不恢复余数阵列除法器的形式化描述和验证方法*)

张欢欢　宋国新
(华东理工大学计算机科学与工程系　上海 200237)

摘　要　本文使用重写技术对不恢复余数阵列除法器进行了形式化描述并结合归纳法对该除法器的正确性进行了验证，整个工作是建立在串行加法器的描述和验证基础上的。不恢复余数阵列除法器的运算和控制有一定的复杂度，适合用大规模集成电路实现。本文成功地用重写归纳法对它进行了描述和验证，说明重写归纳法在硬件电路正确性验证方面有广阔的应用前景。

关键词　重写，归纳，除法器，描述，验证

Formal Specification & Verification of Non Restoring Array Divider

ZHANG Huan-Huan　SONG Guo-Xin
(Department of Computer Science and Engineering, East China University of Science and Technology, Shanghai 200237)

Abstract　Non restoring array divider is a complex arithmetic circuit that can be built using very large scale integrated circuit. In this paper, a specification of non restoring array divider is established using rewrite rules. Rewriting induction techniques are directly used to verify the correctness of the division circuit by proving that it can compute the correct middle result at each step. The specification and verification are based on the correctness of ripple carry adders we have discussed elsewhere in an earlier paper. It shows that rewriting induction has wide applications in the area of complex hardware verifications.

Keywords　Rewriting, Induction, Divider, Specification, Verification

1　引言

近十几年计算机硬件以惊人的速度发展，与硬件正确性相关的技术越来越受到人们的重视。电路正确性验证方面传统的方法分为两类：模拟(simulation)和仿真(emulation)。它们一般是利用硬件编程语言(如 Verilog 和 VHDL)所构造的模型，输入激励向量，通过对比其输出是否与预期的一致来进行硬件的正确性验证。这两种方法的主要缺陷在于：只能对典型的情况加以考察，对系统进行穷尽模拟或仿真是很难甚至是不可能的。形式化方法用于硬件电路的正确性验证正是在这种形势下发展起来[1]。

一般来说，形式化方法就是指用具有形式语义的记号和工具明确地表达所要设计的系统的设计要求或系统特性，即给出系统规范(specification)，并根据系统规范利用上述记号和工具对系统具有的性质和最终实现(能够完成规定功能的具体的系统)的正确性进行严格地证明。

本文在对加法器[3,4]进行验证的基础上，用重写归纳技术[9,10,12]对不恢复余数的阵列除法器进行描述和验证。重写系统是一个重要计算模型，具有计算速度快的优点，在数学和计算机的许多领域(如抽象数据类型、函数式程序设计语言、程序正确性证明和自动推理等)中都有广泛的应用。另一方面，归纳法作为经典的数学证明方法，在人工智能、程序验证和理论计算机科学中有重要的应用。重写归纳技术作为重写系统和归纳法的结合，在软件的开发和验证方面已经取得了较好的进展。但是在硬件验证方面应用还不多，研究较多的有算术电路的建模与验证[2,5~7]、处理器的建模与验证[8,11]、傅里叶变换电路的建模与验证[13~15]等。

本文的第 2 部分介绍不恢复余数除法原理及硬件实现，第 3 部分简单介绍重写系统的原理，第 4 部分给出基于重写技术的不恢复余数阵列除法器的形式化描述，第 5 部分利用重写归纳技术对该除法器进行正确性验证，最后给出结论以及与相关方法的比较。

2　不恢复余数除法原理及硬件实现

计算机中两个原码表示的纯小数相除时，商的符号位由两个运算数的符号异或求得，商的数值部分通过这两个数的数值部分按绝对值求商得到，此外，还得到一个余数。在下面讲述的除法器中，假设参与运算的数是正的规格化的纯小数，且保证运算无溢出，即被除数<除数。

计算机中的除法的商是通过从被除数中减去除数与对应位的商(下面简称位商)的积一位一位计算的，直到商的字长为止。第 i 步的部分余数 R_i、部分商 Q_i、除数 divisor、被除数 dividend、位商值 q_i 之间满足如下关系：

$R_0=\text{dividend}/2, Q_0=0$

$R_{j+1}=2*R_j-q_{j+1}*\text{divisor}$

$Q_{j+1}=2*Q_j+q_{j+1}\quad j=0\cdots n-1$，$n$ 为机器字长

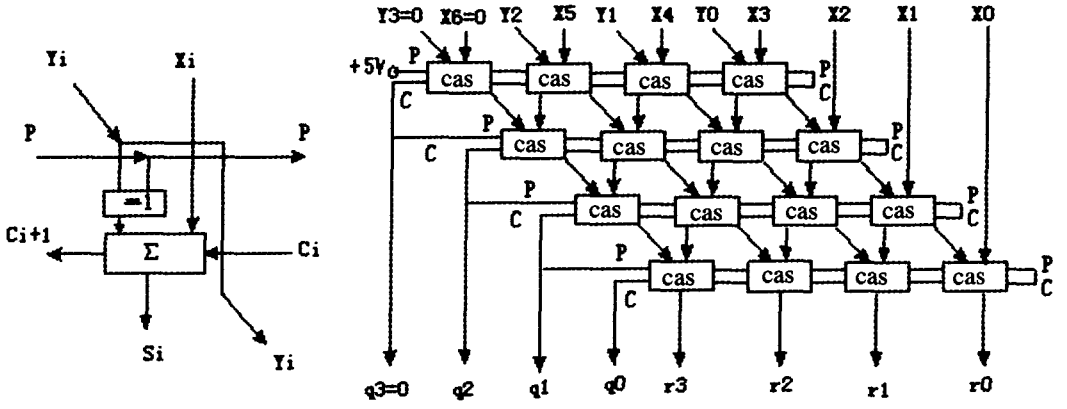
除法器分为恢复余数和不恢复余数两种，不恢复余数除法器具有运算步数固定、控制简单的优点，其不恢复余数的原理是从恢复余数的算法中推出的。恢复余数算法中需要恢复余数的环节为：设某次余数为 r_i，则下一步的余数 $r_{i+1}=2*r_i$

*)本文得到国家自然科学基金资助(No. 60373075)。张欢欢　副教授，博士生，主要从事形式化方法和软件工程方面的研究；宋国新　教授，博士生导师，主要研究领域为形式化方法和软件验证理论。

$-y$，当 $r_{i+1}<0$ 时表明不够减，商上 0 并且余数加 y 恢复原来的余数，即 $(2*r_i-y)+y=2*r_i$。若继续下面的求商，则再下一部的余数 $r_{i+2}=2*(2*r_i)-y=4*r_i-y$。

当 $r_{i+1}<0$ 时，若商仍上 0 但不进行加 y 恢复余数操作，而是把 $(2r_i-y)$ 左移一位，然后加上除数 y，即 $2*(2*r_i-y)+y=4r_i-y$，所得效果相同。所以，当比较结果小于 0 时，求下面的商时仍将未修正的余数左移一位，然后加 y 进行比较就是所谓的不恢复余数法又称为加减交替法。从这一原理可得如下引理。

引理 不恢复余数算法中，第 i 步的对应位商上 0 时 $(q_i=0)$，该步实际的余数为计算出的余数与除数的和(实际余数=Ri+divisor)。



(a) 可控加减单元　　(b) 由 CAS 组成的不恢复余数阵列除法器

图 1　不恢复余数阵列除法器

不恢复余数阵列除法器的结构图如图 1。其中图 1(a)是可控加减单元 CAS 的逻辑框图，输入信号 $P=0$ 时，完成全加器 $X_i+Y_i+C_i$ 功能。$P=1$ 时，完成 $X_i+\overline{Y}_i+C_i$ 功能。图 1(b)是由 CAS 组成的不恢复余数阵列除法器，被除数 X 沿竖线送入 CAS，除数沿斜线送入 CAS，信号 P 控制加减运算。由于是原码除法，所以第一步进行减法运算 $P=1$，由于 P 同时连接到该行末尾进位位，所以第一行执行 $X+\overline{Y}+1=X-Y$ 运算，在被除数<除数的条件下，该位的商一定是 $q=C=0$，该位商还被连到下一行的 P 用于控制下一行的加减。显而易见其后各行的加减取决于前一行的进位位 C(够减不够减)，规则如下：

1)若够减，则 $C=q=$下一行的 $P=1$，本步计算出的部分余数左移一位供下一行做减法操作。

2)若不够减，则 $C=q=$下一行的 $P=0$，本步计算出的部分余数左移一位供下一行做加法操作。

3　重写系统简介

重写系统(Term Rewriting System)定义为一个二元组 $\langle R,S\rangle$，其中 S 代表项的集合，R 代表形如 $l\rightarrow r$ 的重写规则集合，其中 l 和 r 是 S 中的项，可称为规则的左项和右项。

项 s 可以被重写或归约为 t(记为 $s\rightarrow t$)，它的含义是存在 s 的子项 s'，它依据重写规则可以转换成 s''，而将 s 中的 s' 出现都用 s''代替就得到 t。重写系统的其他重要概念，如范式、自反闭包、自反传递闭包、终止和合流等请参阅参考文[10]。

在进行硬件形式化建模和验证时，项和重写规则分别对应硬件系统的状态和状态转换规则。

4　基于重写技术的不恢复余数阵列除法器的形式化描述

为了给出不恢复余数的阵列除法器的形式化描述，我们首先定义最基本的 bool 类型，本文用逻辑常元符号 0(表示真)和 1(表示假)分别表示逻辑位 0 和 1，非、或、与、与非、异或逻辑操作则分别用函数符号 non、or、and、nonand、xor 表示，它们可以用重写系统进行刻画，限于篇幅这里不再给出详细描述，可参阅文[3]。用归纳法可以证明：non、or、and、nonand、xor 分别实现了非门、或门、与门、与非门、异或门的逻辑功能。

我们用逻辑位 0 和 1 组成的串表示二进制数，可以用基类型为布尔类型的列表进行描述，其构造算子为 nil 和 cons。

List=

S：：list

∑：：nil：list

cons：bool,list→list

由于 cons(x,y)表示在列表 y 之前插入元素 x，所以这里的描述方法与传统的表示法相反：列表的第 1 个元素作为最低位。这样，二进制数纯小数 0.11(或整数 011)表示为 cons(1,cons(1,cons(0,nil)))，此外，列表的左(右)移对应实际操作数的除以(乘以)2 运算也与传统的表示法相反。

由上面的叙述可知不恢复余数阵列除法器的基本运算单元是 CAS(可控加减单元)，其核心结构是全加器，从图中可得它实现的功能是：fulladder$(c,x,xor(y,p))$，所以可描述为：

Cas$(x,y,c,p)\rightarrow$fulladder$(c,x,xor(y,p))$

可见在图 1 所示的阵列除法器中，$n+1$ 位的两个纯小数 X、Y 相除的阵列除法器可以看成 $n+1$ 行由 CAS 组成的串行加法器，每行串行加法器进行 $n+1$ 位加(减)法。各行的最高进位位 C 与下一行的 P 相连构成进位反馈控制下一行的加减。依据文[3]的结论，当 $p=0$ 时计算 $X+Y+0$，当 $p=1$ 时计算 $X+(\overline{Y}+1)=X-Y$。串行加法器可用函数符号 rca：bool,list,list→list 描述[3]，其中参数中的 bool 对应低位的进位位、两个 list 对应被加数和加数，结果为和所对应的表。其形式化描述为：

rca：bool,list,list→list

rca$(x,cons(y,nil),$cons$(z,nil))\rightarrow$

if-then-else(non(fulladdercarry(x,y,z)),

cons(fulladdersum(x,y,z), cons

(0,*nil*)),

cons(fulladdersum(*x*, *y*, *z*), cons(1,*nil*)))

rca(*x*,cons(*y*1,cons(*y*2,*y*)),cons(*z*1,cons(*z*2,*z*))→cons(fulladdersum(*x*,*y*1,*z*1),rca(fulladdercarry(*x*,*y*1,*z*1),cons(*y*2,*y*),cons(*z*2,*z*))

其中 if-then-else(0,*x*,*y*)→*y* if-then-else(1,*x*,*y*)→*x*

不失一般性，设被除数 $X=0.x_{n-1}..x_1x_0$，除数 $Y=0.y_{n-1}\cdots y_1y_0$（不足位补 0），商 $Q=0.q_{n-1}\cdots q_1q_0$，余数 $R=0.r_{n-1}\cdots r_1r_0$ 且保证运算无溢出，即 $0<X<Y$，则商的最高位（小数点前的值）一定为 0。

将被除数和部分余数表示为双倍字长的表 $R=X=\{0\cdots0,x_0,x_1\cdots x_{n-1},0\}$，除数表示为 $Y=\{y_0,y_1\cdots y_{n-1},0\}$，商表示为 $Q=\{q_0,q_1,\cdots,q_{n-1},0\}$，初始控制位 $p=1$，初始进位位 $c=p=1$。可用如下函数符号描述不恢复余数的阵列除法器：

tail:*list*→*bool*

XOR:*list*,*bool*→*list*

P:*int*→*bool*

R:*int*→*list*

Quotient:*int*→*list*

分别表示取表尾元素值、表中元素的逐位异或运算、控制位、部分余数、商，其具体形式化描述如下：

tail(*cons*(*x*,*nil*))→*x*

tail(*cons*(*x*,*y*))→*tail*(*y*)

XOR(*nil*,*b*)→*nil*

XOR(*cons*(*y*0,*y*),*b*)→*cons*(*xor*(*y*0,*b*),*XOR*(*y*,*b*))

P(0)→1

P(*j*+1)→*tail*(*R*(*j*))

R(0)→*rca*(1,*X*,*XOR*(*Y*,1))

R(*j*+1)→*rca*(*P*(*j*+1),*R*(*j*)*2,*XOR*(*Y*,*P*(*j*+1)))

Quotient(0)→*cons*(0,*nil*)

Quotient(*j*+1)→*cons*(*tail*(*R*(*j*+1)),*Quotient*(*j*))

5 不恢复余数阵列除法器的正确性验证

不恢复余数阵列除法器的正确性可通过是否满足下列定理来验证：

定理：一个正确的除法器的被除数 X、除数 Y、第 i、$i+1$ 步的部分商 Q_i、Q_{i+1} 和部分余数 R_i、R_{i+1} 之间满足如下关系：

$Q_0*Y+R_0==X$

$Q_{i+1}*Y+R_{i+1}==(Q_i*Y+R_i)*2 \quad i=0\cdots n-1$

下面我们用重写归纳法来验证不恢复余数阵列除法器的正确性。由前面的引理阵列除法器的对应位商上 0 时，实际的余数＝计算的部分余数＋Y。为了书写方便将实际余数记为 $R(i)+t$，其中 t 可取 0 或 Y。

证明：1）奠基：

除法器的第一行运算输入为被除数 X，商为 *Quotient*(0)，部分余数为 $R(0)$

∵ $Y*Quotient(0)+(R(0)+t)$

→$Y*0+(R(0)+t)$

→$Y*0+(rca(1,X,XOR(Y,1))+t)$

→$0+X-Y+t$

依据引理可知商上 0 时，需要修正余数才能求得该步真正的余数：

∴上式→$0+X-Y+Y$→X

∴结果正确。

2）假设第 i 步运算上述关系成立，则第 $i+1$ 步运算：

$Quotient(i+1)*Y+(R(i+1)+t)$

→$cons(tail(R(i+1)),Quotient(i))*Y+(rca(P(i+1),R(i)*2,XOR(Y,P(i+1)))+t)$

①当 $P(i+1)=1$ 时，上式→$cons(b,Quotient(i))*Y+(rca(1,R(i)*2,XOR(Y,1))+t)$

→$cons(b,Quotient(i))*Y+(R(i)*2-Y+t)$

其中 $b=tail(R(i+1))$ 代表第 $i+1$ 步的位商是一个布尔值，下面对 b 进行归纳：

当 $b=0$ 时，上式→$cons(0,Quotient(i))*Y+(R(i)*2-Y+t)$

→$(Quotient(i)*2+0)*Y+(R(i)*2-Y+t)$

→$2*Quotient(i)*Y+(R(i)*2-Y+t)$

→$(Quotient(i)*Y+R(i))*2-Y+t$

由引理，$b=0$ 说明第 $i+1$ 步运算需要修正余数才能得到真正的余数

∴上式→$(Quotient(i)*Y+R(i))*2-Y+Y$

→$(Quotient(i)*Y+R(i))*2$

当 $b=1$ 时，$Quotient(i+1)*Y+(R(i+1)+t)$

→$cons(1,Quotient(i))*Y+(R(i)*2-Y+t)$

→$(Quotient(i)*2+1)*Y+(R(i)*2-Y+t)$

→$2*Quotient(i)*Y+Y+R(i)*2-Y+t$

→$2*Quotient(i)*Y+R(i)*2+t$

而此时的余数即为实际除法的余数，所以 $t=0$

∴上式→$(Quotient(i)*Y+R(i))*2$

②当 $P(i+1)=0$ 时，$Quotient(i+1)*Y+(R(i+1)+t)$

→$cons(b,Quotient(i))*Y+(rca(0,R(i)*2,XOR(Y,0))+t)$

→$cons(b,Quotient(i))*Y+(R(i)*2+Y+t)$

当 $b=0$ 时，上式→$cons(0,Quotient(i))*Y+(R(i)*2+Y+t)$

→$(Quotient(i)*2+0)*Y+(R(i)*2+Y+t)$

→$2*Quotient(i)*Y+(R(i)*2+Y+t)$

→$(Quotient(i)*Y+R(i))*2+Y+t$

由引理一，$b=0$ 说明第 $i+1$ 步运算需要修正余数才能得到真正的余数

∴上式→$(Quotient(i)*Y+R(i))*2+Y+Y$

→$(Quotient(i)*Y+R(i))*2+2*Y$

→$(Quotient(i)*Y+R(i)+Y)*2$

当 $b=1$ 时，$Quotient(i+1)*Y+(R(i+1)+t)$

→$cons(1,Quotient(i))*Y+(R(i)*2+Y+t)$

→$(Quotient(i)*2+1)*Y+(R(i)*2+Y+t)$

→$2*Quotient(i)*Y+Y+(R(i)*2+Y+t)$

而此时的余数即为实际除法的余数，所以 $t=0$

∴上式→$(Quotient(i)*Y+R(i))*2+2*Y$

→$(Quotient(i)*Y+R(i)+Y)*2$

又∵$P(i+1)=0$ 并且 $P(i+1)$→$tail(R(i))$

∴第 i 步的商的位值为 $tail(R(i))=P(i+1)=0$

∴上一步的余数需修正才是真正的余数，即 $R(i)+Y$ 为上一步真正的余数。

∴由①②不恢复余数的阵列除法器满足定理，由此证明它是正确的。

（下转第 291 页）

```
    ST_PlayUIState,
    ST_BrowseState,
    ST_BrowsePlayState,
    ST_LastState, /* Keep this Last */
} stbStateIndex_t;
```

stbStateIndex_t 定义了状态的种类，它与机顶盒真实状态间是一一对应的，是机顶盒预先定义的状态。

```
typedef enum _control_status_{
    ST_UnloadStatus=0,
    ST_StandbyStatus,
    ST_ActiveStatus
}stbCtrlStatus_t;
typedef struct {
    stbCtrlStatus_t controls [ST_NUM_CONTROLS];
} stbState_t;
```

stbState_t 定义了状态节点。状态节点是控制器的状态的集合。

```
typedef struct _MSG_s_{
    unsigned int type; //Identity of message
    unsigned int msgId; //Message value
    unsigned int session; //Unique session key for a whole request
    unsigned int pid; //Sender PID
    unsigned int size; //Size of content
    int reserved [3];
    unsigned char content
    [MSG_MAXSIZE - MSG_HEADER_SIZE];
}Msg_t;
```

Msg_t 定义了事件。这里事件是消息，状态机通过响应消息来改变自身状态。

```
typedef struct _controls_s_{
    unsigned int pid;
    stbStatus_t status;
    stbMsgFunc_t load;
    stbMsgFunc_t activate;
    stbMsgFunc_t deactivate;
    stbMsgFunc_t process;
    stbMsgFunc_t unload;
} stbControl_t;
```

stbControl_t 定义了动作，来实现状态的迁移。

3.3.2 接口函数定义

int InitSM(void)

初始化状态机。

getNextState(int curState, msg_t * pMsg, int msgHandled)

检查消息是否引起状态迁移。

void transitState(msg_t * pMsg)

目前的状态转换到下一个状态。

3.3.3 状态机的驱动

```
void MC_loop(){
    while(1){
        rcvMsg(&msg);
        NextState=getNextState(CurrentState,&msg);
        if(NextState ! =CurrentState)
        transitState(NextState);
        sleep(1);
    }
}
```

抛去一些不必要的细节，状态机通过不断接收消息来响应外界的各种事件，实现状态转移，确保系统正确运行。

结 论 嵌入式系统中，资源稀少，实时性要求高，同时随着技术的发展，应用需求越来越高，使得应用程序复杂程度也越来越高。利用有限状态机模型，可以大大简化编程，尤其是在数字电视系统中，状态通常是确定的和有限的，因此，利用有限状态机模型，可以减轻开发人员的工作量，提高软件的可读性，增强软件的可维护性和扩充性。

参 考 文 献

1 熊振云，阮俊波，金惠华. 嵌入式软件中状态机的抽象与实现. 计算机应用，23(10)

2 徐小良，等. 有限状态机的一种实现框架. 工程设计学报，10(5)

(上接第 285 页)

结论 本文在串行加法器的基础上用重写系统对不恢复余数阵列除法器进行了严格的描述和直观的证明，说明了重写归纳技术在硬件正确性验证方面存在优势。与传统的模拟、仿真相比，其明显的优点是一方面它解决了由于实际的计算机字长较长、状态较多传统方法很难穷尽模拟这一问题，另一方面它使用的数学方法证明更加严格可靠。

本文的验证方法还未实现自动化，进一步的工作包括研究使用验证工具进行自动化验证、过程中的引理推测和效率改进等。

参 考 文 献

1 韩俊刚，杜慧敏. 数字硬件的形式化验证. 北京：北京大学出版社，2001

2 Kapur D, Subramaniam M. Mechanizing Verification of Arithmetic Circuits: SRT Division. In: Proc. 17th FSTTCS, Vol. 1346 of LNCS, Springer Verlag, 1997. 103～122

3 张欢欢，邵志清，宋国新. 基于重写归纳技术的串行加法器的描述和验证[J]. 华东理工大学学报，2003，1(29)：59～64

4 张欢欢，邵志清，宋国新. 基于幂表的并行加法器的归纳验证[J]. 电子学报，2003，6(31)：932～937

5 Kapur D, Subramaniam M. Mechanically verifying a family of multiply circuits [A]. In: Proc. of 8th Conf on Computer Aided Verification [C]. Berlin: Springer-Verlag, 1996. 135～146

6 Kapur D, Subramaniam M. Using and Induction Prover for Verifying Arithmetic Circuits. J. of Software Tools for Technology Transfer. Springer-Verlag, 2000, 3(1): 32～65

7 Akbarpour B, Tahar S, Dekdouk A. Formalization of Fixed-Point Arithmetic in HOL. Formal Methods in Systems Design, Springer Verlag, 2005, 27(1-2): 173～200

8 Tahar S, Zobair M. H, Song X. Formal Verification of a SONET Data Stream Processor. IEE Proceedings - Computers and Digital Techniques, 2004, 151(1): 71～81

9 Kort S, Tahar S, Curzon P. Hierarchical Formal Verification Using a Hybrid Tool. International Journal on Software Tools for Technology Transfer, Springer Verlag, 2002, 4: 1～10

10 Baader F, Nipkow T. Term Rewriting and All That. Cambridge University Press, 1998

11 Arvind, Shen Xiaowei. Using Term Rewriting Systems to Design and Verify Processors. IEEE Micro Special Issue on Modeling and Validation of Microprocessors, 1999

12 Hoe J C, Arvind. Hardware Synthesis from Term Rewriting Systems. In: Proceedings of X IFIP International Conference on VLSI (VLSI 99), Lisbon, Portugal, November 1999

13 Mauricio Ayala-Rincon, Nogueira R B, Llanos C H, Jacobi R P. Modeling a Reconfigurable System for Computing the FFT in Place via Rewriting-Logic. In: Proceedings of the SBCCI'03. IEEE Computer Society Press, São Paulo, SP, Brazil, 2003. 205～210

14 Mauricio A-R, Nogueira R B, Llanos C H, Jacobi R P, Hartenstein R. Efficient Computation of Algebraic Operations over Dynamically Reconfigurable Systems Specified by Rewriting-Logic Environments. Conferencia Internacional de Ciencia de la Computación, Bio-Bio. 23rd SCCC/2003. 60～69

15 Mauricio A-R, et al. Modeling and Prototyping Dynamically Reconfigueable Systems of Efficient Computation of Dynamic Programming methods. In: 17th Symp. On Integrated Circuits and Systems Design. ACM Press, 2004. 248～253