SooPAT

基于single-port SRAM的转置矩阵的地址映射算法

申请号：201410499728.3
申请日：2014-09-25

申请(专利权)人 复旦大学
地址 200433 上海市杨浦区邯郸路220号
发明(设计)人 范益波 谢峥 程魏 曾晓洋

主分类号 H04N19/625(2014.01)I
分类号 H04N19/625(2014.01)I H04N19/122(2014.01)I H04N19/13(2014.01)I
公开(公告)号 104270643A
公开(公告)日 2015-01-07
专利代理机构 上海正旦专利代理有限公司 31200
代理人 陆飞 盛志范

www.soopat.com

注：本页蓝色字体部分可点击查询相关专利

(19)中华人民共和国国家知识产权局

(12)发明专利申请

(10)申请公布号 CN 104270643 A
(43)申请公布日 2015.01.07

(21)申请号 201410499728.3

(22)申请日 2014.09.25

(71)申请人 复旦大学
地址 200433 上海市杨浦区邯郸路 220 号

(72)发明人 范益波 谢峥 程魏 曾晓洋

(74)专利代理机构 上海正旦专利代理有限公司 31200
代理人 陆飞 盛志范

(51)Int.Cl.
H04N 19/625(2014.01)
H04N 19/122(2014.01)
H04N 19/13(2014.01)

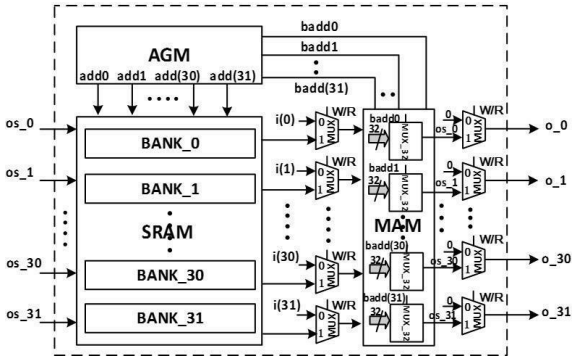
权利要求书4页 说明书5页 附图2页

(54)发明名称

基于single-port SRAM的转置矩阵的地址映射算法

(57)摘要

本发明属于高清数字视频压缩编解码技术领域,具体为一种适用于HEVC视频编码标准下2D-DCT/IDCT中基于single-port SRAM的转置矩阵的地址映射算法。地址映射算法基于对矩阵分块求转置的运算,即先对矩阵进行分块,然后分别以小尺寸块矩阵和基本元素为单元对整个矩阵和小尺寸块矩阵求转置,小尺寸块矩阵的转置可以直接通过排序实现。本发明基于变换单元(TU)进行,支持HEVC允许的4种TU大小并可实现固定的吞吐率:32pixes/cycle,适用于高吞吐率的2D-DCT/IDCT及高性能的视频编解码器中。本发明硬件结构实现可以减小40%左右的面积;相比于已有的基于single-port SRAM的转置矩阵的地址映射算法,可以在不增加硬件开销的情况下,获得更高的吞吐率,实现高清视频的实时编码。



CN 104270643 A

1. 一种基于 Single-port SRAM 的转置矩阵的地址映射算法，基于矩阵分块求转置的基本运算：设对 NxN 矩阵 A，求矩阵 A 的转置矩阵的步骤如下：

①把 NxN 矩阵划分为以 MxM 矩阵为基本单元的 (N/M)x(N/M) 块矩阵；

②对 (N/M)x(N/M) 的块矩阵求转置；

③对每个 MxM 矩阵求转置；

$$A^T = \begin{pmatrix} A_{00} & A_{01} & A_{02} & A_{03} \\ A_{10} & A_{11} & A_{12} & A_{13} \\ A_{20} & A_{21} & A_{22} & A_{23} \\ A_{30} & A_{31} & A_{32} & A_{33} \end{pmatrix}^T = \begin{pmatrix} A_{00}^T & A_{10}^T & A_{20}^T & A_{30}^T \\ A_{01}^T & A_{11}^T & A_{21}^T & A_{31}^T \\ A_{02}^T & A_{12}^T & A_{22}^T & A_{32}^T \\ A_{03}^T & A_{13}^T & A_{23}^T & A_{33}^T \end{pmatrix}$$

其中，A_{ij} 为 MxM 基本单元矩阵，i，j ＝ 0,1,2，…33；

基于 Single-port SRAM 的转置矩阵硬件结构，由如下三部分模块组成：AGM，SRAM，MAM；其中：(1)SRAM 模块，是存储单元，划分为 32 个 Bank，每个 Bank 的深度为 5，位宽为 16bit；(2)AGM 模块，根据映射算法产生地址 add(i) 和地址 badd(i)，数据的映射通过地址 add(i) 和地址 badd(i) 确定，i ＝ 0,1,2…,31；add(i) 是第 i 个 Bank 的输入信号，控制将数据写入第 i 个 Bank 的指定字节；badd(i) 通过 MAM 模块对数据进行排序：写操作时通过对输入数据排序将输入数据分块并写入指定的 Bank；读操作时通过对读出的数据排序实现对块矩阵的转置及正序输出；(3)MAM 模块，由 32 个 32：1 的选择器 MUX 组成，控制信号为 badd(i)，对数据进行排序；

所述地址映射算法，通过 add(i) 和 badd(i) 确定，具体映射如下：

① 4x4 输入矩阵：算法支持同时处理两个 4x4 输入矩阵，4x4 矩阵的转置直接通过 badd 实现，不经过 SRAM 的存储；

映射如下：

```
for i = 0,1,2,...,7
    for j = 0,1,2,3
      add(4*i+j) = k;        k 任意常数;
      badd(4*i+j) = 4*j + (i%4) + (i/4)*16;
    end
end
```

② 8x8 输入矩阵：8x8 输入矩阵每次输入连续 4 行或 4 列，共输入两次：k ＝ 0,1；

写操作时地址映射如下：

```
fork=0:1
    badd(0)=4*k;    badd(1)=1+4*k; badd(2)=2+4*k; badd(3)=3+4*k;
    badd(4)=8+4*k;  badd(5)=9+4*k; badd(6)=10+4*k; badd(7)=11+4*k;
    badd(8)=16+4*k;  badd(9)=17+4*k; badd(10)=18+4*k; badd(11)=19+4*k;
    badd(12)=24+4*k; badd(13)=25+4*k; badd(14)=26+4*k; badd(15)=27+4*k;
    badd(16)=4-4*k;  badd(17)=5-4*k; badd(18)=6-4*k; badd(19)=7-4*k;
    badd(20)=12-4*k; badd(21)=13-4*k; badd(22)=14-4*k; badd(23)=15-4*k;
    badd(24)=20-4*k; badd(25)=21-4*k; badd(26)=22-4*k; badd(27)=23-4*k;
    badd(28)=28-4*k; badd(29)=29-4*k; badd(30)=30-4*k; badd(31)=31-4*k;
  for  i=0:3
    for j=0:7
      add(8*i+j)=(k+i/2)%2;
    end
  end
end
```

读操作时地址映射如下：

```
fork=0:1
  for i=0:31
    add(i)=k;
  end
badd(0)=(16*k)%32;   badd(1)= (4+16*k)%32; badd(2)= (8+16*k)%32; badd(3)= (12+16*k)%32;
badd(4)= (16+16*k)%32;  badd(5)= (20+16*k)%32; badd(6)= (24+16*k)%32; badd(7)= (28+16*k)%32;
badd(8)= (1+16*k)%32;  badd(9)= (5+16*k)%32; badd(10)= (9+16*k)%32; badd(11)= (13+16*k)%32;
badd(12)= (17+16*k)%32; badd(13)= (21+16*k)%32; badd(14)= (25+16*k)%32; badd(15)= (29+16*k)%32;
badd(16)= (2+16*k)%32;  badd(17)= (6+16*k)%32; badd(18)= (10+16*k)%32; badd(19)= (14+16*k)%32;
badd(20)= (18+16*k)%32; badd(21)= (22+16*k)%32; badd(22)= (26+16*k)%32; badd(23)= (30+16*k)%32;
badd(24)= (3+16*k)%32; badd(25)= (7+16*k)%32; badd(26)= (11+16*k)%32; badd(27)= (15+16*k)%32;
badd(28)= (19+16*k)%32; badd(29)= (23+16*k)%32; badd(30)= (27+16*k)%32; badd(31)= (31+16*k)%32;
end
```

③ 16x16 输入矩阵：16x16 输入矩阵每次输入连续 2 行或 2 列，共输入 8 次：k = 0, 1, 2, . . . , 7；

写操作时地址映射如下：

```
for k=0:7
  for i=0:7
    badd(4*i)=2*f8((8-k)%8,i);          badd(4*i +1)= 2*f8((8-k)%8,i)+1;
    badd(4*i +2)= 2*f8((8-k)%8,i)+16;   badd(4*i +3)= 2*f8((8-k)%8,i)+17;
    add(4*i)=f8((8-k)%8,i);             add(4*i +1)= f8((8-k)%8,i);
    add(4*i +2)= f8((8-k)%8,i);         add(4*i +3)= f8((8-k)%8,i);
  end
end
```

读操作时地址映射如下：

```
for k=0:7
    for i=0:31
        add(i)=k;
    end
    for i=0:7
       badd(2*i)=4*f8(k,i);            badd(2*i +1)= 4*f8(k,i)+2;
       badd(2*i +16)= 4*f8(k,i)+1;     badd(2*i +17)= 4*f8(k,i)+3;
    end
end
```

④ 32x32 输入矩阵:32x32 输入矩阵每次输入 1 行或 1 列,共输入 32 次:k = 0,1,2,...,31 ;

写操作时地址映射如下 :

```
for k=0:31
   for i=0:31
        badd(i)=f32((32-k)%32,i);
         add(i)=f32((32-k)%32,i);
   end
end
```

读操作时地址映射如下 :

```
for k=0:31
    for i=0:31
        add(i)=k;
        badd(i)=f32(k,i);
    end
end
```

其中 :

⑴% :取余数的操作 ;M% N 表示 M 除 N 的余数 ;

⑵ / :取整操作 ;M/N 表示 M 除 N 的商的整数部分 ;

⑶ $f_N(i,j)$ 是一个 NxN 的二维矩阵 ;

$$f_N(i,j)=\begin{cases} i+j & j\le(N-1-i) \\ i+j-N & j>(N-1-i) \end{cases}$$

f_8 如下所示 :

$$f_8 = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 0 \\ 2 & 3 & 4 & 5 & 6 & 7 & 0 & 1 \\ 3 & 4 & 5 & 6 & 7 & 0 & 1 & 2 \\ 4 & 5 & 6 & 7 & 0 & 1 & 2 & 3 \\ 5 & 6 & 7 & 0 & 1 & 2 & 3 & 4 \\ 6 & 7 & 0 & 1 & 2 & 3 & 4 & 5 \\ 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \end{pmatrix}。$$

基于 single-port SRAM 的转置矩阵的地址映射算法

技术领域

[0001] 本发明属于高清数字视频压缩编解码技术领域，针对 HEVC 视频编解码标准，具体涉及一种适用于 HEVC 视频编码标准下，视频编码器和解码器中 2D-DCT/2D-IDCT 转置矩阵的地址映射算法。

背景技术

[0002] HEVC(High Efficiency Video Coding) 是由国际电信组织 (ITU) 和运动图像专家组 (MPEG) 联合成立的组织 JCTVC 提出的下一代视频编解码标准。目标是在相同的视觉效果的前提下，相比于上一代标准 H.264/AVC，压缩率提高一倍。为达到目标，HEVC 的运算复杂度相比 H.264 大大提高，因此 HEVC 编码器和解码器的硬件开销和功耗较大。降低硬件开销和功耗是 HEVC 编解码器设计的研究热点。

[0003] 绝大部分图像中直流和低频区占很大一部分，而高频区则占很小一部分。DCT 变换可将图像从空间域变换到频域，产生相关性很小的一些变换系数，有利于图像压缩。为进一步提高图像压缩律，HEVC 编码标准支持 16x16 和 32x32 的二维整形 DCT/IDCT。

[0004] 2D-DCT/IDCT 可以分解为两次一维 DCT/IDCT 运算 :①行 (列) 方向的 DCT/IDCT 变换 ;②对由①的中间结果产生的矩阵中列 (行) 方向做 DCT/IDCT 变换。计算过程可以由下式得到，大尺寸的 2D-DCT/IDCT 硬件实现需要转置矩阵模块。

[0005] $Y_N = A_N * F_N * A_N^T$

[0006] $= (A_N * (A_N * F_N)^T)^T$

[0007] 其中，F_N :NxN 的输入矩阵 ;Y_N 为 NxN 的变换后输出矩阵。A_N 为 HEVC 中 NxN 变换的矩阵。

[0008] 转置矩阵可以基于寄存器阵列实现，对于大尺寸的转置矩阵实现，基于寄存器阵列的实现会消耗大量的硬件资源和功耗。HEVC 中转置矩阵中存储结果为 16bit，4x4 的 2D-DCT 的转置矩阵仅需要 256-bit 的寄存器，而 32x32 的 2D-DCT 转置矩阵需要 16384-bit 的寄存器阵列。当存储大量数据时，SRAM 中单位比特数据的存储面积小于寄存器阵列中单位比特数据的存储面积，因此相比于寄存器阵列，SRAM 更适合于实现大尺寸的转置矩阵。

[0009] 2D-DCT/IDCT 中 , 第一次一维变换的结果按行 (列) 为单元写入转置矩阵，第二次一维变换时将存储的中间结果以列 (行) 为单元读出。寄存器阵列可以很容易的实现行数据和列数据的读写，而 single-port SRAM 只能实现行方向或列方向的数据读写，所以行 (列) 数据必须按一定的规则写入 SRAM 中，列 (行) 数据才能从 SRAM 中读出。大尺寸矩阵的转置会产生大量地址，映射方法不当会提高硬件实现的复杂度。因此映射方法必须便于硬件实现。

[0010] 为了实现实时编码，几种高吞吐率的 DCT/IDCT 架构实现已经被提出，对于各种尺寸的 TU, 吞吐率可以达到 32pixes/cycle。在这些 2D-DCT/IDCT 的设计中，转置矩阵都是基于寄存器阵列 , 硬件开销较大，数据在寄存器中的移动，导致功耗较大。一种基于 single-port SRAM 的转置矩阵的地址映射算法已经提出，但该算法仅适用于低吞吐率的

2D-DCT/IDCT 架构。本发明提出的映射算法适用于 HEVC 支持的所有基于变换单元 (TU) 大小,对于不同的 TU 尺寸可以实现固定的吞吐率 :32pixes/cycle。

发明内容

[0011] 本发明的目的在于提供一种适用于HEVC标准下2D-DCT/IDCT中基于Single-port SRAM的转置矩阵的地址映射算法。

[0012] 本发明提出的基于Single-port SRAM的转置矩阵的地址映射算法,是基于矩阵分块求转置的基本运算。设对 NxN 矩阵 A,求矩阵 A 的转置矩阵的步骤如下 :① NxN 矩阵划分为以MxM矩阵为基本单元的 (N/M)x(N/M) 块矩阵 ;②对 (N/M)x(N/M) 的块矩阵求转置 ;③对每个 MxM 矩阵求转置 ;即 :

[0013] $$A^T = \begin{pmatrix} A_{00} & A_{01} & A_{02} & A_{03} \\ A_{10} & A_{11} & A_{12} & A_{13} \\ A_{20} & A_{21} & A_{22} & A_{23} \\ A_{30} & A_{31} & A_{32} & A_{33} \end{pmatrix}^T = \begin{pmatrix} A_{00}^T & A_{10}^T & A_{20}^T & A_{30}^T \\ A_{01}^T & A_{11}^T & A_{21}^T & A_{31}^T \\ A_{02}^T & A_{12}^T & A_{22}^T & A_{32}^T \\ A_{03}^T & A_{13}^T & A_{23}^T & A_{33}^T \end{pmatrix}$$

[0014] 其中,A_{ij} 为 MxM 基本单元矩阵,i,j = 0,1,2, …33。

[0015] 基于 Single-port SRAM 的转置矩阵硬件结构,由如下三部分模块组成 :AGM,SRAM,MAM ;其中 :

[0016] (1)SRAM :该模块是存储单元,划分为 32 个 Bank, 每个 Bank 的深度为 5,位宽为 16bit ;(2)AGM :该模块根据映射算法产生地址 add(i) 和 badd(i),数据的映射通过 add(i) 和 badd(i) 确定 (i = 0,1,2…,31)。add(i) 是第 i 个 Bank 的输入信号,控制将数据写入第 i 个 Bank 的指定字节 ;badd(i) 通过 MAM 模块对数据进行排序 :写操作时通过对输入数据排序将输入数据分块并写入指定的 Bank ;读操作时通过对读出的数据排序实现对块矩阵的转置及正序输出 ;(3)MAM 模块由 32 个 32 :1 的选择器 MUX 组成, 控制信号为 badd(i), 对数据进行排序。

[0017] 本发明的地址映射算法,可以通过 add(i) 和 badd(i) 确定,具体映射如下 :

[0018] ① 4x4 输入矩阵 :算法支持同时处理两个 4x4 输入矩阵,4x4 矩阵的转置可以直接通过 badd 实现,不用经过 SRAM 的存储 ;

[0019] 映射如下 :

[0020]

```
for i = 0,1,2,...,7
    for j = 0,1,2,3
      add(4*i+j) = k;        k 任意常数;
      badd(4*i+j) = 4*j+(i%4)+(i/4)*16;
    end
end
```

[0021] ② 8x8 输入矩阵 :8x8 输入矩阵每次输入连续 4 行 (列),共输入两次 (k = 0,1)。

[0022] 写操作时地址映射如下 :

[0023]

```
for k=0:1
    badd(0)=4*k;     badd(1)=1+4*k; badd(2)=2+4*k; badd(3)=3+4*k;
    badd(4)=8+4*k;   badd(5)=9+4*k; badd(6)=10+4*k; badd(7)=11+4*k;
    badd(8)=16+4*k;  badd(9)=17+4*k; badd(10)=18+4*k; badd(11)=19+4*k;
    badd(12)=24+4*k; badd(13)=25+4*k; badd(14)=26+4*k; badd(15)=27+4*k;
    badd(16)=4-4*k;  badd(17)=5-4*k; badd(18)=6-4*k; badd(19)=7-4*k;
    badd(20)=12-4*k; badd(21)=13-4*k; badd(22)=14-4*k; badd(23)=15-4*k;
    badd(24)=20-4*k; badd(25)=21-4*k; badd(26)=22-4*k; badd(27)=23-4*k;
    badd(28)=28-4*k; badd(29)=29-4*k; badd(30)=30-4*k; badd(31)=31-4*k;
   for   i=0:3
      for j=0:7
        add(8*i+j)=(k+i/2)%2;
      end
   end
end
```

[0024] 读操作时地址映射如下：

[0025]

```
 for k=0:1
    for i=0:31
      add(i)=k;
    end
  badd(0)=(16*k)%32; badd(1)= (4+16*k)%32; badd(2)= (8+16*k)%32; badd(3)= (12+16*k)%32;
  badd(4)= (16+16*k)%32; badd(5)= (20+16*k)%32; badd(6)= (24+16*k)%32; badd(7)= (28+16*k)%32;
  badd(8)= (1+16*k)%32; badd(9)= (5+16*k)%32; badd(10)= (9+16*k)%32; badd(11)= (13+16*k)%32;
  badd(12)= (17+16*k)%32; badd(13)= (21+16*k)%32; badd(14)= (25+16*k)%32; badd(15)= (29+16*k)%32;
  badd(16)= (2+16*k)%32; badd(17)= (6+16*k)%32; badd(18)= (10+16*k)%32; badd(19)= (14+16*k)%32;
  badd(20)= (18+16*k)%32; badd(21)= (22+16*k)%32; badd(22)= (26+16*k)%32; badd(23)= (30+16*k)%32;
  badd(24)= (3+16*k)%32; badd(25)= (7+16*k)%32; badd(26)= (11+16*k)%32; badd(27)= (15+16*k)%32;
  badd(28)= (19+16*k)%32; badd(29)= (23+16*k)%32; badd(30)= (27+16*k)%32; badd(31)= (31+16*k)%32;
 end
```

[0026] ③ 16x16 输入矩阵：16x16 输入矩阵每次输入连续 2 行（列），共输入 8 次（k = 0, 1, 2, . . . , 7）。

[0027] 写操作时地址映射如下：

[0028]

```
for k=0:7
  for i=0:7
      badd(4*i)=2*f8((8-k)%8,i);          badd(4*i +1)= 2*f8((8-k)%8,i)+1;
      badd(4*i +2)= 2*f8((8-k)%8,i)+16;   badd(4*i +3)= 2*f8((8-k)%8,i)+17;
      add(4*i)=f8((8-k)%8,i);             add(4*i +1)= f8((8-k)%8,i);
      add(4*i +2)= f8((8-k)%8,i);         add(4*i +3)= f8((8-k)%8,i);
  end
end
```

[0029] 读操作时地址映射如下：

[0030]

```
for k=0:7
    for i=0:31
        add(i)=k;
    end
    for i=0:7
        badd(2*i)=4*f8(k,i);          badd(2*i +1)= 4*f8(k,i)+2;
        badd(2*i +16)= 4*f8(k,i)+1;   badd(2*i +17)= 4*f8(k,i)+3;
    end
end
```

[0031] ④ 32x32 输入矩阵：32x32 输入矩阵每次输入 1 行（列），共输入 32 次（k = 0, 1, 2, . . . , 31）。

[0032] 写操作时地址映射如下：

[0033]

```
for k=0:31
    for i=0:31
        badd(i)=f32((32-k)%32,i);
        add(i)=f32((32-k)%32,i);
    end
end
```

[0034] 读操作时地址映射如下：

[0035]

```
for k=0:31
    for i=0:31
        add(i)=k;
        badd(i)=f32(k,i);
    end
end
```

[0036] 其中：

[0037] (1)% ：取余数的操作；M% N 表示 M 除 N 的余数；

[0038] (2) / ：取整操作；M/N 表示 M 除 N 的商的整数部分；

[0039] (3) $f_N(i, j)$ 是一个 NxN 的二维矩阵。

[0040] $$f_N(i,j)=\begin{cases} i+j & j \le (N-1-i) \\ i+j-N & j > (N-1-i) \end{cases}$$

[0041] f_8 如下所示：

[0042] $$f_8 = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 0 \\ 2 & 3 & 4 & 5 & 6 & 7 & 0 & 1 \\ 3 & 4 & 5 & 6 & 7 & 0 & 1 & 2 \\ 4 & 5 & 6 & 7 & 0 & 1 & 2 & 3 \\ 5 & 6 & 7 & 0 & 1 & 2 & 3 & 4 \\ 6 & 7 & 0 & 1 & 2 & 3 & 4 & 5 \\ 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \end{pmatrix}$$

[0043] 本发明以适用于HEVC支持的所有TU大小:4x4,8x8,16x16,32x32;对于不同的TU尺寸可以实现固定的吞吐率:32pixes/cycle,适用于高吞吐率的2D-DCT/IDCT及高性能的视频编解码器中。相比基于寄存器阵列实现的转置矩阵,该硬件结构可以实现40%左右的面积减小;相比于已有的基于single-port SRAM的转置矩阵的地址映射算法,该算法可以在不增加硬件开销的情况下,获得更高的吞吐率,从而高效的实现高清视频的实时编码。

附图说明

[0044] 图1:8x8矩阵地址映射。

[0045] 图2:SRAM划分图。

[0046] 图3:基于SRAM的转置矩阵硬件结构图。

具体实施方式

[0047] 下面通过实例并结合附图,以8x8输入矩阵为例进一步具体描述本发明方法。

[0048] 对于8x8的输入矩阵,每次输入4行,分两次输入,对应的地址映射add(i)和badd(i)如图一所示,写操作时W/R＝0,输入数据经过MAM模块,MAM根据badd(i)对输入数据进行排序以指定输入数据写入的Bank,然后数据经过add(i)写入第i个Bank的指定字节;输入数据存储完成后,开始读操作W/R＝1,根据add(i)从SRAM中读出列数据,然后根据badd(i)对读出的列数据进行排序输出,每次输出四列数据,分两次输出。

[0049] 本发明采用一种适用于HEVC视频编码标准下2D-DCT/IDCT中基于single-port SRAM的转置矩阵的地址映射算法及硬件实现。,可以有效的减少芯片的存储单元的面积,减小了硬件的开销。相比基于寄存器阵列实现的转置矩阵,该硬件结构可以实现40%左右的面积减小。相比于已有的基于single-port SRAM的转置矩阵的地址映射算法,该算法可以在不增加硬件开销的情况下,获得更高的吞吐率,从而高效的实现高清视频的实时编码。

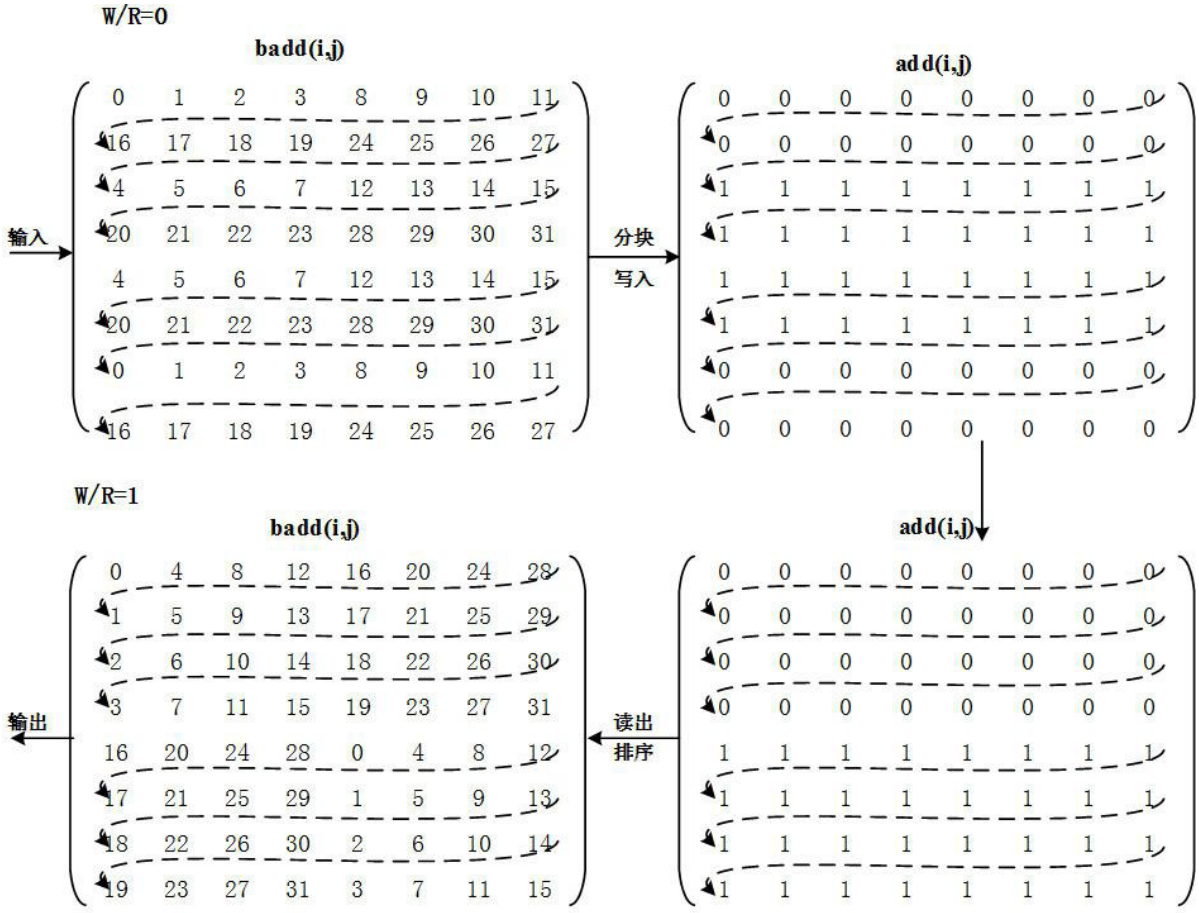

W/R=0
badd(i,j)
add(i,j)
输入
分块
写入
W/R=1
badd(i,j)
add(i,j)
输出
读出
排序

图 1

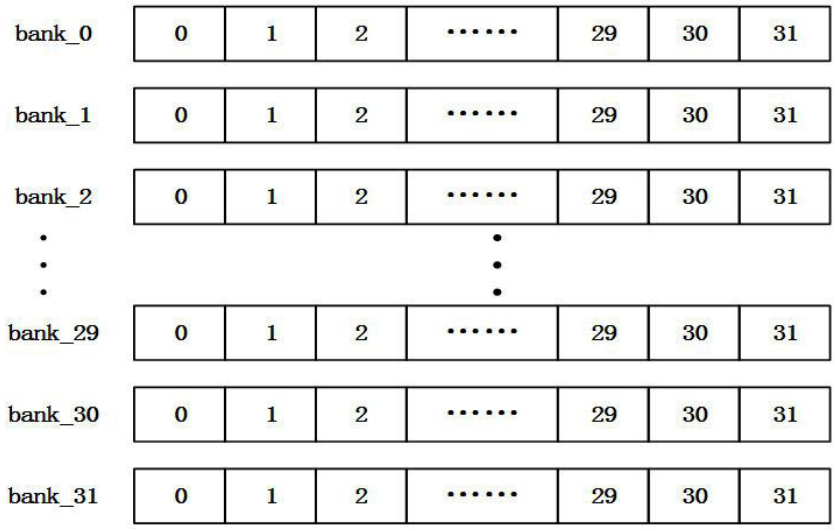

bank_0
bank_1
bank_2
bank_29
bank_30
bank_31
0
1
2
29
30
31

图 2

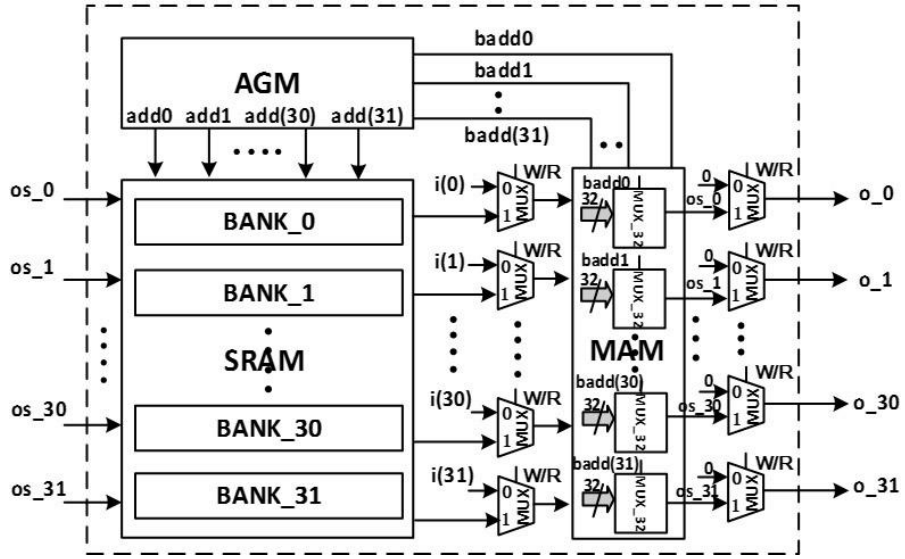

AGM
add0 add1 add(30) add(31)
badd0
badd1
badd(31)
os_0
os_1
os_30
os_31
BANK_0
BANK_1
SRAM
BANK_30
BANK_31
i(0)
i(1)
i(30)
i(31)
W/R
MUX
MAM
MUX_32
o_0
o_1
o_30
o_31

图 3