ИНФОРМАТИКА

2006 **январь-март** **№ 1**

УДК 519.7

Д.А. Кочанов

VHDL-МОДЕЛИ МАТЕМАТИЧЕСКИХ ФУНКЦИЙ, ОСНОВАННЫЕ НА КУСОЧНО-ПОЛИНОМИАЛЬНОЙ ИНТЕРПОЛЯЦИИ

Предлагается способ создания VHDL-моделей математических функций с использованием стандартных математических пакетов. Математические функции реализуются в виде макроэлементов цифровых сверхбольших интегральных схем.

Введение

При проектировании цифровых заказных сверхбольших интегральных схем (СБИС) возникает проблема создания макроблоков (макроэлементов), предназначенных для вычисления значений математических функций на заданных интервалах изменения аргумента [1, 2]. Характеристики таких макроблоков (сложность, быстродействие) зависят от эффективности реализованного в них вычислительного алгоритма и требуемой точности вычислений.

Для вычисления значений математических функций в составе цифровых СБИС предложены различные методы. Одним из них является табличный метод [3], основная идея которого заключается в хранении значений искомой функции в виде двоичной таблицы. Главное преимущество этого метода – высокая точность вычислений значений функции в заданных точках, недостаток – большая площадь генерируемой схемы.

В настоящей статье предлагается решать проблему создания описаний цифровых подсхем для вычисления значений математических функций путем полиномиальной интерполяции [4]. В работе приводится VHDL-алгоритм кусочно-полиномиальной интерполяции математических функций, а также осуществляется сравнительный анализ сложности схем, реализующих кусочно-полиномиальную интерполяцию математических функций и схем, реализующих те же функции табличным методом.

1. Алгоритм схемы Горнера

Реализация алгоритма полиномиальной интерполяции аналитической функции может быть осуществлена различными способами. Это обусловлено, прежде всего, существованием разных форм записи полинома и, как следствие, разных способов вычисления его значения. В основе изложенного ниже подхода лежит задание интерполяционного полинома в виде схемы Горнера, общий вид которой для полинома степени n следующий:

$$p_n(x) = a_0 + x(a_1 + x(...(a_{n-1} + xa_n)...)). \quad (1)$$

Преимущество такой формы записи перед каноническим представлением полинома $p_n(x) = a_0 + a_1 x + ... + a_n x^n$ заключается в меньшем количестве операций умножения, которые необходимо выполнить, чтобы вычислить значение полинома в требуемой точке, а значит в большей скорости вычислений и меньшей погрешности окончательного результата.

Пусть на отрезке $[a, b]$ задан полином степени n, представленный в виде формулы (1), тогда его значения в некоторой точке $x \in [a, b]$ можно вычислить при помощи итерационной формулы

$$t_i(x) = a_{n-i} + xt_{i-1} : t_0 = a_n, i = 1, 2, ..., n, \quad (2)$$

где a_i – i-й коэффициент; n – степень полинома (1).

Для того чтобы на основе формулы (2) можно было синтезировать цифровое устройство, необходимо записать ее на синтезируемом подмножестве некоторого языка описания аппаратуры (в работе для этих целей использован язык VHDL) [5]. В этом случае возникают две сложности, связанные с ограничениями синтезируемого подмножества языка VHDL:

– отсутствие поддержки данных вещественного типа;

– возможность выполнения деления только на числа вида 2^n, $n \in Z$. Здесь и далее символом Z обозначается множество целых чисел.

Модифицируем формулу (2) с учетом данных ограничений. Для этого введем следующие обозначения: пусть k – целочисленная константа, задающая точность представления дробей (число двоичных разрядов после запятой, отводимое под дробную часть аргумента) и пусть $x' = 2^k x$, $a'_i = [2^k (a_i + 2^{-k-1})]$ (квадратные скобки в выражении обозначают взятие целой части аргумента). Кроме того, введем в рассмотрение бинарную операцию $\circ$:

$$x \circ y = \begin{cases} \left[\left[\dfrac{xy + 2^{k-1}}{2^k}\right]\right], xy > 0; \\ \left[\left[\dfrac{xy - 2^{k-1}}{2^k}\right]\right], xy < 0, \end{cases} \tag{3}$$

тогда для выражения $q_n(x) = a'_0 + x' \circ (a'_1 + x' \circ (... \circ (a'_{n-1} + x' \circ a'_n)...))$ справедливо отношение $q_n(x) = 2^k (p_n(x) + \varepsilon)$, где ε некоторая погрешность вычислений. Для того чтобы вычислить значение $q_n(x)$ в некоторой точке, можно воспользоваться следующей итерационной формулой:

$$u_i(x') = a'_{n-i} + x' \circ u_{i-1} : u_0 = a'_n, i = 1, 2, ..., n. \tag{4}$$

Вычислительная погрешность формулы (4) больше, чем погрешность формулы (2), так как после выполнения каждой операции умножения результат округляется до требуемой точности, однако преимущество формулы (4) заключается в том, что ее можно задать в виде алгоритма на языке VHDL. Одна из возможных алгоритмических реализаций формулы 4 (алгоритм 1) приведена в листинге 1.

Листинг 1. **Алгоритм 1 (VHDL-реализация формулы (4))**

```
Library IEEE, work;
use work.idat.all;
package pack is
function mul (x, y, prec: integer) return integer;
end pack;

package body pack is
function mul (x, y, prec: integer) return integer is
variable tmp: integer;
begin
 tmp := x*y;
 if(tmp > -1) then tmp := tmp + 2**(prec-1);
 else tmp := tmp - 2**(prec-1);
 end if;
 tmp := tmp / 2**(prec);
 return tmp;
end mul;
end pack;

Library IEEE, work;
use work.pack.all;
use work.idat.all;
entity pisin is
```

```
port (signal x: in integer; signal y: out integer);
end pisin;

architecture arch of pisin is
begin
p0: process(x)
 variable tmp: integer;
begin
 tmp := a_factor(0);
 for i in 1 to arr_len loop
       tmp := a_factor(i) + mul(tmp, x, prec);
 end loop;
 y <= tmp;
end process;
end arch;
```

Для того чтобы использовать алгоритм 1 для полиномиальной интерполяции математических функций, необходимо:

1) построить для требуемой функции полиномиальное приближение на заданном интервале изменения аргумента, после чего записать полином в виде скобочной схемы Горнера (1);

2) создать VHDL-файл с входными данными для алгоритма 1 с информацией об интерполяционном полиноме (его степени, интервале изменения аргумента, точности представления дробей, полиномиальных коэффициентах и др.).

Пример такого файла с комментариями для полинома

$$p_6(x) = 1{,}00003\,x + 0{,}00269\,x^2 - 0{,}16566\,x^3 - 0{,}00186\,x^4 + 0{,}01017\,x^5 - 0{,}00093\,x^6$$

приведен в листинге 2. Полином $p_6(x)$ приближает функцию sin на отрезке $[0, \pi/2]$ по точкам $\{0, \pi/12, \pi/6, .., \pi/2\}$ с точностью до 18 двоичных разрядов после запятой.

Листинг 2. **Пример VHDL-файла с информацией о полиноме *p6(x)***

```
package idat is
constant prec: integer := 18; --число двоичных разрядов после запятой
constant arr_len: natural := 6; --степень интерполяционного полинома
type fact_arr is array (0 to arr_len) of integer;
constant a_factor: fact_arr := (-244,2665,-487,-43427,-71,262151,0);
--список коэффициентов полинома, умноженных на 2**prec и округленных
end idat;
```

2. Модифицированный алгоритм схемы Горнера

Особенностью алгоритма 1 является округление промежуточного результата после каждой итерации. Необходимость в округлении обусловлена тем, что число двоичных разрядов дробной части аргумента ограничено константой k. Кроме того, погрешность возникает из-за округления полиномиальных коэффициентов и погрешности аппроксимирующего полинома.

Надо сказать, что при $x \to \infty$ погрешность алгоритма 1 растет со скоростью геометрической прогрессии, поэтому алгоритм 1 может найти практическое применение лишь для приближения функций на интервале $[a,b] \subseteq [-1,1]$. Чтобы расширить границы применимости алгоритма на множество всех действительных чисел, сделаем в полиноме (1) следующее преобразование независимой переменной:

$$x = \frac{a+b}{2} + \frac{a-b}{2}\,y. \tag{5}$$

Данная линейная замена осуществляет взаимооднозначное соответствие между $x \in [a,b]$ и $y \in [-1,1]$. После проведения замены запишем полученный полином в виде формулы (1), для

этого пересчитаем значения полиномиальных коэффициентов, после чего получим полином $p_n(y)$ степени *n* от переменной *y*: $p_n(y) = b_0 + y(b_1 + y(...(b_{n-1} + yb_n)...))$.

Введем следующие обозначения:

$$c_0 = \frac{a+b}{b-a},\ c_1 = \frac{2}{a-b}. \tag{6}$$

После этого запишем итерационную формулу (7), являющуюся модифицированным вариантом формулы (4) и позволяющую вычислить значения полинома $p_n(x)$ на заданном интервале изменения аргумента:

$$u_i(x) = b_{n-i} + (c_0 + c_1 x)u_{i-1} : u_0 = b_n, i = 1, 2, ..., n. \tag{7}$$

Замена переменной (5) приводит к появлению в формуле (7) величины $c_0 + c_1 x$, вычисление которой, в свою очередь, приводит к появлению в формуле (7) дополнительной вычислительной погрешности. Уменьшить погрешность можно, наложив ограничения на выбор отрезка [*a*, *b*], а именно выбрав его границы таким образом, чтобы для констант c_0 и c_1 (6) выполнялись следующие соотношения:

$$c_0 = 2^{-l};\ c_1 = 2^{-m};\ l < k;\ m < k;\ l, m \in Z.$$

При таком выборе константы можно хранить без потери точности и погрешность вычисления промежуточной величины $y(x) = c_0 + c_1 x$ будет равна погрешности округления произведения $x \times c_1$ до *k* двоичных разрядов после запятой. Данная величина не превосходит константы 2^{-k-1}.

Воспользуемся введенной ранее бинарной операцией $\circ$ (3), чтобы адаптировать формулу (7) к реализации на языке VHDL. Для этого введем обозначения: $x' = 2^k x$, $b'_i = 2^k b_i$. Запишем формулу для вычисления значения полинома в точке x' с округлением:

$$u_i(x') = b'_{n-i} + (c_0 + c_1 \circ x') \circ u_{i-1} : u_0 = b_n, i = 1, 2, .., n. \tag{8}$$

Поскольку формула (8) содержит только те арифметические операции, которые поддерживаются синтезируемым подмножеством языка VHDL, на ее основе можно записать синтезируемый VHDL-алгоритм (алгоритм 2).

После проведения ряда экспериментов по синтезу VHDL-реализаций алгоритма 1 выяснилось, что сложность получаемых схем сильно зависит от степени интерполяционного полинома и, в меньшей мере, от размеров интервала интерполяции. Поэтому при построении VHDL-реализации алгоритма 2 использовалась кусочно-полиномиальная интерполяция, при которой интервал изменения аргумента [*a*, *b*] разбивается на подынтервалы и строится полиномиальное приближение требуемой функции на каждом из полученных отрезков разбиения. VHDL-реализация формулы (8) (алгоритм 2) приведена в листинге 3.

Листинг 3. **Алгоритм 2 (VHDL-алгоритм кусочно-полиномиальной интерполяции математических функций)**

```
Library IEEE, work;
use work.idat.all;
package pack is
function mul (x, y: integer) return integer;
end pack;

package body pack is
function mul (x, y: integer) return integer is
```

```
variable tmp: integer;
begin
 tmp := x*y;
 if(tmp > -1) then
        tmp := tmp + 2**(prec-1);
 else
        tmp := tmp - 2**(prec-1);
 end if;
 tmp := tmp / 2**(prec);
 return tmp;
end mul;
end pack;

Library IEEE, work;
use IEEE.std_logic_signed.all;
use work.pack.all;
use work.idat.all;

entity poly is
port (signal x: in x_range; signal y: out y_range);
end poly;

architecture arch of poly is
begin
p0: process(x)
variable tmp: integer;
 variable x1: integer;
 variable k: natural range 1 to segs;
begin
 for i in 1 to segs loop
        if ((x >= fragm(i-1)) and (x <= fragm(i))) then
               k := i;
        else
               null;
        end if;
 end loop;
 x1 := c_factor(k)(0) + mul(c_factor(k)(1), x);
 tmp := a_factor(k)(0);
 for i in 1 to power loop
        tmp := a_factor(k)(i) + mul(tmp, x1);
 end loop;
 y <= tmp;
end process;
end arch;
```

Входными данными для алгоритма 2 служит VHDL-пакет с информацией о кусочно-полиномиальной интерполяции для требуемой функции. Пример такого пакета для функции $y(x) = \sqrt{x}$ приведен ниже.

Листинг 4. **VHDL-пакет с данными для алгоритма 2 об интерполяции функции $y(x) = \sqrt{x}$ на интервале изменения аргумента [1, 160]**

```
package idat is
constant prec: natural := 12; --точность приближения
constant power: natural := 4; --степень интерполяционного полинома
constant segs: natural := 8; --количество интервалов, на которые разбит отре-
зок [a,b]

type a_vect is array (0 to power) of integer;
type a_matrix is array (1 to segs) of a_vect;
constant a_factor: a_matrix :=
 ((-3, 12, -70, 836, 5017), (-4, 17, -98, 1182, 7094),
 (-5, 25, -139, 1672, 10033), (-7, 35, -197, 2364, 14189),
 (-10, 49, -279, 3344, 20066), (-15, 70, -394, 4729, 28378),
```

```
  (-21, 99, -557, 6688, 40132), (0, 4, -76, 2731, 49152)); --коэффициенты
интерполяционных полиномов

type c_vect is array (0 to 1) of integer;
type c_matrix is array (1 to segs) of c_vect;
constant c_factor: c_matrix :=
  ((-12288, 8192), (-12288, 4096), (-12288, 2048),
  (-12288, 1024), (-12288, 512), (-12288, 256), (-12288, 128),
  (-36864, 256)); --коэффициенты c0 и c1

type f_vect is array (0 to segs) of integer;
constant fragm: f_vect :=
  (4096, 8192, 16384, 32768, 65536, 131072, 262144, 524288, 655360); --точки
разбиения [a,b]

subtype x_range is integer range fragm(0) to fragm(segs); --диапазон изменения
аргумента
subtype y_range is integer range 0 to 51814; --диапазон изменения функции
end idat;
```

Таким образом, алгоритм 2 можно применять для приближения функций на любых интервалах изменения аргумента. Подтверждением этому служит рис. 1, на котором изображены график функции $\sin(x) - p_n(x)$ и график погрешности алгоритма 2. Первая функция обозначает погрешность полиномиальной интерполяции функции $\sin(x)$, а вторая функция – погрешность алгоритма 2.

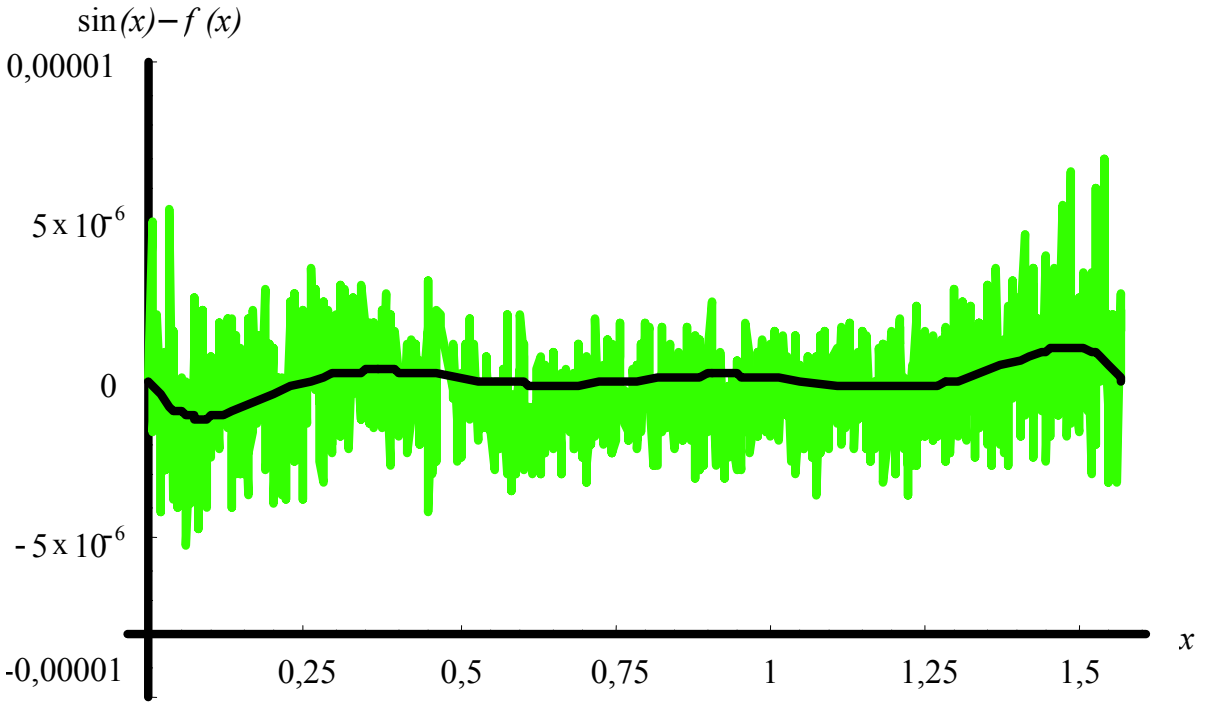

Рис. 1. Графики погрешностей полиномиальной аппроксимации функции y(x)=sin(x) на отрезке [0,π/2] с точностью до 18 двоичных разрядов после запятой и ее схемной реализации по алгоритму 2

3. Генерация входных данных в системе Mathematica

Ранее уже упоминалось, что входными данными для алгоритма 2 служит VHDL-пакет с информацией о кусочно-полиномиальной интерполяции требуемой функции. Для того чтобы создать такой пакет, необходимо провести ряд относительно трудоемких математических расчетов, таких, как получение полиномиального приближения функции на выбранном разбиении,

пересчет значений полиномиальных коэффициентов с учетом замены (5) и др. Чтобы сократить время вычислений, можно использовать какой-либо из математических пакетов, например систему Mathematica [6]. В листинге 5 приведена программа, выполнение которой в системе Mathematica позволит автоматически сгенерировать VHDL-пакет с информацией о кусочно-полиномиальной интерполяции функции $y(x) = \sqrt{x}$.

Листинг 5. **Программа для генерации в системе Mathematica VHDL-пакета с информацией о кусочно-полиномиальной интерполяции функции $y(x) = \sqrt{x}$ полиномами четвертой степени на отрезке [1, 256]**

```
Fun[x_]:=Sqrt[x] (* Требуемая функция *)
net:={1,2,4,8,16,32,64,128,256} (* Разбиение интервала интерполяции *)
n:=4 (* Степень интерполяционного полинома *)
dir="d:\\asp\\work2\\gor1\\temp\\sqrt\\"; (* Директория для генерации VHDL-
пакета *)
k=-1; (* Точность вычисления дробной части числа. Если k < 1, то выбор точно-
сти будет произведен автоматически *)

fname:=StringJoin[dir,"idat_",ToString[n],"_",ToString[k],".vhd"]
For[pts=List[];i=1,i<Length[net],++i,pts=Append[pts,({#,Fun[#]}&/@N
[Range[net[[i]],net[[i+1]],(net[[i+1]]-net[[i]])/n])]];
For[mpoly=List[];i=1,i≤Length[pts],++i,mpoly=Append[mpoly,InterpolatingPolynom
ial[
pts[[i]],x]]]
For[canpoly=List[];i=1,i≤Length[pts],++i,canpoly = Append[canpoly,
Collect[mpoly[[i]]/.x→(net[[i]]+net[[i+1]])/2+t*(net[[i+1]]-net[[i]])/2,t]]]
If[k<1,Block[{Ea},
For[Ea=List[];i=1,i≤Length[pts],++i,
Ea=Append[Ea,Max[Abs[Minimize[Fun[(net[[i]]+net[[i+1]])/2+ t*(net[[i+1]] -
net[[i]])/2] - canpoly[[i]],(t>=-1)&& (t<=1), {t}]][[1]],
Abs[Maximize[Fun[(net[[i]]+net[[i+1]])/2+t*(net[[i+1]]-net[[i]])/2]-
canpoly[[i]],(t>=-1)&&(t<=1),{t}]][[1]]]];
Emax=Max[Ea];
k=Abs[Round[Log[2,Emax]]]]
dotp=2^k;
mas1=Round[CoefficientList[canpoly,{t}]*dotp];
For[mas2=List[];i=1,i≤Length[mas1],++i,mas2=Append[mas2,Reverse[mas1[[i]]]]]
For[tp=List[];i=1,i<Length[net],i++,tp=Append[tp,Collect[N[(2x-
(net[[i]]+net[[i+1]]))/     (net[[i+1]]-net[[i]])],x]]]
mas3=Round[dotp*CoefficientList[tp,{x}]];
alg[x_]:=Block[{tm1,t,i,j},
For[j=1,j<Length[net],++j,If[x≥net[[j]]&&x≤net[[j+1]],tm1=mas1[[j]];t1[x1_]=
mas3[[j]][[1]]+Round[mas3[[j]][[2]]*x1/dotp];Break[]]];
t=tm1[[Length[tm1]]];
i=Length[tm1]-1;
Do[t=f[tm1[[i]],t,t1[dotp*x]];--i;
If[i<=0,Return[N[t/dotp]]],{Length[tm1]}]]
choospoly[x_]:=
For[i=1,i<Length[net],++i,If[x≥net[[i]]&&x≤net[[i+1]],Return[mpoly[[i]]]]]
Plot[{Fun[x]-alg[x],Fun[x]-choospoly[x]}, {x,First[net],Last[net]},
PlotRange→All,
PlotStyle→{{Hue[1]},{Thickness[0.006],Hue[0.7]}},
AxesOrigin→{First[net], -2Abs[Emax]}];
```

Для того чтобы на основе программы из листинга 5 сгенерировать VHDL-пакеты для других математических функций, необходимо внести соответствующие изменения в первые пять строк алгоритма 2: задать имя интерполируемой функции, разбиение, выходную директорию и др., после этого командой Kernel→Evaluation→Evaluate Notebook запустить алгоритм на выполнение. В результате будет сгенерирован VHDL-пакет с информацией о кусочно-полиномиальной интерполяции для требуемой функции. Кроме того, будет построен график

погрешности алгоритма 2, аналогичный графику на рис. 1, по которому можно оценить эффективность будущей схемной реализации и, при необходимости, внести корректировки в параметры генерации: изменить степень интерполяционного полинома, разбиение либо точность представления дробей.

Условно процесс создания макроблоков можно разбить на два этапа (рис. 2). На первом этапе разработчик генерирует в системе Mathematica VHDL-пакет с входными данными для алгоритма 2, на втором – осуществляет синтез алгоритма 2 совместно с созданным пакетом в системе Leonardo.

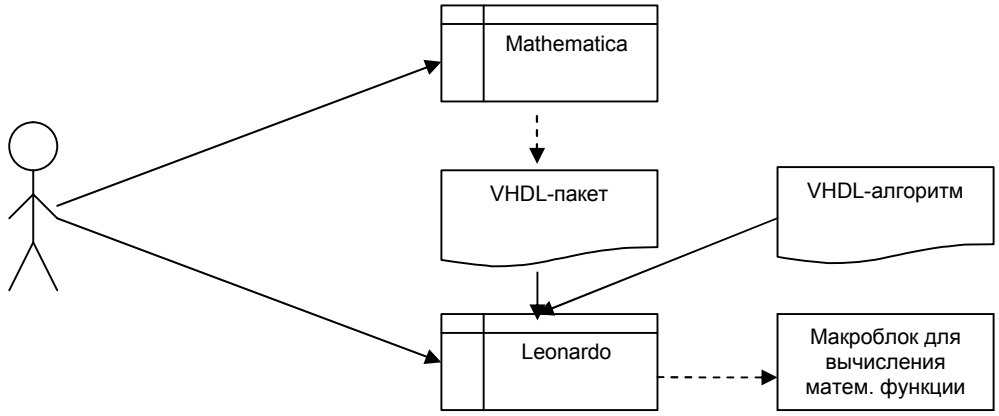

Рис. 2. Процесс создания макроблока для вычисления математических функций в составе СБИС

4. Анализ сложности и быстродействия синтезированных схем

Эксперименты проводились на ПК с процессором Intel Pentium III, 500 МГц, имеющим 256 Мб оперативной памяти. По алгоритму 2 и табличному методу было сгенерировано несколько VHDL-описаний цифровых устройств, предназначенных для вычисления значений элементарных математических функций на интервале изменения аргумента [*a*, *b*] с точностью *k* двоичных разрядов после запятой (табл. 1, 2).

Таблица 1

Зависимость сложности синтезированных по алгоритму 2 устройств от параметров алгоритма

f	[a,b]	k	I	O	n	l	E	e	LUTs	ns	t (м:с)
ln(x)	[1, 256]	5	14	8	1	9	0,081	0,06	299	61	0:08
ln(x)	[1, 256]	6	15	9	1	8	0,072	0,06	279	56	0:08
ln(x)	[1, 256]	7	16	10	2	8	0,018	0,006	778	103	0:20
ln(x)	[1, 256]	6	15	9	3	8	0,026	0,0008	1247	149	0:46
ln(x)	[1, 256]	11	20	14	3	8	0,0012	0,0008	1387	152	1:00
ln(x)	[1, 256]	12	21	15	4	8	0,00055	0,00013	1850	192	1:08
ln(x)	[1, 256]	12	21	15	5	8	0,00055	0,00002	2320	219	1:49
$\sqrt{x}$	[1, 160]	12	20	16	4	8	0,0007	0,00017	1939	169	0:51
$\sqrt{x}$	[1, 160]	10	18	14	4	8	0,00158	0,00017	1942	173	0:54
$\sqrt{x}$	[1, 160]	10	18	14	5	8	0,0018	0,00003	2393	201	1:06
sin(x)	[0, π/2]	15	16	16	5	1	0,00005	0,00001	2148	211	1:19

Символы столбцов таблицы имеют следующие значения:
f – имя функции;
[*a*, *b*] – интервал изменения аргумента;
k – точность приближения;
I – число входных полюсов схемы;
O – число выходных полюсов схемы;
n – степень интерполяционного полинома;
l – количество отрезков, на которые разбивается интервал изменения аргумента;
E – максимальная погрешность алгоритма;
e – максимальная погрешность интерполяционного полинома;
LUTs – количество элементарных логических вентилей, в которых измеряется сложность схемы в библиотеке логических элементов SPARTAN2E;
ns – задержка схемы в наносекундах;
t – время проведения синтеза.

Таблица 2

Результаты синтеза устройств, реализующих аналитические функции табличным методом

f	[a,b]	k	I	O	E	LUTs	ns	t (м:с)
ln(x)	[1, 256]	4	12	7	0,0625	434	25	00:42
ln(x)	[1, 256]	5	13	8	0,03125	850	22	02:34
ln(x)	[1, 256]	6	14	9	0,015625	1944	24	06:11
ln(x)	[1, 256]	7	15	10	0,0078125	3979	26	16:27
$\sqrt{x}$	[1, 160]	4	12	8	0,0625	636	22	01:38
$\sqrt{x}$	[1, 160]	5	13	9	0,03125	1464	26	05:42
$\sqrt{x}$	[1, 160]	6	14	10	0,015625	3153	24	11:09
sin(x)	[0, π/2]	8	9	8	0,00390625	341	18	00:21
sin(x)	[0, π/2]	10	11	10	0,000976563	1132	25	03:19
sin(x)	[0, π/2]	11	12	11	0,000488281	2498	24	08:54

Основное назначение табличного метода – создание VHDL-моделей макроблоков, реализующих математические функции в составе цифровых СБИС.

Заключение

Результаты проведенных экспериментов показали, что сложность синтезируемых по алгоритму 2 цифровых устройств сильно зависит от степени интерполяционного полинома и, в меньшей мере, от разбиения отрезка [*a*, *b*]. Степень полинома можно уменьшить (сохранив при этом требуемую точность вычислений), увеличивая количество интервалов разбиения, однако сильное дробление может также оказаться неэффективным.

Таким образом, успех практического применения алгоритма 2 во многом зависит от удачного компромисса между погрешностью вычислений, степенью интерполяционного полинома и разбиением отрезка интерполяции. Оптимальный результат достигается, если порядок точности приближения (величина 2^{-k}) сравним с порядком максимальной погрешности аппроксимирующего полинома.

Выигрыш метода кусочно-полиномиальной интерполяции, реализованного по алгоритму 2, перед табличным методом становится ощутимым с ростом точности приближения и размеров интервала интерполяции.

Список литературы

1. Байков В.Д., Смолов В.Г. Специализированные процессоры: Итерационные алгоритмы и структуры. – М.: Радио и связь, 1985. – 288 с.

2. Балашов Е.П., Пузанков Д.В. Проектирование информационно-управляющих систем. – М.: Радио и связь, 1987. – 256 с.

3. Бибило П.Н., Кочанов Д.А. Получение VHDL-моделей схем для вычисления функций, заданных в табличной форме // Информатика. – 2004. – № 3. – С. 29–38.

4. Демидович Б.П., Марон И.А. Основы вычислительной математики. – М.: Наука, 1970. – 664 с.

5. Бибило П.Н. Синтез логических схем с использованием языка VHDL. – М.: Солон-Р, 2002. – 384 с.

6. Прокопеня А.Н., Чичурин А.В. Применение системы Mathematica к решению обыкновенных дифференциальных уравнений: учеб. пособие. – Мн.: БГУ, 1999. – 265 с.

Поступила 18.11.05

Объединенный институт проблем
информатики НАН Беларуси,
Минск, Сурганова, 6
email: bibilo@newman.bas-net.by

D.A. Kochanov

VHDL MODELS OF MATHEMATICAL FUNCTIONS THAT ARE BASED ON PIECEWISE-POLYNOMIAL INTERPOLATION

This article proposes a solution of a problem of creating VHDL-models of macroelements, which are designed to calculate the values of analytical functions. The initial function definition is given in analytical form in mathematical packages. The resulting function description is provided in table form with conversion to VHDL form.