|  |
| --- |
| iVip |
| BNN处理器 |
| 模块说明手册 |

|  |
| --- |
| 林宁超  [日期] |

模块说明

目录

[一.BNN Core 2](#_Toc67674283)

[1.指令综述： 2](#_Toc67674284)

[2.各模块介绍 4](#_Toc67674285)

[BNN\_Core 4](#_Toc67674286)

[BPU\_Group 6](#_Toc67674287)

[BPU 8](#_Toc67674288)

[BPUE 10](#_Toc67674289)

[二、指令译码模块 11](#_Toc67674290)

[模块说明 11](#_Toc67674291)

[译码原理 12](#_Toc67674292)

[汇编器使用方法 13](#_Toc67674293)

[三、SRAM模块 14](#_Toc67674294)

[四、向BNN模块写入数据 15](#_Toc67674295)

[读写SRAM 16](#_Toc67674296)

一.BNN Core

## 1.指令综述：

|  |  |  |
| --- | --- | --- |
| 信号 | **具体指令及功能** 按 [19:0]格式 | 各个信号的功能 |
| Instruction 20bits  Bnn\_Core指令 | **给BPUG的指令** | 控制BPUG里计算和数据读取 |
| **PSUM进行一次累加：**  [9]=1，[4:1]=某个数, others=0  (某个数为选通的BPUG列)  Next posedge inst[9]=0 | [9]bpug\_psum\_add:  Psum进行一次累加  [4:1]bpug\_sel：选出需要做累加的BPUG单元 |
| **将二值化计算结果写入某个寄存器：**  [10]=1  Next posedge [10] = 0 | [10]：Cal\_bin\_wr 1bit  将二值化计算结果写入reg\_bins中  [12]：是否pooling  [6][13]：写入pooling reg的位置 |
| **写入BIAS：** | [11]：Bias\_wr 1bit：BIAS REG写使能 |
| **写入img或wgt数据**  [8] or [7] =1, [2:1]=某个数,others = 0  Next posedge all = 0 | [8:7]en img和wgt reg的使能信号  [2:1]：bpug\_sel Bpug列选信号，他将控制BPUG的输入信号chip\_sel |
| **PSUM\_REG赋为BIAS**  [0]=1 Next posedge [0]=0 | [0]psum\_rst它会与**BPUG中PSUM置0**同时发生。有紫色指令即可。 |
| **输出有效**  [14]=1, [6]=1 or 0, others=0 | [14]: store, 0时输出高阻  [6]: 0时输出result\_bins[3:0]，1时输出[7:4] |
| BPUG指令  13bits | **给BPU的计算指令** | Inst\_to\_bpu给BPU的指令 |
| **Wgt\_reg使能** | [19:17]选择写入哪个WGT\_REG |
| **从8列img\_reg中选7列**  [6]=0 or 1 | [6] Data\_sel选img\_reg前七行的某七列 |
| **Img\_reg上移一位**  [15]=1, others=0 | [15]: img\_reg\_up 指示img\_reg整体向上移位1bit |
| **数据写入哪部分img\_reg**  [16]=1 or 0 | [16]: img\_reg\_sel 0时，写入[7:0]的img\_reg,，1时写入[15:8]的img\_reg |
| BPU指令  5bits | **PSUM寄存器置零** | [0]：Psum\_rst置高时，PSUM置零 |
| **算某个BPUE同或结果的popcnt**  [3:1] ≤3’b111 | [3:1]：Lut\_sel 3bits  选通某个BPUE结果做LUT地址 |
| **Popcnt值进行累加** | [5]：Psum\_add 1bit  LUT结果进入PSUM累加 |

**关于几个复用的信号**

[0]：控制BNN\_Core中PSUM赋值为BIAS，控制BPU中PSUM的复位。计算过程从BPU复位开始，BPU计算完成后BNN Core中开始计算，且BNN Core计算期间无需再给psum赋值，因此他们两个功能可以复用一个信号。

[4:1]：控制数据写入的列选信号，控制计算时BPUE选通，控制计算时BPUG选通。这三个功能都不是同时进行的，因此[4:1]理论上可以复用，经仿真验证也没有问题。

[6]：控制BPUG中img\_reg的选通，BNN Core中输出4×8bits的选通。这两个操作不会同时进行，因此这个信号可以复用。

[8]和[15]：[8]是图像写入，[15]是图像寄存器移位。[8]置高，[15]置低时读入图像，[15]置高，[8]置低时图像寄存器移位。当他们同时置高时，读入本轮计算的配置参数。

## 2.各模块介绍

### BNN\_Core



功能：通过MUX逐个选出BPUG的计算结果并进行求和，求和结果存于PSUM中。求和结果可选做2x2 Pooling。

其中，有一个chip\_sel片选信号，用作写入数据时选通一列四个BPUG。

输入信号：

|  |  |  |
| --- | --- | --- |
| 信号 | 具体 按inst[13:0]格式 | 功能 |
| Instruction 20bits  Bnn\_Core指令 | **给BPUG的指令** | [16:15][8:5][3:0]Inst\_to\_bpug给BPUG的指令, Core的MUX信号复用前八位中[4:1], Core中PSUM\_RST也复用[0] |
| **PSUM进行一次累加：**  [9]=1，[4:1]=某个数, others=0  (某个数为选通的BPUG列)  Next posedge inst[9]=0 | [9]bpug\_psum\_add:  Psum进行一次累加  [4:1]bpug\_sel：选出需要做累加的BPUG单元 |
| **将二值化计算结果写入某个寄存器：**  [10]=1  Next posedge [10] = 0 | [10]：Cal\_bin\_wr 1bit  将二值化计算结果写入reg\_bins中  [12]：是否pooling  [6][13]：写入pooling reg的位置 |
| **写入BIAS：**  [11] = 1, others = 0 | [11]：Bias\_wr 1bit  BIAS REG写使能 |
| **写入img或wgt数据**  [8] or [7] =1, [2:1]=某个数,others = 0  Next posedge all = 0 | [8:7]en img和wgt reg的使能信号  [2:1]：bpug\_sel Bpug列选信号，他将控制BPUG的输入信号chip\_sel |
| **PSUM\_REG赋为BIAS**  [0]=1 Next posedge [0]=0 | [0]psum\_rst它会与**BPUG中PSUM置0**同时发生。有紫色指令即可。 |
| **输出有效**  [14]=1, [6]=1 or 0, others=0 | [14]: store, 0时输出高阻  [6]: 0时输出result\_bins[3:0]，1时输出[7:4] |
| clk | - | 时钟信号 |
| rst | - | 复位信号，寄存器全部置零 |
| Data\_in 32 bits | 32bits数据总线 | EN信号控制写入 |

输出信号

|  |  |  |
| --- | --- | --- |
| 信号 | 具体 | 功能 |
| result\_bins 8×8bits | 每次写8bits的移位寄存器 | 每个BPU的计算结果 |

寄存器：

|  |  |
| --- | --- |
| 名称 | 作用 |
| Pooling\_reg 3×8bits | 存放要做pooling的数据 |
| Bias 8×7bits | 存放bias数值 |
| Cal\_intern 8×11bits | 存放全精度计算结果，它就是框图中的PSUM |
| Reg\_bins 8×8bits | 存放计算结果的移位寄存器  每次输出4列 |
| Bpug\_enable 16bits | 控制BPUG是否工作 |
| Bpu\_enable 3bits | 控制BPU的工作个数 |
| Height 3bits | 控制kernel边长 |

### BPU\_Group



功能：将图像REG中，选出一块7x7连到BPU进行后续计算，结果通过OUT输出。

输入信号：

|  |  |  |
| --- | --- | --- |
| 信号 | 具体 按inst[13:0]格式 | 功能 |
| Instruction 13bits  指令 | Depends | Inst\_to\_bpu给BPU的指令 |
| **从8列img\_reg中选7列**  [6]=0 or 1 | [6] Data\_sel选img\_reg前七行的某七列 |
| **Img\_reg上移一位**  [15]=1, others=0 | [15]: img\_reg\_up 指示img\_reg整体向上移位1bit |
| **数据写入哪部分img\_reg**  [16]=1 or 0 | [16]: img\_reg\_sel 0时,写入[7:0]的img\_reg, 1时写入[15:8]的img\_reg |
| **数据写入哪个wgt\_reg** | [19:17] |
| clk | - | 时钟信号 |
| rst | - | 复位信号，寄存器全部置零 |
| Sel 1bits | 控制数据写入的 |  |
| Data\_in 8bits | 连到IMG和WGT REG | EN信号控制写入哪个REG |
| Chip\_sel 1bits | 片选信号 | 控制是否读入数据 |
| Height 3bits |  |  |
| Bpu\_enable 3bits | 控制工作的BPU的数量 |  |

输出信号

|  |  |  |
| --- | --- | --- |
| 信号 | 具体 | 功能 |
| Bpu\_out 6×7bits | 6×7bits 有符号数 | 每个BPU的计算结果 |

寄存器

|  |  |
| --- | --- |
| 名称 | 作用 |
| Img\_reg 16×8bits | 存放图像数值的寄存器 |

**img\_reg的组织方式：**

采取这种组织方式，是因为SRAM每次最小读8bits，因此卷积窗向下平移的操作，无法通过读下一bit的数据来实现。

比如读完[7:0]，下一波该读[2:9]，但SRAM无法错出2bit读取。

因此我们用16×8bits的移位寄存器，并且有向上移位的操作（高位在上）。

### BPU



功能：BPUE对图像和权重的一列数据进行同或计算，结果逐列通过MUX进入查找表进行popcount计算。 BPUE进行1×7bits的同或计算。

输入信号：

|  |  |  |
| --- | --- | --- |
| 信号 | 具体 按inst[13:0]格式 | 功能 |
| Instruction 5bits  指令 | **PSUM寄存器置零**  [0]=1, others = 0 | [0]：Psum\_rst 1bit  置高时，PSUM置零 |
| **算某个BPUE同或结果的popcnt**  [3:1]={0,1,2,3,4,5,6,7} | [3:1]：Lut\_sel 3bits  选通某个BPUE结果进入LUT |
| **Popcnt值进行累加**  [5]=1,[3:1]不变，其余=0 | [5]：Psum\_add 1bit  LUT结果进入PSUM累加 |
| clk | - | 时钟信号 |
| rst | - | 复位信号，寄存器全部置零 |
| Img 7×7bits |  | 计算的图像数据 |
| Wgt\_input 7bits | 每7bits接入一个BPUE | 写入的一行wgt数据 |
| Wgt\_en | - | Wgt写入使能 |
| Height 3bits | - | 控制kernel的边长 |
| Sel 1bit |  | Wgt\_reg是否能写入 |

输出信号

|  |  |  |
| --- | --- | --- |
| 信号 | 具体 | 功能 |
| Popcnt\_add 7bits | 7bits 有符号数 | POPCOUNT计算结果 |

寄存器

|  |  |
| --- | --- |
| 名称 | 作用 |
| Wgt | 存放7×7wgt数据 |
| popcnt | 存放1×7popcount计算结果 |
| Popcount\_add | 存放7×7popcount计算结果 |

### BPUE



功能：BPUE对输入的7bits图像数据进行处理，和权重数据进行同或计算，并输出结果

输入信号

|  |  |
| --- | --- |
| 信号 | 功能 |
| Wgt 7bits | 输入的weight值 |
| IMG7bits | 输入的imaget值 |
| Height 3bits | Kernel边长 |

输出信号

|  |  |
| --- | --- |
| 信号 | 功能 |
| Xnor\_out 7bits | 同或计算结果 |

**说明**

输入的1×7img 数据为[6:0]img

当height为7时，给同或单元的img数据就是[6:0]img本身。

当height为5时，给同或单元的img数据的最高位和最低位为1，中间五位是输入的img数据。{1,img[5:1],1}

当height为3时，给同或单元的img数据的最高两位和最低两位为1，中间三位是输入的img数据。{00, img [4:2],00}

# 二、指令译码模块

## 模块说明

输入信号：

|  |  |
| --- | --- |
| 信号 | 功能 |
| Inst[16:0]指令 | 指令 |
| clk | 时钟信号 |
| rst | 复位信号，寄存器全部置零 |

输出信号

|  |  |  |
| --- | --- | --- |
| 信号 | 具体 | 功能 |
| Bnncore\_ctrl[16:0] | 详见第一章 | 给BNNcore的指令 |
| Datasram\_ctrl[14:0] | [14]WEN  [13]CEN  [12:0]地址 | 给DataSram的指令 |
| Instsram\_ctrl[12:0] | [12]WEN  [11]CEN  [10:0]地址 | 给InstSram的指令 |

寄存器

|  |  |
| --- | --- |
| 名称 | 作用 |
| Pc1 | 用作inst\_sram的地址 |
| Pc2 | 用作data\_sram的地址 |
| Pc3 | 用作选通BPUG和BPUE计算的地址 |
| Pc4 |  |

|  |  |
| --- | --- |
| 名称 | 作用 |
| R1 | 保存CMP结果  JUMP指令条件 |
| R2 | 通用寄存器 |
| R3 | 通用寄存器 |
| … | … |
| R12 | 通用寄存器 |

## 译码原理

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 指令类型inst[15:11] | 说明 | 操作数 | 下降沿0 | 上升沿1 | 下降沿1 | 上升沿2 |
| A类指令 | LOAD1H 00001 | 选中Ctrler的R1~R4的高八位保存一个值 | 1：选择写入的寄存器  2：写入的值 | 指令SRAM  输出此指令 | 寄存器Rx存immed  给指令SRAM地址（PC1）、读信号 PC1+1 |  |  |
| A类指令 | LOAD1L 00010 | 选中Ctrler的R1~R4的低八位保存一个值 | 1：选择写入的寄存器  2：写入的值 | 指令SRAM  输出此指令 | 寄存器Rx存immed  给指令SRAM地址（PC1）、读信号 PC1+1 |  |  |
| B类指令 | LOAD2 00011 | 从数据SRAM向BNNCore取32bits，地址和控制信号由Controller给，数据地址存在PC2中，MUX地址是立即数 | 1：图像/权值/偏置  2：选择写入的BPUG列  （3）：选择图像写入REG的哪一部分  最后一个：1则PC2自加，0则PC2自减 | 同上 | 给数据SRAM地址（PC2）、读信号  给BNN Core地址（MUX）、写信号  给指令SRAM地址（PC1）、读信号  PC1+1，PC2+1或-1，PC3+1 | 数据SRAM读出信号 | BNNCore完成写入 |
| A类指令 | ADD1 00100 | Controller中某一寄存器加一个立即数 | 1：选择目标寄存器  2：要加的整数 | 同上 | 寄存器Rx完成加Immed  给指令SRAM地址（PC1）、读信号 PC1+1 |  |  |
| A类指令 | CMP 00101 | 将一个寄存器中的数与立即数比较，结果写回到R1  寄存器值大于立即数，R1赋0，反之赋1 | 1：:作比较的寄存器  一般是PC\*  2：作比较的立即数 | 同上 | 寄存器Rx完成与immed的比较  R1完成写入  给指令SRAM地址（PC1）、读信号  PC1+1 |  |  |
| A类指令 | JUMP 00110 | 条件跳转，如果R1==1，则修改PC中的地址，  PC = PC - immed， 否则PC=PC+1 | 1：跳转的值 | 同上 | 完成PC1=Immed  给指令SRAM地址（Immed）、读信号  PC1+1 |  |  |
| B类指令 | EMPT 00111 | BNN Core里的PSUM置BIAS，BPU里的PSUM置0 | NULL | 同上 | 给BNN Core相应控制信号  给指令SRAM地址（PC1）、读信号  PC1+1 |  | 完成复位 |
| B类指令 | BPUEADD 01000 | 所有BPUE做一次加法，1 of 7 MUX地址由立即数给出 | NULL | 同上 | 给BNN Core相应控制信号  给指令SRAM地址（PC1）、读信号  PC1+1, |  | 完成加法 |
| B类指令 | BPUCADD 01001 | BNNCore内做一次加法，1 of 16 MUX地址由立即数给出 | NULL | 同上 | 给BNN Core相应控制信号  给指令SRAM地址（PC1）、读信号  PC1+1, |  | 完成加法 |
| B类指令 | BNNOUT 01010 | BNN Core内计算结果输出，使得输出端有效（非高阻） | 不池化就没有操作数  1：POOL  2：写入第几个POOL\_REG |  | 给BNN Core相应控制信号  给指令SRAM地址（PC1）、读信号PC1+1 |  |  |
| B类指令 | STORE 01011 | 数据写回数据SRAM，地址由通用寄存器给出 | 1：决定给出哪一部分  2：决定PC2自加或自减 | 同上 | 给数据SRAM地址（Rx）、写信号  给指令SRAM地址（PC1）、读信号  PC1+1, pc2+1或减一 | 完成数据SRAM的写入 |  |
| B类 | SHIFTUP 01100 | 将图像REG上移1bit | NULL |  |  |  |  |
| B类 | MOV 01101 | 将一个寄存器的值赋给另一个寄存器 |  |  |  |  |  |
| A类 | LOAD3H 01110 | 选中Ctrler的R5~R12的高八位保存一个值 | 1：选择写入的寄存器  2：写入的值 | 指令SRAM  输出此指令 | 寄存器Rx存immed  给指令SRAM地址（PC1）、读信号 PC1+1 |  |  |
| A类 | LOAD3L 01111 | 选中Ctrler的R5~R12的低八位保存一个值 | 1：选择写入的寄存器  2：写入的值 | 指令SRAM  输出此指令 | 寄存器Rx存immed  给指令SRAM地址（PC1）、读信号 PC1+1 |  |  |
|  | NULL 00000 | 空指令，不执行任何操作 |  | 同上 | 给指令SRAM地址（PC1）、读信号 PC1+1 |  |  |

## 汇编器使用方法

一行汇编是一条指令，操作数间用空格隔开

1. LOAD1L、LOAD1H、LOAD3L、LOAD3H  
   例1：LOAD1L R3 5 最后的操作数是赋的值，小于等于255即可  
   例2：LOAD1L PC1 31
2. LOAD2：  
   LOAD2 WGT 3 2 1 其中“3” 指第四列BPUG，“2”代表选通第3个BPU的wgt，“1”代表PC2自加1  
   LOAD2 BIAS 1 “1”代表PC2自加1  
   LOAD2 IMG 3 1 1 “3”代表第四列BPUG，第一个1写入[15:8]行，后代表PC2自加1
3. ADD:  
   ADD R3 1  
   ADD PC3 2 后面的操作数是给寄存器加的值，介于-128~127之间
4. CMP  
   CMP R2 3 比较的数，介于0~511
5. JUMP:  
   JUMP 3 ：判断条件，PC1减3
6. EMPT：  
   EMPT 没有其他操作数
7. BPUEADD：  
   BPUEADD 3 1 ：“3”代表选通第4个BPUE，“1”代表选IMG\_REG的[7:1]列
8. BPUCADD:  
   BPUCADD 7：选通 第8个BPUG的计算结果求和
9. BNNOUT:  
   BNNOUT 直接将结果输出到BNN CORE的结果寄存器里  
   BNNOUT POOL 3 后面的数是放入第几位POOL\_REG，0~3
10. STORE:  
    STORE 1 0 ：1是输出第[7:4]列结果REG，0 PC4（存放 结果的写入地址）自减
11. SHIFTUP:  
    SHIFTUP ：IMG\_REG上移一位，没有其他操作数
12. MOV

将一个寄存器的值保存至另一个寄存器中

MOV R3 PC1：把PC1的值保存至R3中

# 三、SRAM模块

DATASRAM:保存数据和权值，宽度32位，深度8192（13bit）

INSTSRAM:保存指令，宽度16位，深度2048（11bit）

SRAM端口说明

input [(ADDRWIDTH-1):0] ADDR; //地址

input [(DATAWIDTH-1):0] D; //输入数据端

output reg [(DATAWIDTH-1):0] Q; //输出数据端

input CEN; //片选信号，低有效

input WEN; //写使能信号，低—写，高—读

input CLK; //时钟

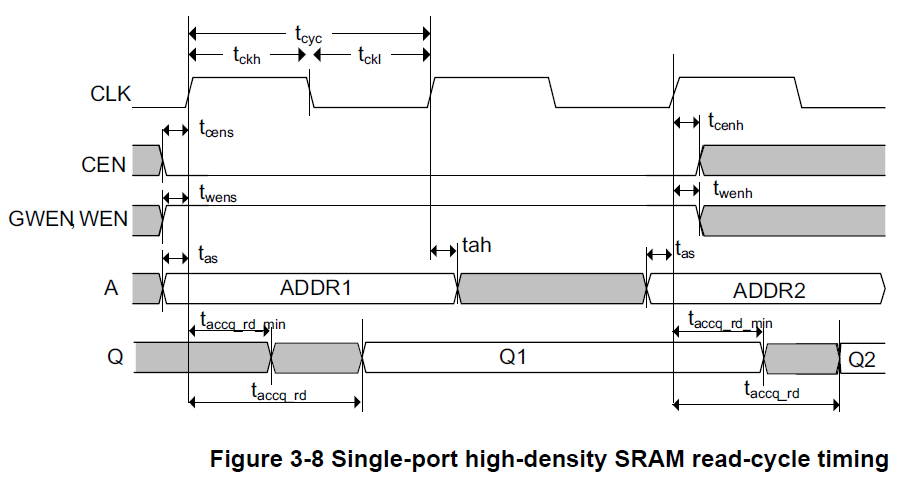
input [2:0] EMA; //延时信号，置3’b010

input [1:0] EMAW; //延时信号，置2’b00

input EMAS; //延时信号，置0

input RET1N; //数据保持信号，置1

SRAM读取时序



SRAM写入时序

# 四、向BNN模块写入数据

TOP模块接口

|  |  |  |
| --- | --- | --- |
| 端口名称 | 说明 |  |
| CLK | 系统总时钟，10M |  |
| RST | 系统复位信号，高有效，置高时指令从头开始执行，即所有寄存器为0 |  |
| PAUSE | 系统暂停信号，高有效，置高时指令执行暂停，即所有寄存器数据保持不变，停止取指令 |  |
| SRAMSEL | 选通SRAM的连接方式，与外部端口还是内部内部 | =1，DATA/INST SRAM的地址和数据端接到芯片端口（由SRAM\_MUX选通）;=0，DATA SRAM地址和控制接到CTRL，数据接到BNN Core，INST SRAM地址、控制和数据接到CTRL |
| SRAMMUX | 仅当SRAM\_SEL=1时有效，选通DATA/INST SRAM哪个与外部接口相连 | =1，DATA SRAM的地址、控制和数据接到芯片端口  =0，INST SRAM的地址、控制和数据接到芯片端口 |
| SRAMDIN | 串行数据输入，连接方式由SRAM\_SEL/MUX决定，只有在SRAMSEL=1且SRAMWEN=0时有效 | SRAMSEL=1时，连续32个周期输入到DATASRAM中，LSB在前  SRAMSEL=0时，连续16个周期输入到INSTSRAM中，LSB在前 |
| SRAMA[13:0] | 并行地址输入，[12:0]给DATASRAM，[10:0]给INSTSRAM |  |
| SRAMDOUT | 串行数据输出，连接方式由SRAM\_SEL/MUX决定 | SRAMSEL=1,时，连续32个周期输出DATASRAM数据，LSB在前  SRAMSEL=0,时，连续16个周期输出INSTSRAM数据，LSB在前 |
| SRAMCEN | SRAM片选使能，低有效 |  |
| SRAMWEN | SRAM写使能，低有效 | =1，SRAM读；=0，SRAM写 |

### 读写SRAM

#### 写DATASRAM

SRAMSEL=1/SRAMCEN=1/SRAMWEN=1（初始化状态）（下降沿）——SRAMCEN=0/SRAMMUX=1/SRAMA给地址（下降沿）——SRAMWEN=0/SRAMDIN给LSB（下降沿）——后面连续给接下来31位数据，由低到高（下降沿）

#### 写INSTSRAM

SRAMSEL=1/SRAMCEN=1/SRAMWEN=1（初始化状态）（下降沿）——SRAMCEN=0/SRAMMUX=0/SRAMA给地址（下降沿）—— SRAMWEN=0/SRAMDIN给LSB （下降沿）——后面连续给接下来31位数据，由低到高（下降沿）

#### 读DATASRAM

SRAMSEL=1/SRAMCEN=1/SRAMWEN=1（初始化状态）（下降沿）——SRAMSEL=1/SRAMCEN=0/SRAMWEN=1/SRAMMUX=1/SRAMA给地址/（下降沿）——SRAMDOUT输出一个0（上升沿）——SRAMDOUT连续输出32位数据，由LSB到MSB（上升沿）

#### 读INSTSRAM

SRAMSEL=1/SRAMCEN=1/SRAMWEN=1（初始化状态）（下降沿）——SRAMSEL=1/SRAMCEN=0/SRAMWEN=1/SRAMMUX=0/SRAMA给地址/（下降沿）——SRAMDOUT输出一个0（上升沿）——SRAMDOUT连续输出16位数据，由LSB到MSB（上升沿）