基于 FPGA 的 DES 加密芯片的设计

蒋存波,孙朝华,杜婷婷,陈 铭,程小辉
JIANG Cun-bo,SUN Chao-hua,DU Ting-ting,CHEN Ming,CHENG Xiao-hui

桂林工学院,广西 桂林 541004
Guilin University of Technology,Guilin,Guangxi 541004,China
E-mail:sunchaohua1976@163.com

JIANG Cun-bo,SUN Chao-hua,DU Ting-ting,et al.Hardware design of DES algorithm based on FPGA.Computer Engineering and Applications,2008,44(16):83-86.

Abstract: This paper introduces the basic principles of the DES encryption system,achieves the optimum FPGA reconstructing design on the DES with the help of giving the algorithm of DES encryption under the Verilog language and the C language conditions and the effective dealing with the key issues.Then it gives a new dynamic management design programme of the keys.Finally,the results of the design function simulation and test analysis show that the design of the whole thesis has higher practical significance and value.

Key words: Data Encryption Standard(DES);FPGA;Verilog language;C language

摘 要:通过对 DES 加密原理的分析,推导出了 DES 的算法公式,通过对算法中核心部分的数学分析和化简,借助 Verilog 语言与 C 语言编程以及 EDA 设计软件的帮助,实现了 DES 算法的 FPGA 条件下的重构设计,同时对密钥的动态管理提出了新的设计方案。最后,通过对设计结果的功能仿真和测试分析,论证了整个设计过程的正确性。

关键词:DES;FPGA;Verilog 语言;C 语言
DOI:10.3778/j.issn.1002-8331.2008.16.025 **文章编号**:1002-8331(2008)16-0083-04 **文献标识码**:A **中图分类号**:TP336

1 引言

DES(Data Encryption Standard)[1]是应用得最为广泛的分组加密系统之一,除穷举法外尚未发现其它快速破解方法的成功案例。目前 DES 算法通常是使用基于软件的实现方法,尽管软件实现加密/解密过程十分方便而且设计灵活,但加/解密过程计算量大,会过多地消耗处理器资源从而降低系统的整体性能,因此,在许多对实时性要求较高的数据加密领域,软件加密方法存在缺陷。当前已开始出现硬加密专用芯片(例如 HT32A256)应用于高端 IC 卡等领域,利用 CPLD/FPGA 实现加/解密功能则是一种在灵活性、安全性等方面更具优势的选择。本文介绍通过借助相关工具软件,利用可编程芯片实现 DES 加密接口芯片的优化设计方法。

2 DES 工作原理介绍和设计分析

2.1 工作原理介绍

DES 的工作原理如图 1 所示,加密过程主要分为三个步骤:

(1)64 位(bit)的明文分组进行操作,进行初始变换 IP(Initial Permutation),搅乱序列数原来的顺序后,再分为 L_0 与 R_0 两个 32 位的分组序列数;

(2)R_0 与第一个子密钥 $K1$ 一同经过加密函数($f(R_i,K_j)$)运算,得到 32 位的输出再与 L_0 做逐位异或运算,其结果成为下一轮的 R_1,R_0 则成为下一轮的 L_1,如此连续 16 轮,而后得到的 R_{16} 与 L_{16} 不须再互换,直接连接成 64 位的序列数;

(3)最后,对 64 位的序列数一起经过逆初始变换(IP^{-1}),从而实现密文输出。

解密过程与加密相同,只是子密钥的使用顺序刚好相反,取 $K_j(16\cdots1)$即可实现。

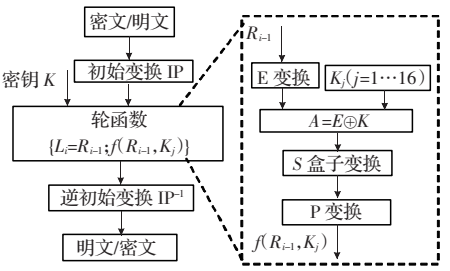

图 1 DES 算法原理图

2.2 算法描述与设计分析

(1)IP 变换公式:初始变换 IP 的目的是将 64 位明文 $B=(b_0,b_1,\cdots,b_{62},b_{63})$按一定规则换位得到位序改变的 64 位序列

基金项目:广西省自然科学基金(the Natural Science Foundation of Guangxi Province of China under Grant No.0447099)。

作者简介:蒋存波(1962-),男,副教授,主要研究方向:智能控制技术,嵌入式系统与应用;孙朝华(1976-),男,电气工程师,主要研究方向:嵌入式系统及应用,微型计算机应用。

收稿日期:2008-01-17 **修回日期**:2008-04-21

数$(L_0,R_0)=M=(m_0,m_1,\cdots,m_{63})=(b_{\tau(0)},b_{\tau(1)},\cdots,b_{\tau(62)},b_{\tau(63)})=(b_{57},b_{49},b_{41},\cdots,b_{14},b_6)$,其中$b_i$、$m_i$表示对应位序列的位编号。$L_0$和$R_0$分别为序列数$M$的左半部分和右半部分,$M$中序号$n$与明文中序号$\tau(n)$的关系由公式(1)确定(其中,int为取整运算符号)。

$$\tau(n)=57-8n+66\cdot\text{int}(n/8)-9\cdot\text{int}(n/32)\quad(n=0,1,\cdots,63)\tag{1}$$

例如,当$n=0$时,代入公式(1)有$\tau(0)=57$,即$m_0=b_{\tau(0)}=b_{57}$,这说明经过IP变换后,明文中的序列位b_{57}作为序列数M的第0位序位输出。依次将$n=1,2,\cdots,63$分别代入公式(1),可以求得M的所有位序号值。

(2)IP^{-1}为变换IP的逆变换,目的是将轮函数最后一轮的运算结果再进行一次位序变换后作为最终结果输出,变换后的位序$\tau'(n)$与变换前的位序n的关系由公式(2)确定。

$$\tau'(n)=\begin{cases}39+4n-33\cdot\text{int}(n/8) & (n=0,2,\cdots,62)\\ 3+4n-33\cdot\text{int}(n/8) & (n=1,3,\cdots,63)\end{cases}\tag{2}$$

计算过程与公式(1)类似,通过取值计算可以求得变换后序列数中的每一位数值,从而得到最终输出值。

(3)轮函数的数学公式:轮函数是DES算法的核心,它是一个Feistel网络[1](替代-置换网络),即取一个长度为N(N为偶数)的分组,把它分成长度为$N/2$的L和R左右两部分,通过16轮循环运算后实现对数据的扩散、混淆和混乱。通过分析可以得到一个迭代型的轮函数算法公式:

$$\begin{cases}L_{i+1}=R_i\\ R_{i+1}=L_{i+1}\oplus f(R_i,K_j)\end{cases}(i=0,1,\cdots,15;j=1,2,\cdots,16)\tag{3}$$

轮函数第$i+1$轮的结果(L_{i+1},R_{i+1})取决于上一轮的输出,加密运算时输出$R_{i+1}=L_{i+1}\oplus f(R_i,K_j)$,解密时,对收到的信息进行相同的运算,得到$[R_{i+1}]\oplus f(R_{i+1},K_j)=[L_{i+1}\oplus f(R_i,K_j)]\oplus f(R_{i+1},K_j)=L_{i+1}$,从而恢复$L_{i+1}$,从而保证算法的可逆性。

通过原理图1可以知道,公式(3)中$f(R_i,K_j)$由R_i、E变换$(f_E(R_i))$、P变换(f_P)、子密钥K_j和S盒子变换共同决定,其中$R_i$$(R_i=[r_i(01),r_i(02),\cdots,r_i(32)])$的初值取$R_0$,E变换的作用是对32位的序列$R_i$在特定的二进制位位置重复使用某一位信息来进行位扩充,从而实现数据位的乱序,数据位对应关系如表1所示。从表中可以看出,$f_E(R_i)$的前二位分别是R_i的第32和01位,而$f_E(R_i)$中的最后一位则是R_i中第01位。P变换的作用是将经S盒子运算后的结果进行换位变换后作为轮函数的结果输出,它的求解与$f_E(R_i)$类似($S=[s_1,s_2,\cdots,s_{32}]$为S盒子运算后的序列数)。数据位对应关系如表2所示。

表1　E变换表

E位编号	R_j位编号	E位编号	R_j位编号	E位编号	R_j位编号	E位编号	R_j位编号
1	$r_j(32)$	13	$r_j(08)$	25	$r_j(16)$	37	$r_j(24)$
2	$r_j(01)$	14	$r_j(09)$	26	$r_j(17)$	38	$r_j(25)$
3	$r_j(02)$	15	$r_j(10)$	27	$r_j(18)$	39	$r_j(26)$
4	$r_j(02)$	16	$r_j(11)$	28	$r_j(19)$	40	$r_j(27)$
5	$r_j(04)$	17	$r_j(12)$	29	$r_j(20)$	41	$r_j(28)$
6	$r_j(05)$	18	$r_j(13)$	30	$r_j(21)$	42	$r_j(29)$
7	$r_j(04)$	19	$r_j(12)$	31	$r_j(20)$	43	$r_j(28)$
8	$r_j(05)$	20	$r_j(13)$	32	$r_j(21)$	44	$r_j(29)$
9	$r_j(06)$	21	$r_j(14)$	33	$r_j(22)$	45	$r_j(30)$
10	$r_j(07)$	22	$r_j(15)$	34	$r_j(23)$	46	$r_j(31)$
11	$r_j(08)$	23	$r_j(16)$	35	$r_j(24)$	47	$r_j(32)$
12	$r_j(09)$	24	$r_j(17)$	36	$r_j(25)$	48	$r_j(01)$

表2　P变换表

P位编号	S位编号	P位编号	S位编号	P位编号	S位编号	P位编号	S位编号
1	S_{16}	9	S_{01}	17	S_{02}	25	S_{19}
2	S_{07}	10	S_{15}	18	S_{08}	26	S_{13}
3	S_{20}	11	S_{23}	19	S_{24}	27	S_{30}
4	S_{21}	12	S_{26}	20	S_{14}	28	S_{06}
5	S_{29}	13	S_{05}	21	S_{32}	29	S_{22}
6	S_{12}	14	S_{18}	22	S_{27}	30	S_{11}
7	S_{28}	15	S_{31}	23	S_{03}	31	S_{04}
8	S_{17}	16	S_{10}	24	S_{09}	32	S_{25}

通过以上分析,可以得到求解$f(R_{i-1},K_j)$的方程组。

$$\begin{cases}f(R_i,K_j)=f_P(A_i')\\ A_i'=f_S(A)\\ A=f_E(R_i)\oplus K_j\end{cases}\quad(i=0,1,\cdots,15;j=1,2,\cdots,16)\tag{4}$$

公式(4)中,密钥K_j和S盒子的运算是求解$f(R_i,K_j)$的关键,同时也是轮函数运算的核心。子密钥K_j需要经过循环移位和换位变换后求得,变换公式为:

$$\begin{cases}m=\sum_{t=0}^{i}\sigma_t\\ C_j=2^m\cdot C_0,D_j=2^m\cdot D_0\\ K_j=f_{PC2}(G_j),G_0=f_{PC1}(Key)\end{cases}\quad(i=0,1,\cdots,15;j=i+1)\tag{5}$$

G_j与C_j的关系如下式所示,它借用集合来表示位序列的位信息(每个元素是一个二进制位信息)。

$$\boldsymbol{G}_j=\begin{bmatrix}g_j(01),g_j(02),g_j(03),g_j(04),g_j(05),g_j(06),g_j(07),g_j(08)\\ g_j(09),g_j(10),g_j(11),g_j(12),g_j(13),g_j(14),g_j(15),g_j(16)\\ g_j(17),g_j(18),g_j(19),g_j(20),g_j(21),g_j(22),g_j(23),g_j(24)\\ g_j(25),g_j(26),g_j(27),g_j(28),g_j(29),g_j(30),g_j(31),g_j(32)\\ g_j(33),g_j(34),g_j(35),g_j(36),g_j(37),g_j(38),g_j(39),g_j(40)\\ g_j(41),g_j(42),g_j(43),g_j(44),g_j(45),g_j(46),g_j(47),g_j(48)\\ g_j(49),g_j(50),g_j(51),g_j(52),g_j(53),g_j(54),g_j(55),g_j(56)\end{bmatrix}=$$

$$\begin{bmatrix}c_j(01),c_j(02),c_j(03),c_j(04),c_j(05),c_j(06),c_j(07),c_j(08)\\ c_j(09),c_j(10),c_j(11),c_j(12),c_j(13),c_j(14),c_j(15),c_j(16)\\ c_j(17),c_j(18),c_j(19),c_j(20),c_j(21),c_j(22),c_j(23),c_j(24)\\ c_j(25),c_j(26),c_j(27),c_j(28),d_j(01),d_j(02),d_j(03),d_j(04)\\ d_j(05),d_j(06),d_j(07),d_j(08),d_j(09),d_j(10),d_j(11),d_j(12)\\ d_j(13),d_j(14),d_j(15),d_j(16),d_j(17),d_j(18),d_j(19),d_j(20)\\ d_j(21),d_j(22),d_j(23),d_j(24),d_j(25),d_j(26),d_j(27),d_j(28)\end{bmatrix}$$

$$\boldsymbol{K}_j=\begin{bmatrix}k_j(01),k_j(02),k_j(03),k_j(04),k_j(05),k_j(06),k_j(07),k_j(08)\\ k_j(09),k_j(10),k_j(11),k_j(12),k_j(13),k_j(14),k_j(15),k_j(16)\\ k_j(17),k_j(18),k_j(19),k_j(20),k_j(21),k_j(22),k_j(23),k_j(24)\\ k_j(25),k_j(26),k_j(27),k_j(28),k_j(29),k_j(30),k_j(31),k_j(32)\\ k_j(33),k_j(34),k_j(35),k_j(36),k_j(37),k_j(38),k_j(39),k_j(40)\\ k_j(41),k_j(42),k_j(43),k_j(44),k_j(45),k_j(46),k_j(47),k_j(48)\end{bmatrix}=$$

$$\begin{bmatrix}g_j(14),g_j(17),g_j(11),g_j(24),g_j(01),g_j(05),g_j(03),g_j(28)\\ g_j(15),g_j(06),g_j(21),g_j(10),g_j(23),g_j(19),g_j(12),g_j(04)\\ g_j(26),g_j(08),g_j(16),g_j(07),g_j(27),g_j(20),g_j(13),g_j(02)\\ g_j(41),g_j(52),g_j(31),g_j(37),g_j(47),g_j(55),g_j(30),g_j(40)\\ g_j(51),g_j(45),g_j(33),g_j(48),g_j(44),g_j(49),g_j(39),g_j(56)\\ g_j(34),g_j(53),g_j(46),g_j(42),g_j(50),g_j(36),g_j(29),g_j(32)\end{bmatrix}$$

其中:

$$G_0=\begin{bmatrix} key_{57},key_{49},key_{41},key_{33},key_{25},key_{17},key_{09},key_{01} \\ key_{58},key_{50},key_{42},key_{34},key_{26},key_{18},key_{10},key_{02} \\ key_{59},key_{51},key_{43},key_{35},key_{27},key_{19},key_{11},key_{03} \\ key_{60},key_{52},key_{44},key_{36},key_{63},key_{55},key_{47},key_{39} \\ key_{31},key_{23},key_{15},key_{07},key_{62},key_{54},key_{46},key_{38} \\ key_{30},key_{22},key_{14},key_{06},key_{61},key_{53},key_{45},key_{37} \\ key_{29},key_{21},key_{13},key_{05},key_{28},key_{20},key_{12},key_{04} \end{bmatrix}$$

式中,Key(令 $Key=[key_{01},key_{02},\cdots,key_{64}]$)为初始密钥序列,$m$ 为循环移位的次数,密钥 Key 通过阵列变换运算 f_{PC1} 得到 G_0。令 $C_0(C_0=[c_0(1),c_0(2),\cdots,c_0(28)])$取 G_0 前 28 位,而 $D_0(D_0=[d_0(1),d_0(2),\cdots,d_0(28)])$取其后 28 位,$f_{PC1}$、$f_{PC2}$ 作用与 IP 变换一样,所不同的是 Key 的第 8、16、24、32、40、48、56、64 位不参与变换,f_{PC1}、f_{PC2} 可以通过以上的 K_j 和 G_j 变换式实现。密钥通过 16 轮次的移位和变换,得到子密钥 $K_1,K_2,\cdots,K_{16}$。密钥的整个变换过程是线性变换。

S 盒子变换是一个 6 输入 4 输出的非线性函数,它的作用是使明文实现较好的"混乱",从而保证加密过程具有较强的安全性。

3　DES 的硬件实现过程

3.1　S 盒子的实现过程

S 盒子变换可以用 $Y(y_1,y_2,y_3,y_4)=H(a,b,c,d,e,f)$表示,输出 Y 的每一个分量 $y_u(u=1,2,3,4)$由 a、b、c、d、e、f 6 个变量确定,在编程时,如果直接调用 6 变量的函数来实现变换,在用 EDA 软件(如 quartusII)进行综合时将占用较多的资源。为了合理利用设计资源,在此,首先使用卡诺图手工进行化简,得出 S 盒每一位输出的简化逻辑表达式,考虑到变换内部结构,化简时应固定 2 个变量,化简另外 4 个变量,同时还应考虑到多个输出逻辑函数之间的乘积项共用问题。下面以第一个 S 盒子变换为例写出其化简后的逻辑表达式。其中 y_1、y_2、y_3、y_4 为输出,a、b、c、d、e、f 为输入。

$$y_1=\bar{a}\bar{f}(b\bar{d}\bar{e}+c\bar{d}e+bce+\bar{b}\bar{c}d)+\bar{a}f(bc\bar{d}+\bar{b}cd+\bar{b}d\bar{e}+cd\bar{e}+\bar{b}\bar{c}\bar{d}e+b\bar{c}de)+af(b\bar{e}+c\bar{d}\bar{e}+b\bar{c}d+\bar{b}\bar{c}\bar{d}e+\bar{b}cde)+af(\bar{c}\bar{d}e+b\bar{c}\bar{e}+b\bar{c}\bar{d}+\bar{b}ce+\bar{b}cd+cde)$$

$$y_2=\bar{a}\bar{f}(\bar{b}\bar{d}e+\bar{b}\bar{c}\bar{e}+\bar{b}cd+b\bar{c}\bar{e}+b\bar{c}\bar{d}+bcde)+\bar{a}f(\bar{b}c\bar{d}+b\bar{c}\bar{d}+\bar{b}d\bar{e}+b\bar{c}e+\bar{b}\bar{c}d\bar{e}+bcde)+af(b\bar{d}+\bar{b}ce+b\bar{c}e+\bar{b}d\bar{e}+\bar{b}cde)+af(\bar{b}de+b\bar{c}e+b\bar{c}d+\bar{b}\bar{c}\bar{d}e+bd\bar{e}+bc\bar{d}\bar{e})$$

$$y_3=\bar{a}\bar{f}(\bar{b}c\bar{e}+\bar{b}de+bde+b\bar{c}d+bc\bar{d}\bar{e})+\bar{a}f(\bar{b}c\bar{e}+\bar{b}\bar{c}e+b\bar{d}e+\bar{c}d\bar{e})+af(\bar{b}\bar{c}\bar{e}+\bar{b}cd+b\bar{c}\bar{d}+bcd\bar{e}+b\bar{c}e)+af(c\bar{d}\bar{e}+\bar{b}\bar{c}\bar{d}+\bar{b}\bar{d}\bar{e}+cde+bcd+bde)$$

$$y_4=\bar{a}\bar{f}(\bar{b}\bar{c}e+c\bar{d}e+b\bar{c}e+\bar{b}cd)+\bar{a}f(\bar{b}c\bar{e}+b\bar{d}\bar{e}+bde+b\bar{c}d+\bar{b}\bar{c}\bar{d}e)+af(b\bar{c}\bar{e}+b\bar{d}e+\bar{b}de+\bar{b}\bar{c}d+b\bar{c}\bar{d}e)+af(\bar{b}\bar{c}\bar{e}+\bar{b}\bar{d}e+b\bar{c}e+bde+bc\bar{d}\bar{e})$$

经过以上化简后,输入输出可以表示为:$Y(y_1,y_2,y_3,y_4)=H'[(a,f)*(b,c,d,e)]$,当确定了 a、f 的取值后,S 盒子的功能就能通过 4 输入的选择器来实现而不需要使用 6 输入选择器,整个变换可以使用双重嵌套的选择方式来实现,外层使用 2 个变量,内层使用 4 个变量,经过综合后,单个 S 盒子的实现仅占用 24 个逻辑单元,相对于直接使用 6 个变量的 CASE 语句的实现结果而言,占用资源约减少了 50%。

3.2　子密钥的实现过程

通过对公式(5)的分析,可以得到子密钥的求解过程如图 2 所示。

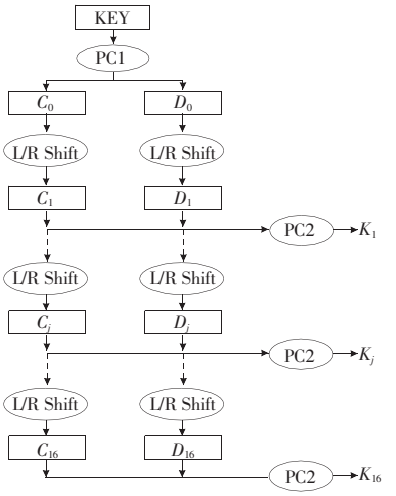

图 2　子密钥生成原理图

表 3　子密钥运算移位规律表

轮次 (i)	移位数 (σ_i)	轮次 (i)	移位数 (σ_i)	轮次 (i)	移位数 (σ_i)	轮次 (i)	移位数 (σ_i)
0	1	4	2	8	1	12	2
1	1	5	2	9	2	13	2
2	2	6	2	10	2	14	2
3	2	7	2	11	2	15	1

密钥 Key 通过 PC1 变换运算,得到 56 位序列数(C_0,D_0)。根据公式(5)及表 3 中每轮次左移运算(如果是解密,则进行右移运算)的 σ_i 值,可以得到 C_j 和 D_j,C_j 和 D_j 经过阵列变换运算 PC2 后,最终求得对应轮次的子密钥 $K_j(j=1,\cdots,16)$。

从以上求解过程可以知道,每一个子密钥的得到都须经过繁琐的移位和换位变换后才能实现,如果直接通过硬件描述语言(DHL)来实现子密钥的求解,过程将十分繁杂,同时源程序也将非常庞大,不利于编译和检查。由于求解过程都是线性变换,通过推导可以知道,子密钥中的每一位(bit)与原始密钥的每一位(bit)存在固定的对应关系,只要确定了这个关系,也就求出了所有的子密钥。

初始密钥与子密钥之间固定关系的确定利用 C 语言编程实现,主程序如下文所示,设计思路是用数组模拟密钥的位组,数组中的每一个存储单元表示密钥的一个 bit,数组中存储的数据是所需要密钥的 bit 对应得原始密钥的 bit 序号。将数组经过相应的置换和移位后,就可以得到所需密钥和原始密钥的对应关系。

```
#include <stdio.h>
void main(void)
{
void GetKeyVal(char KeySel);
int KeySel=1;
//一输入需要的密钥号------
while(KeySel>0&&KeySel<=16){
printf("please enter key code");
scanf("%d",&KeySel);
GetKeyVal(KeySel);
.....
printf("key %d is \n",KeySel);
for(i=0;i<=47;i++){
}
```

```
}
if(K[i]>=10)  printf("%3d",K[i]);
else          printf("%3d",K[i]);
if((i+1)%8==0) printf(":%d\n",i+1);
//换行显示
}
```

将上面的程序编译执行后,可以求得子密钥 K_j,例如,在输入 K_1 后得到如下的结果:

$$K_1=\begin{bmatrix}k_j(01),k_j(02),k_j(03),k_j(04),k_j(05),k_j(06),k_j(07),k_j(08)\\k_j(09),k_j(10),k_j(11),k_j(12),k_j(13),k_j(14),k_j(15),k_j(16)\\k_j(17),k_j(18),k_j(19),k_j(20),k_j(21),k_j(22),k_j(23),k_j(24)\\k_j(25),k_j(26),k_j(27),k_j(28),k_j(29),k_j(30),k_j(31),k_j(32)\\k_j(33),k_j(34),k_j(35),k_j(36),k_j(37),k_j(38),k_j(39),k_j(40)\\k_j(41),k_j(42),k_j(43),k_j(44),k_j(45),k_j(46),k_j(47),k_j(48)\end{bmatrix}_{j=1}=$$

$$\begin{bmatrix}10,51,34,60,49,17,33,57\\02,09,19,42,03,35,26,25\\44,58,59,01,36,27,18,41\\22,28,39,54,37,04,47,30\\05,53,23,29,61,21,38,63\\15,20,45,14,13,62,55,31\end{bmatrix}$$

这便是求得的子密钥 K_1 和原始密钥的对应关系,同理,可以分别求得 $K_2,\cdots,K_{16}$ 的对应关系。这种位置的确定关系在 DHL 语言里可以通过数据选择器的设计来实现,从而简化了密钥的实现过程,同时系统资源也得到了优化利用。此外,这种设计为实现密钥保护和随机更换提供了一个非常重要的手段。利用微处理器管理密钥时,可以将以上程序移植到处理器,处理器将计算的结果通过接口总线送到加密芯片的密钥输入端口,如果密钥的选取是随机的,则可以实现动态加密。

3.3　加密算法的实现过程

加密算法的实现采用资源优先的设计方案,仅用硬件实现一套密钥变换轮函数和数据加密钥运算轮函数,通过反复 16 次调用这一硬件结构来实现一次 DES 加密运算,具体原理图如图 3 所示。

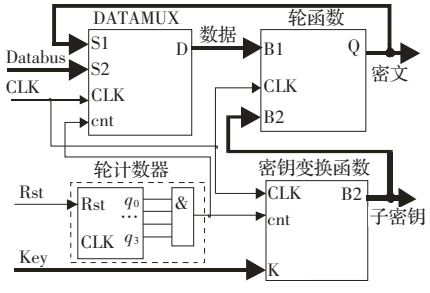

图 3　DES 算法实现原理图

外部的数据(Databus,Key)在轮计数器的控制下,在第一个 CLK 到来时将数据分别经过变换处理后送到轮函数的入口,经过第一轮变换后数据被寄存器锁存,在下一个 CLK 到来时,与相应轮的子密钥一起再次被送至轮函数的入口,这样循环 16 轮后,在轮计数器的控制下,将最终的计算结果输出。轮计数器是整个加密过程的状态控制核心,控制整个加密运算的进程和输出,DATAMUX 是多路数据选择器,选择进入轮函数运算的数据是加密中间数据还是原始输入数据。如果是解密,则计算过程与加密完全一样,对此不再赘述。

整个加密系统的综合结果如图 4 所示。从综合结果可以清楚地知道,DES 系统的实现所占用的逻辑单元数仅为 391,设计资源得到了极大的优化利用,这对于降低整个设计成本是极其重要的。

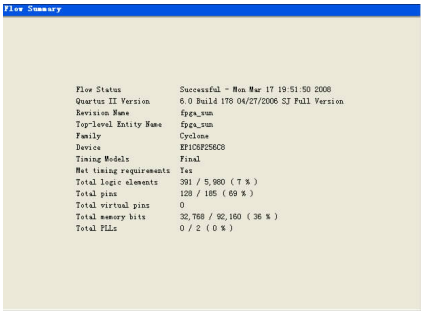

图 4　DES 加密算法综合结果图

4　仿真波形及结果

对 DES 算法的测试步骤为:

(1)测试加密功能模块:在加密工作模式下,系统的仿真结果如图 5 所示。在明文(dataIn)为 64 位的十六进制数 636F6D7075746572 时,当取密钥(key)为 64 位的十六进制数 7365637572697479 的条件下,通过 DES 加密(decrypt=0),输出得到的密文(dataOut)为 64 位的十六进制数 379CB171B20D7A23。

图 5　DES 加密功能仿真波形图

(2)测试解密功能模块:在解密工作模式下,系统的仿真结果如图 6 所示。在密文(dataIn)为 64 位的十六进制数 379CB171B20D7A23 时,当取密钥(key)为 64 位的十六进制数 7365637572697479 的条件下,通过 DES 解密(decrypt=1),输出得到的明文(dataOut)为 64 位的十六进制数 636F6D7075746572。

图 6　DES 解密功能仿真波形图

(3)对测试结果的分析验证:通过对图 5、图 6 的结果比较可以知道,在密钥为 64 位的十六进制数 7365637572697479 时,模块对 64 位的明文十六进制数 636F6D7075746572 加密

(下转 113 页)