

真值关系逻辑原理及实现

何 群

(南京工业大学 电子与信息工程学院, 江苏 南京 210009)

摘 要: 为有效地提高运算系统的运行速度, 以关系运算为理论基础, 提出真值关系逻辑的二维两层的设计算法: 将变量作为元组(行)而变量的各值域作为属性(列)形成真值关系表. 根据运算式或子程序的要求分别对表采用行、列二维的关系运算得到相应的概念模型; 应用 FPGA(尤其是 ISP)技术对概念模型进行底层编程(VHDL)及配置为固件, 使得运算式或子程序的运行直接由固件完成, 即用函数变量的真值由固化硬件直接产生出函数的真值, 极大地提高其运行速度. 给出算法原理并用各类实例以证明.

关键词: 真值关系; FPGA; 关系运算; VHDL; 逻辑化简

中图分类号: TP302

文献标识码: A

文章编号: 1000-7180(2015)04-0040-04

The Principle and Implementation of Logic Relationship by True Value

HE Qun

(College of Electronics and Information Engineering, Nanjing University of Technology, Nanjing 210009, China)

Abstract: In order to improve the speed of computing systems effectively, on the basis of relationship operating, the algorithm of logic relationship of true value for two-dimensional simplified on two layers is put forward in the paper. The algorithm is that: Variables is defined as tuples (rows) and each domain of variables is defined as attribute (columns), and it's a relationship table of true value. According to the requirement of the formula or subroutine operations on the table for corresponding relationship, the conceptual model is given out. The model is programmed for VHDL and configured to firmware through FPGA technology, especially the ISP. The formula or the subroutine directly are performed by firmware. The true value of function directly is produced by the firmware with the true value of variables, the function running speed can be greatly improved. The experiments show that it is efficient.

Key words: true value relationship; FPGA; relationship operation; VHDL; logic minimization

1 引言

计算机的运算部件(硬件)是运行速度的核心器件, 而运算方法(软件)则是在其基础上实现各种功能的程序, 显然增加针对各类应用的特定运算部件可有效地提高程序运行速度. 随着 FPGA(尤其是 ISP)^[1-3]技术的发展, 可以像软件编程一样进行运算部件(硬件)的设计且固化为实体(固件), 固件可以像子程序一样直接调用, 实现一种由变量的原始代码通过固件直接生成函数值代码过程. 设计固件的实质是大型的逻辑化简, 传统的方法是真值表化简,

是一种一维逻辑表达方式, 它需要给出所有变量的全部状态, 当变量长且数量多时, 建立这样的真值表化简模式是很难实现的. 针对这种情况, 提出一种二维二层设计逻辑模型的算法. 首先应用关系运算原理^[4], 将各变量的二进制同权位值(列)作为一个域, 而每个变量则为一个元组(行). 采用投影(project)将整体集化为部分集; 对部分集采用域内选择(select)和扩域连接(join)运算两个步骤生成这部分的逻辑概念模型, 由该模型再扩展为整体逻辑概念模型; 根据该模型采用 VHDL 语言^[5]设计出其逻辑模型, 由相应的开发系统将逻辑模型固化至具体芯片

中从而形成固件,这样相关的程序就可以通过接口直接调用固件得到该逻辑模型所实现的运行结果.可将常用算法或函数进行设计、固化和调用,由于固件几乎完全是由硬件实现且可在系统动态地切换,所以可有效地提高程序的运行速度.

2 算法理论和实现

以关系 $R^{[4]} = (\{A_n\}, V, F) (n = 1, 2, \dots, n)$ 给出真值关系的扩展定义.

定义 1(真值关系)

$Tr(K, \{D_m\}, V, f) (m = m, m-1, \dots, 1)$,

式中, Tr 为真值关系名; K 是变量名属性(关键字); $\{D_m\}$ 为变量的值域, D_m 最高位, D_1 最低位; $V = \{1, 0\}$ 为 $\{D_n\}$ 的值域; f 是条件函数.

以关系运算^[4]的定义(选择;投影;连接)给出真值关系运算的扩展定义.

定义 2(真值关系运算)

(1) 选择(select): $\delta f(Tr) = \{r \mid r \in Tr \wedge f\}$.

选择出满足 f 条件的元组(行);

(2) 投影(project): $\pi_{dr}(Tr) = \{Tdr \mid d \in D \wedge r \in K\}$. 列(D)、行(K)双向投影得真值子关系 Tdr ;

(3) 连接(join): $(Ts \omega Tr)f = \{sr\}$. 按条件 f 将 Tr 中元组 r 的 D 域扩展至 Ts 得对应元组 sr .

在定义 1 中, Tr 给出了各变量及变量分量的全部概念上可能的取值而非全部真值. 主要根据算式要求进行概念逻辑建模, 这是与传统直值表表示^[6]的区别之一. 定义 2 的(1)可以对所选出的元组进行列向计数运算, 得到相同域内同类值的数量; 定义 2 中的(3)可以根据这些数值量实现行向扩展出新的计算结果. 即采用列向、行向两个步骤建立逻辑模型. 因为列向中所有值是在同域内, 所以可作数量上的计算, 而行向运算则是跨域运算, 要根据数值大小进行权位调整. 这是与直值表逻辑化简^[6]的区别之二. 当变量多且字长较长时可以用定义 2 中的(2)将投影为较小的 Tr 进行局部逻辑模型的建立, 因为变量的权位具有递增(或降)的特点, 所以可以由该局部模型扩展至原表全集的概念逻辑模型.

真值关系及运算只给出逻辑模型是概念上的描述, 所以还要将这种模型建立具体的逻辑模型, 即用 VHDL 语言^[5]建立实际的逻辑模型.

建立逻辑模型的步骤为:

(1) 根据变量的数量及其值域列出 Tr ;

(2) 对 Tr 进行投影(project)操作得真值子关系

Sr ;

(3) 对 Sr 进行选择(select)操作及相应域运算;

(4) 用连接(join)对域运算结果进行跨域调整得局部概念逻辑模型;

(5) 将 Sr 的式(3)和式(4)扩充至 Tr ;

(6) 用 VHDL 对式(5)和进行实际编程;

(7) 再对该 VHDL 语句进行编译得整体逻辑模型.

综上述, 从语言层面进行两层描述: 概念层(关系符号)和应用层(VHDL), 前者使得描述清晰、简洁; 后者依据准确、易于实现. 而逻辑化简层面进行二维描述: 域内运算(列向)和跨域调整运算(行向), 形成二维建立逻辑模型的算法.

3 实例分析

3.1 连加(Σ)运算

设真值关系逻辑 T_j (3 变量) 如表 1 所示.

表 1 连加真值逻辑关系

K	D_n	$\dots$	D_i	$\dots$	D_1
A	a_n	$\dots$	a_i	$\dots$	a_1
B	b_n	$\dots$	b_i	$\dots$	b_1
C	c_n	$\dots$	c_i	$\dots$	c_1
J_1	j_{n-1}	$\dots$	j_{i-1}	$\dots$	j_{1-1}
J_2	j_{n-2}	$\dots$	j_{i-2}	$\dots$	j_{1-2}

表中 $\{a_i, b_i, c_i\} (i = 1, \dots, n)$ 的值域为 $\{0, 1\}$; J_1 和 J_2 是用于存放低域对本域进位的临时变量.

(1) 关系语言描述如下:

设 j_{i-1}, j_{i-2} 是后 1、2 位向本位的进位; j_{i+1}, j_{i+2} 是本位向前 1、2 位的进位.

① $j_{i-1}, j_{i-2} = \{0\}$ // 设最初 1、2 级进位初值

// 对表中每个 D_i 域循环

② for $T_j(D_i) (i = 0, \dots, n) \{$

// 计算出 D_i 域中为 1 的数量记入 d_i

③ select $d_i = \text{count}(D_i) \text{ when } (D_i = 1)$

// 将后 1、2 级向本位的进位和本位的数量 d_i 一起

// 进行二进制加(2+) 调整生成本位 s_i 及向前进位.

④ join $j_{i+2}, j_{i+1}, s_i \text{ when } (\{j_{i-1}, j_{i-2}, d_i\} 2+)$

(2) VHDL 语言描述如下:

// 从相应寄存器得 i 位(同域)值

signal D ; STD_LOGIC_VECTOR

(4 to 0); = $\{j_{i-2}, j_{i-1}, a_i, b_i, c_i\}$;

// 存放对本域的计算结果 $\{j_{i+2}, j_{i+1}, s_i\}$

signal C ; STD_LOGIC_VECTOR(2 to 0);

signal JS ; INTEGER range 0 to 4;

for i in 1 to 4 loop // 计算本域内为 1 的数量

if($D(i)$) then $JS \leq JS + 1$; endif;

end loop;

// 将 JS 转换为本位计算的结果(逻辑类型)

$C \leq (STD_LOGIC_VECTOR)JS$;

(3) 由 VHDL 生成 D_i 位的逻辑表达式:

$$s_i = a_i \oplus b_i \oplus c_i \oplus j_{i-1} \oplus j_{i-2};$$

$$j_{i+1} = \overline{a_i} \cdot \overline{b_i} \cdot j_{i-2} \cdot j_{i-1} \vee a_i \cdot b_i \cdot \overline{j_{i-2}} \cdot \overline{j_{i-1}}$$

$$\vee (j_{i-2} \oplus j_{i-1}) \cdot (\overline{a_i} \cdot c_i \vee b_i \cdot \overline{c_i} \vee a_i \cdot \overline{b_i})$$

$$\vee (a_i \oplus b_i)(c_i \oplus j_{i-1});$$

$$j_i + 2 = j_{i-1} \cdot j_{i-2} \cdot (a_i \cdot b_i \vee a_i \cdot c_i \vee b_i \cdot c_i)$$

$$\vee a_i \cdot b_i \cdot c_i \cdot (j_{i-1} \vee j_{i-2})$$

通过真值关系和 VHDL 语言两级描述使逻辑模型设计清晰,对列方向采用同域计数,只对逻辑 1 计数(标量);对行方向采用跨域调整,且将计数值直接转为相应的逻辑值(矢量).在表 1 中扩展变量的数量即可得连加($\sum$)运算.

3.2 连乘($\prod$)运算

设被乘数为 $A = (a_{3j}, \dots, a_{2j}, \dots, a_j, \dots, a_1)$, 乘数为 $B = (b_{2j}, \dots, b_j, \dots, b_1)$, A 和 B 的乘积为 $C = (c_{5j}, c_{4j}, \dots, c_{3j}, \dots, c_{2j}, \dots, c_j, \dots, c_1)$. 得两变量乘法算式的一般关系式 T_m , 如表 2 所示.

表 2 乘数、被乘数和积的真值逻辑关系(局部)

C_{5j}	C_{4j}	C_{3j}	C_{2j}	C_j	C_1
$\underline{a_{5j}} \cdot b_1$	$\underline{a_{4j}} \cdot b_1$	$a_{3j} \cdot b_1$	$a_{2j} \cdot b_1$	$a_j \cdot b_1$	$a_1 \cdot b_1$
$\underline{a_{4j}} \cdot b_j$	$a_{3j} \cdot b_j$	$a_{2j} \cdot b_j$	$a_j \cdot b_j$	$a_1 \cdot b_j$	$\underline{a_0} \cdot b_j$
$a_{3j} \cdot b_{2j}$	$a_{2j} \cdot b_{2j}$	$a_j \cdot b_{2j}$	$a_1 \cdot b_{2j}$	$\underline{a_0} \cdot b_{2j}$	$\underline{a-1} \cdot b_{2j}$
J_{5j-1}	J_{4j-1}	J_{3j-1}	J_{2j-1}	J_{1j-1}	$\underline{J_{0j-1}}$
J_{5j-i}	J_{4j-i}	J_{3j-i}	J_{2j-i}	J_{1j-i}	$\underline{J_{0j-i}}$

表中分量 $a \cdot b$ 表示二者相与, J 是用于寄存后域向本域进位的临时变量;为算法描述方便引入冗余为 0 的分量(表中用下划线示).

各列(C_i)的分量之和为 C_i 域的总量从而得积 C 的各域总量.对 C 的各域作二进制加法调整得 C 本次操作的部分积.该积又为下步乘法的被乘数.这样递进生成被乘数与下一个乘数变量相乘.最后得全集的总积.

关系语言描述如下:

设连乘变量名为 $A_i (i = 1, \dots, n)$

creat tab T_{A_k} ; // 建变量表,形如表 1

// 建乘积表且对应引入第一个变量(被乘数) A_1

creat tab CN link $T_{A_k} \cdot A_1$;

for(A_k)($k = 2, \dots, n$) { // 对每个乘数变量循环

// 按表 2 算式计算每列的累加和存入对应 C_i

insert ($C_i = C_s \cdot A_t$)($s = i, \dots, i-m; t = 1, \dots,$

m) from (CN, $T_{A_k} \cdot A_k$) into C_i

// 对各 C_i 二进制加法调整生成本次的乘积

join CN when $\{(C_i)2+\}$ }

从描述可知,CN 表存放本次乘法乘积的中间结果又是下次乘法的被乘数且逐级增长.采用了直接从乘、被乘数寄存器的位值"逻辑与"式得同域数量,而非加法时所采用的以同列为一个域的方式.这样可节省空间和提高速度.VHDL 语言描述见 3.3 节,扩展相应的变量数量及进位变量即得连乘($\prod$)运算.

3.3 积分($\int$)函数运算

设积分式 $\int_a^b f(x)dx, f(x) = x^3, a = 1, b = 10$.

$f(x)$ 函数是 3 个 x 连乘的积,根据 3.2 节中的关系描述,连乘的 VHDL 主要语句如下:

function fc(x) return INTEGER is

signal x :STD_LOGIC_VECTOR

(m to 1) := $\{x_m, \dots, x_i, \dots, x_1\}$; // x 的初值

signal fx:STD_LOGIC_VECTOR

(3($n+1$) to 1);

signal JS :INTEGER ;

for i in 1 to 3($m+1$) loop // 计算全域

for j in i to $i * m$ loop // 计算本域

if ($x_j, x(j-m+1)$) then JS: = JS + 1;

endif

end loop;

JS: = JS+fx. jj;

// 转换为本位积(逻辑类型)

fx $\leq$ (STD_LOGIC_VECTOR) JS ;

end loop;end fc ;

用复化梯形求积分^[7]是一个相乘后再累加的算法(C 语言描述):

for ($h = (b-a)/n, t=0, i=1; i=n; i++$)

$t = t + h/2 * (f(i) + f(i+1))$;

根据 3.1 节和 3.2 节中的关系描述,其对应的主要 VHDL 语句:

function fjf($a, b, n, fc(x)$)

return INTEGER is

signal h :INTEGER:= $(b-a)/n$;

signal t, t_1 :INTEGER;

signal y_i, y_n :STD_LOGIC_VECTOR;

```

for i in a to b loop // 对每个步进循环
// fj 函数实现两数加法, 参见 3.2 节
 $t_1 := f_j(fc(i), fc(i+1));$ 
// fC 函数实现两数乘法, 参见上文的  $fc(x)$ 
 $t_1 := fC(t_1, h);$ 
 $t_1 := t_1 \text{ srl } 1;$  // srl 逻辑右移, 实现  $(t_1/2)$  运算
 $t := f_j(t, t_1);$  // 累加, 参见 3.1 节
end loop; end fjf;

```

可见, 积分数值运算的逻辑模型可以用基本运算的逻辑模型进行组合实现; 也可以将整个函数的模型形成一个固件, 可以更大提高运行速度, 但因为该固件只针对这一种函数, 所以降低了其应用的灵活性。

综上所述, 通过对函数的输入变量各种取值所产生的函数值建立逻辑模型, 其实质是一个逻辑性化简编程的大型翻译器。这种方法可以扩展至其他类型的变量及其各种函数和过程运算。

4 测试比较

采用 C 语言环境及 Lattice ispEXPERT(lattice 公司可编程逻辑器件开发软件)^[8] 分别开发 3.3 节中的积分运算(复化梯形公式)程序, 进行测试比较。Lattice ispEXPERT 对 ispLSI/pLSI6192 进行编程配置并接入 PC 的 IO 接口卡; 将 C 语言直接调用积分程序(语言级实现)和 C 语言通过嵌入汇编调用 IO 接口卡访问 isp 芯片(固件级实现)进行比较:

// 下限, 上限, 步长, 值;

integer ia=0, ib=10, n=10, it=0;

(1) C 语言直接实现

function iy(ix){ $iy = ix^3$ }; // 被积函数

for ($i=1; i < n+1; i++$) {

// 复化梯形公式

{ $t = t + ih/2 * (iy(i) + iy(i+1));$ }

(2) C 语言嵌入汇编访问接口实现

function iv(ix){ // 被积函数(访问接口得函数值)

// 660H~669H 为 I/O 地址

asm{mov DX, 660H; mov AX, ix; out DX, AX;

mov DX, 663H; in AX, DX; mov iv, AX; }

for ($i=1; i < n+1; i++$) {

$i_1 = iv(i); i_2 = iv(i+1);$ // 得两点函数值

// 实现 $(i_1 + i_2) * h/2$ 固化逻辑

{ -asm{mov DX, 664H; mov AX, i_1 ; out DX,

A; ...;

mov DX, 666H; mov AX, h; out DX, A;

mov DX, 669H; in AX, DX; mov t, AX; }

分别各自调用对应的函数 1 000 万次, 前者用时约 1 s, 平均每次用时 100 ns; 后者用时约 0.6 s, 平均每次用时 60 ns。去除相同用时部分, 后者比前者约快 1 倍左右, 而且后者在芯片选择和接口设计方面还有提速的空间。

5 结束语

提高 CPU 时钟的频率是提高系统运行速度的重要技术, 但不论 CPU 的频率如何提高, 只靠其几种运算部件(比如加法器)很难对各类应用环境达到高速。首次提出以真值关系逻辑进行逻辑模型设计, 通过对真值关系的两层(关系级和 VHDL 语言级)编程, 可对任何特定的运算(或函数)进行前期的设计和固化从而形成固件。

在真值关系逻辑的两层(关系级和 VHDL 语言级)编程中, 由于 VHDL 语言级设计完全是以关系级描述为基准, 所以更进一步的工作是将 VHDL 设计编译化, 即设计一个由关系级语言编译出 VHDL 语言的系统, 只要给出前者即可直接获得后者, 从而进一步提升建立逻辑模型的方法。当前可编程器件及相应的开发手段正向大规模、高速和在系统可编程等方面发展, 其在总体提高运算部件运行速度方面将更加广阔。

参考文献:

- [1] 胡敏. 基于层次化重复单元的 FPGA 结构描述方法[J]. 计算机工程, 2013, 39(7): 318-321.
- [2] Kuon I, Tessier R, Rose J. FPGA architecture: survey and challenges[J]. Foundations and Trends in Electronic Design Automation, 2008, 2(2): 135-253.
- [3] 王若成. 基于 ISP 技术的存储器实验系统改进与设计[J]. 实验室研究与探索, 2010, 29(8): 64-66.
- [4] 王进鹏. SPARQL 查询的关系代数表示与转换方法[J]. 计算机工程与应用, 2011, 47(22): 110-113.
- [5] 李云松. Xilinx FPGA 设计基础: VHDL 版[M]. 西安: 西安电子科技大学出版社, 2008: 64-146.
- [6] 李高斌. 组合逻辑电路自动合成算法的硬件实现[J]. 微电子学, 2011, 41(6): 876-882.
- [7] 张士勤. 带端点导数的梯形修正公式[J]. 河南师范大学学报: 自然科学版, 2009, 37(3): 25-26.
- [8] 尹立贤. HDB3 码的 ISP 技术实现研究[J]. 现代电子技术, 2009, 269(5): 176-178.

作者简介:

何 群 男, (1955-), 高级工程师, 硕士生导师。研究方向为粗糙集、粒计算和可编程器件。E-mail: hequn2000 @ 126.com.