模块说明

目录

一.BNN Core

1.指令综述:

信号	**具体指令及功能** 按 [19:0]格式	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$\begin{cases} others = 0, no\ pooling \\ [12] = 1, [6] and [13] depends \end{cases}$ 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。

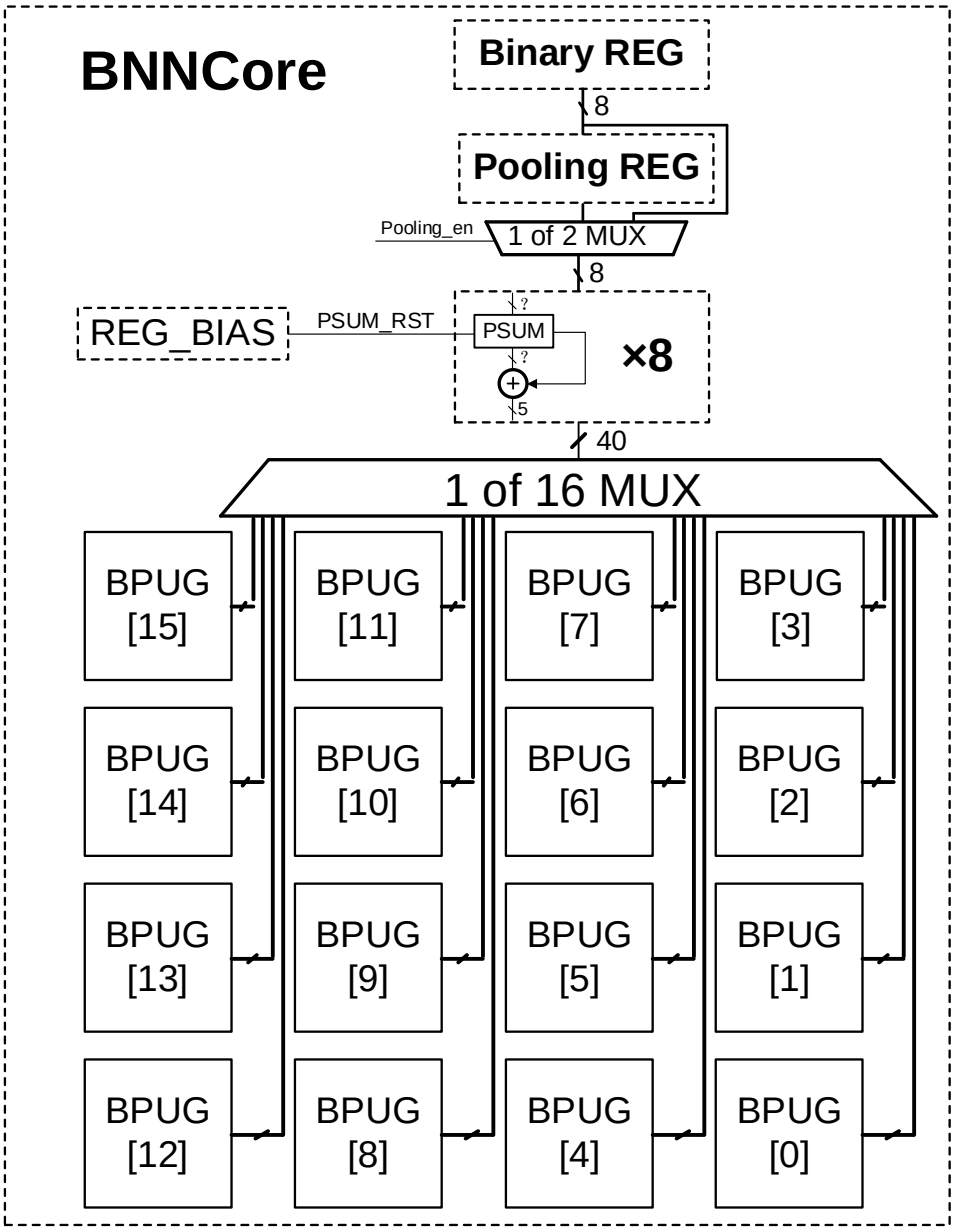
[4:1]：控制数据写入的列选信号，控制计算时 BPUE 选通，控制计算时 BPUG 选通。这三个功能都不是同时进行的，因此[4:1]理论上可以复用，经仿真验证也没有问题。

[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$\begin{cases} others = 0, no\ pooling \\ [12] = 1, [6] and [13] depends \end{cases}$ 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 信号控制写入哪个 REG

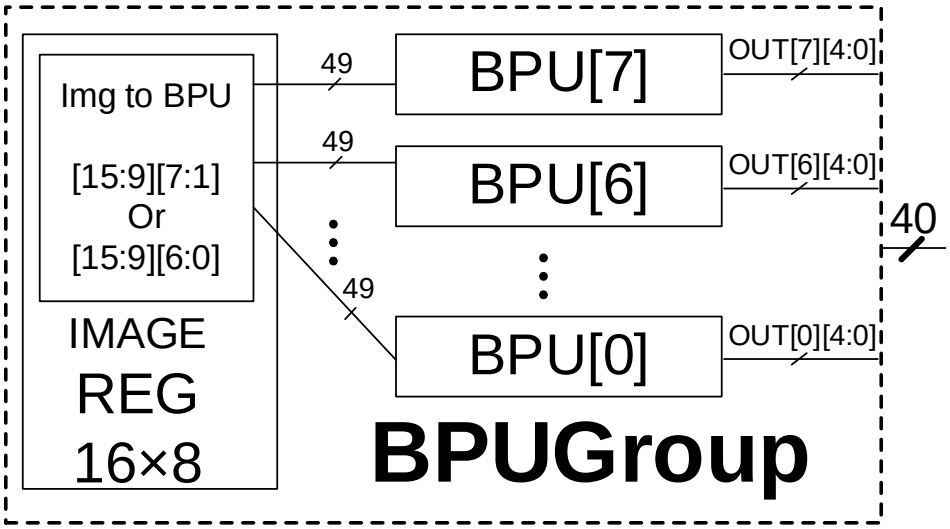
输出信号

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	由 bpug_sel 控制生成
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

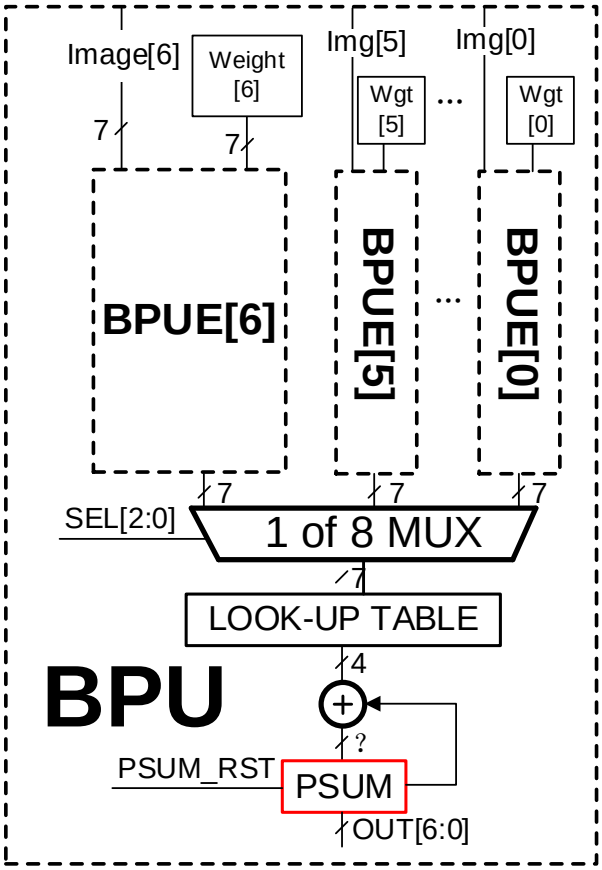
寄存器

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	每 7bits 接入一个 BPUE	计算的图像数据
Wgt_input 7bits		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

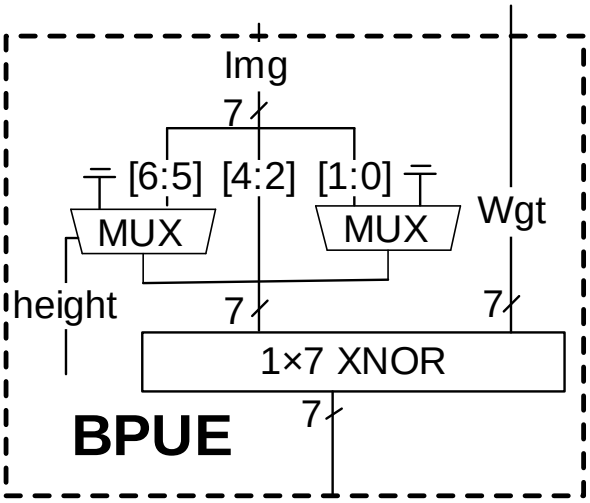
输出信号

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

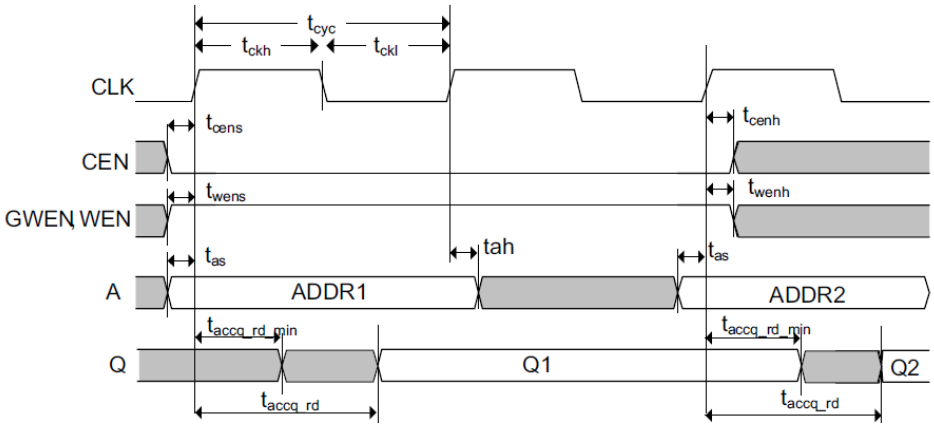


Figure 3-8 Single-port high-density SRAM read-cycle timing

SRAM 写入时序