**数字电路与数字系统实验**

**2020 秋学期**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

MD5 摘要

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

191220152 张城铨

191220029 傅小龙

目录

[前言 4](#_Toc60751657)

[实现功能 4](#_Toc60751658)

[软件部分（傅小龙） 4](#_Toc60751659)

[硬件部分（张城铨） 4](#_Toc60751660)

[第一部分 MD5模块 5](#_Toc60751661)

[1.1 概述 5](#_Toc60751662)

[1.2 发展历史 5](#_Toc60751663)

[MD2 5](#_Toc60751664)

[MD4 5](#_Toc60751665)

[MD5 6](#_Toc60751666)

[1.3 MD5算法原理 6](#_Toc60751667)

[1.3.1 填充 6](#_Toc60751668)

[1.3.2 初始化变量 7](#_Toc60751669)

[1.3.3 处理分组数据 7](#_Toc60751670)

[1.3.4 输出 10](#_Toc60751671)

[1.4 MD5应用 10](#_Toc60751672)

[1.4.1 一致性验证 10](#_Toc60751673)

[1.4.2 数字签名 11](#_Toc60751674)

[1.4.3 安全访问认证 11](#_Toc60751675)

[1.5 MD5特点 12](#_Toc60751676)

[1.6 java代码实现MD5摘要算法 12](#_Toc60751677)

[参考： 15](#_Toc60751678)

[1.7 Verilog实现MD5摘要算法 16](#_Toc60751679)

[1.7.1数字抽象 16](#_Toc60751680)

[1.7.2建立模型 17](#_Toc60751681)

[第二部分——io输入输出 23](#_Toc60751682)

[\*模块划分 23](#_Toc60751683)

[2.1键盘输入模块 24](#_Toc60751684)

[2.1.1 原理 24](#_Toc60751685)

[参考： 24](#_Toc60751686)

[2.1.2 具体代码 25](#_Toc60751687)

[2.1.3遇到的一些问题（时序） 26](#_Toc60751688)

[2.1.4 data\_2\_asc代码 27](#_Toc60751689)

[2.2状态控制模块 29](#_Toc60751690)

[2.2.1 状态转移图 29](#_Toc60751691)

[2.2.2 具体实现 29](#_Toc60751692)

[2.2.3 与MD5模块的交互 31](#_Toc60751693)

[2.3 显示模块 32](#_Toc60751694)

[2.3.1 原理 32](#_Toc60751695)

[2.3.2 字符rom & 显存ram 33](#_Toc60751696)

[2.3.3地址计算及处理 33](#_Toc60751697)

[2.3.4 遇到的问题 34](#_Toc60751698)

[三、实验启示 35](#_Toc60751699)

# 前言

## 实现功能

### 软件部分（傅小龙）

·实现对字符串（< 56位）加密处理

·verilog与java程序的对拍测试

·工作量：>30h

### 硬件部分（张城铨）

·有限状态机控制软件、硬件间的交互

·vga显示与键盘间的交互

工作量：>25h

# 第一部分 MD5模块

## 1.1 概述

　　MD5即Message-Digest Algorithm 5（信息-摘要算法5），用于确保信息传输完整一致。是计算机广泛使用的杂凑算法之一（又译[摘要算法](https://baike.baidu.com/item/%E6%91%98%E8%A6%81%E7%AE%97%E6%B3%95)、[哈希算法](https://baike.baidu.com/item/%E5%93%88%E5%B8%8C%E7%AE%97%E6%B3%95)），主流编程语言普遍已有MD5实现。将数据（如汉字）运算为另一固定长度值，是杂凑算法的基础原理，MD5的前身有MD2、[MD3](https://baike.baidu.com/item/MD3)和[MD4](https://baike.baidu.com/item/MD4)。

## 1.2 发展历史

### MD2

　　Rivest在1989年开发出MD2算法。在这个算法中，首先对信息进行数据补位，使信息的字节长度是16的倍数。然后，以一个16位的检验和追加到信息末尾，并且根据这个新产生的信息计算出散列值。后来，Rogier和Chauvaud发现如果忽略了检验将和MD2产生冲突。MD2算法[加密](https://baike.baidu.com/item/%E5%8A%A0%E5%AF%86)后结果是唯一的（即不同信息加密后的结果不同）。

### MD4

　　为了加强算法的安全性，Rivest在1990年又开发出MD4算法。MD4算法同样需要填补信息以确保信息的比特位长度减去448后能被512整除（信息比特位长度mod 512 = 448）。然后，一个以64位[二进制](https://baike.baidu.com/item/%E4%BA%8C%E8%BF%9B%E5%88%B6)表示的信息的最初长度被添加进来。信息被处理成512位damg?rd/merkle迭代结构的区块，而且每个区块要通过三个不同步骤的处理。Den boer和Bosselaers以及其他人很快的发现了攻击MD4版本中第一步和第三步的漏洞。Dobbertin向大家演示了如何利用一部普通的个人电脑在几分钟内找到MD4完整版本中的冲突（这个冲突实际上是一种漏洞，它将导致对不同的内容进行加密却可能得到相同的加密后结果）。毫无疑问，MD4就此被淘汰掉了。

　　尽管MD4算法在安全上有个这么大的漏洞，但它对在其后才被开发出来的好几种信息安全加密算法的出现却有着不可忽视的引导作用。

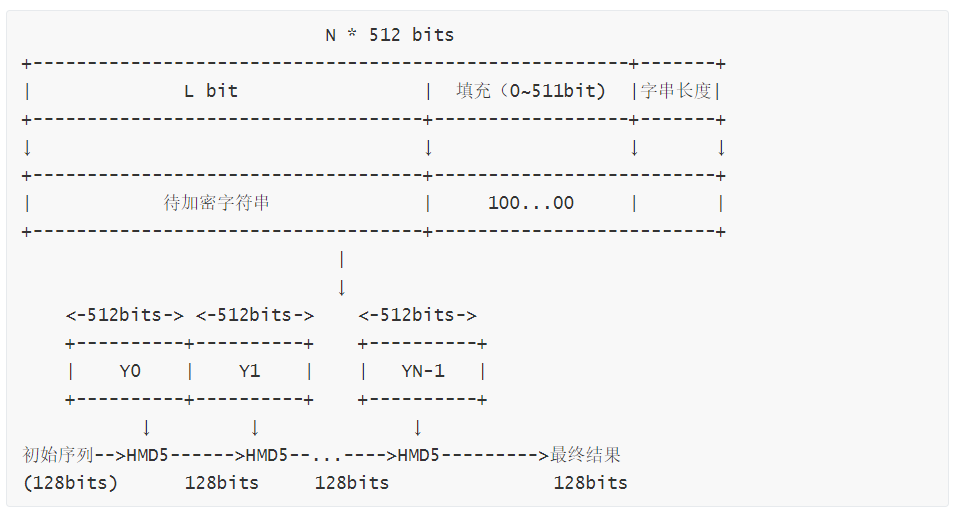
### MD5

　　1991年，Rivest开发出技术上更为趋近成熟的md5算法。它在MD4的基础上增加了"安全-带子"（safety-belts）的概念。虽然MD5比MD4复杂度大一些，但却更为安全。这个算法很明显的由四个和MD4设计有少许不同的步骤组成。在MD5算法中，信息-摘要的大小和填充的必要条件与MD4完全相同。Den boer和Bosselaers曾发现MD5算法中的假冲突（pseudo-collisions），但除此之外就没有其他被发现的加密后结果了。

## 1.3 MD5算法原理

　　对于MD5算法可以简要的概括为：MD5是以512位分组来处理输入信息，且每一分组又被划分为16个32位子分组，经过了一系列的处理后，算法的输出由四个32位分组组成，将这四个32位分组级联后将生成一个128位散列值。

总体流程图如下：



### 1.3.1 填充

　　在MD5算法中，首先需要对信息进行填充，使其位长对512求余的结果等于448，并且填充必须进行，即使其位长对512求余的结果等于448。因此，信息的位长（Bits Length）将被扩展至N \* 512 + 448，N为一个非负整数，N可以是零。

　　填充的方法如下：

　　1) 在信息的后面填充一个1和无数个0，直到满足上面的条件时才停止用0对信息的填充。

　　2) 在这个结果后面附加一个以64位二进制表示的填充前信息长度（bit），如果二进制表示的填充前信息长度超过64位，则取低64位。

　　经过这两步的处理，信息的位长为N \* 512 + 448 + 64 = (N + 1) \* 512，即长度恰好是512的整数倍。这样做的原因是为满足后面处理中对信息长度的要求。

### 1.3.2 初始化变量

　　初始的128位值为初始链接变量，这些参数用于第一轮的运算，以大端[字节序](https://baike.baidu.com/item/%E5%AD%97%E8%8A%82%E5%BA%8F)来表示，他们分别为： A=0x01234567，B=0x89ABCDEF，C=0xFEDCBA98，D=0x76543210。

（每一个变量给出的数值是高字节存于内存低地址，低字节存于内存高地址，即大端字节序。在程序中变量A、B、C、D的值分别为0x67452301，0xEFCDAB89，0x98BADCFE，0x10325476）

### 1.3.3 处理分组数据

　　每一分组的算法流程如下：

　　第一分组需要将上面四个链接变量复制到另外四个变量中：A到a，B到b，C到c，D到d。从第二分组开始的变量为上一分组的运算结果，即A = a，B = b， C = c， D = d。

主循环有四轮（MD4只有三轮），每轮循环都很相似。第一轮进行16次操作。每次操作对a、b、c和d中的其中三个作一次非线性函数运算，然后将所得结果加上第四个变量，文本的一个子分组和一个常数。再将所得结果向左环移一个不定的数，并加上a、b、c或d中之一。最后用该结果取代a、b、c或d中之一。

　　以下是每次操作中用到的四个非线性函数（每轮一个）。

F( X ,Y ,Z ) = ( X & Y ) | ( (~X) & Z )  
G( X ,Y ,Z ) = ( X & Z ) | ( Y & (~Z) )  
H( X ,Y ,Z ) = X ^ Y ^ Z  
I( X ,Y ,Z ) = Y ^ ( X | (~Z) )

　　这四个函数的说明：如果X、Y和Z的对应位是独立和均匀的，那么结果的每一位也应是独立和均匀的。

　　F是一个逐位运算的函数。即，如果X，那么Y，否则Z。函数H是逐位奇偶操作的函数。

　　假设Mj表示消息的第j个子分组（从0到15），常数ti是4294967296 \* abs(sin(i))的整数部分，i 取值从1到64，单位是弧度(rad)。(4294967296=232)

　　现定义：

　　FF(a ,b ,c ,d ,Mj ,s ,ti ) 操作为a = b + ((a + F(b,c,d) + Mj + ti) << s)

　　GG(a ,b ,c ,d ,Mj ,s ,ti ) 操作为a = b + ((a + G(b,c,d) + Mj + ti) << s)

　　HH(a ,b ,c ,d ,Mj ,s ,ti) 操作为 a = b + ((a + H(b,c,d) + Mj + ti) << s)

　　II(a ,b ,c ,d ,Mj ,s ,ti) 操作为 a = b + ((a + I(b,c,d) + Mj + ti) << s)

　　注意：这里的<<表示循环左移位，不是左移位。

　　这四轮（共64步）是：

　　第一轮

FF(a ,b ,c ,d ,M0 ,7 ,0xd76aa478)  
FF(d ,a ,b ,c ,M1 ,12 ,0xe8c7b756)  
FF(c ,d ,a ,b ,M2 ,17 ,0x242070db)  
FF(b ,c ,d ,a ,M3 ,22 ,0xc1bdceee)  
FF(a ,b ,c ,d ,M4 ,7 ,0xf57c0faf)  
FF(d ,a ,b ,c ,M5 ,12 ,0x4787c62a)  
FF(c ,d ,a ,b ,M6 ,17 ,0xa8304613)  
FF(b ,c ,d ,a ,M7 ,22 ,0xfd469501)  
FF(a ,b ,c ,d ,M8 ,7 ,0x698098d8)  
FF(d ,a ,b ,c ,M9 ,12 ,0x8b44f7af)  
FF(c ,d ,a ,b ,M10 ,17 ,0xffff5bb1)  
FF(b ,c ,d ,a ,M11 ,22 ,0x895cd7be)  
FF(a ,b ,c ,d ,M12 ,7 ,0x6b901122)  
FF(d ,a ,b ,c ,M13 ,12 ,0xfd987193)  
FF(c ,d ,a ,b ,M14 ,17 ,0xa679438e)  
FF(b ,c ,d ,a ,M15 ,22 ,0x49b40821)

　　第二轮

GG(a ,b ,c ,d ,M1 ,5 ,0xf61e2562)  
GG(d ,a ,b ,c ,M6 ,9 ,0xc040b340)  
GG(c ,d ,a ,b ,M11 ,14 ,0x265e5a51)  
GG(b ,c ,d ,a ,M0 ,20 ,0xe9b6c7aa)  
GG(a ,b ,c ,d ,M5 ,5 ,0xd62f105d)  
GG(d ,a ,b ,c ,M10 ,9 ,0x02441453)  
GG(c ,d ,a ,b ,M15 ,14 ,0xd8a1e681)  
GG(b ,c ,d ,a ,M4 ,20 ,0xe7d3fbc8)  
GG(a ,b ,c ,d ,M9 ,5 ,0x21e1cde6)  
GG(d ,a ,b ,c ,M14 ,9 ,0xc33707d6)  
GG(c ,d ,a ,b ,M3 ,14 ,0xf4d50d87)  
GG(b ,c ,d ,a ,M8 ,20 ,0x455a14ed)  
GG(a ,b ,c ,d ,M13 ,5 ,0xa9e3e905)  
GG(d ,a ,b ,c ,M2 ,9 ,0xfcefa3f8)  
GG(c ,d ,a ,b ,M7 ,14 ,0x676f02d9)  
GG(b ,c ,d ,a ,M12 ,20 ,0x8d2a4c8a)

　　第三轮

HH(a ,b ,c ,d ,M5 ,4 ,0xfffa3942)  
HH(d ,a ,b ,c ,M8 ,11 ,0x8771f681)  
HH(c ,d ,a ,b ,M11 ,16 ,0x6d9d6122)  
HH(b ,c ,d ,a ,M14 ,23 ,0xfde5380c)  
HH(a ,b ,c ,d ,M1 ,4 ,0xa4beea44)  
HH(d ,a ,b ,c ,M4 ,11 ,0x4bdecfa9)  
HH(c ,d ,a ,b ,M7 ,16 ,0xf6bb4b60)  
HH(b ,c ,d ,a ,M10 ,23 ,0xbebfbc70)  
HH(a ,b ,c ,d ,M13 ,4 ,0x289b7ec6)  
HH(d ,a ,b ,c ,M0 ,11 ,0xeaa127fa)  
HH(c ,d ,a ,b ,M3 ,16 ,0xd4ef3085)  
HH(b ,c ,d ,a ,M6 ,23 ,0x04881d05)  
HH(a ,b ,c ,d ,M9 ,4 ,0xd9d4d039)  
HH(d ,a ,b ,c ,M12 ,11 ,0xe6db99e5)  
HH(c ,d ,a ,b ,M15 ,16 ,0x1fa27cf8)  
HH(b ,c ,d ,a ,M2 ,23 ,0xc4ac5665)

　　第四轮

II(a ,b ,c ,d ,M0 ,6 ,0xf4292244)  
II(d ,a ,b ,c ,M7 ,10 ,0x432aff97)  
II(c ,d ,a ,b ,M14 ,15 ,0xab9423a7)  
II(b ,c ,d ,a ,M5 ,21 ,0xfc93a039)  
II(a ,b ,c ,d ,M12 ,6 ,0x655b59c3)  
II(d ,a ,b ,c ,M3 ,10 ,0x8f0ccc92)  
II(c ,d ,a ,b ,M10 ,15 ,0xffeff47d)  
II(b ,c ,d ,a ,M1 ,21 ,0x85845dd1)  
II(a ,b ,c ,d ,M8 ,6 ,0x6fa87e4f)  
II(d ,a ,b ,c ,M15 ,10 ,0xfe2ce6e0)  
II(c ,d ,a ,b ,M6 ,15 ,0xa3014314)  
II(b ,c ,d ,a ,M13 ,21 ,0x4e0811a1)  
II(a ,b ,c ,d ,M4 ,6 ,0xf7537e82)  
II(d ,a ,b ,c ,M11 ,10 ,0xbd3af235)  
II(c ,d ,a ,b ,M2 ,15 ,0x2ad7d2bb)  
II(b ,c ,d ,a ,M9 ,21 ,0xeb86d391)

　　所有这些完成之后，将a、b、c、d分别在原来基础上再加上A、B、C、D。

　　即a = a + A，b = b + B，c = c + C，d = d + D;

　　然后用下一分组数据继续运行以上算法。

### 1.3.4 输出

　　最后的输出是a、b、c和d的级联。

　　以下是前面所述MD5算法实现的一些运算结果:

MD5 ("") = d41d8cd98f00b204e9800998ecf8427e  
MD5 ("a") = 0cc175b9c0f1b6a831c399e269772661  
MD5 ("abc") = 900150983cd24fb0d6963f7d28e17f72  
MD5 ("message digest") = f96b697d7cb7938d525a2f31aaf161d0  
MD5 ("abcdefghijklmnopqrstuvwxyz") = c3fcd3d76192e4007dfb496cca67e13b  
MD5 ("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz") = f29939a25efabaef3b87e2cbfe641315  
MD5 ("8a683566bcc7801226b3d8b0cf35fd97") =cf2cb5c89c5e5eeebef4a76becddfcfd

## 1.4 MD5应用

### 1.4.1 一致性验证

　　MD5的典型应用是对一段信息（Message）产生信息摘要（Message-Digest），以防止被篡改。比如，在Unix下有很多软件在下载的时候都有一个文件名相同，文件扩展名为.md5的文件，在这个文件中通常只有一行文本，大致结构如：

　　MD5 (tanajiya.tar.gz) = 38b8c2c1093dd0fec383a9d9ac940515

　　这就是tanajiya.tar.gz文件的数字签名。MD5将整个文件当作一个大文本信息，通过其不可逆的字符串变换算法，产生了这个唯一的MD5信息摘要。为了让读者朋友对MD5的应用有个直观的认识，笔者以一个比方和一个实例来简要描述一下其工作过程：

　　大家都知道，地球上任何人都有自己独一无二的指纹，这常常成为司法机关鉴别罪犯身份最值得信赖的方法；与之类似，MD5就可以为任何文件（不管其大小、格式、数量）产生一个同样独一无二的“数字指纹”，如果任何人对文件做了任何改动，其MD5值也就是对应的“数字指纹”都会发生变化。

　　我们常常在某些软件下载站点的某软件信息中看到其MD5值，它的作用就在于我们可以在下载该软件后，对下载回来的文件用专门的软件（如Windows MD5 Check等）做一次MD5校验，以确保我们获得的文件与该站点提供的文件为同一文件。

　　具体来说文件的MD5值就像是这个文件的“数字指纹”。每个文件的MD5值是不同的，如果任何人对文件做了任何改动，其MD5值也就是对应的“数字指纹”就会发生变化。比如下载服务器针对一个文件预先提供一个MD5值，用户下载完该文件后，用我这个算法重新计算下载文件的MD5值，通过比较这两个值是否相同，就能判断下载的文件是否出错，或者说下载的文件是否被篡改了。

　　利用MD5算法来进行文件校验的方案被大量应用到软件下载站、论坛数据库、系统文件安全等方面。

### 1.4.2 数字签名

　　MD5的典型应用是对一段Message(字节串)产生fingerprint(指纹），以防止被“篡改”。举个例子，你将一段话写在一个叫 readme.txt文件中，并对这个readme.txt产生一个MD5的值并记录在案，然后你可以传播这个文件给别人，别人如果修改了文件中的任何内容，你对这个文件重新计算MD5时就会发现（两个MD5值不相同）。如果再有一个第三方的认证机构，用MD5还可以防止文件作者的“抵赖”，这就是所谓的数字签名应用。

### 1.4.3 安全访问认证

　　MD5还广泛用于操作系统的登陆认证上，如Unix、各类BSD系统登录密码、数字签名等诸多方面。如在Unix系统中用户的密码是以MD5（或其它类似的算法）经Hash运算后存储在文件系统中。当用户登录的时候，系统把用户输入的密码进行MD5 Hash运算，然后再去和保存在文件系统中的MD5值进行比较，进而确定输入的密码是否正确。通过这样的步骤，系统在并不知道用户密码的明码的情况下就可以确定用户登录系统的合法性。这可以避免用户的密码被具有系统管理员权限的用户知道。MD5将任意长度的“字节串”映射为一个128bit的大整数，并且是通过该128bit反推原始字符串是困难的，换句话说就是，即使你看到源程序和算法描述，也无法将一个MD5的值变换回原始的字符串，从数学原理上说，是因为原始的字符串有无穷多个，这有点象不存在反函数的数学函数。所以，要遇到了md5密码的问题，比较好的办法是：你可以用这个系统中的md5()函数重新设一个密码，如admin，把生成的一串密码的Hash值覆盖原来的Hash值就行了。

## 1.5 MD5特点

　　MD5算法具有以下特点：

　　1、压缩性：任意长度的数据，算出的MD5值长度都是固定的。

　　2、容易计算：从原数据计算出MD5值很容易。

　　3、抗修改性：对原数据进行任何改动，哪怕只修改1个字节，所得到的MD5值都有很大区别。

　　4、强抗碰撞：已知原数据和其MD5值，想找到一个具有相同MD5值的数据（即伪造数据）是非常困难的。

　　MD5的作用是让大容量信息在用数字签名软件签署私人密钥前被"压缩"成一种保密的格式（就是把一个任意长度的字节串变换成一定长的十六进制数字串）。除了MD5以外，其中比较有名的还有sha-1、RIPEMD以及Haval等。

## 1.6 java代码实现MD5摘要算法

java实现MD5摘要算法的相关文件位于Quartus项目文件夹的java/文件夹内, 用于检验Verilog语言实现的MD5算法正确性.

import java.io.\*;  
public class MD5 {  
   //四个链接变量  
       private final int A = 0x67452301;  
       private final int B = 0xefcdab89;  
       private final int C = 0x98badcfe;  
       private final int D = 0x10325476;  
       //ABCD's temp val  
       private int Atemp,Btemp,Ctemp,Dtemp;  
​  
       /\*  
       \*常量ti  
       \*公式:floor(abs(sin(i+1))×(2pow32)  
       \*/  
       private final int K[]={  
           0xd76aa478,0xe8c7b756,0x242070db,0xc1bdceee,  
           0xf57c0faf,0x4787c62a,0xa8304613,0xfd469501,0x698098d8,  
           0x8b44f7af,0xffff5bb1,0x895cd7be,0x6b901122,0xfd987193,  
           0xa679438e,0x49b40821,0xf61e2562,0xc040b340,0x265e5a51,  
           0xe9b6c7aa,0xd62f105d,0x02441453,0xd8a1e681,0xe7d3fbc8,  
           0x21e1cde6,0xc33707d6,0xf4d50d87,0x455a14ed,0xa9e3e905,  
           0xfcefa3f8,0x676f02d9,0x8d2a4c8a,0xfffa3942,0x8771f681,  
           0x6d9d6122,0xfde5380c,0xa4beea44,0x4bdecfa9,0xf6bb4b60,  
           0xbebfbc70,0x289b7ec6,0xeaa127fa,0xd4ef3085,0x04881d05,  
           0xd9d4d039,0xe6db99e5,0x1fa27cf8,0xc4ac5665,0xf4292244,  
           0x432aff97,0xab9423a7,0xfc93a039,0x655b59c3,0x8f0ccc92,  
           0xffeff47d,0x85845dd1,0x6fa87e4f,0xfe2ce6e0,0xa3014314,  
           0x4e0811a1,0xf7537e82,0xbd3af235,0x2ad7d2bb,0xeb86d391};  
       //cnt of shifting left or right   
       private final int s[]={7,12,17,22,7,12,17,22,7,12,17,22,7,  
           12,17,22,5,9,14,20,5,9,14,20,5,9,14,20,5,9,14,20,  
           4,11,16,23,4,11,16,23,4,11,16,23,4,11,16,23,6,10,  
           15,21,6,10,15,21,6,10,15,21,6,10,15,21};  
​  
​  
       private void init(){  
           Atemp = A;  
           Btemp = B;  
           Ctemp = C;  
           Dtemp = D;  
      }  
       /\*  
       \*shift right  
       \*/  
       private int shift(int a,int s){  
           return(a << s) | (a >>> (32 - s));// higher bits set to 0 when shifting right  
      }  
       /\*  
       \*main loop  
       \*/  
       private void MainLoop(int M[]){  
           int F, g;  
           int a = Atemp;  
           int b = Btemp;  
           int c = Ctemp;  
           int d = Dtemp;  
           for(int i = 0; i < 64; i ++){  
               if(i < 16){  
                   F = (b & c) | ((~b) & d);  
                   g = i;  
              }else if(i < 32){  
                   F = (d & b) | ((~d) & c);  
                   g = (5 \* i + 1) % 16;  
              }else if(i < 48){  
                   F = b ^ c ^ d;  
                   g = (3 \* i + 5) % 16;  
              }else{  
                   F = c ^ (b | (~d));  
                   g = (7 \* i) % 16;  
              }  
               int tmp = d;  
               d = c;  
               c = b;  
               b = b + shift(a + F + K[i] + M[g], s[i]);  
               a = tmp;  
          }  
           Atemp = a + Atemp;  
           Btemp = b + Btemp;  
           Ctemp = c + Ctemp;  
           Dtemp = d + Dtemp;  
      }  
       //填充函数  
       private int[] add(String str){  
           int num = ((str.length() + 8) / 64) + 1;//512bits, 64bytes as a group  
           int strByte[] = new int[num \* 16];//64/4=16 integers  
           for(int i = 0; i < num \* 16; i++){//init 0  
               strByte[i] = 0;  
          }  
           int i;  
           for(i = 0; i < str.length(); i++){  
               strByte[i >> 2] |= str.charAt(i) << ((i % 4) \* 8);//one integer store 4 bytes  
          }  
           strByte[i >> 2] |= 0x80 << ((i % 4) \* 8);//add '1' at the end  
           //add origin length  
           strByte[num\*16-2]=str.length()\*8;  
               return strByte;  
      }  
       //call func to encode  
       public String getMD5(String source){  
           init();  
           int strByte[] = add(source);  
           for(int i = 0; i < strByte.length / 16; i++){  
           int num[] = new int[16];  
           for(int j = 0; j < 16;j ++){  
               num[j] = strByte[i \* 16 + j];  
          }  
           MainLoop(num);  
          }  
           return changeHex(Atemp) + changeHex(Btemp) + changeHex(Ctemp) + changeHex(Dtemp);  
​  
      }  
       //integer to string  
       private String changeHex(int a){  
           String str = "";  
           for(int i = 0; i < 4; i++){  
               str += String.format("%2s", Integer.toHexString(((a >> i \* 8) % (1<<8)) & 0xff)).replace(' ', '0');  
          }  
           return str;  
      }  
         
     
       private static MD5 instance;  
       public static MD5 getInstance(){  
           if(instance == null){  
               instance = new MD5();  
          }  
           return instance;  
      }  
​  
       private MD5(){};  
​  
       public static void main(String[] args)throws IOException{  
BufferedReader  buf;  
buf = new BufferedReader(new InputStreamReader(System.in));  
String in = new String("");  
System.out.println("Input a string:");  
in = buf.readLine();  
while(!in.equals(">exit")){  
String str = MD5.getInstance().getMD5(in);  
System.out.println(str);  
System.out.println("Input a string:");  
in = buf.readLine();  
}  
      }  
}

## 参考：

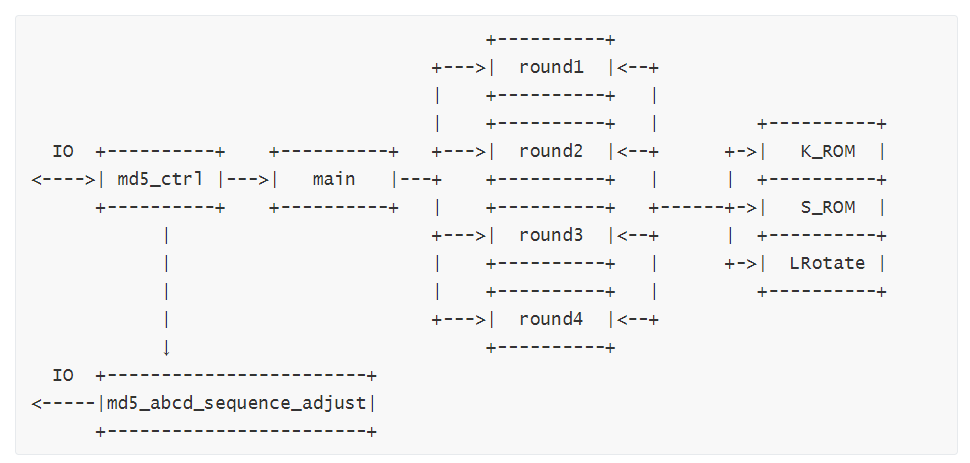
<https://en.wikipedia.org/wiki/MD5>

<http://hubingforever.blog.163.com/blog/static/171040579201210781650340/>

## 1.7 Verilog实现MD5摘要算法

## 1.7.1数字抽象

运算部分负责将用户输入的字符串进行加密. 其各模块间关系示意图如下所示：

**md5\_ctrl**: 从io模块获取用户输入的字串[511:0] indata和开始运算的信号start. 将由main模块提供的加密好的字串[127:0] MD5code传给md5\_abcd\_sequence\_adjust模块做调序处理, 并将结束信号finish发送给io模块. 在本模块内实现对待加密字串的长度填充, 将填充完毕的字串分块, 以512bits为一块依次送给main模块进行MD5运算, 并以start, blockstart作为开始处理信号传给main模块.

**main**: 接收由md5\_ctrl模块传入的待加密块[511:0] indata和开始信号start, blockstart. 处理该分组数据, 过程及原理详见1.3.3处理分组数据, 这里不再赘述. 其中4轮16次的操作分别封装于round1, round2, round3, round4模块中, main模块将待加密块和当前处理次数[3:0] in\_round传给这四个模块进行运算.

**round1**: 对输入[511:0] datain分为四块a、b、c和d并做(a + (b & c) | ((~b) & d) + Mj + K) << S运算(详见1.3.3处理分组数据中的介绍), 运算结果以[31:0] result返回给main模块. 其中, Mj是当前处理字串的第j个子分组, K, S分别存放于K\_ROM, S\_ROM中, 由操作次数[3:0]in\_round作为选择信号. 这里的<<操作为循环移位, 由模块LRotate实现, [4:0] S为左移次数.

**round2**: 对输入[511:0] datain做(a + (d & b) | ((~d) & c) + Mj + K) << S运算(详见1.3.3处理分组数据中的介绍), 运算结果以[31:0] result返回给main模块.

**round3**: 对输入[511:0] datain做(a + b ^ c ^ d + Mj + K) << S运算(详见1.3.3处理分组数据中的介绍), 运算结果以[31:0] result返回给main模块.

**round4**: 对输入[511:0] datain做(a + c ^ (b | (~d)) + Mj + K) << S运算(详见1.3.3处理分组数据中的介绍), 运算结果以[31:0] result返回给main模块.

**K\_ROM**: 存放4294967296 \* abs(sin(i))的整数部分，i 取值从1到64.

**S\_ROM**: 存放每次运算所需的左循环移位数S.

**LRotate**: 将传入的操作数[31:0] in循环左移[4:0] s位.

**md5\_abcd\_sequence\_adjust**: 对md5\_ctrl传入的计算结果[127:0] MD5\_output的四部分a、b、c和d调整为d、c、b和a, 将处理后的MD5串[1023:0] MD5\_result和长度[7:0] MD5\_result\_len传给io相关模块进行输出.

## 1.7.2建立模型

**md5\_ctrl**:

//接口设计  
module md5\_ctrl(in\_clk, indata, start, len, MD5code, finish);  
input start, in\_clk;//开始运算信号和时钟信号  
   input[63:0] len;//待加密字串的长度  
   input[511:0] indata;//输入  
output[127:0] MD5code;//运算结果  
output reg finish = 1'b0;//运算结束信号

//时序控制信号  
reg start\_cope = 1'b1, start\_block = 1'b1; //总开始信号和分块处理信号  
reg last = 1'b0, first = 1'b0, over = 1'b0;  
reg blocknop = 1'b0, times = 1'b0;  
reg additional = 1'b0, last\_finish = 1'b0;   
//若输入多于56个字符,则填充的信息会溢出至第二块, 需要对第二个512bits的信息块进行运算, 使用reset信号来重置以上状态变量并计算第二个信息块的MD5散列值  
reg [1:0]reset = 2'b0;

在start信号为1时, md5\_ctrl模块将对输入进行填充, 填充规则详见1.3.1填充节中的说明. 这里只考虑输入字符数量小于63个的情况.

if (start) begin  
last = 1'b1;  
copedata0 = indata;  
if(len[8:0] < 9'd448) begin  
/\*填充  
    \*处理后应满足bits≡448(mod512),字节就是bytes≡56（mode64)  
    \*填充方式为先加一个1,其它位补零  
    \*最后加上64位的原来长度  
\*/  
copedata0[len[8:0] + 7] = 1'b1;  
copedata0[511:448] = len[63:0];  
start\_block = 1'b1;  
blocknop = 1'b1;  
end  
else begin  
times = 1'b1;  
if(len[8:0] < 9'd504) begin  
copedata0[len[8:0] + 7] = 1'b1;  
end  
else begin  
copedata1[7] = 1'b1;  
end  
copedata1[511:448] = len[63:0];  
start\_block = 1'b1;  
blocknop = 1'b1;  
end  
end

具体的运算交给main模块实现.

**md5\_ctrl**:

//接口设计  
module main(clk, start, blockstart, indata, finish, MD5code, over);  
input clk, start, blockstart, over;  
//时钟信号, 总开始信号, 分块开始信号, 结束控制信号  
input[511:0] indata;//待加密串  
output reg finish = 1'b0;//运算结束信号  
output[127:0] MD5code;//运算结果

参照1.3.2初始化变量节, 初始链接变量[127:0] linkvar初始化为：

initial begin  
   linkvar[31:0] = 32'h67452301;  
   linkvar[63:32] = 32'hefcdab89;  
   linkvar[95:64] = 32'h98badcfe;  
   linkvar[127:96] = 32'h10325476;  
end

参照1.3.3处理分组数据节, 需要将输入的字串做FF(), GG(), HH(), II()四种运算各16次, 共64次. 以模块round1, round2, round3, round4分别实现上面这四个函数中的((a + X(b,c,d) + Mj + ti) << s)部分, 返回结果为[31:0] result1, [31:0] result2, [31:0] result3, [31:0] result4. 以[6:0] cnt记录已运算次数, integer endrounds = 7'd64为总运算次数. 运算用到的4个变量a,b,c,d值的更新实现如下：

if(cnt < endrounds) begin  
a\_next = d;  
d\_next = c;  
c\_next = b;  
if(cnt <= 7'd15)  
b\_next = b + result1;  
else if(cnt <= 7'd31)  
b\_next = b + result2;  
else if(cnt <= 7'd47)  
b\_next = b + result3;  
else if(cnt <= 7'd63)  
b\_next = b + result4;  
end  
else begin  
a\_next = a;  
b\_next = b;  
c\_next = c;  
d\_next = d;  
end

每完成一次运算cnt将加1, 当cnt >= endrounds时将finish信号置1, 标志着运算结束, 将由[127:0] MD5code返回计算结果:

assign MD5code = { a, b, c, d };

**round1**:

//接口设计  
module round1(indata, in\_round, link\_var, result);  
   input [511:0] datain; //输入字串  
   input [3:0] rounds; //当前运算次数  
   input [127:0] link\_var; //链接变量  
   output [31:0] result; //当前运算结果

round1模块返回表达式(a + (b & c) | ((~b) & d) + Mj + K) << S的值.其中的a、b、c、d分别为

链接变量中的4个int型变量:

assign A = link\_var[31:0];  
assign B = link\_var[63:32];  
assign C = link\_var[95:64];  
assign D = link\_var[127:96];

Mj的值由下式求得:

assign M = datain[{g, 5'd0} +: 32];   
// 算符'+:' 变量[起始地址 +: 数据位宽] <–等价于–> 变量[(起始地址+数据位宽-1)：起始地址]  
//变量[结束地址 -: 数据位宽] <–等价于–> 变量[结束地址：(结束地址-数据位宽+1)]

K, S的值由K\_ROM, S\_ROM模块根据rounds的值查表给出.

单次的运算结果如下求得：

assign F = ((B & C) | (~B & D)) + A + K + M;  
LRotate lr1(.in(F), .s(S), .out(result));

LRotate模块将中间结果F循环左移S位即可得到结果result.

**round2**, **round3**, **round4** 的实现与**round1**类似, 只是对中间结果F的求值方式有所不同. 具体的表达式在1.3.3 处理分组数据节和3.1 数字抽象中已有所介绍, 这里不再赘述.

**K\_ROM**:

供**round[i]**模块查询当前运算用到的K值. 为缩短运算时长, 使用case语句实现.

module K\_ROM(cnt, K);  
   input [5:0] cnt;  
   output reg [31:0] K;  
​  
   always @(\*)  
       case(cnt)  
           6'd0: K = 32'hd76aa478;  
           6'd1: K = 32'he8c7b756;  
           6'd2: K = 32'h242070db;  
           //............  
           6'd63: K = 32'heb86d391;  
       endcase  
endmodule

**S\_ROM**:

供**round[i]**模块查询当前运算用到的S值. 为缩短运算时长, 使用case语句实现.

module S\_ROM(cnt, S);  
input [5:0] cnt;  
   output reg [4:0] S;  
​  
   always @(\*)  
       case(cnt)  
           6'd0: S = 5'd7;  
           6'd1: S = 5'd12;  
           6'd2: S = 5'd17;  
           //............  
           6'd63: S = 5'd21;  
       endcase  
endmodule

**LRotate**:

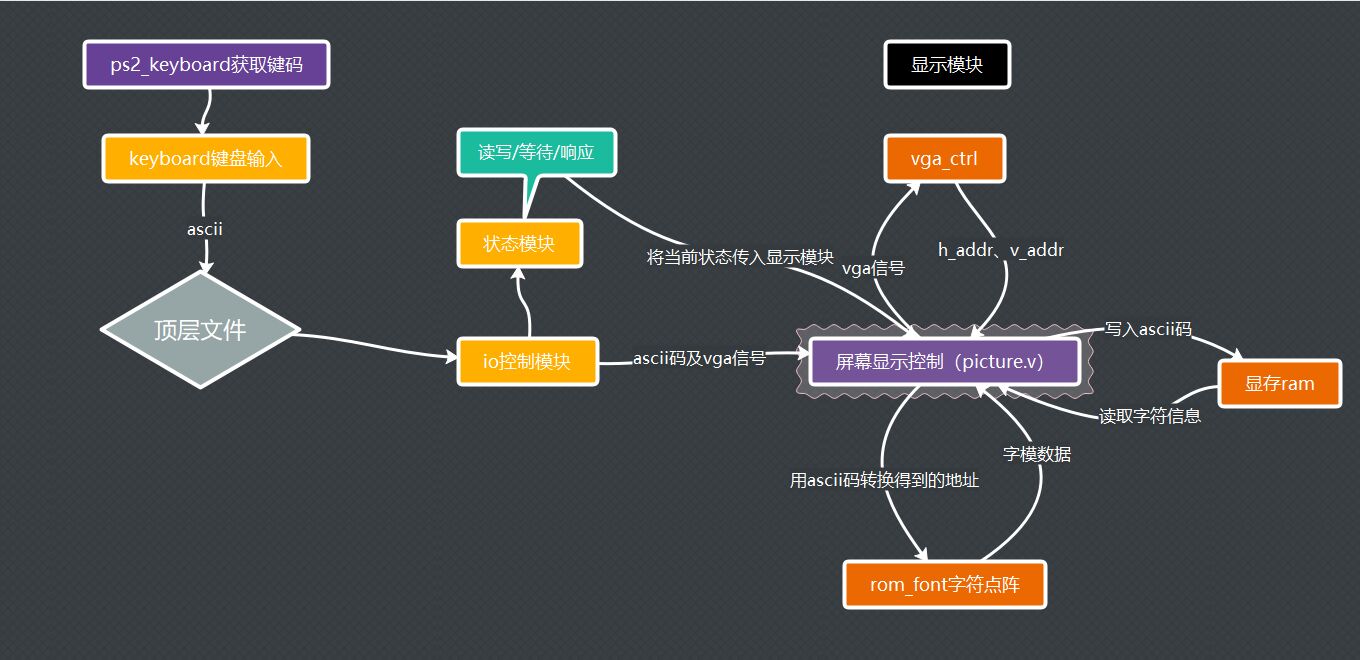
//接口设计  
module LRotate(in, s, out); //left shift bit in rotation  
   input [31:0] in; //待旋转数据  
   input [4:0] s; //左旋s位  
   output reg [31:0] out; //旋转结果

左旋位数s满足0 <= s <= 31, 表达式通式为out = {in[30 - s: 0], in[31: 31-s]};模块内对不同的s值用case语句实现.

case(s)  
5'd0: out = in;  
5'd1: out = {in[30:0], in[31:31]};  
5'd2: out = {in[29:0], in[31:30]};  
5'd3: out = {in[28:0], in[31:29]};  
//.............  
5'd31: out = {in[0], in[31:1]};  
endcase

# 第二部分——io输入输出

## \*模块划分



## 2.1键盘输入模块

此模块在之前的几次基础实验中多次复用到

在音频实验中使用的是一段式状态机实现多个按键和声

本实验中套用了二段式状态机的模板

### 2.1.1 原理

二段式状态机：用两个always块来描述状态转移和输出的状态机

其中第一个always模块采用同步时序描述状态转移；

第二个always模块采用组合逻辑判断状态转移条件，描述状态转移规律以及输出

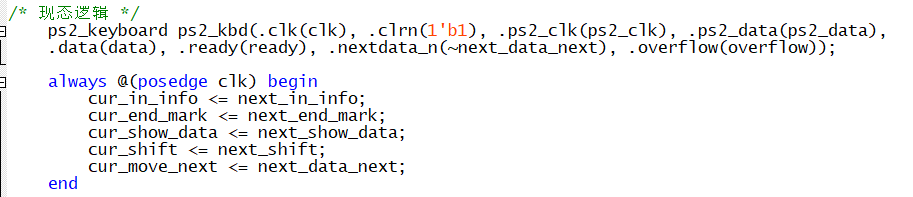
较一段式状态机而言，更利于代码的优化与条件的添加。

其他具体的内容与一段式的相近，不再详细赘述

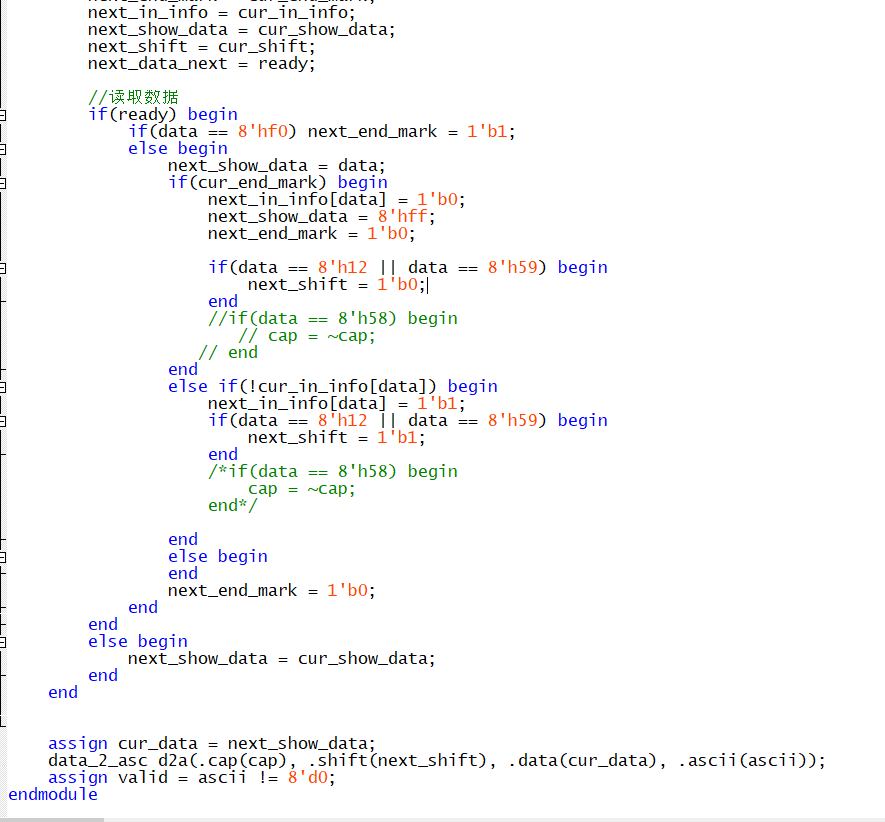
## 参考：

<http://web.stanford.edu/class/ee183/tutorial/ee183_tutorial.pdf>

### 2.1.2 具体代码



现态逻辑中调用ps2\_keyboard模块获取按键码。



次态逻辑下进行数据的转换（data\_2\_asc）与输出（ascii）

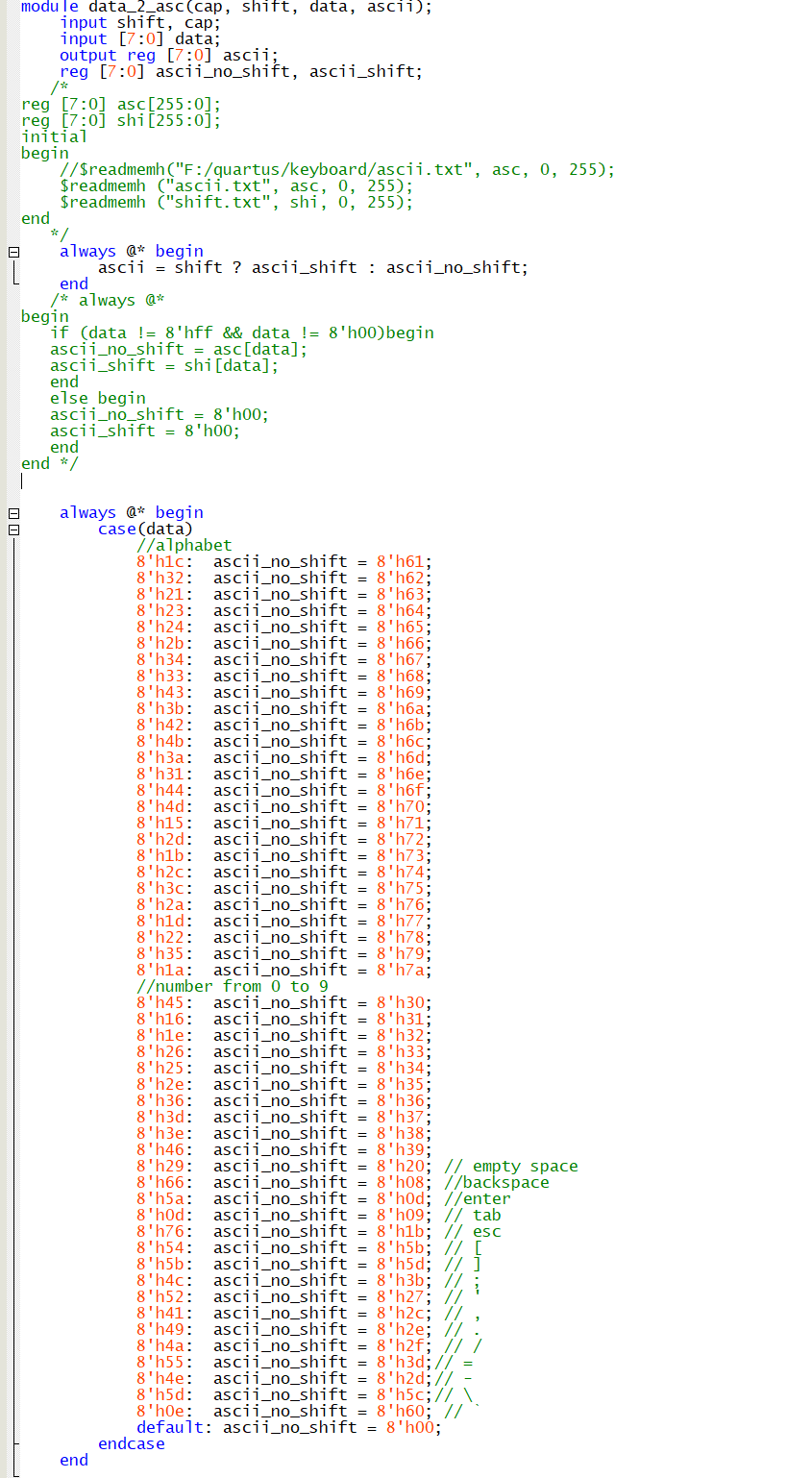
### 2.1.3遇到的一些问题（时序）

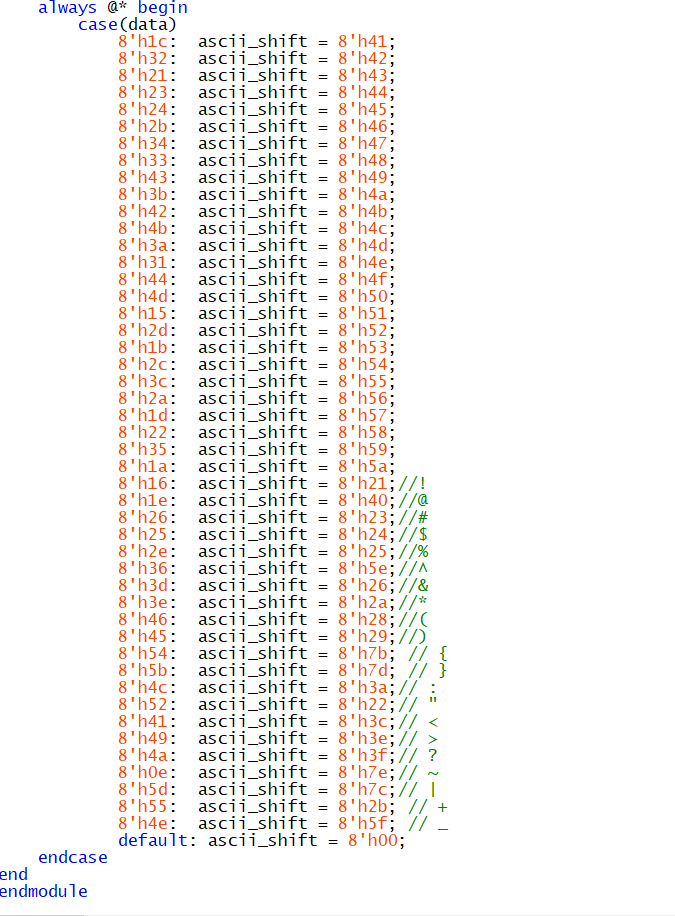
1. 在最初调试过程中，起初data\_2\_asc模块沿用了之前的readmemh读取.txt文件中的键码对应ascii数据的方式，发现与显示模块配合时，读取时间过长。

故直接改成case语句对ascii码进行赋值，分为shift按下和shift松开两种情况（放弃了cap实现大小写）。

2.接1中放弃通过判断cap大写锁定是否打开来输入大写字符的想法，是因为加入对caps状态的赋值判断后，字符输入时的显示屏上会偶尔出现一些像素点闪烁的现象（未知bug），猜测依旧是时序除了小问题，故删除了这一状态，直接改为用shift进行大小写输入。

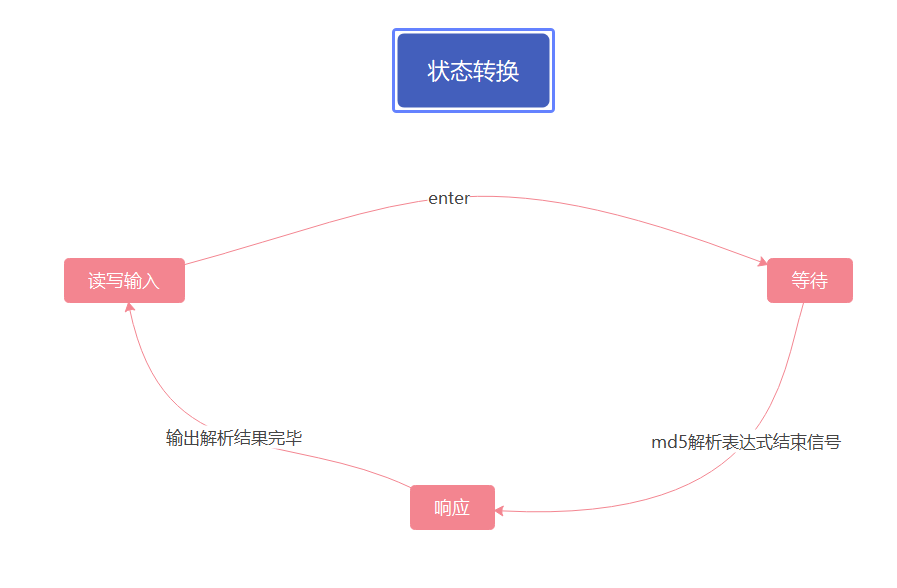
### 2.1.4 data\_2\_asc代码





## 2.2状态控制模块

### 2.2.1 状态转移图

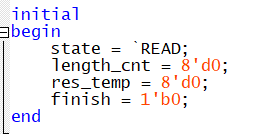


一共分为3个状态：读写输入READ，等待WAIT，响应RESPONSE

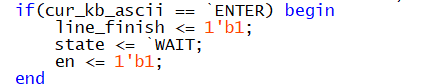
分别代表：从键盘读取输入、等待md5算法部分解析、接收md5解析结果

### 2.2.2 具体实现

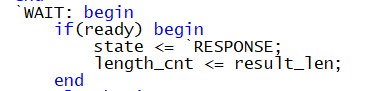
首先状态初始化为READ



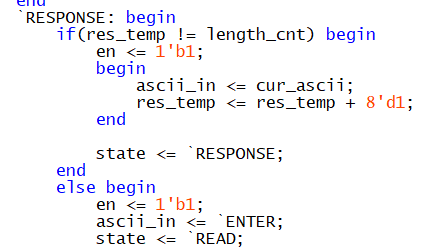
1. **READ状态下** 读入键盘输入并传给显示模块进行显存读写输入



1. 接收到回车键enter后状态切换到WAIT等待阶段，此时状态模块控制显示模块什么也不做，等待MD5内核模块进行解析操作。



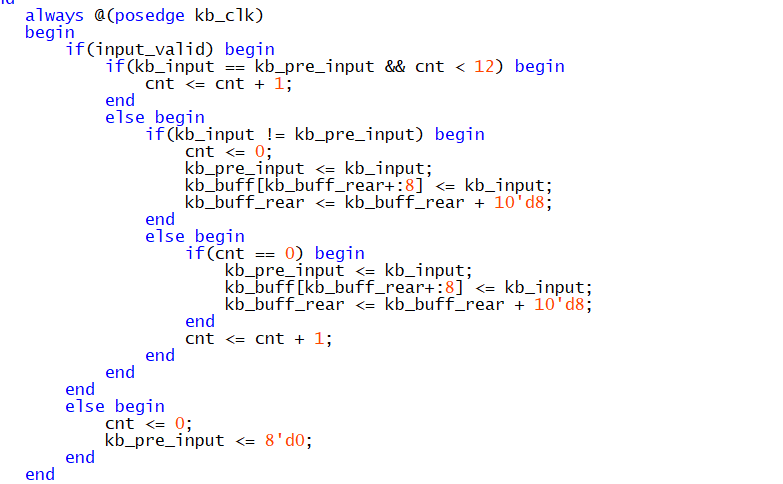
1. 解析完成后，将传回一个完成的ready信号和解析结果result，状态模块接收到该信号有效时，切换到响应阶段。
2. 相应阶段下由io控制模块逐个字符读出result块中的数据并传入显示模块进行显示。



当result块中的数据全部读完之后，响应阶段切换到读取键盘输入阶段，继续从键盘获取输入，完成一个状态转移的周期。

### 2.2.3 与MD5模块的交互

**键盘输入有效时**（接受的按键码都是常规字符且处于READ状态），需要将读入的ascii码存入一个**数据块并输出给MD5模块（MD5模块将进行对该数据块的处理）**，在接收到回车enter状态转移到WAIT阶段的同时将finish信号置为1，MD5模块接收到finish信号高有效时开始对该数据块进行处理，同时停止接收键盘输入



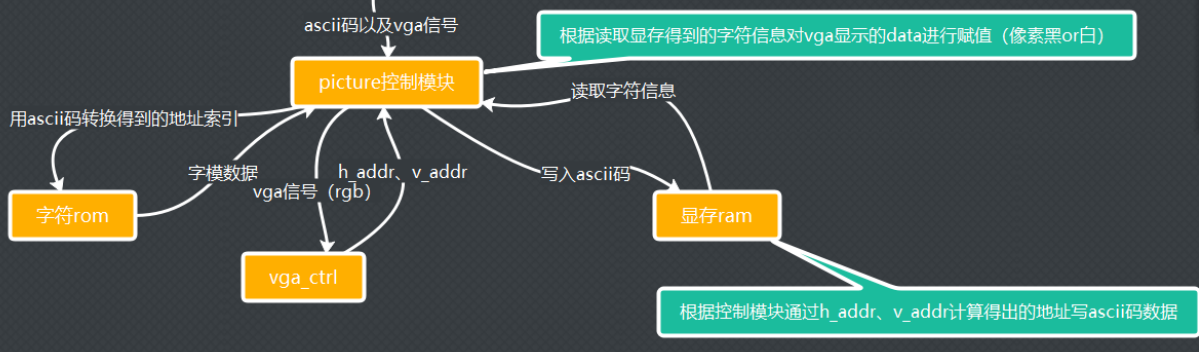
数据块用一个kb\_buff的数组模拟一个队列实现。

其中用一个整型变量cnt处理按键长按问题

与实验十一中处理长按的原理相同，当当前读取按键的码与前一个相同时，cnt++，直到cnt大于等于12时，认为是在长按该按键，进行数据块的存储与显示器的输出，在cnt小于12的过程中，始终默认只输出、存储一次该按键数据，其他文件设备不进行任何操作。

## 2.3 显示模块

### 2.3.1 原理



实验十一原理模块划分图（部分）

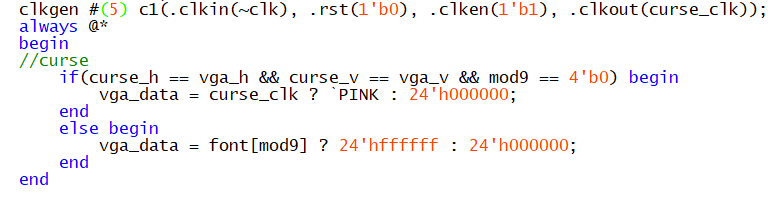
1.我们用vga\_ctrl（时钟信号是用clkgen生成的VGA\_CLK）获取 h\_addr 和 v\_addr 的数据并做一系列运算后，要去 rom 中读取字模数据，这两个过程始终是一起的，共用同一个时钟信号clk。





2.之后是写显存、读显存的过程，选择双核ram。

3.显示光标则又需要一个频率比以上慢得多的时钟信号



### 2.3.2 字符rom & 显存ram

根据vga\_font.txt生成的mif文件大小如图所示（可以通过excel和quartus 中的txt文件实现用txt文件完成mif文件的初始化），建立单核只读rom

再将字符 rom 中读取的字模信息写入显存，同时再从显存中读取数据，是一个边写边读的过程。所以要建立一个双核的读写 ram

（具体内容同实验十一）

### 2.3.3地址计算及处理

显示器 640x480 的像素映射到显存中 70x30 块，

v\_addr 要除以480/30（=16） 得到显存 ram 中对应位置的横坐标，

h\_addr 除以 9 得到显存ram中对应位置的纵坐标。除以16即右移4位实现。 **再将所得的（横坐标，纵坐标）转换为一维线性地址，即横坐标 \* 70 + 纵坐标，即为显存ram中的地址**。

但是，代码测试时发现，乘法处理过慢会导致一系列的时序问题，**故直接改成一行显示64个字符，即转换为一维线性地址时，用横坐标 \* 64 + 纵坐标，乘64**可以用移位快速实现。



如图，vga\_v是v\_addr右移四位的结果，vga\_h是h\_addr处以九后的结果。

vga\_v与vga\_h拼起来恰好就是显存地址。

根据改地址从显存 ram 中读取出对应字符的 ascii 码，再到字符 rom 中取出对应的字模信息。 注意：字模点阵 mif 文件中每个字符占了 16x9 的大小，且字符逐行存储，故读取地址应是8位 ascii 码作为高 8 位，低4位是列地址模16即低四位

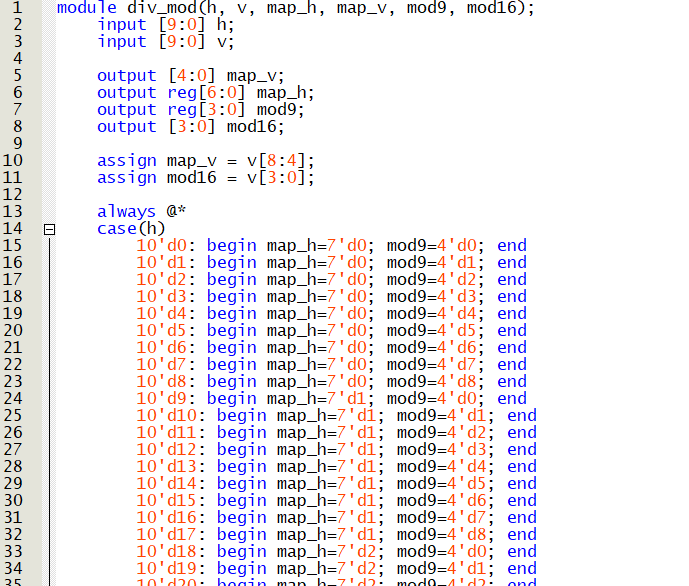


### 2.3.4 遇到的问题

上述计算过程中，只有一个除以9无法通过移位实现。

实验十一中代码量小、其他要处理的内容也少，可以直接除以9，而本实验中由于其他模块内容较多，直接除以9 处理器来不及算，显示屏上会出现奇怪的bug。

故开了一个模块用于使用case语句对除9模9地址的计算进行直接赋值。



（有近千行，就不全部复制了）

# 三、实验启示

1. 团队合作一定要多交流，之前基础实验的一些好的模板和代码框架可以互相借鉴。

2. 时间与空间往往不能兼得，往往要通过反复的测试调整找到一个能够实现较高效率的平衡点。

3. FSM 永远滴神

4. 善于利用工具是个好习惯（比如弄个java对拍）