УДК 621.391.837:681.3

А. Х. Мурсаев
Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В. И. Ульянова (Ленина)

Реализация функционального преобразования в микросхемах программируемой логики

Функциональное преобразование, т. е. формирование кода, численный эквивалент которого есть заданная функция от численного эквивалента входного кода, является широко распространенной операцией в измерительных, управляющих и подобных системах. Реализация таких преобразований в вычислителях общего назначения (например, микропроцессорах) зачастую не удовлетворяет требованиям потребителей по производительности или энергоемкости. Приводится обзор методов функционального преобразования как аппроксимационных (плоская, иначе табличная, аппроксимация, полиномиальная аппроксимация), так и не обладающих методической погрешностью (вычисления по методу цифра за цифрой), а также их реализация в специализированных структурах, в том числе микросхемах программируемой логики. Рассмотрены варианты пространственного и временного распределения вычислений в их сравнении при использовании современной технической базы. Показано, что при реализации в микросхемах программируемой логики и умеренных затратах достижимы показатели производительности до 10^7 преобразований в секунду.

Функциональное преобразование, аппаратурная реализация, программируемые логические интегральные схемы

Функциональное преобразование, т. е. вычисление функции одного аргумента $y = F(x)$, – важная, но весьма трудоемкая составляющая выполнения многих алгоритмов.

В 1970–80 гг. аппаратным реализациям функционального преобразования уделялось весьма пристальное внимание, но в дальнейшем в связи с развитием и внедрением микропроцессорной техники интерес к проблемно-ориентированным вычислительным средствам значительно сократился.

В настоящее время для выполнения многих задач, требующих повышенной производительности (обработка сигналов, управление в реальном времени и т. д.), широко применяются специализированные архитектуры, базирующиеся на структурной организации вычислений (использование специализированных блоков, распараллеливание, конвейеризация). Современные технологии и номенклатура изделий микроэлектроники, особенно программируемые логические интегральные схемы (ПЛИС), позволяют строить специализированные вычислители, которые во многих применениях превосходят устройства на базе универсальных процессоров по соотношению производительности и стоимости, а также массогабаритным показателям.

Однако и сейчас аппаратный подход к реализации функционального преобразования используется недостаточно и разработчики предпочитают выполнять функциональные преобразования в программируемом ядре системы. В [1], посвященной аппаратным реализациям задач, прямо высказано мнение, что удобны для аппаратной реализации лишь алгоритмы, «не содержащие вычислений трансцендентных функций».

Попробуем с новых позиций взглянуть на «хорошо забытое старое».

В соответствии с классификацией проф. В. Б. Смолова [2] цифровые функциональные преобразователи можно отнести к следующим классам:

– табличные;

– алгоритмические;

– таблично-алгоритмические.

Табличные методы реализуются схемами с памятью или через логическую декомпозицию.

Алгоритмические обычно строятся как модель некоей (часто физической) системы, описы-

ваемой уравнениями, подобными заданным. Они представлены широким разнообразием алгоритмов и структурных решений (цифровые дифференциальные анализаторы [3], [4], цифро-время-импульсные [5], [6], развертывающие [4], [7] и др.), но в данной статье мы ограничимся полиномиальной аппроксимацией и реализаций вычислений методом «цифра за цифрой» как наиболее универсальным и обеспечивающим хорошие показатели по критерию затраты/производительность. Общим для этих методов является возможность рекуррентного построения вычислительной процедуры, что порождает сходные структурные решения. Причем для обоих методов можно предложить сходные ряды структур, члены которых отличаются по производительности и аппаратурным затратам. Таблично-алгоритмические выполняют некий алгоритм с использованием настроек (например, коэффициентов), выбираемых из дополнительных таблиц.

Табличные функциональные преобразователи. Реализация табличного представления достаточно тривиальна. Значения функции для всех допустимых аргументов заносятся в память в порядке возрастания численного эквивалента кода аргументов. На адресный вход памяти подается код аргумента, тогда на выходе получим отсчет функции этого аргумента.

Такой подход следует считать наиболее эффективным для устройств с относительно невысокой точностью (до 16–18 двоичных разрядов аргумента, что соответствует относительной погрешности воспроизведения $2 \cdot 10^{-5} \ldots 4 \cdot 10^{-6}$). Но даже в автономных измерительных и управляющих системах такая точность может оказаться недостаточной. Например, при преобразовании системы координат в системах управления движением или компьютерной графики приходится иметь дело со слабо обусловленными системами уравнений преобразования, при решении которых погрешности многократно усиливаются и накапливаются.

При высоких требованиях по точности и разрядности аргументов табличная реализация требует значительных объемов памяти. И хотя современные ПЛИС (программируемые логические интегральные схемы) содержат блоки памяти сравнительно большой емкости, затраты могут оказаться чрезмерными и другие подходы получают преимущества. При этом в ПЛИС имеются достаточные ресурсы для имплементации экономичных и высокопроизводительных блоков функционального преобразования на базе иных подходов.

Полиномиальные аппроксиматоры. В диапазоне требований по относительной погрешности $10^{-6} \ldots 10^{-8}$ (20–25 разрядов) наиболее эффективными представляются полиномиальные аппроксиматоры.

При выборе способа аппроксимации используют аппроксимацию отрезком степенного ряда или оптимальное (чебышевское или среднеквадратическое) приближение. Преимуществом аппроксимации отрезком степенного ряда служит наличие простых рекуррентных алгоритмов вычисление очередного коэффициента ряда. Например, любой коэффициент ряда Тейлора для вычисления функции

$$Z = \exp(x)$$

выражается как

$$a_i = 1/i!$$

и легко определяется в процессе вычислений по рекуррентной формуле

$$a_i = a_{i-1}/i.$$

Подобные соотношения известны для многих аналитических функций.

Однако при реализации в ПЛИС коэффициенты аппроксимирующего полинома a_i предпочтительно вычислить заранее и сохранять в постоянной памяти или в форме соответствующей коммутации. При таком подходе структура устройства слабо зависит от способа аппроксимации. Оптимальные приближения обеспечивают такую же точность при меньшем порядке полинома по сравнению с разложением в ряд и в этом смысле при реализации в ПЛИС предпочтительны, хотя структурно решения эквивалентны.

Воспользуемся представлением аппроксимирующего полинома в виде схемы Горнера:

$$Z = a_0 + x\left\{a_1 + x \times \left[a_2 + x(a_3 \ldots + x\, an)\ldots\right]\right\}.$$

Для последовательной реализации это соотношение удобно представить в рекуррентной форме:

$$z_0 = a_n;$$
$$z_i = a_n - i + xz_i - 1;\ i = 1 \ldots n;$$
$$Z = z_n,$$

где z_i – приближенное значение функции после i-й итерации.

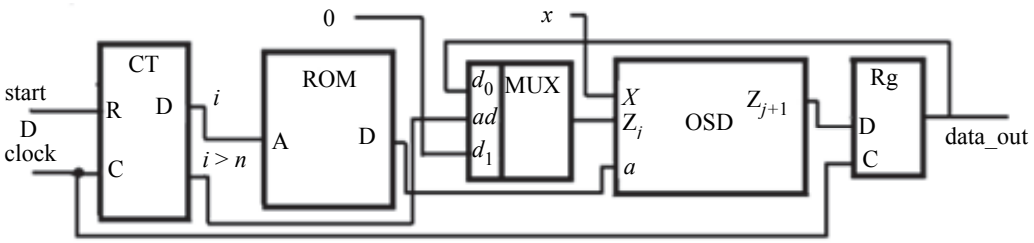

Рис. 1

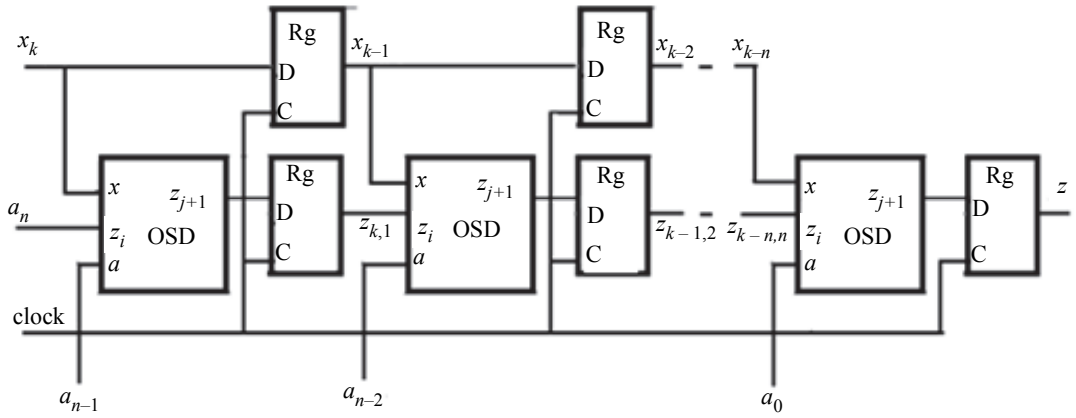

Рис. 2

Реализация блока, реализующего один любой шаг алгоритма, представляется очевидной – умножитель-накопитель является типовым узлом большинства развитых ПЛИС. На приведенных далее рисунках этот блок обозначен OSD (One Step Device). Однако выполнять совокупность шагов преобразования можно разными способами, и это порождает разнообразные структуры, отличающиеся аппаратурными затратами и производительностью [1], [8].

Наиболее компактную, но, соответственно, относительно «медленную» реализацию обеспечивает «итеративная декомпозиция» (ИД по терминологии [1]). Все вычисления выполняются в одном блоке за несколько шагов, при этом исходными данными для каждого шага служат результаты выполнения предыдущего. Типовая структура блока, реализующего ИД, представлена на рис. 1, где OSD реализует один шаг алгоритма преобразования. Счетчик CT, получив сигнал start, означающий появление нового отсчета аргумента, переходит в исходное состояние, после чего пробегает множество состояний от 0 до n, а затем затормаживается. Выход счетчика (содержит номер такта i) обеспечивает выборку из памяти ROM соответствующего коэффициента аппроксимации. OSD складывает очередной коэффициент с произведением аргумента преобразования на результат предыдущего шага.

При этом результат каждого шага фиксируется в регистре RG и становится исходным данным для следующего шага. На первом шаге вместо такого произведения за счет переключения мультиплексора MUX подается ноль.

Конвейерная схема на примере вычислителя полинома представлена на рис. 2. Здесь x_k – отсчет в последовательности входных данных ($z_{r,\text{p}}$) и результат вычисления p членов аппроксимирующего полинома для r-го отсчета. Каждый решающий блок выполняет один шаг алгоритма и в каждый момент использует не свои результаты, а результаты, сформированные его предшественником в предыдущем такте работы. Рассмотрим некоторый интервал $t_k - t_{k+1}$. На вход в этом интервале поступил отчет x_k последовательности входных данных. Легко проверить, что каждый i-й OSD получает на вход результат, соответствующий вычислению i–1 первых шагов преобразования входного отсчета x_{k-i}. Результаты всех ступеней синхронно фиксируются в регистрах. Таким образом, одновременно обрабатывается n отсчетов входного потока данных.

Устройство, как и устройство, представленное рис. 1, выдает результат через n тактов после поступления нового данного. Но в системах, характеризуемых плотным потоком поступающих отсчетов, конвейерные схемы обеспечивают максимальную производительность потому, что отсче-

ты могут поступать с периодом лишь незначительно превышающем время преобразования в одной ступени. Затраты, конечно, возрастают, но при погрешности аппроксимации порядка 10^{-6} (разрядности операндов 20–25) в большинстве случаев достаточен полином менее 8-го порядка, и для реализации в ПЛИС затраты не кажутся катастрофическими. Интервал между поступлениями данных лишь незначительно превышает время умножения. Причем этот интервал может быть еще более сокращен при конвейризации умножителя.

Потоковые схемы, в которых между ступенями отсутствуют регистры, обеспечивают минимальную задержку получения результата, но производительность при плотных потоках данных значительно ниже, так как новые данные могут поступать только после завершения переходных процессов, необходимых для получения результата для предшествующего отсчета.

Возможно использование смешанных структур для оптимального удовлетворения конкретных требований по затратам или производительности. Ступенями конвейера могут быть не только исполнители одного шага, но и блоки, каждый из которых исполняет несколько шагов преобразования с использованием ИД, передавая промежуточные результаты следующему подобному блоку, реализующему совокупность следующих шагов.

Таблично-алгоритмические вычислители. Чаще всего в устройствах этого типа предполагается «замешивание» результатов функциональных преобразований для групп разрядов. Пусть аргумент представлен в виде $a + b$, причем слагаемые представляют числовые эквиваленты содержимого старших и младших разрядов с учетом их относительных весов. Тогда, например,

$$\begin{aligned} \exp(a+b) &= \exp(a)\times(b); \\ \sin(a+b) &= \sin(a)\cos(b)+\sin(b)\cos(a); \\ \cos(a+b) &= \cos(a)\cos(b)-\sin(a)\sin(b). \end{aligned} \quad (1)$$

При этом значительно сокращается объем таблиц. Однако такие относительно простые разложения известны лишь для небольшого числа аналитических функций.

Более универсальный подход предполагает использование некоторого вида кусочной аппроксимации. Наиболее широко известна кусочно-полиномиальная. Диапазон изменений аргумента разбивается на поддиапазоны, нумеруемые численным эквивалентом старших разрядов аргумента. Во всех диапазонах используется одинаковое аппроксимирующее выражение (функция от численного эквивалента содержимого оставшихся младших разрядов), но для каждого поддиапазона используются свои коэффициенты аппроксимации, выбираемые из блока памяти в соответствии с тем, в каком диапазоне находится аргумент. Сокращение интервала аппроксимации позволяет при той же требуемой точности использовать более простое аппроксимирующее выражение. Так при полиномиальной аппроксимации удается снизить порядок аппроксимирующего полинома, а значит, и число умножений. Можно использовать структуры, подобные структурам полиномиальных аппроксиматоров. При этом блоки памяти хранят коэффициенты аппроксимации для различных диапазонов, а диапазон определяется старшими разрядами аргумента.

Однако при реализации в ПЛИС выигрыш кусочной аппроксимации по сравнению со сквозной далеко не однозначен. Если для реализации методом ИД существенных отличий по оборудованию устройств, использующих сквозную или кусочную аппроксимацию, нет, а производительность за счет сокращения числа умножений все же выше, то в потоковых реализациях приходится использовать относительно большое число блоков памяти маленькой емкости (а в ПЛИС все равно приходится использовать блоки памяти большей емкости). Это может породить худшие по производительности и по аппаратурным затратам варианты по сравнению с использованием сквозной аппроксимации несмотря на сокращение числа умножителей. Поэтому, хотя такой подход в ряде случаев может быть оправдан, при выборе способа реализации требуется тщательное сопоставление.

Устройства, реализующие метод «цифра за цифрой». Устройства, реализующие метод «цифра за цифрой», не имеют конкурентов при представлении аргументов и данных с точностью, соответствующей 25 разрядам и выше, но могут использоваться и при меньшей разрядности.

Этот итерационный алгоритм сводит прямые вычисления сложных функций к выполнению простых операций сложения и сдвига. Здесь, как и в полиномиальных аппроксиматорах, применяется рекуррентное построение вычислителя с использованием табличных значений коэффициентов.

Рассмотрим метод «цифра за цифрой» в версии Волдера [9], [10], известный также как Cordic, на примере задачи расчета косинуса и синуса задаваемого угла φ.

Сначала выбирается некоторый исходный угол β_0, для которого известны значения функций. Далее выполняется *n*-шаговая итерационная процедура, где *n* – разрядность результата. На каждом *i*-м шаге значение угла изменяется в большую или меньшую сторону на величину

$$a_i = \pm \mathrm{arctg}\left(2^{-i}\right)$$

так, чтобы приближаться к φ:

$$\beta_i = \sum_{j=0}^{i} \alpha_i;\ \left|\beta_{i+1} - \varphi\right| < \left|\beta_i - \varphi\right|.$$

При этом на каждом шаге для нового значения аргумента β_i вычисляют значения функций синуса и косинуса, используя тригонометрические преобразования вида (1),

$$\sin(\beta_i + \alpha_i) = \cos(\alpha_i) \times$$
$$\times \left\{\sin(\beta_i) \pm \cos(\beta_i)\mathrm{tg}\left[\mathrm{arctg}\left(2^{-i}\right)\right]\right\}$$

и аналогично

$$\cos(\beta_i + \alpha_i) = \cos(\alpha_i)\left[\cos(\beta_i) \pm \sin(\beta_i) \cdot 2^{-i}\right].$$

Умножение на $\cos(\alpha_i)$ формально присутствует на каждом шаге. Но рекурентность процедуры позволяет лишь однажды умножить результат на произведение косинусов фиксированного набора углов – для фиксированной разрядности это константа. Обычно она используется как модификатор функции от начального значения угла и называется коэффициентом искажения.

На основании изложенного блок, выполняющий один шаг итерационного алгоритма, может быть представлен Vhdl-кодом, приведенным в следующем листинге.

Листинг. VHDL-описание блока, выполняющего один шаг алгоритма

```
Entity one_step_cordic is
  Generic (n: integer :=32; -- разрядность преобразования
        Corr:integer:= 652032874 );--  коэффициент  искажения ( по умолчанию для 32 разрядов)
    Port ( clock:bit;
       FI:  in integer range -2**(n-1) to 2**(n-1)-1;  --аргумент преобразования
       sin_out, cos_out: inout  integer range -2**(n-1) to 2**(n-1)-1; --результаты очередного шага
       k: integer range 0 to n-1; -- номер шага итерации
      art_tg: in integer  range -2**(n-1) to 2**(n-1)-1;  -- модификатор аргумента для очередного шага
                                                          -- выбирается из памяти
End entity;

Arcitecrure rtl of one_step_cordic is
   Function barrel_shift
                ( m:integer range 0 to n-1; -- разрядность сдвигателя
                   K: integer range 0 to n-1 ;--  сдвиг на число разрядов
                    v : integer range -2**(n-1) to 2**(n-1)-1  ;--  сдвигаемое данное
       return integer range -2**(n-1) to 2**(n-1)-1)   is
begin variable res:integer;
   res:=v; --- операторная часть опущена
 return res;
end function;
signal sin_in, cos_in, remainder_in, remainder_out:   integer range -2**(n-1) to 2**(n-1)-1;
                          --результаты предыдущего шага
begin
   sin_in<= 0 when k=0 else sin_out;   --входной мультиплексер
  cos_in<= corr when k=0 else cos_out;
  remainder_in<= Fi when k=0 else remainder_out;
process(  clock)
begin
  if  clock='1' and clock'event   then
      if remainder_in >0 then        sin_out<= sin_in  + barrel_shift(n,k,cos_in);
                                  cos_out<= cos_in - barrel_shift(n,k,sin_in);
                                  remainder_out<=remainder_in - arc_tg(k);
       Else                      sin_out<= sin_in  - barrel_shift(n,k,cos_in);
                                   cos_out<= cos_in +barrel_shift(n,k,sin_in);
                                    remainder_out<=remainder_in + arc_tg(k);
      end if;
    end if;
end process;
end rtl;
```

Данные представляются в дополнительном коде в формате фиксированной запятой. Старший, крайний левый разряд обозначает знак, запятая – справа от знакового разряда.

Функция barrel_shifter – сдвиг на произвольное число разрядов – реализуется типовым цифровым узлом (см., например, [11]). Реализация зависит от используемой платформы и имеющихся библиотек и здесь не приводится. Например, для проектов на ПЛИС фирмы «Altera», можно воспользоваться модулем из библиотеки параметризованных LPM_CLSHIFT.

Модуль one_step_cordic может непосредственно включаться в структуру, приведенную на рис. 1, подобно OSD. Отличие состоит лишь в том, что по линии обратной связи Z передаются три параметра, а блок ROM хранит отсчеты функции арктангенса для соответствующих шагов.

Результаты синтеза с использованием пакета Quartus^{TM} показали, что для реализации 32-разрядного функционального преобразователя достаточно всего 600 ячеек микросхем программируемой логики семейства CyclonII, т. е. менее 10 % микросхемы минимальной емкости. Полный цикл преобразования составляет 230 нс (7 нс на каждый шаг).

Для повышения производительности при плотном потоке данных можно перейти к конвейерной реализации, соединяя модули one_step_cordic так, как на рис. 2. Заметим, что управляемая схема сдвига (barrel shifter) для каждой ступени вырождается в сдвигатель на постоянное число битов. Значения сдвига, как и arctg (α_i), – это индивидуальные для каждого модуля константы, которые могут задаваться как параметр настройки. Объем затрат, конечно, увеличивается (до 4000 тысяч ячеек для микросхем семейства CyclonII), что, впрочем, для многих применений приемлемо. Максимально допустимая частота поступления данных составляет 150 мГц.

Вычислители целого ряда аналитических функций (показательных, логарифмических, обратных тригонометрических и др.) во многом аналогичны [12].

Описанные в представленной работе реализации легко реконфигурируются по разрядности данных и структуре (итеративная декомпозиция, поток, конвейер, конвейер из ИД-блоков) в зависимости от требований и ограничений. Реализации протестированы и выполнена их загрузка в микросхемы программируемой логики. Используя приведенную программу и имеющийся прототип на ПЛИС, можно включать такие модули в состав систем на кристалле в качестве сопроцессора, разгружая ядро системы от невыгодных операций, и обеспечить производительность до сотен миллионов отсчетов за секунду, что может удовлетворить требования практически любых современных систем управления и цифровой обработки сигналов.

СПИСОК ЛИТЕРАТУРЫ

1. Kaeslin H. Digital Integrated Circuit Design. From VLSI Architectures to CMOS Fabrication. New York: Cambridge University Press, 2008.

2. Смолов В. Б., Фомичев В. С. Аналого-цифровые нелинейные вычислительные устройства. Л.: Энергия, 1974. 264 с.

3. Каляев А. В. Теория цифровых интегрирующих машин и структур. М.: Энергия, 1970.

4. Mayorov F. V. Electronic digital integrating computers. Digital Differential Analyzers. London: Iliffe Books Ltd, 1964.

5. Время-импульсные вычислительные устройства / В. Б. Смолов, Е. П. Угрюмов, А. Б. Артамонов и др.; под ред. В. Б. Смолова, Е. П. Угрюмова. М.: Радио и связь, 1983. 288 с.

6. Данчеев В. П. Цифро-частотные вычислительные устройства. М.: Энергия, 1976. 176 с.

7. Темников Ф. Е., Славинский В. Л. Математические развертывающие системы. М.: Энергия, 1970. 121 с.

8. Мурсаев А. Х. Представление в языках проектирования аппаратуры потоковых, конвейерных и микропрограммных реализаций операционных устройств // Изв. СПбГЭТУ «ЛЭТИ». 2010. № 5. С. 73–78.

9. Volder J. E. The CORDIC trigonometric computing technique // IRE Transactions on Electronic Computers. 1959. № 9. P. 189–197.

10. Байков В. Д., Смолов В. Б. Аппаратурная реализация элементарных функций в ЦВМ. Л.: Изд-во Ленингр. ун-та, 1975. 96 с.

11. 16_bit_barrel_shifter. ACADEMIA. URL: http://www.academia.edu/9391495/.

12. Байков В. Д., Смолов В. Б. Специализированные процессоры: итерационные алгоритмы и структуры. М.: Радио и связь, 1985. 288 с.

A. Kh. Mursaev
Saint Petersburg Electrotechnical University «LETI»

IMPLEMENTATION OF FUNCTIONAL TRANSFORMATION
IN THE PROGRAMMABLE LOGIC CHIPS

Functional transformation, that is, the formation of the code, which numerical equivalent is a defined function of the input code numerical equivalent, is a widespread operation in the measurement, control and similar systems. The implementation of such transformations in universal calculators (e. g. microprocessors) often does not meet the requirements of consumers in terms of performance or energy effectiveness. A review of the functional transformation methods is presented: tabular approximation, polynomial approximation, calculations according Volder method. The implementations into program logic microchips using different variants of spatial and temporal distribution of calculations are considered. It is shown that performance in the range up to 107 transformations per second with moderate costs is achievable.

Functional transformation, hardware implementation, programmable logic

УДК 519.7

С. И. Тодиков
Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В. И. Ульянова (Ленина)

Минимизация мультиопераций в классе ключевых стандартных форм

Рассматривается проблема минимизации мультиопераций в классе ключевых стандартных форм. В основе работы лежит разработанный алгоритм минимизации мультиопераций для n = 2, 3 в классе ключевых стандартных форм. Данный алгоритм основан на анализе мультиопераций и последующим наилучшим заменам всех нулевых элементов для получения минимального представления мультиоперации в классе ключевых стандартных форм. С помощью разработанного алгоритма получены все минимальные представления мультиопераций для n = 2, 3 в классе ключевых стандартных форм, средняя сложность минимального представления мультиопераций в классе ключевых стандартных форм и количественное распределение мультиопераций по сложностям полученных минимальных представлений. Произведено сравнение полученных результатов минимизации мультиопераций в классе ключевых стандартных форм с минимизацией мультиопераций в классе стандартных форм. При сравнении результатов делается вывод, что минимальное представление мультиопераций в классе стандартных форм лучше, чем в классе ключевых стандартных форм.

Мультиоперация, суперклоны, ключевая стандартная форма, алгоритм, минимизация

В теории дискретных функций, помимо всюду определенных функций k-значной логики, изучаются и функции, определенные не на всех наборах. В этом случае неопределенность понимается по-разному, в зависимости от рассматриваемого класса задач. Не всюду определенными функциями на конечном множестве являются и мультиоперации. У данного вида функций неопределенность понимается как неодноэлементное (в том числе и пустое) подмножество конечного множества, на котором эти функции заданы. В настоящее время изучаются разные вопросы теории мультиопераций, в частности вопросы полноты [1], связь клонов функций и клонов мультиопераций [2], рассматриваются вопросы функциональной разделимости [3].

В данной работе затрагивается вопрос минимизации мультиоперации в классе ключевых стандартных форм, приводится алгоритм минимизации и результаты минимизации мультиопераций ранга 2.

Основные понятия. Пусть 2^A – множество всех подмножеств A. Отображение из A^n в 2^A называется n-местной мультиоперацией на A.