоичном дереве (рис. 2).

Шаг 6 (блок 9): Сохранить множества D_{2k-1} в вершине с номером $2k-1$ и D_{2k} в вершине с номером $2k$ двоичного дерева.

Шаг 7 (блок 10): Повторять шаги 2–6 для множества D_w до тех пор, пока количество векторов в D_{2k-1}: $N_{D_{2k-1}} \geq 2$.

Шаг 8 (блок 11): Вычислить количество общих операций «⊕» для реализации заданного линейного отображения по формуле $N = Nxor_{D_0} - CO$ и закончить алгоритм.

Так как вес Хемминга векторов в D_w уменьшается в ходе выполнения минимизации, то алгоритм всегда сходится, т.е. удовлетворяет свойству результативности. Блок-схема этапов вычисления числа операций «⊕» и определения данных по алгоритму минимизации представлена на рис. 3.

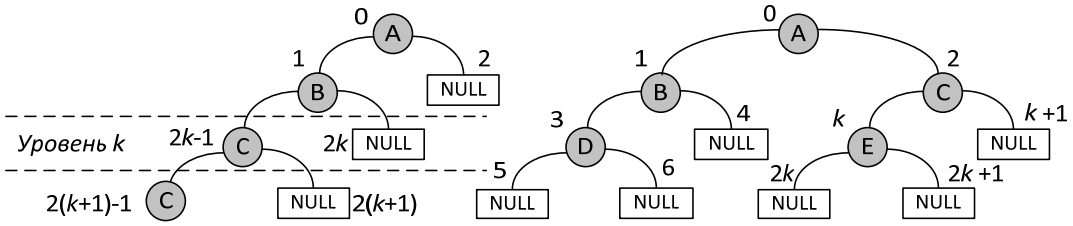
В [6] Н.П. Борисенко и Хоанг Дык Тхо предложили алгоритм минимизации логических схем для реализации *S*-блоков. Наш алгоритм минимизации, разработанный для реализации линейного отображения, является дальнейшим его развитием. В [6] булевы функции являются нелинейными. Количество булевых функций небольшое, но их вес Хемминга значительно больше, чем в нашем случае. Идея алгоритма также основана на разделении исходного множества D_0 векторов, составляющих *S*-блоки, на два подмножества D_1 и D_2. Результатом работы алгоритма также является двоичное дерево, в котором сохраняется вся информация о минимизации.

Однако в алгоритме [6] после каждого вычисления общего вектора $A_{2k-1,t} = (A_{w,i} \,\&\, A_{w,j})$ (см. шаг 4) пара новых векторов $(A_{w,\,i} \oplus A_{2k-1,\,t},\, A_{w,\,j} \oplus A_{2k-1,\,t})$ не участвует в процессе вычисления следующих значений общего вектора $A_{2k-1,\,t+1}$. Значит, общий вектор создается только между двумя оригинальными векторами в множестве D_w. Это снижает количество общих логических «⊕» *CO*, так как значение *m* (см. шаг 4) между оригинальным вектором в D_w и одним из новых векторов $(A_{w,\,i} \oplus A_{2k-1,\,t},\, A_{w,\,j} \oplus A_{2k-1,\,t})$ может быть больше, чем между двумя оригинальными векторами в множестве D_w. Наш алгоритм учитывает это. Кроме того, в [6] пара новых векторов $(A_{w,\,i} \oplus A_{2k-1,\,t},\, A_{w,\,j} \oplus A_{2k-1,\,t})$ удаляется от множества D_w, они дополняются в множестве D_{2k} (множество оставшихся векторов). Следовательно, на следующем этапе необходимо вычислять наследников не только для множества D_{2k-1}, но и для множества D_{2k}. Примеры деревьев, полученных алгоритмами, представлены на рис. 4. Все это усложняет процесс реализации на языках программирования и процесс синтеза логических схем. В [6] за счет удаления пары новых векторов $(A_{w,\,i} \oplus A_{2k-1,\,t},\, A_{w,\,j} \oplus A_{2k-1,\,t})$ от D_w максимальное количество $N_{D_{2k-1}}$ общих векторов в D_{2k-1} в два раза меньше количества векторов в D_w, а в нашем $N_{D_{2k-1}}$ во много раз больше, чем N_{D_w}. Это приводит к расходу памяти для сохранения D_{2k-1}.

В качестве примера рассмотрим линейное отображение $Y = M \cdot X$, где $X = (x_0, x_1, ..., x_7)$, $Y = (y_0, y_1, ..., y_7)$, $x_i, y_i \in GF(2)$, $0 \leq i \leq 7$, матрица M и соответствующее ей множество ЛБФ заданы выражением (13) и в виде векторов (14):

$$M=\begin{pmatrix}0&1&0&0&0&1&1&0\\0&0&1&0&1&0&1&1\\1&0&0&1&0&1&0&1\\1&0&0&0&1&1&0&0\\1&1&0&0&1&1&1&0\\0&1&1&0&1&1&1&1\\1&0&1&1&0&1&1&1\\1&0&0&1&1&1&0&1\end{pmatrix}\Leftrightarrow\begin{cases}f_0=x_1\oplus x_5\oplus x_6\\f_1=x_2\oplus x_4\oplus x_6\oplus x_7\\f_2=x_0\oplus x_3\oplus x_5\oplus x_7\\f_3=x_0\oplus x_4\oplus x_5\\f_4=x_0\oplus x_1\oplus x_4\oplus x_5\oplus x_6\\f_5=x_1\oplus x_2\oplus x_4\oplus x_5\oplus x_6\oplus x_7\\f_6=x_0\oplus x_2\oplus x_3\oplus x_5\oplus x_6\oplus x_7\\f_7=x_0\oplus x_3\oplus x_4\oplus x_5\oplus x_7\end{cases}, \quad (13)$$

$$D_0=\{A_{0,0},A_{0,1},A_{0,2},A_{0,3},A_{0,4},A_{0,5},A_{0,6},A_{0,7}\}=\{\{0,1,0,0,0,1,1,0\},\{0,0,1,0,1,0,1,1\},\{1,0,0,1,0,1,0,1\},$$
$$\{1,0,0,0,1,1,0,0\},\ \{1,1,0,0,1,1,1,0\},\ \{0,1,1,0,1,1,1,1\},\ \{1,0,1,1,0,1,1,1\},\ \{1,0,0,1,1,1,0,1\}\}. \quad (14)$$



а *б*

Рис. 4. Пример двоичного дерева: *а* – по разрабатываемому алгоритму и *б* – по алгоритму в [6]

Количество требуемых логических операций «⊕» при реализации по каждой ЛБФ вычисляется по формуле (5) и равно $N_{D_0}=\sum_{i=0}^{7}N_{A_{0,i}}=28$.

Применив разработанный алгоритм минимизации к D_0, получим двоичное дерево (рис. 5).

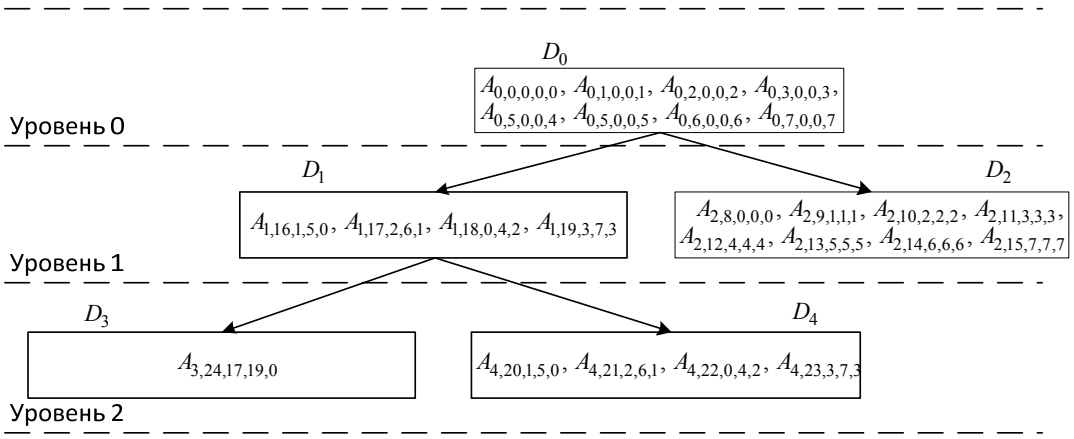


Рис. 5. Полное двоичное дерево линейного отображения (13)

Количество общих логических операций «⊕» *CO* равно 11. Следовательно, количество логических операций «⊕» после минимизации равно $N_{D_{0,0}}-CO=28-11=17$.

Ясно, что задача определения общих элементов при совместной реализации множества всех векторов $D_0=\{A_0,A_2,\ldots,A_{n-1}\}$ может быть выполнена различными способами. В общем случае требуется тотальный перебор (*из всех возможных сочетаний* C_n^k, где $k=2,\ldots,n$). Очевидно, что при реальных линейных отображениях такая задача становится неразрешимой. Кроме того, после нахождения всех сочетаний повторяющихся элементов необходимо еще синтезировать логическую схему реализации линейных булевых функций. Как показал опыт, это тоже очень сложная проблема. Наш алгоритм решает ее, так как он базируется на структурированных данных для построения дво-

ичного дерева, которое оказалось удобным для синтеза логических схем линейных отображений большой размерности. В следующем разделе рассмотрим алгоритм синтеза логических схем линейных отображений по результатам работы алгоритма минимизации.

Алгоритм синтеза логических схем линейных отображений. Для синтеза схемы, на основе полученных результатов (оставшихся и общих компонентов) после применения алгоритма минимизации, будем двигаться по направлению «снизу вверх» и «слева направо». Это процесс программирования при чтении информации двоичного дерева, полученного после применении алгоритма минимизации. В общем алгоритм можно разделить на 2 этапа:

Этап 1: Строят логическую схему для нижнего уровня *k*: сначала получают логическую схему для общих компонентов в множестве D_{2k-1} (левые наследники), а потом – схему для оставшихся компонентов в D_{2k} (правые наследники).

Этап 2: По названиям векторов и их соответствующим номерам пар родителей строят логическую схему для уровней с номерами от $k-1$ до $k=0$. Также по правилу «слева направо». При этом схема для левого наследника уровня $k-1$ синтезируется схемой левого и правого наследников уровня k.

Блок-схема синтеза линейного отображения по дереву представлена на рис. 6.

ВХОД: двоичное дерево, построенное алгоритмом, представленным на рис. 3.

ВЫХОД: схема реализации линейных булевых функций заданного линейного отображения с минимизированным количеством логических элементов.

Шаг 1 (блок 2): Сначала строят схему каждого вектора $A_{2k-1, z_i, r_{0,i}, r_{1,i}, i}$, где $0 \le i \le N_{D_{2k-1}} - 1$, в множестве D_{2k-1} для левых наследников уровня *k*, а потом – схему каждого вектора $A_{2k, z_j, r_{0,j}, r_{1,j}, j}$, где $0 \le j \le N_{D_{2(k-1)-1}} - 1$, в множестве D_{2k}.

Шаг 2 (блок 3): Присваивая $p = k-1$, где p – номер уровня, переходят к синтезу следующего уровня.

Шаг 3 (блок 4): Проверяют условие $p \ge 0$. Если условие выполнено, то переходят к выполнению операций в блоке 5. Иначе алгоритм синтеза заканчивается.

Шаг 4 (блок 5): Инициализация значений: $i = 0$, $w = 2p - 1$ – индекс множества D_w для левых наследников уровня p и $w = 0$ для множества D_0, и $q = p + 1$ – номер предыдущего уровня.

Шаг 5 (блок 6): Проверяют условие $p = 0$, чтобы определить значение индекса w.

Шаг 6 (блок 8): Строят схему каждого вектора в D_w проверкой условия $i < N_{D_w}$. Если условие выполнено, то переходят к блоку 9, иначе – к блоку 16.

Шаг 7 (блок 9): Осуществляется синтез схемы вектора $A_{w,\,i}$ в множестве D_w уровня *p* на основе схемы вектора $A_{2q,i}$ в множестве D_{2q}, т.е. $A_{w,i} = A_{2q,i}$. Именно в этом шаге выполняется синтез каждого вектора в D_w уровня $p = k - 1$ оставшимися компонентами в D_{2q} уровня $p + 1$.

Шаг 8 (блок 10): Инициализация значения $j = 0$.

Шаг 11 (блок 13): Дополняют схему вектора $A_{w,\,i}$ в D_w, полученную на шаге 7 (блок 9), схемой вектора $A_{2q-1,j}$ в D_{2q-1}. В результате получают вектор $A_{w,i} = A_{2q,i} \oplus A_{2q-1,j}$.

Шаг 12 (Блок 14): Увеличивая $j = j + 1$, выполняют дополнение схемы на шагах 9–11. Переходят к шагу 9 (блок 11).

Шаг 9 (блок 11): Проверяют условие $j < N_{D_{2q-1}}$. Если условие выполнено, то переходят к блоку 12, иначе – блоку 15.

Шаг 10 (блок 12): Проверяют выполнение условия $z_{i,D_w} = r_{0,j,D_{2q-1}}$ или $z_{i,D_w} = r_{1,j,D_{2q-1}}$, т.е. проверяют условие того, что вектор в D_w является вектором родителя вектора в D_{2q-1}. Если условие выполнено, то переходят к блоку 13, иначе – к блоку 14.

Шаг 13 (блок 15): Увеличивая $i = i + 1$, продолжают строить схему следующего в D_w на уровне p. Переходят к шагу 6 (блок 8).

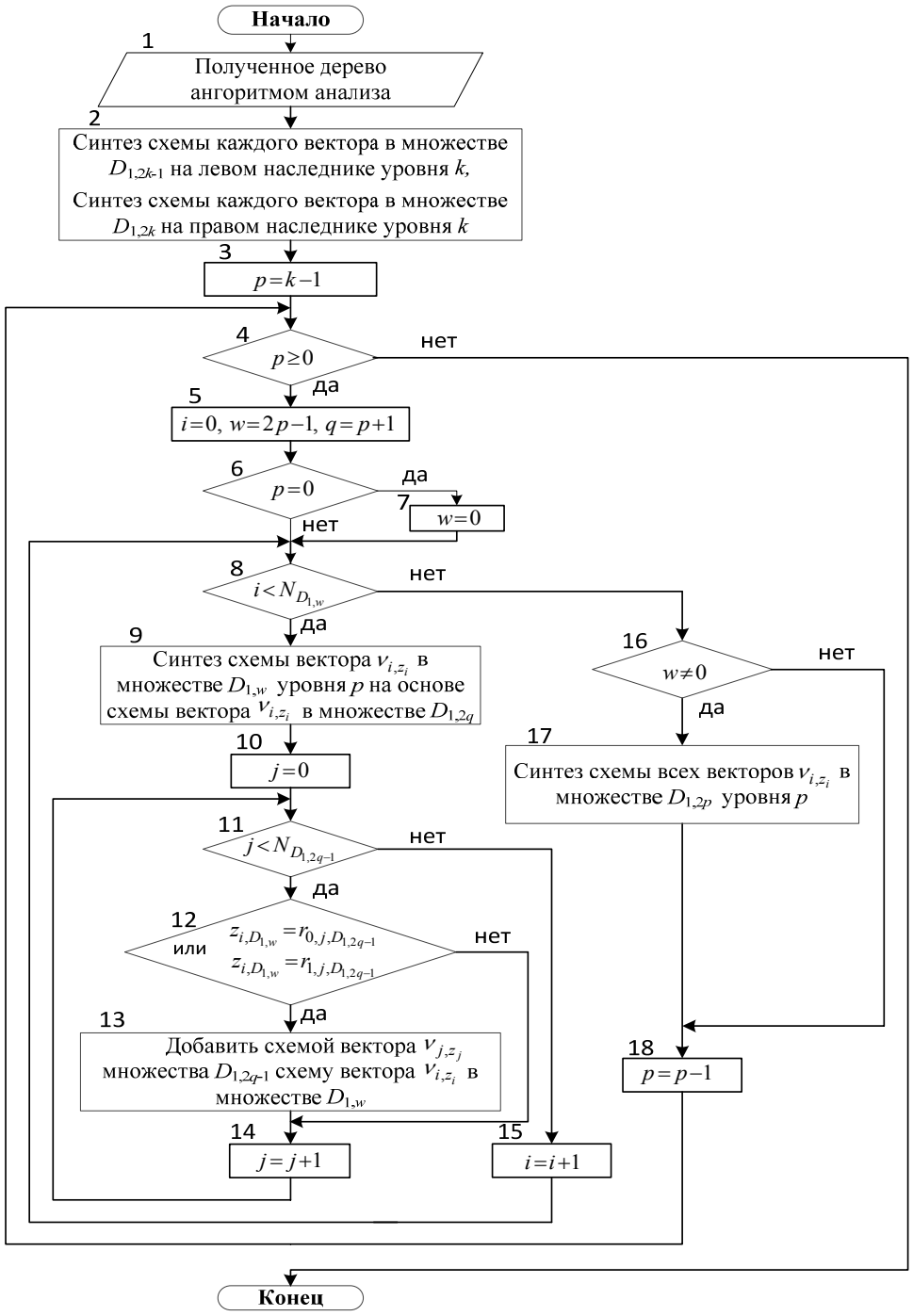


Рис. 6. Блок-схема синтеза логической схемы линейного отображения по двоичному дереву

Шаг 14 (блок 16): Проверяют условие $w\neq 0$. Если условие выполнено, то осуществляется синтез схем всех векторов в D_{2p} (блок 17), т.е. строят схемы всех векторов правых наследников уровня p. Переходят к блоку 18.

Шаг 15 (блок 18): Выполняя $p=p-1$, осуществляется синтез следующего уровня. Переходят к шагу 3 (блок 4).

Применяя алгоритм синтеза к дереву, полученному в примере (см. рис. 5), получают логическую схему данного линейного отображения (рис. 7). Для удобства на рис. 7 обозначены $x_i\in GF(2),\ i=0,...,7$ – восемь входных переменных, вектор во множестве левого наследника (D_{2k-1}), имеющий номер z, обозначается через L_z, вектор правых наследников (D_{2k}) – R_z, а вектор в D_0 – f_z.

Результаты эксперимента. Здесь представлены результаты применения разработанного алгоритма минимизации для реализации линейного отображения рис. 1 блочного шифра «Кузнечик» [7].

Если его построить по каждой ЛБФ, то количество требуемых логических операций «⊕» равно 7879 [5]. Применив алгоритм минимизации, предложенный в [6], количество требуемых «⊕» не превышает 3276, а при применении разработанного алгоритма – 3146.

Если на основе внешнего неприводимого полинома (1) построить линейное отображение по схеме РСЛОС Галуа, то количество требуемых «⊕» также равно 7879 [5]. Применив алгоритм минимизации [6], количество требуемых «⊕» не превышает 3297, а при применении разработанного алгоритма – 3149.

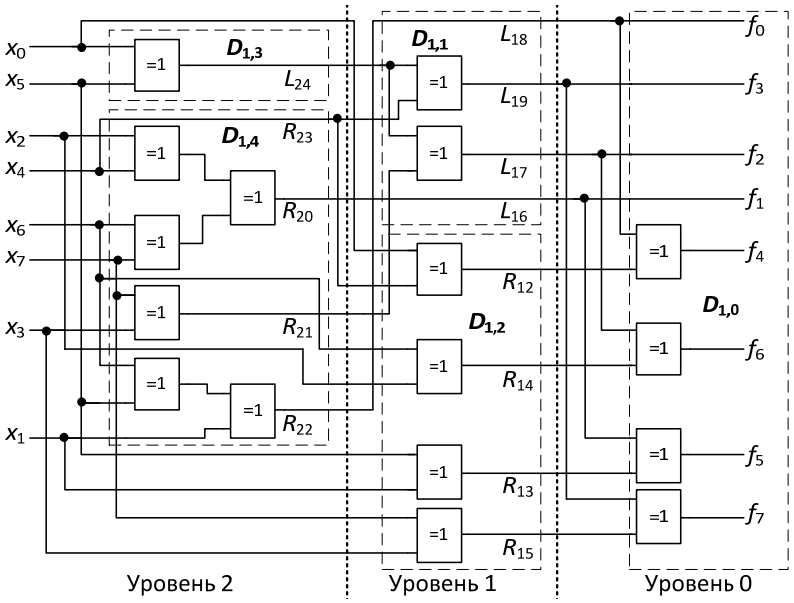


Рис. 7. Синтезированная схема реализации линейного отображения для рассмотренного примера с минимизированным количеством логических элементов

Глубина схемы линейного отображения шифра «Кузнечик» при реализации по отдельным ЛБФ равна $\log_2 128 = 7$. Полученная по алгоритмам минимизации и синтеза схема имеет глубину 15.

Дальнейшее повышение эффективности рассмотренных алгоритмов возможно путем учета совпадающих координат не только в парах, но и в тройках, четверках и т.д. векторов. Однако сложность алгоритма анализа растет почти экспоненциально. О процессе синтеза сложно что-либо сказать, т.к. прямой перенос идеи «двоичного дерева» не совсем очевиден. Кроме того, как показано в [6], тотальный перебор несущественно изменил достижимую границу минимизации, а глубина схемы растет существенно.

Заключение. В статье рассмотрены алгоритм минимизации числа логических элементов при реализации линейных отображений и соответствующий ему алгоритм синтеза логических схем этих отображений. Разработанные алгоритмы применены для минимизации системы ЛБФ, составляющих линейные отображения, но они могут быть использованы и для минимизации системы нелинейных булевых функций. То есть эффективность применения алгоритмов зависит не от вида булевых функций, а от веса векторов, при их представлении в виде полиномов Жегалкина.

Алгоритм минимизации является усовершенствованным вариантом алгоритма, предложенного в [6]. Изменение алгоритма позволяет получить простое двоичное дерево, что облегчает процесс синтеза логических схем линейных отображений и уменьшает количество необходимых логических элементов «⊕».

Анализ полученных результатов для минимизации линейного отображения шифра «Кузнечик» показывает, что уменьшение числа требуемых логических элементов «⊕» влечет за собой увеличение глубины схемы. Количество логических «⊕» при минимизации уменьшается больше чем в 2 раза и глубина схемы увеличивается во столько же, по сравнению со случаем, когда отображение строится по отдельным ЛБФ.

Литература

1. Логачёв О.А. Булевы функции и теории кодирования и криптографии / О.А. Логачёв, А.А. Сальников, В.В. Ященко. – М.: МЦНМО, 2004. – 470 с.

2. Кузьмин А.С. Линейные рекуррентные последовательности над кольцами Галуа / А.С. Кузьмин, А.А. Нечаев // Алгебра и логика. – 1995. – Т. 3, № 2. – С. 169–189.

3. Paar C. Efficient VLSI Architectures for Bit-Parallel Computation in Galois Fields: Ph.D. Theses. – Essen, Germany: Universität, Institute for experimental mathematics, 1994.

4. Зензин О.С. Стандарт криптографической защиты AES. Конечные поля / О.С. Зензин, М.А. Иванов; под ред. М.А. Иванова. – М.: КУДИЦ-ОБРАЗ, 2002. – 175 с.

5. Borisenko N.P. Developing Algorithm for Software and Hardware Implementation of Large Size Linear Mapping / N.P. Borisenko, V.L. Nguyen, A.A. Bulygin. // 2nd Workshop on Current Trends in Cryptology (CTCrypt 2013). – June 23–25, 2013, Ekaterinburg, Russia. Pre-proceedings. – Ekaterinburg, 2013. – P. 192–204.

6. Borisenko N.P. Algorithm for minimization of the number of logic elements in a circuit implementing S-box Boolean functions / N.P. Borisenko, Hoang Duc Tho. 3nd Workshop on Current Trends in Cryptology (CTCrypt 2014). – June 5–6, 2014, Moscow, Russia. Pre-proceedings. – Moscow, 2014. – P. 125–144.

7. Low-Weight and Hi-End Draft Russian Encryption Standard / V. Shishkin, D. Dygin, I. Lavrikov et al. // 3nd Workshop on Current Trends in Cryptology (CTCrypt 2014). – June 5–6, 2014, Moscow, Russia. Pre-proceedings. – Moscow, 2014. – P. 183–188.

8. Гагарина Л.Г. Алгоритмы и структуры данных: учеб. пособие / Л.Г. Гагарина, В.Д. Колдаев. – М.: Финансы и статистика; ИНФРА-М, 2009. – 304 с.

9. Программа Logic-Friday [Электронный ресурс]. – Режим доступа: http://www.logic-friday.software.informer.com, свободный (дата обращения: 03.2014).

10. Muhammad Reza Z'aba. Analysis of Linear Relationship in Block Ciphers / Muhammad Reza Z'aba. Bachelor of science (Computer). – Kuala Lumpur: Universiti Teknologi Malaysia, 2003. – 256 p.

Борисенко Николай Павлович
Канд. техн. наук, доцент, сотр. Академии Федеральной службы охраны (ФСО) России
Тел.: (486-2) 54-99-33
Эл. почта: npbor@yandex.ru

Нгуен Ван Лонг
Инженер, сотр. Академии ФСО России
Тел.: 8-903-637-73-93
Эл. почта: nvlong.bcy@gmail.com

Borisenko N.P., Nguyen V.L.
Minimization algorithms of number logic elements in implementation of linear mappings

In the article, we propose an efficient algorithm to minimize the number of logic elements in the hardware implementation of large size linear mapping, presented by linear Boolean functions (LBF). The essence of the algorithm is to use the structural data on the description of the binary tree to determine the overall logic elements XOR. An algorithm based on obtained binary tree is constructed to synthesize logic schema.
Keywords: linear mapping, Boolean function, minimization, implementation.
