

功能

- 核心：两个先进的 RISC-V 32-bit RI5CY 内核，全面支持 RV32I 基本整数指令集，RV32C 压缩标准扩展指令集和 RV32M 整数乘除法扩展指令集。
 - 硬件循环，增加后存取指令，DSP 和向量操作指令，这些指令增强了内核在信号处理应用中的效率。
 - 大核频率最高至 200Mhz，IPC 接近于 1
- AI 内核：一个支持 ANN 算法的专用指令集处理器。
 - 8 个 PE 组成脉动阵列支持 800M 次每秒累加运算及硬件激活函数。
- 内存：哈佛架构内存组织
 - 大核：64Kb 数据和指令 RAM
 - 小核：32kb 数据和指令 RAM
 - 独立的 Bootloader 内存空间
 - 外部 DDR 控制器分别具有 16 根地址和数据线
- 电源和时钟：
 - 1.2 V 内核电压、1.8 V DDR 电压
 - 3.3 V IO 供电
 - 25Mhz 外部时钟输入，内部 PLL 可调
- 功耗：< 1W
 - 时钟树可关闭核心和所有外设时钟
 - 内核支持睡眠和停止模式
- 大核外设：
 - SPI x 2，支持协议映射
 - UART x 2，最高 12.5 Mbps
 - I2C x 2
 - EMMC/SDIO/SD x 2

- DDRCTL x 1
 - Camera Bridge x 1，8bit 并行摄像头接口，最大带宽 37 Mbyte/s
 - PWM x 4，32bit 宽度计数器，1-8 分频
 - Timer x 2，32bit 宽度计数器，1-8 分频
 - GPIO x 6
- 小核外设：
 - SPI x 1，支持协议映射
 - UART x 1，最高 5 Mbps
 - I2C x 1
 - PWM x 4，32bit 宽度计数器，1-8 分频
 - Timer x 2，32bit 宽度计数器，1-8 分频
 - GPIO x 5
- 编程与调试接口：
 - JTAG 接口
 - SPI 调试接口
 - 串口编程接口，通过 Bootloader 加载
- 封装：11 x 11mm BGA 封装，256 Pin

特性

- ΦPU 物端处理器内核基于 RISC-V 开源架构，具有高效的运算效率，CoreMark 评分 2.1。
- SoC 处理器内核经过了工业实用检验，使得其完全可信赖。
- 双核心的配置，使得多实时任务处理更加便捷，配合时钟控制降低了功耗，实现多种 Always-On 功能。
- AI 内核可对神经网络，图像处理等多种算法进行加速，并且具有完整的编译器。

目录

1 总体描述	3	4.5.2 CTRL(Timer Control)	29
2 CPU	5	4.5.3 CMP(Timer Compare)	30
2.1 概述	5	4.6 Event Unit.....	30
2.2 接口	6	4.6.1 IER(Interrupt Enable).....	31
2.2.1 接口定义	6	4.6.2 IPR(Interrupt Pending)	31
2.2.2 接口时序	8	4.6.3 ISP(Interrupt Set Pending)	31
2.3 主要流程	9	4.6.4 ICP(Interrupt Clear Pending)...	31
3 ANN 内核	11	4.6.5 EER(Event Enable).....	32
3.1 概述	11	4.6.6 EPR(Event Pending).....	32
3.2 接口	13	4.6.7 ESP(Event Set Pending)	32
3.3 控制寄存器配置	15	4.6.8 ECP(Event Clear Pending)	33
3.4 操作控制流程	16	4.6.9 SCR(Sleep Control).....	33
4 外设	19	4.6.10 SSR(Sleep Status).....	33
4.1 UART	19	4.7 SoC Control	33
4.2 GPIO	19	4.7.1 PAD Mux.....	34
4.2.1 PADDR (Pad Direction)	20	4.7.2 CLK Gate	34
4.2.2 PADIN(Input Values)	20	4.7.3 Boot Address.....	34
4.2.3 PADOUT(Output Values)	20	4.7.4 Info	34
4.2.4 INTEN (Interrupt Enable)	20	4.7.5 Status	34
4.2.5 INTTYPE0(Interrupt Type 0)	21	4.7.6 PAD Configuration	35
4.2.6 INTTYPE1(Interrupt Type 1)	21	4.8 Debug Port	35
4.2.7 INTSTATUS(Interrupt Status) ..	21	4.9 Pulse Width Modulation(PWM).....	36
4.2.8 PADCFG0-7(Pad Configuration		5 高级调试单元	37
Registers 0-7).....	21	6 SPI slave	38
4.3 SPI Master	22	7 摄像头接口	39
4.3.1 STATUS(Status Register)	22	7.1 概述	39
4.4 I2C	26	7.2 接口时序	39
4.4.1 CPR (Clock Prescale Register)		7.3 内部结构	40
.....	27	8 EMMC 接口	41
4.4.2 CTRL(Control Register)	27	8.1 主要功能	41
4.4.3 TX(Transmit Register).....	27	8.2 eMMC 接口总体结构.....	41
4.4.4 RX (Receive Register)	28	9 内存控制器	42
4.4.5 CMD (Command Register).....	28	9.1 概述	42
4.4.6 STATUS (Status Register)	28	10 引脚定义	44
4.5 Timer	29	11 存储映射	45
4.5.1 TIMER(Current Timer Value)...	29		

1 总体描述

ΦPU 物端处理器包含两个基于 RISC-V 的 RI5CY 内核的片上系统，采用大小核设计，大核的运行频率为 200Mhz，小核运行频率为 80Mhz。内核使用分开的数据和指令单口 RAM，并包含一个启动 ROM 用于存储 boot loader，来通过 SPI 读取外部 flash 芯片存储的程序或者使用串口对芯片进行编程。

配备了一个支持 ANN 操作的深度学习内核，峰值运算能力为 800M flops，可用于对图像分类，图像处理等应用的加速。ANN 内核挂载在 AHB 总线上，共享系统内存。该平台集成了摄像头接口、LPDDR 控制器、eMMC 控制器等，因此非常适合于视频存储与分析，实时运动控制等应用。

系统时钟树如图 1.1 所示：

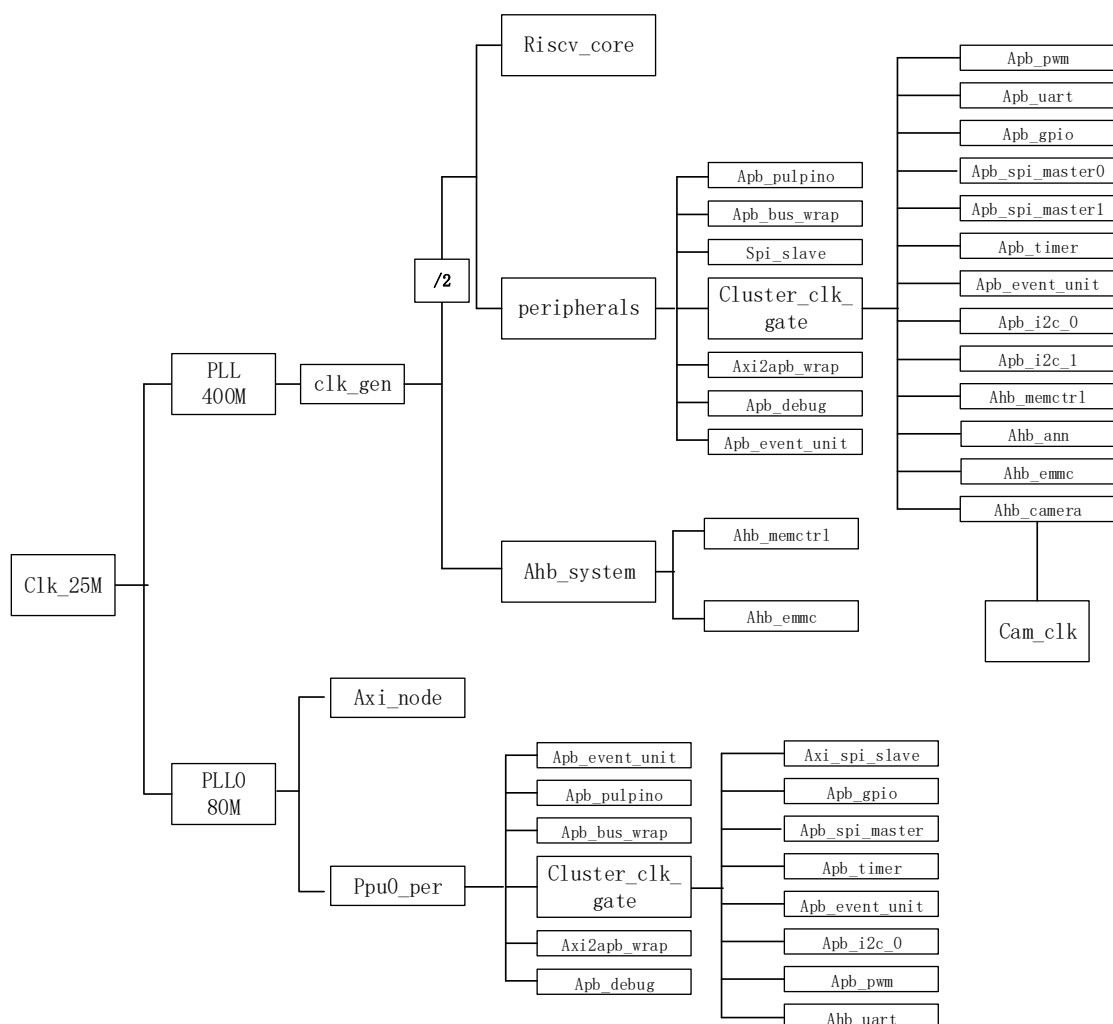


图 1.1 系统时钟树

系统框图如图 1.2 所示。

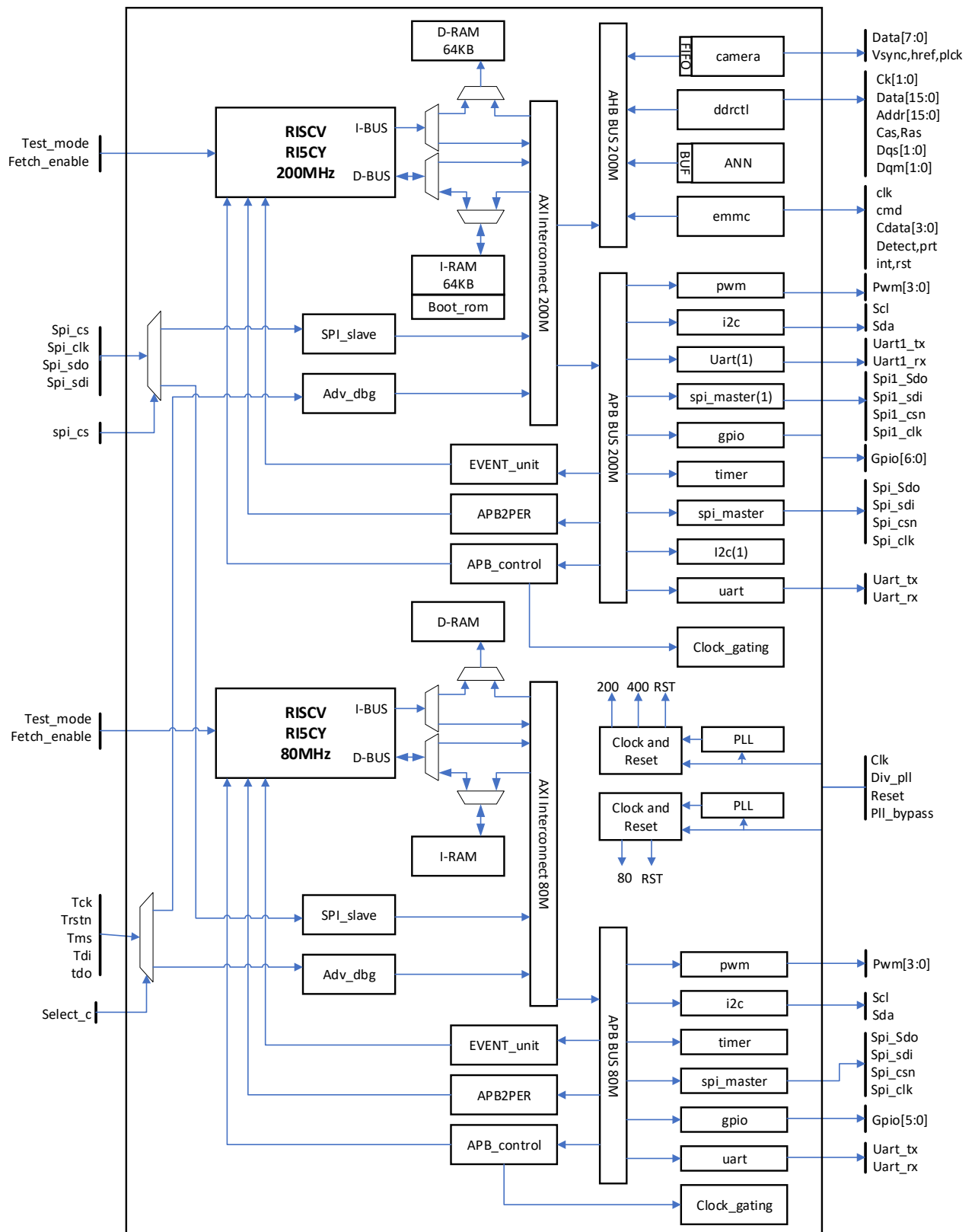


图 1.2 φPU 系统框图

2 CPU

2.1 概述

RI5CY 内核是一个基于 RISC-V 架构的 32 位先进处理器。RI5CY 内核的 IPC(每时钟周期的指令数)接近于 1，完全支持基本整数指令集 (RV32I)，压缩指令集 (RV32C) 和乘法指令集扩展 (RV32M)。其也实现了一些扩展的 DSP 指令，比如：硬件循环，增加后存取指令，乘累加和向量操作指令，这些指令增强了内核在信号处理应用中的效率。

RI5CY 内核使用独立的指令和数据总线，指令和数据内存 SRAM 均为 64k 字节（可选）。其中指令总线为单向接口，数据为双向接口。在接口处可以使用内核/总线协议转换单元，将一部分地址映射至总线，实现处理器对总线，外部存储器及各种外设的访问。

RI5CY 内核使用一个具有四级流水线的顺序发射架构，其结构如图 2.1 所示。内核的取指单元可以每周期给指令解码级提供一条指令，为了优化性能和时间裕度，使用了一个指令预取单元来从指令内存或者缓存提取指令，其具有一个深度为 4 的 32 位宽的 FIFO 来存储预取的指令。各级流水线之间完全独立，后续流水级无论前序流水级是否阻塞都可以执行，数据可以在流水线间尽可能的传输，避免了不必要的阻塞。

RI5CY 内核也支持很多低功耗特性，内核空闲时可以被时钟门控进入低功耗模式，当有外部事件触发时才通过事件单元激活内核。

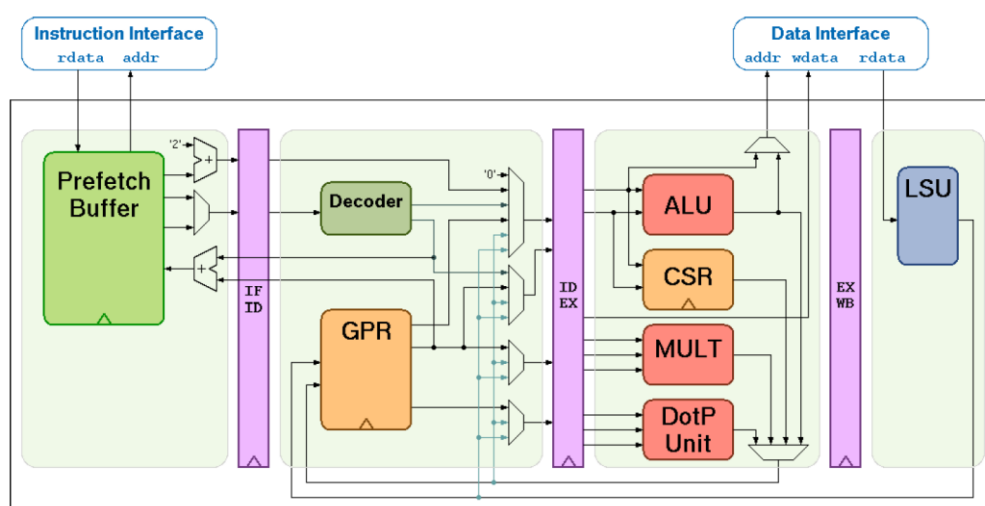


图 2.1 RI5CY 内核架构框图

2.2 接口

2.2.1 接口定义

表 2-1 CPU 指令接口信号描述了内核的取指令信号，其协议与 LSU（载入与存储单元）使用的协议相同，但其没有写操作因此信号较少，协议时序参看 2.2.2 小节。

表 2-1 CPU 指令接口信号

信号	方向	描述
instr_req_o	output	请求准备好，必须保持高直到 instr_gnt_i 保持高一个周期
instr_addr_o[31:0]	output	地址
instr_rdata_i[31:0]	input	从内存读取的数据
instr_rvalid_i	input	当 instr_rdata_i 具有有效数据时，该信号为高。在每次请求中，该信号将只有一个周期的高电平时间。
instr_gnt_i	input	另一侧接受了请求。instr_addr_o 可以在下个周期改变。

内核的 LSU 单元负责接入数据内存。以字（32bit），半字（16bit）和字节（8bit）载入和存储都是支持的。表 2-2 描述了内核 LSU 单元的接口信号。

表 2-2 CPU 数据接口信号

信号	方向	描述
data_req_o	output	请求准备好，必须保持高直到 data_gnt_i 保持高一个周期
data_addr_o[31:0]	output	地址
data_we_o	output	写使能，高为写，低为读，和 data_req_o 一起发送。
data_be_o[3:0]	output	字节使能，对要读写的字节设置，和 data_req_o 一起发送。

data_wdata_o[31:0]	output	要写进内存的数据，和 data_req_o 一起发送。
data_rdata_i[31:0]	input	从内存读取的数据
data_rvalid_i	input	当 data_rdata_i 具有有效数据时，该信号为高。在每次请求中，该信号将只有一个周期的高电平时间。
data_gnt_i	input	另一侧接受了请求。 data_addr_o 可以在下个周期改变。

LSU 单元可以执行非对齐接入，接入没有对齐到自然边界的字。这个操作需要执行两次分开的对齐字的访问。所以，至少需要两个周期来实现非对齐的读写。

内核具有针对 adv_debug_if 的调试接口，其接口信号如表 2-3 所示

表 2-3 CPU 调试端口信号

信号	方向	描述
debug_req_i	input	请求
debug_gnt_o	output	授权/同意
debug_rvalid_o	output	读数据有效
debug_addr_i[14:0]	input	读写地址
debug_we_i	input	写使能
debug_wdata_i[31:0]	input	写数据
debug_rdata_o[31:0]	output	读数据
debug_halted_o	output	当内核在调试模式时输出高
debug_halt_i	input	当需要内核进入调试模式设置为高
debug_resume_i	input	当需要内核退出调试模式设置为高

debug_halted_o, debug_halt_i 和 debug_resume_i 是用来作为在多个内核间循环触发。在单核调试时不是必须的，因此，debug_halt_i 和 debug_resume_i 可以接到 0。若使用这两个信号，它们应当分别只有一个

周期为高来避免锁死。

内核接口的其余信号可分为时钟与复位，静态的 ID 和启动地址，中断输入和内核的控制信号，其列于表 2-4。

表 2-4 内核功能信号

信号	方向	描述
clk_i	input	输入时钟
rst_ni	input	复位
clock_en_i	input	使能时钟，否则其为关闭状态
test_en_i	input	为测试使能所有时钟门控
boot_addr_i	input	起始地址输入
core_id_i	input	内核 id 输入
cluster_id_i	input	内核群 id 输入
irq_i	input	电平敏感的中断寄存器输入
fetch_enable_i	input	控制 cpu 是否进行取值操作
core_busy_o	output	内核是否为忙的输出
ext_perf_counters_i	input	外部性能计数器

2.2.2 接口时序

CPU 与内存通信的协议工作流程如下：

LSU 单元在 data_addr_o 上提供有效地址并设置 data_req_o 为高。之后内存一旦准备好提供需求，立即将 data_gnt_i 设置为高作为响应。响应可能发生在与发出请求的同一周期，或者几个周期之后。当接收到授权信号，LSU 就可以在下一个周期改变地址。并且，data_wdata_o, data_we_o 和 data_be_o 信号也可以改变，基于对内存已经处理和存储了信息的假设。在接收了授权之后，如果 data_rdata_i 有效内存将 data_rvalid_i 设置为高以响应。响应可能在接收到授权信号之后的一个或多个周期发生。注意当一个写操作执行时 data_rvalid_i 也必须置位，尽管此时 data_rdata_i 没有意义。

图 2.2、图 2.3、图 2.4 为协议的典型时序图。

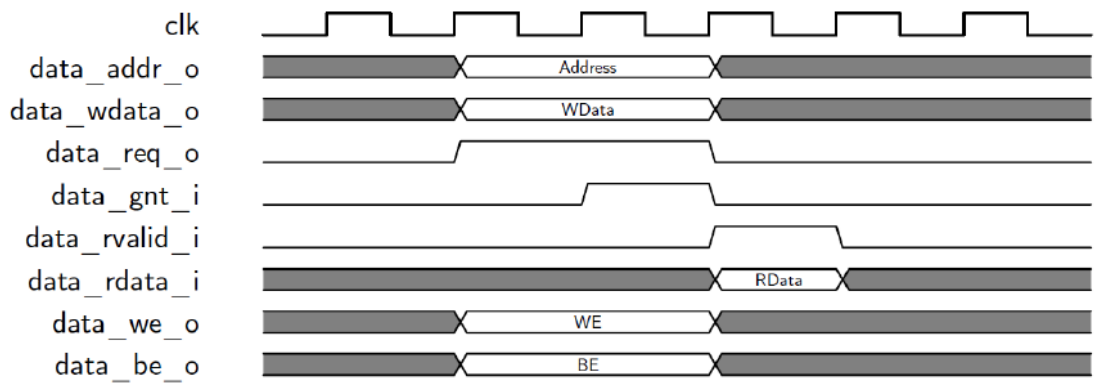


图 2.2 基本的内存传输

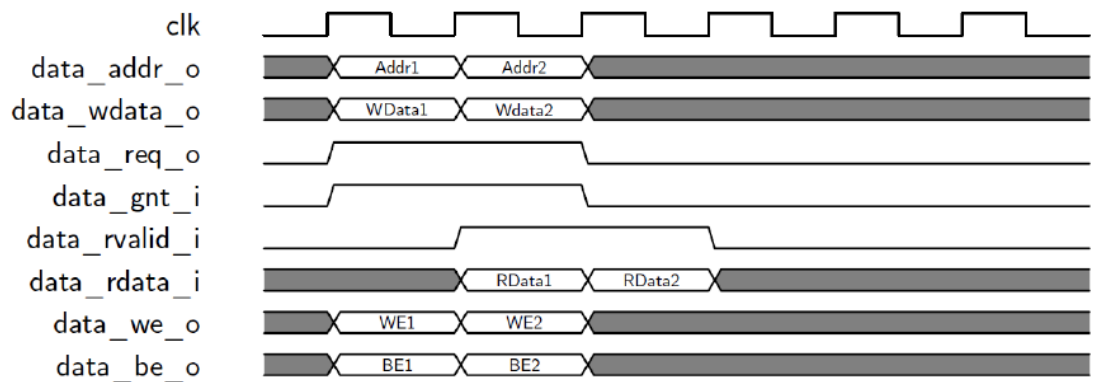


图 2.3 连续的内存传输

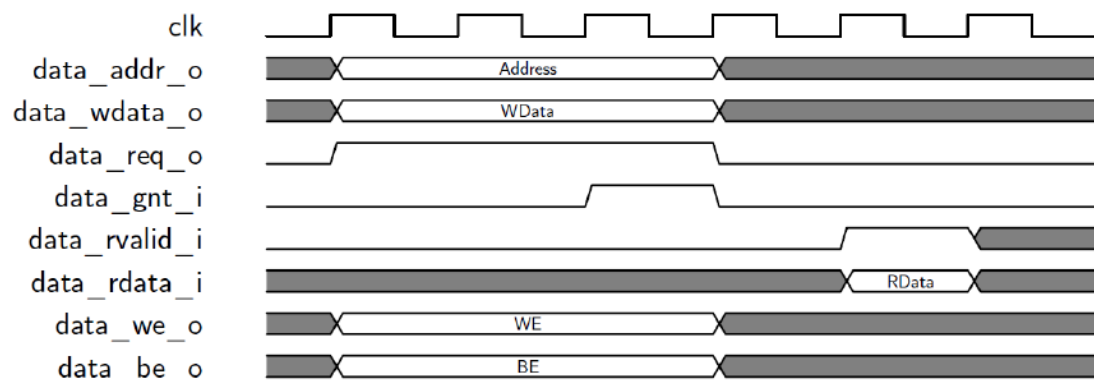


图 2.4 慢响应的内存传输

2.3 主要流程

CPU 在启动或者复位之后，程序指针 PC 将指向 BOOT_ROM 所在的内存区域，启动地址由 boot_addr 寄存器的默认参数控制。处理器随后执行存储在 BOOT_ROM 中的 BOOT_LOADER 代码从外部存储器提取用户程序代码到内存空间，并跳转到相应地址执行程序。

程序的例外包含向量中断，错误和特权。其中复位，非法指令，特权和 LSU 错误为固定的地址，用户中断地址可以在前 32 个地址空间中分配，中断地址以 boot_addr 寄存器内容为基地址，其分布如表 2-5 所

示，程序应根据中断向量表的位置重新分配 boot_addr 地址以使得中断系统可以正常运作。

表 2-5 中断/例外向量表

Base address: boot_addr

00h-7Ch	IRQ0-31
80h	RST
84h	ILLINSN
88h	ECALL
8Ch	LSUERR

3 ANN 内核

3.1 概述

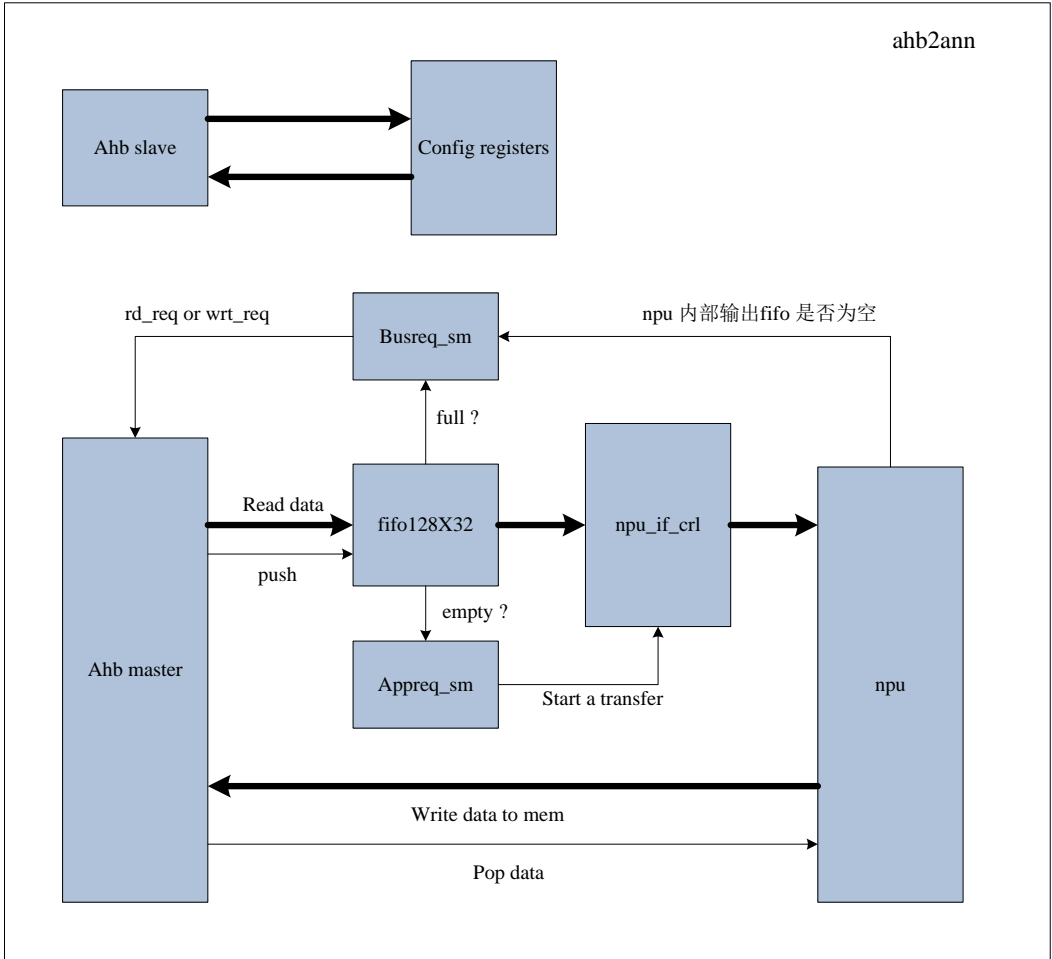


图 3.1 Ahb2ann 结构图

AI 内核支持 ANN 的拓扑结构和算法，其执行一套专用指令集，对 ANN 的拓扑进行运行规划和计算。ANN 内核内具有 8 个处理单元，每个处理单元为一个乘加核心，具有本地数据缓存，并以脉动阵列的方式传递数据，每个神经元所对应的数据都会经过硬件激活函数以得到输出。数据的输入输出特点：

1. 在运算开始之前，需要对 Npu 内部的多个统一编址的 ram 进行数据初始化。通过 ram 端口输入数据。输入最大数据量 22.5KB。此时 RST_N 维持低电平。
2. 在运算开始之后，需要输入运算数据，通过端口 npu_din 输入。输入最大数据量 640*480*2B。此时 RST_N 维持高电平。
3. 运算完成，运算结果数据输出，通过 npu_dout 输出。数据量较小。此时 RST_N 维持高电平。

表 3-1 内核模块端口说明

方向	名称	width	描述
input	CLK		时钟
input	RST_N		复位，运算单元参数初始化过程该信号为低电平。运算过程中该信号为高。
input	npu_din	[31: 0]	数据输入
input	npu_enq		写输入数据 fifo
input	npu_deq		读输出数据 fifo
Input	ram_din	[15: 0]	参数输入
Input	ram_reg_adr	[10: 0]	参数地址
Input	ram_mem_adr	[2: 0]	参数地址（ram 块地址）
Input	ram_we		参数写使能
output	npu_dout	[31:0]	数据输出
output	npu_dout_valid		数据输出有效，输出 fifo 非空
output	npu_full		数据队列满
output	input_count	[10:0]	数据输入 fifo 计数器
output	output_count	[6:0]	数据输出 fifo 计数器

数据的输入和输出的时序如图 3.2，图 3.3 及图 3.4 所示。

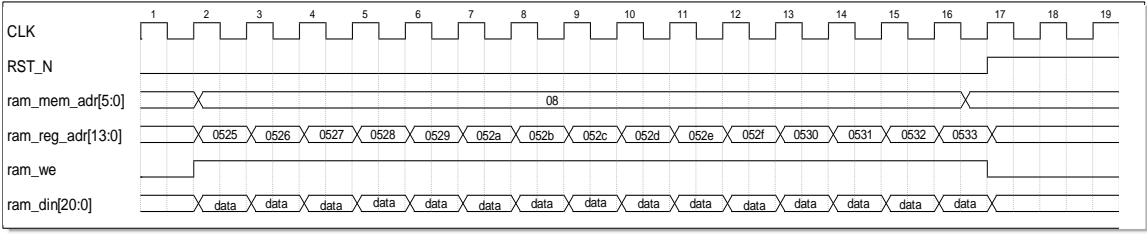


图 3.2 运算前运算参数输入时序图

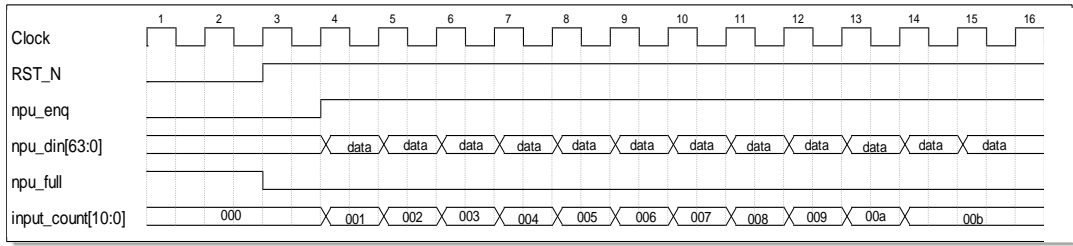


图 3.3 运算中运算数据输入时序图

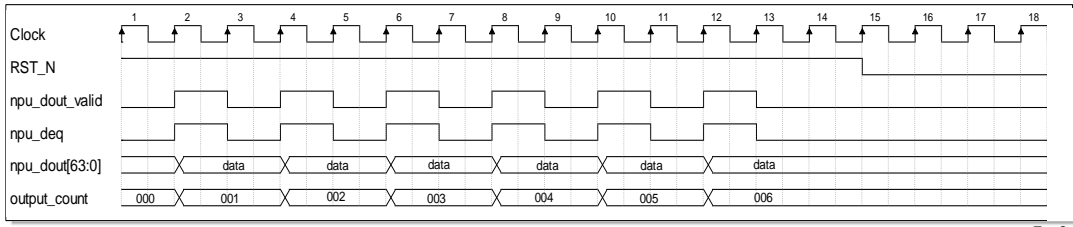


图 3.4 运算中运算结果输出时序图

3.2 接口

表 3-2 Ahb2ann 接口信号

信号名	位宽	方向	说明
全局信号			
hclk	1	input	时钟
hresetn	1	input	复位
ahb master 接口信号			
ahbm_hresp	2	input	okey error retry split
ahbm_haddr	32	output	访问地址
ahbm_htrans	2	output	空闲 忙 10- 非连续 连续

ahbm_hwrite	1	output	1 写操作，0 读操作
ahbm_hsize	3	output	表示传输大小 只产生 32 位的写操作
ahbm_hburst	3	output	Burst 类型：设计中只使用了不定长突发 incr 和 single 两种类型。读请求的时候是不定长突发，突发长度为 block_size 长度。写请求是突发长度固定为 8 的 INCR。剩余写数据长度不足 8 时，INCR 写完剩余长度即可。
ahbm_hwdata	32	output	写数据
ahbm_hrdata	32	input	读数据
ahbm_hbusreq	1	output	总线请求
ahbm_hgrant	1	input	总线授权
ahbm_hport	4	output	固定给出 0001
ahbm_hlock	1	output	不产生 lock 请求
ahbm_hready_in	1	input	Ready 信号（握手信号）
ahb slave 接口信号			
ahbs_hsel	1	input	从机选中
ahbs_haddr	32	input	地址
ahbs_htrans	2	input	传输类型
ahbs_hwrite	1	input	读写选择
ahbs_hsize	3	input	传输大小（always WORD 010）
ahbs_hburst	3	input	突发类型 寄存器访问不使用 sHBURST，每拍只根据 sHTRANS（NONSEQ/SEQ）和地址进行读写。
ahbs_hwdata	32	input	写数据
ahbs_hrdata	32	output	读数据

			寄存器访问只支持 32 位操作，默认地址最低两位为零，即按 32 位对界。
ahbs_hready_out	1	output	Ahb 数据总线准备好（输出）
ahbs_hresp	2	output	传输响应 只产生 OKAY 响应
ahbs_hready_in	1	input	Ahb 数据总线准备好（输入）
interrupt	1	output	中断（运算过程或初始化过程结束时，发出中断）

3.3 控制寄存器配置

表 3-3 内核寄存器配置

地址		名称	类型	描述
控制寄存器				
0x00	[0]	start	W1, 0x0	启动 npu 开始工作
	[1]	init_ram	W1, 0x0	启动初始化操作
	[2]	dma_en	RW, 0x0	1.dma 使能
	[3]	int_clr	W1, 0x0	清除中断
	[4]	stop	W1,0x0	强制退出当前过程(init 过程或 npu 运算过程)
	[31:4]			保留
中断使能				
0x04	[0]	int_en_init	RW, 0x1	初始化完成中断使能
	[1]	int_en_npu	RW, 0x1	运算完成中断使能
	[2]	err_int_en	RW, 0x1	运算异常中断使能
	[31:3]			保留

Dma 参数				
0x08		src_addr	RW, 0x0	读数据源地址
0x0c		dst_addr	RW, 0x0	写数据目的地址
0x10	[15:0]	block_count	RW, 0x0	
	[21:16]	block_size	RW, 0x0	
运算过程控制参数				
0x14		weight_depth	RW, 0x0	权重数据深度
0x18		bias_depth	RW, 0x0	偏置数据深度
0x1c		im_depth	RW, 0x0	指令数据深度
0x20		sig_depth	RW, 0x0	查找表数据深度
0x24		npu_datain_depth	RW, 0x0	运算数据输入深度
0x28		npu_dataout_depth	RW, 0x0	运算数据输出深度
状态寄存器				
0x2c	[0]	init_end	RO,0x0	初始化操作完成
	[1]	npu_end	RO,0x0	运算操作完成
0x30	[0]	init_on	RO,0x0	初始化操作正在进行
	[1]	npu_on	RO,0x0	npu 运算操作正在进行

3.4 操作控制流程

运算单元分为初始化过程，运算过程。

表 3-4 初始化过程流程

	操作	寄存器
Step0 检查加速器状态	使用加速器之前先检查加速器是否处于工作状态。	0x30 [30'b0,npu_on, init_on]

	状态寄存器 npu_on 和 init_on 都为 0 时可以配置及启动	
Step1 配置参数寄存器	源地址	src_addr
	传输块大小	block_size
	传输块个数	block_count
	参数中权重数据深度	weight_depth
	参数中偏置数据深度	bias_depth
	参数中指令数据深度	im_depth
	参数中查找表数据深度	sig_depth
Step2 启动初始化过程		init_ram & dma_en
Step3 初始化过程结束	发中断 cpu 收到中断通过读取状态寄存器 0x2c [npu_end,init_end] 判断是哪一处理过程结束引发中断 cpu 清除中断 & 关闭 dma_en	dma_en int_clr

表 3-5 运算过程流程

	操作	寄存器
Step0 检查加速器状态	使用加速器之前先检查加速器是否处于工作状态。 状态寄存器 npu_on 和 init_on 都为 0 时可以配置及启动	0x30 [30'b0,npu_on, init_on]
Step1 配置参数寄存器	源地址	src_addr

4 外设

在 PPU0 中所有的外设都是连接 APB 总线上的，除了 SPI Slave 之外。在第六章有更多关于 SPI Slave 的介绍。PPU 和 PPU0 类似，区别是新增加的 memctl（内存控制器），Emmc（外存控制器），Camera capture（图像数据采集），ANN（神经网络加速器）是连接在 AHB 总线上的。

4.1 UART

UART 有六个输入输出接口分别是 uart_tx,uart_rx,uart_rts,uart_cts,uart_dtr,uart_dsr.UART 连接在 APB 总线上，APB 总线的时钟频率是 200Mbit/s,当过采样频率为 16 时，Max.baud rate in Mbits is 12.UART 接口的传输速率可以达到 0.1152Mbit/s。

表 4-1 串口外部信号

Signal	Direction	Description
uart_tx	output	Transmit Data
uart_rx	input	Receive
uart_rts	output	Request to
uart_cts	input	Clear to
uart_dtr	output	Data Terminal
uart_dsr	input	Data Set

4.2 GPIO

表 4-2 GPIO 信号

Signal	Direction	Description
Gpio_in[31:0]	input	Transmit Data
Gpio_out[31:0]	output	Receive Data
Gpio_dir[31:0]	output	Request to Send
Gpio_padcfg[5:0][31:0]	output	Pad Configuration
interrupt	output	Interrupt(Rise or Fall or Level)

4.2.1 PADDIR (Pad Direction)

Reset Value:0x0000_0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	PADDIR

Bit 31:0 PADDIR: Pad Direction.

控制每个 GPIO Pad 的值。值 1 表示将其配置为输出，而将 0 配置为输入

4.2.2 PADIN(Input Values)

Reset Value:0x0000_0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	PADIN

Bit 31:0 PADIN:Input Values.

4.2.3 PADOUT(Output Values)

Reset Value:0x0000_0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	PADOUT

Bit 31:0 PADOUT:Output Values.

4.2.4 INTEN (Interrupt Enable)

Reset Value:0x0000_0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
IT	IT	IT	IT	IT	IT	IT	IT	IT	IT	IT	IT	IT	IT	IT	IT	IT	IT	IT	IT	IT	IT	IT	IT	IT	IT	IT	IT	IT	IT	IT	IT	INTEN

Bit 31:0 INTEN: Interrupt Enable.

每个输入信号的中断使能，INTTYPE0 and INTTYPE1 控制中断触发。

There are four triggers available

INTTYPE0 = 0, INTTYPE1 = 0: Level 1

INTTYPE0 = 1, INTTYPE1 = 0: Level 0

INTTYPE0 = 0, INTTYPE1 = 1: Rise

INTTYPE0 = 1, INTTYPE1 = 1: Fall

4.2.5 INTTYPE0(Interrupt Type 0)

Reset Value:0x0000_0000


31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
T0	T0	T0	T0	T0	T0	T0	T0	T0	T0	T0	T0	T0	T0	T0	T0	T0	T0	T0	T0	T0	T0	T0	T0	T0	T0	T0	T0	T0	T0	T0	T0	INTTYPE0

Bit 31:0 **INTTYPE0**: Interrupt Type 0.

与 INTTYPE1 一起控制中断触发。使用 INTEN 首先启用中断。

4.2.6 INTTYPE1(Interrupt Type 1)

Reset Value:0x0000_0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
T1	T1	T1	T1	T1	T1	T1	T1	T1	T1	T1	T1	T1	T1	T1	T1	T1	T1	T1	T1	T1	T1	T1	T1	T1	T1		T1	T1	T1	T1	

INTTYPE1

Bit 31:0 **INTTYPE1**: Interrupt Type 1.

与 INTTYPE0 一起控制中断触发。使用 INTEN 首先启用中断。

4.2.7 INTSTATUS(Interrupt Status)

Reset Values:0x0000_0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	INTSTATUS

Bit 31:0 **INTSTATUS**:Interrupt Status.

包含每个 GPIO 的中断状态。读取状态寄存器时清零。类似地，中断线在中断状态下被设置为高电平，并且在状态寄存器被读取时解除。

4.2.8 PADCFG0-7(Pad Configuration Registers 0-7)

Reset Value:0x0000_0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	PADCFG0-7

Bit 31:0 **PADCFG0-7**: Pad Configuration Registers.

引脚配置寄存器控制通常在 ASIC 中使用的引脚的各个方面，例如，驱动强度，施密特触发器，摆率等。由于这些配置参数取决于所使用的确切的引脚，因此每个实现都可以以任何方式使用 PADCFG0-7 寄存器，

如果不需要，也可以使其不连接。

4.3 SPI Master

表 4-3 SPI Signals

Signal	Direction	Description
spi_clk	output	Master Clock
spi_csn0	output	Chip Select 0
spi_csn1	output	Chip Select 1
spi_csn2	output	Chip Select 2
spi_csn3	output	Chip Select 3
spi_mode[1:0]	output	SPI Mode
spi_sdo0	output	Output Line 0
spi_sdo1	output	Output Line 1
spi_sdo2	output	Output Line 2
spi_sdo3	output	Output Line 3
spi_sdi0	input	Input Line 0
spi_sdi1	input	Input Line 1
spi_sdi2	input	Input Line 2
spi_sdi3	input	Input Line 3
events_o[1:0]	output	Event/Interrupt

4.3.1 STATUS(Status Register)

1. Status (Status Register)

Offset: 0x00

Reset Value: 0x0000_0000

Write define as:

31	11	8	5	4	3	2	1	0
0	CS	0	SWRST	QWR	QRD	WR	RD	

Bit 11:8 CS: Chip Select.

指定下一次传输应使用的芯片选择信号。

Bit 4 SRST: Software Reset.

清除 FIFO 并中断传输

Bit 3 QWR: Quad Write Command.

在 Quad SPI 模式下执行写操作.

Bit 2 QRD: Quad Read Command.

在 Quad SPI 模式下执行读操作.

Bit 1 WR: Write Command.

在 standard SPI 模式下执行写操作.

Bit 0 RD: Read Command.

在 standard SPI 模式下执行读操作.

Read define as:

31	28	24	19	16	6	0
0	Tx_fifo	0	Rx_fifo	0	ctrl_statu	

Bit 6:0 ctrl_statu: spi 模块处理的任务.

ctrl_statu bit map table

ctrl_statu[0]	IDLE
ctrl_statu[1]	CMD
ctrl_statu[2]	ADDR
ctrl_statu[3]	MODE
ctrl_statu[4]	DUMMY
ctrl_statu[5]	DATA_TX
ctrl_statu[6]	DATA_RX

Bit 19:6 Rx_fifo: the elements contain in rx fifo.

Bit 28:24 Tx_fifo: the elements contain in tx fifo.

2. CLKDIV(Clock Divider)

Offset: 0x04

Reset Value: 0x0000_0000

31	7	0
	CLKDIV	

Bit 7:0: CLKDIV: Clock Divider.

时钟分频器值用于分割 SPI 传输的 SoC 时钟。这个寄存器在转移过程中不应该被修改。

The frequency of SPI = (APB_bus_clock) / (2 * (CLKDIV + 1)).

3. SPICMD (SPI Command)

Offset: 0x08

Reset Value: 0x0000_0000



Bit 31:0 SPICMD: SPI Command.

当执行读或写操作时，首先发送 SPI commend。SPI commend 的长度可以由 SPILEN 寄存器控制。

4. SPIADR (SPI Address)

Offset: 0x0C

Reset Value: 0x0000_0000



Bit 31:0 SPIADR: SPI Address.

当执行读或写操作时，首先发送 SPI commend, 之后发送 SPI address。SPI address 的长度可以由 SPILEN 寄存器控制。

5. SPILEN (SPI Transfer Length)

Offset: 0x10

Reset Value: 0x0000_0000



Bit 31:16 DATALEN: SPI Data Length.

读或写的位数，注意首先把 SPI 命令和地址写入到 SPI slave。

Bit 13:8 ADDRLEN: SPI Address Length.

SPI 地址应该被发送的位数。

Bit 5:0 CMDLEN: SPI Command Length.

SPI 命令应该被发送的位数。

6. SPIDUM (SPI Dummy Cycles)

Offset: 0x14

31	0
DUMMYWR	DUMMYRD

Dummy cycles(不读不写)在发送 SPI 命令+地址和写数据之间。

Dummy cycles(不读不写)在发送 SPI 命令+地址和读数据之间。

Offset: 0x18

将数据写入 FIFO。FIFO 是 32 位宽和 8 位深度的，写这个地址会把数据压入 FIFO。

Offset: 0x20

从 FIFO 读出数据。FIFO 是 32 位宽和 8 位深度的，写这个地址会从 FIFO 取出数据。

Offset: 0x24

31	30	26	24	18	16	10	8	2	0
EN	CNTEN	0	CNTRX	0	CNTTX	0	THR	0	THTX

Enable interrupts.

Enable Count.

期望在一次中断中接收的数据的位数。这个计数应该与从 FIFO 中取出的数据的位数相同，这些软件

在中断服务功能中执行。该设置使得每次读取操作之后都可以发生中断信号。

Bit 18:16 CNTTX

期望中断的数据的位数。这个计数应该与装入 FIFO 的数据位数相同，这些软件在中断服务功能中执行。该设置使得每次写操作之后都可以发生中断信号。

Bit 10:8 THRX: Threshold Rx

如果 Rx FIFO 中的数据超出此阈值，则会立即发生一次 event_o [0]。只有在 INTSTA 擦除或 RX 计数被访问后，Rx FIFO 中的数据再次超过这个门限时，这个中断才会再次发生。如果大量数据传输，通信的方向应取决于状态寄存器。

Bit 2:0 THTX: Threshold Tx

如果 Tx FIFO 中的数据低于此阈值，则会立即发生一次 event_o [0]。只有在 INTSTA 擦除或 TX 计数达到后，Tx FIFO 中的数据再次超过此阈值，才会再次发生此中断。如果大量数据传输，通信的方向应取决于状态寄存器。

Events_o [0]在 INTCFG 寄存器中的 TX FIFO / RX FIFO 的元件低于/超过 THTX / THRX 之后发生。

Events_o [1]表示传输结束。只有在 Dummy, Data_tx, Data_rx 状态之后才会发生。Cmd 或地址发送不会触发这个信号。该中断一直处于打开状态，不受 INTCFG 寄存器的 EN 位控制。在这个中断之后，程序需要通过获取 STATUS 寄存器中的 Rx_fifo 来读取所有缓冲的数据。

10. INTSTA

Offset: 0x28

Reset Value: 0x0000_0000

该寄存器只能写入。访问该寄存器将刷新 Event0 状态。

4.4 I2C

I2c 可以通过软件设置使输出在高低之间自由转换，I2C 在标准模式下的传输速率可以达到 100Khz。I2C 在快速模式下可以达到 400khz。

表 4-4 I2C Signals

Signal	Direction	Description
scl_pad_i	input	SCL Input
scl_pad_o	output	SCL Output(always 0)
scl_padoen_o	output	SCL Pad Direction

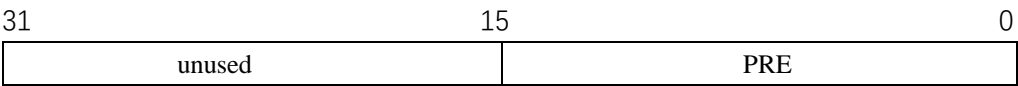
sda_pad_i	input	SDA Input
sda_pad_o	output	SDA Output(always 0)
sda_padoen_o	output	SDA Pad Direction
Interrupt_o	output	Event/Interrupt

4.4.1 CPR (Clock Prescale Register)

Offset: 0x00

Reset Value: 0x0000_0000

Write define as:



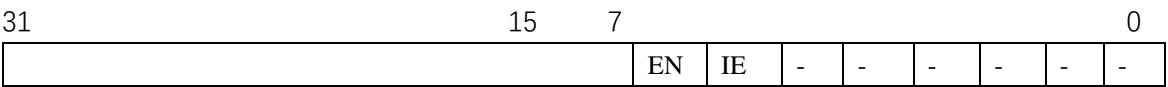
Bit 15:0 PRE: Prescaler.

通过 PRE 中的值设置时钟预分频器，以通过分频实现所需的 I2c 时钟，当前系统时钟用户设置。

4.4.2 CTRL(Control Register)

Offset: 0x04

Reset Value: 0x0000_0000



Bit 7 EN: Enable.

Enable the I2c peripheral.

Bit 6 IE: Interrupt enable.

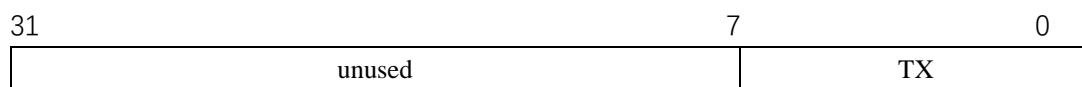
Enable interrupts.

Bit 5:0 Reserved: Set to 0.

4.4.3 TX(Transmit Register)

Offset: 0x08

Reset Value: 0x0000_0000

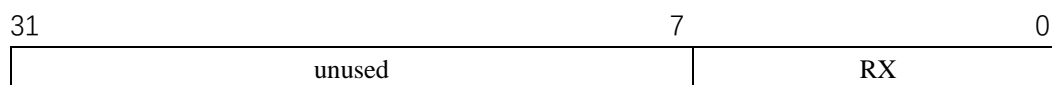


Bit 7:0 TX: Transmit Register.

4.4.4 RX (Receive Register)

Offset: 0x0C

Reset Value: 0x0000_0000

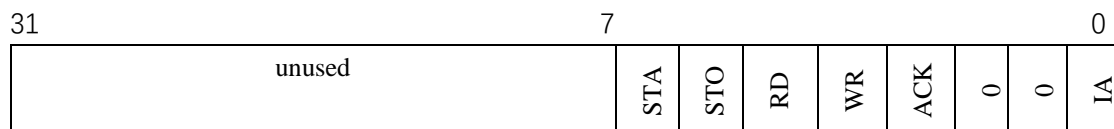


Bit 7:0 RX: Receive Register.

4.4.5 CMD (Command Register)

Offset: 0x10

Reset Value: 0x0000_0000



Bit 7 STA: Send start bit.

Bit 6 STO: Send stop bit.

Bit 5 RD: Read from bus.

Bit 4 WR: Write to bus.

Bit 3 ACK: Acknowledge received data.

Bit 2:1 Reserved: Set to 0.

Bit 0 IA:Interrupt Acknowldge.

设置为 1 来确认中断。传输完成或仲裁失败时清除。

4.4.6 STATUS (Status Register)

Offset: 0x14

Reset Value: 0x0000_0000

31	7							0	
unused		RX	BUS	AL	0	0	0	TIP	IRQ

Bit 7 RXA: Acknowledge from sent data.

Bit 6 BUS: Bus is busy.

Bit 5 AL: Arbitration lost.

Bit 4:2 Reserved: Set to 0.

Bit 1 TIP: Transfer in progress.

Bit 0 IRQ: Interrupt received.

传输结束或总线消失时总是设置该标志，不管是否启用中断。这个标志可能轮询或者通过向 IA 命令寄存器写入 1 来清除。

4.5 Timer

定时器单元每个默认有两个定时器。在实例化定时器时，可以被参数覆盖。

4.5.1 TIMER(Current Timer Value)

Reset Value: 0x0000_0000

31		15		0
TIMER				

Bit 31:0 **TIMER**: Current Timer Value.

定时器的当前值。每个定时器都有一个内部预分频器，用于指定定时器将增加的时间间隔。预分频器由 CTRL 控制。当 TIMER 达到 FFFF_FFFF 时，会产生一个中断。

4.5.2 CTRL(Timer Control)

Reset Value: 0x0000_0000

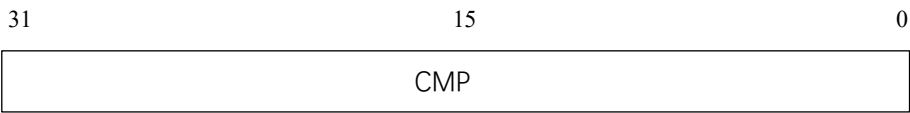
31		15		0
unused		PRE	0	EN

Bit 5:3 PRE:Prescaler value.

Bit 0 EN: Enable the timer.

4.5.3 CMP(Timer Compare)

Reset Value:0x0000_0000



Bit 31:0 CMP:Timer Compare.

当 Timer 到达 CMP 时一个中断将被触发。

4.6 Event Unit

PPU 具有轻量级的事件和中断单元，支持多达 32 行的向量化中断和多达 32 条输入行的事件触发。中断和事件行分别被屏蔽和缓冲，见图 5.1。

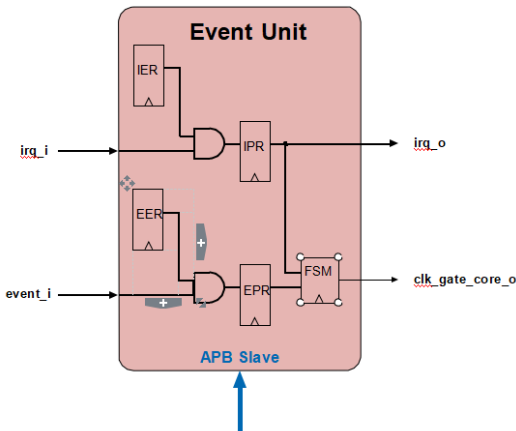


图 4.1 Event Unit.

事件和中断线的当前分配如图 5.2 所示。注意，irq_i 和 event_i 绑定在一起。

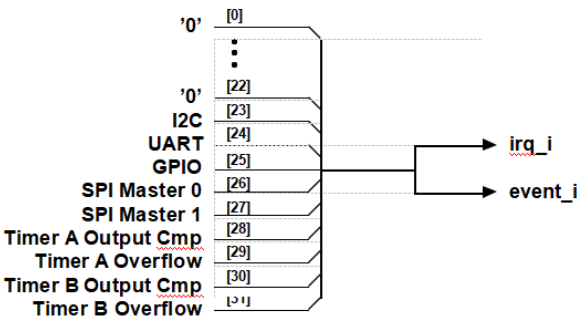


图 4.2 Event Lines.

4.6.1 IER(Interrupt Enable)

Reset Value:0x0000_0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	IER

Bit 31:0 IER:Interrupt Enable.

使能中断的每个线

4.6.2 IPR(Interrupt Pending)

Reset Value:0x0000_0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	IPR

Bit 31:0 IPR:Interrupt Pending.

对中断的每个线读或写

4.6.3 ISP(Interrupt Set Pending)

Reset Value:0x0000_0000

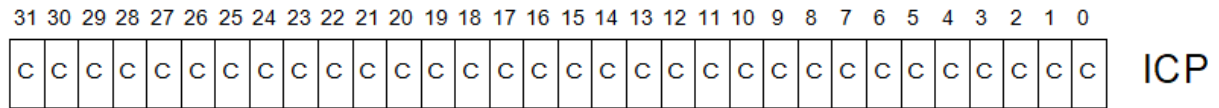
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	ISP

Bit 31:0 ISP: Interrupt Set Pending.

设置中断寄存器的每个线。通过设置这个位，一个中断将被触发在已经选择的线上

4.6.4 ICP(Interrupt Clear Pending)

Reset Value:0x0000_0000

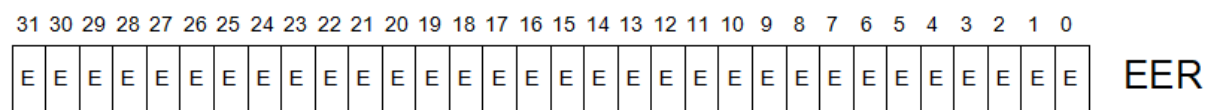


Bit 31:0 ICP:Interrupt Clear Pending.

清除正在发生的中断，通过设置这个位，一个正在发生的中断将被清除。

4.6.5 EER(Event Enable)

Reset Value:0x0000_0000

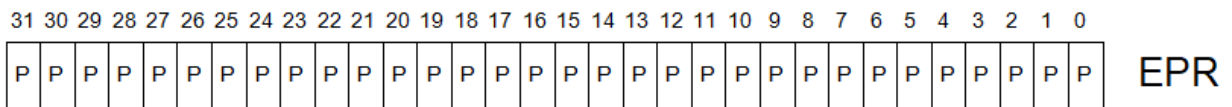


Bit 31:0 EER:Event Enable.

对事件的每个线使能

4.6.6 EPR(Event Pending)

Reset Value:0x0000_0000

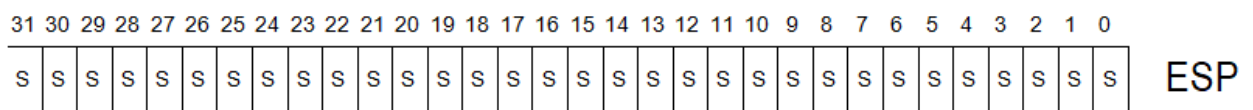


Bit 31:0 EPR:Event Pending.

对正在进行事件的每个线读和写

4.6.7 ESP(Event Set Pending)

Reset Value:0x0000_0000

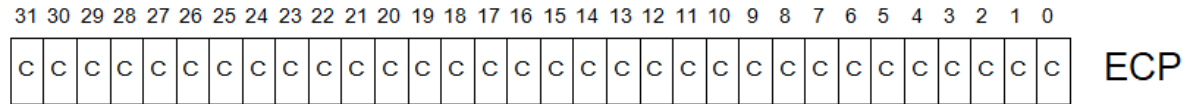


Bit 31:0 ESP:Event Set Pending.

设置正在发生事件寄存器的每个线，通过设置这个位，一个事件将被设置在选中的线上。

4.6.8 ECP(Event Clear Pending)

Reset Value:0x0000_0000

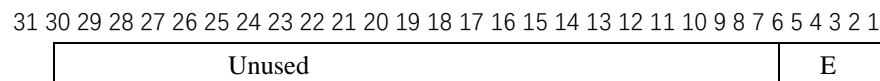


Bit 31:0 ECP:Event Clear Pending.

清除正在发生的事件，通过设置这个位，一个正在发生的事件将被清除。

4.6.9 SCR(Sleep Control)

Reset Value:0x0000_0000

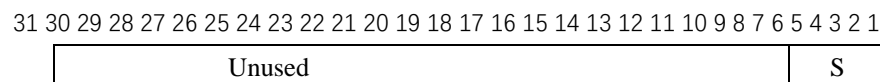


Bit 0 E:Sleep Enabled.

让核进入睡眠状态。当发生中断或事件时，核将再次唤醒。

4.6.10 SSR(Sleep Status)

Reset Value:0x0000_0000



Bit 0 S:Sleep Status.

设置内核如果它在睡眠状态或者有时钟使能

4.7 SoC Control

PPU 提供了一个小而简单的 APB 外围设备，提供有关该平台的信息，并提供在 ASIC 上进行多路复用的方法。

下列寄存器可以被访问。

4.7.1 PAD Mux

Reset Value:0x0000_0000

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

PADMUX																													
--------	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

Bit 31:0 PADMUX:当目标是 ASIC 时，这寄存器的内容通常是用做引脚复用。

4.7.2 CLK Gate

Reset Value:0

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

Unused																												E
--------	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	---

Bit 31:0 CLK GATE: 该寄存器包含用于时钟选通内核的时钟选通使能信号（E）的值。它的值为高。

4.7.3 Boot Address

Reset Value:0x0000_8000

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

Boot Address																													
--------------	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

Bit 31:0 Boot Address: 该寄存器保存启动地址。可以从 ROM 或直接从指令存储器启动。

4.7.4 Info

Reset Value:0

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Unused	D	I	Rom Size	Inst Ram Size	Data Ram Size	Version
--------	---	---	----------	---------------	---------------	---------

Bit 31:0 Info Register:

该寄存器保存有关 PPU 体系结构的信息。Version 包含 ppu 版本。标志 D 和 I 报告当前是否有数据/指令缓存。RomSize 定义 Boot ROM 的大小。最后，Inst.RamSize 和数据 RamSize 定义 RAM 的大小按 8KB 的倍数。

4.7.5 Status

Reset Value:0

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

Unused	S
--------	---

Bit 31:0 Status Register: 状态寄存器位 S 可用于保存测试的最终结果以进行验证。

4.7.6 PAD Configuration

Reset Value:0x0000_0000

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

PAD Configurartion

Bit 31:0 PAD CFG0-7 这八个寄存器可以用作 ASIC 目标来配置寄存器，例如拉低或拉高这个值。

4.8 Debug Port

该块包含一个 apb2per 桥，允许从 APB 命令转换到调试总线。调试总线直接将内核的调试单元连接到 APB 总线，并允许进行内存映射调试。由于所有内核寄存器都是内存映射的，因此可以使用内存接口调试内核，而不是使用高度专业化的接口。

下面是对 debug 信号的描述:

表 4-5 Debug Bus Signals

Signal	Direction	Description
Dbg_req	output	Request signal
Dbg_addr	output	Address
Dbg_we	output	Write enable
Dbg_wdata[31:0]	output	Data to write
Dbg_gnt	input	Grant
Dbg_rvalid	input	Response valid
Dbg_rdata[31:0]	input	Read data

该协议与核心数据和指令接口非常相似。要读取或写入，主机必须通过地址，写入使能和要写入的数据来提出请求。一旦 Slave 将请求信号设置为 1，请求就被执行。在写操作的情况下，有效信号上升到 1 以通知 master 传输成功。在读取操作的情况下，从机在 rdata 行上返回所请求的数据，恰好是在授权之后的一

个周期。

4.9 Pulse Width Modulation(PWM)

利用 PWM 技术可以对外围模拟电路进行调控。PWM 可以生成不同的时钟宽度和不同时钟占空比的时钟信号。

5 高级调试单元

高级调试单元有一个 AXI 主接口来访问外设和存储器。调试单元不再具有专门的调试接口来读取所有内核寄存器。

所有内核寄存器都是内存映射的，这意味着它们可以通过 AXI 接口读取。因此，调试不仅可以通过 JTAG 进行，还可以通过 SPI 或任何其他接口进行。

JTAG 信号链接到了 SoC 的引脚，详细信息可以参考高级调试单元相关文档。

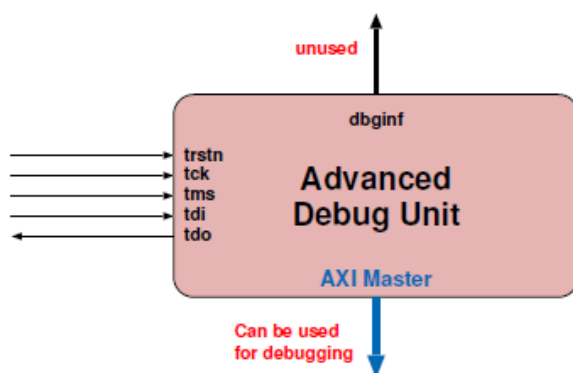


图 5.1 调试单元框图

6 SPI slave

SPIslave 设备是一个有效的外设，它在没有内核的帮助下接收/发送数据。它的目的是作为一个外部接口，系统用户可以从外部访问内部存储器。这种机制可以用来预先将程序加载到存储器中，启动系统，等待程序已经终止的确认，然后检查结果。

SPIslave 有一个 AXI 主机，通过它可以访问所有的外设和存储器。从外部可以通过发送 SPI / QSPI 命令来访问它。QSPI 是 SPI 的扩展，它使用四条数据通道而不是一条，因此是标准 SPI 速度的四倍。

如果 ASIC 上的引脚很少，那么可以将标准 SPI 接口连接到引脚，并接受到 SoC 传输中与 QSPI 模式相比的性能损失。信号 spi_sdi1-3 和 spi_sdo1-3 专门用于 QSPI 模式。

表 6-1 SPI SLAVE 信号表

Signal	Direction	Description
spi_clk	output	Slave Clock
spi_cs	input	Chip Select
spi_mode[1:0]	output	SPI Mode
spi_sdo0	output	Output Line 0
spi_sdo1	output	Output Line 1
spi_sdo2	output	Output Line 2
spi_sdo3	output	Output Line 3
spi_sdi0	input	Input Line 0
spi_sdi1	input	Input Line 1
spi_sdi2	input	Input Line 2
spi_sdi3	input	Input Line 3

7 摄像头接口

7.1 概述

Camera Bridge 提供 AHB-Master 和 AHB-Slave 两套接口。AHB-Master 用于发起对存储器的访问，且根据 Camera 的应用场景，只会发起对存储器的写操作。AHB-Slave 用于 CPU 对 Camera Bridge 的访问，根据 Camera 的应用场景，主要是软硬件交互流程。

AHB-Master 接口属性：

- 支持 Early Burst Termination ；
- 32 位地址位宽，32 位数据位宽；
- 根据 AHB 总线系统的实际需求（memctl 只返回 OKAY 响应），只支持 OKAY 响应。

AHB-Slave接口属性：

- 32位地址位宽，32位数据位宽；
- 只生成OKAY响应，不产生Retry、Split和Error响应；
- 寄存器访问都是32位操作（寄存器按32位进行编址），默认HSIZE为32位操作，且默认最低两位地址为零，即32位对界。

7.2 接口时序

所有满足接口时序的摄像头都可以接入该 Camera Bridge，典型的接口时序如图 7.1 所示：

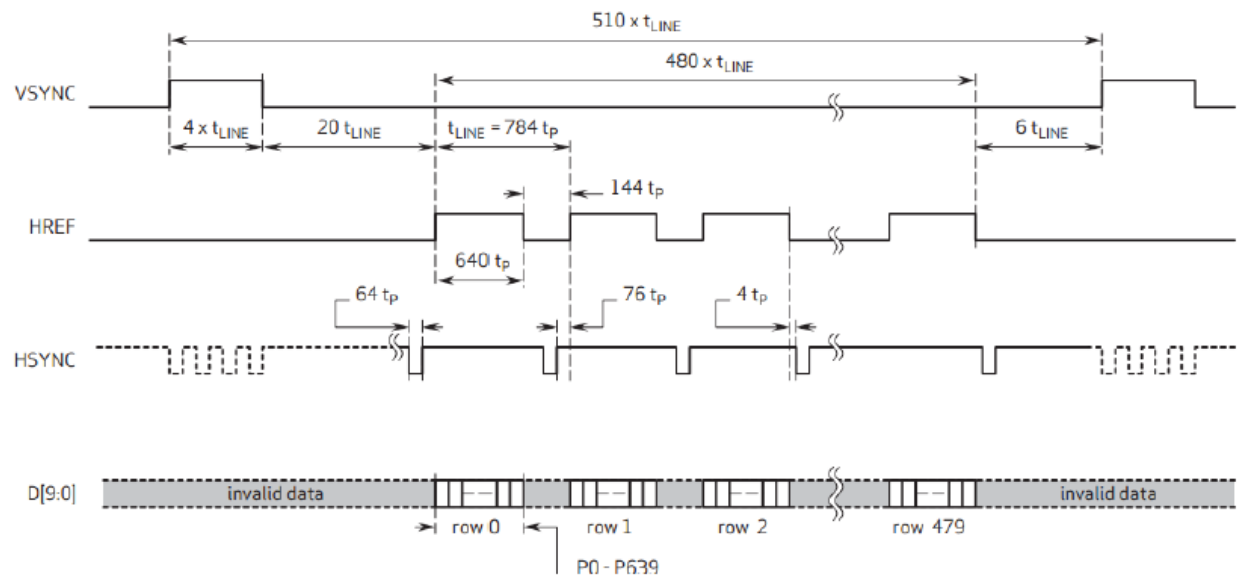


图 7.1 典型的摄像头接口时序

说明：

- 该图中每帧为 640*480 个像素(VGA)，每个像素对应两个字节。
- 对于 YUV/RGB，tp 表示两个 pclk 时钟。

7.3 内部结构

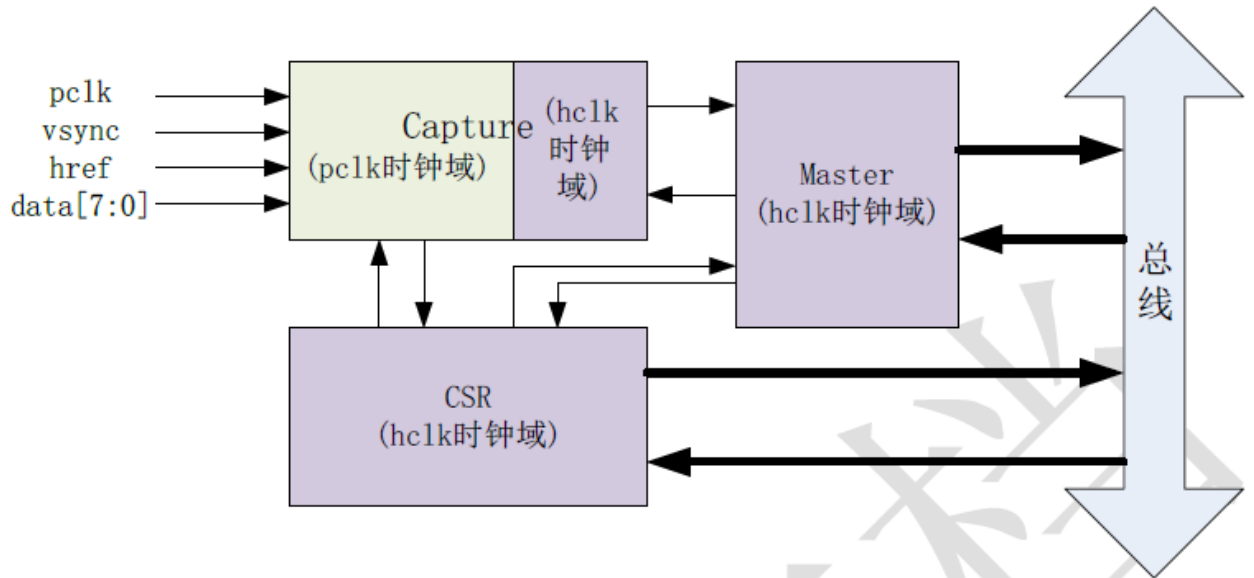


图 7.2 Camera Bridge 结构示意图

说明：

- CSR 模块是 CameraIP 的 AHB Slave 接口，主要用于操作 CSR 寄存器（第三章）；
- Capture 模块是 CameraIP 的数据接收模块，主要用于接收来自 OV7725 的采集数据，采集数据经过 pclk 和 hclk 异步交接后通过 Master 接口发向 AHB 总线；
- Master 模块是 CameraIP 的 AHB Maser 接口，主要用于把采集数据转换内部 AHB 协议写存储器。

关于该部分的详细内容，请参考 Camera Bradge 文档。

8 EMMC 接口

8.1 主要功能

eMMC 接口可用于连接 SD (eSD) 、SDIO (eSDIO) 、CE-ATA、MMC、eMMC。所有的数据交换都由 Host 端的 eMMC 控制器发起。eMMC 接口具有如下特点：

- 接口支持 SD3.01 标准，SDIO3.01 标准，CE-ATA1.1 标准，MMC4.4.1 标准，eMMC4.5.1 标准；
- 支持 SDR 标准传输模式（最高传输率 25MB/s）、SDR 高速传输模式（最高传输率 50MB/s）与 DDR 高速传输模式（最高传输率 100MB/s）；
- 与 MMC (eMMC) 的读写数据位宽可配置为 1bit/4bit/8bit；与 SD/SDIO 的读写数据位宽可配置为 1bit/4bit；
- 集成 DMA 引擎。

8.2 eMMC 接口总体结构

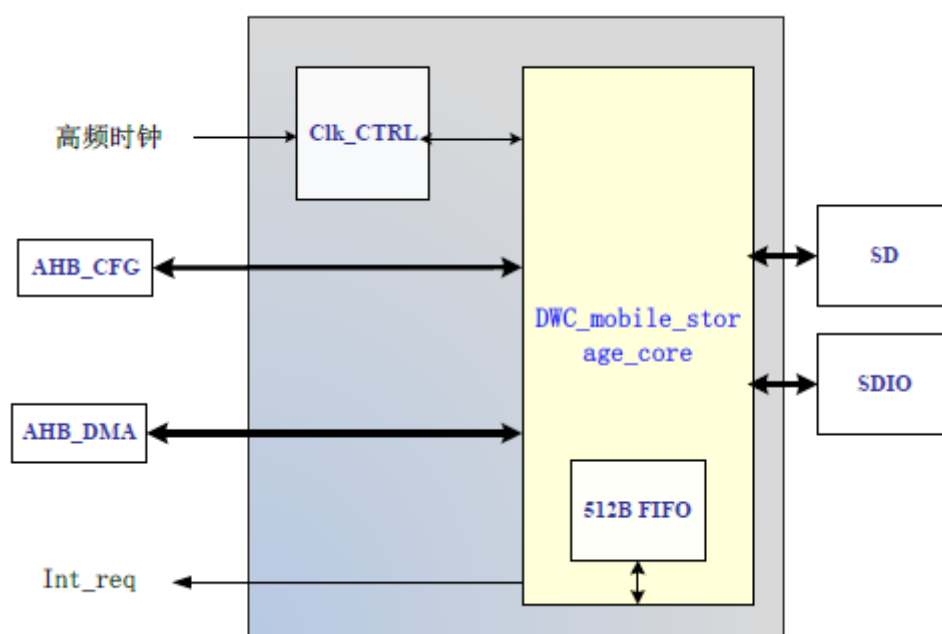


图 8.1 eMMC 接口结构示意图

其中 Clk_CTRL 用于时钟生成和相位控制。

关于该部分的详细内容，请参考 eMMC 文档。

9 内存控制器

9.1 概述

DW_memctl 是连接外部存储设备的接口模块。其作为 AHB 总线上的 Slave，主要功能是将来自 AHB 总线上的读写请求转化成对存储设备的访问，并向 AHB 总线返回响应。DW_memctl 模块直接使用 Synopsys 公司的 IP，该 IP 支持 SDRAM 与 static memory 2 种存储设备。DW_memctl 仅使用其中部分功能，主要特性有：

- 1) 支持 32 位宽 AHB 数据、地址总线；
- 2) 支持最大 4GB 的存储器访问空间；
- 3) 存储设备地址宽度、访问时序可配置；
- 4) 支持 Mobile DDR-SDRAM；
- 5) 支持 1 个片选信号。

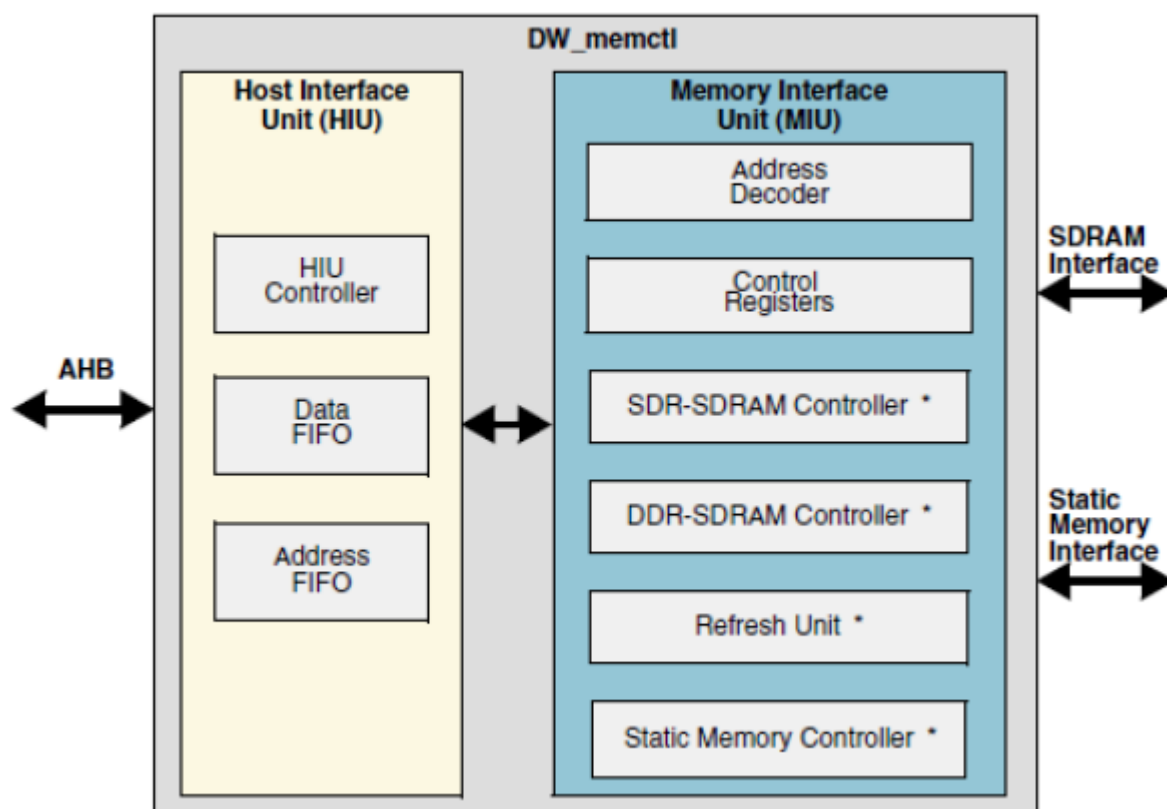


图 9.1 DM_memctl 框图

DM_memctl 主要包括以下部分：HIU(Host Interface Unit):主接口部件，区分 AHB 请求是对内存空间还

是控制寄存器的访问，并生成相应命令转发至 MIU，同时返回相应响应至 AHB 总线（HIU 不区分 SDRAM、SRAM/FLASH 请求）。包含以下子模块：

HIU Controller：生成其他子模块的控制信号；

Data FIFO：读写数据缓冲模块；

Address FIFO：地址缓冲模块。

MIU(Memory Interface Unit)：内存接口部件，主要生成正确的地址、数据和控制信号给存储设备。包含以下子模块：

Address Decoder：将 HIU 提供的地址翻译成存储器可识别的地址；

Control Register：IP 的各种寄存器，包括控制寄存器、配置或时序寄存器；

DDR-SDRAM Controller:生成 DDR-SDRAM 控制信号；

Refresh Unit：按一定间隔生成 SDRAM 的刷新命令；

该模块的详细信息请参考 memctl 手册。

10 引脚定义

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	VSS	VSS	MEMCTL_S_CAS_N	VSS	MEMCTL_S_SEL_N	VSS	MEMCTL_S_ADDR[1]	MEMCTL_S_ADDR[4]	MEMCTL_S_ADDR[5]	MEMCTL_S_ADDR[10]	MEMCTL_S_ADDR[11]	MEMCTL_S_BANK_ADDR[0]	MEMCTL_S_BANK_ADDR[1]	MEMCTL_S_ADDR[12]	MEMCTL_S_ADDR[15]	VSS
B	VSS	DIV_PLL_I[0]	MEMCTL_S_RAS_N	MEMCTL_S_WE_N	MEMCTL_S_CKE	MEMCTL_S_CK_N	MEMCTL_S_CK_P	VSS	MEMCTL_S_ADDR[8]	VSS	MEMCTL_S_ADDR[13]	VSS	MEMCTL_S_ADDR[9]	VSS	MEMCTL_S_DQ[0]	MEMCTL_S_DQ[1]
C	PLL_BPS_I	DIV_PLL_I[1]	CLK_I	VSS	MEMCTL_S_RD_DQS_MASK	MEMCTL_S_RD_DQS_MASK	MEMCTL_S_ADDR[0]	MEMCTL_S_ADDR[2]	MEMCTL_S_ADDR[3]	MEMCTL_S_ADDR[6]	MEMCTL_S_ADDR[7]	VREF	MEMCTL_S_ADDR[14]	MEMCTL_S_DQM[0]	MEMCTL_S_DQS[0]	MEMCTL_S_DQ[2]
D	VSS	VSS	VSSQ_MEM	VSSQ_MEM	VDD	VSS	VDD	VSS	VDD	VSS	VDD	VSS	VDD	MEMCTL_S_DQ[3]	VSS	MEMCTL_S_DQ[5]
E	AVDD_PLL_0	AVDD_PLL_0	AVSS_PLL_0	AVSS_PLL_0	VSSQ_MEM	VDD18	VSSQ_MEM	VDD18	VSSQ_MEM	VDD18	VSSQ_MEM	VDD18	VSS	MEMCTL_S_DQ[4]	MEMCTL_S_DQ[6]	MEMCTL_S_DQ[1]
F	AVDD_PLL	AVDD_PLL	AVSS_PLL	AVSS_PLL	VDD18	VSS	VSS	VSS	VSS	VSS	VSS	VSSQ_MEM	VDD	MEMCTL_S_DQ[7]	VSS	MEMCTL_S_DQM[1]
G	VSS	VSS	VSS	VSS	VSSD33	VSS	VDD	VDD	VDD	VDD	VSS	VDD18	VSS	MEMCTL_S_DQ[9]	VREF	MEMCTL_S_DQ[8]
H	FETCH_ENABLE_I	TESTMODE_I	RSTN_I	VSS	VDD33	VSS	VDD	VSS	VSS	VDD	VSS	VSSQ_MEM	VDD	MEMCTL_S_DQ[11]	VSS	MEMCTL_S_DQ[10]
J	UART_TX	VSS	SPI_MASTER_CLK_0	VDD	VSSD33	VSS	VDD	VSS	VSS	VDD	VSS	VDD18	VSS	MEMCTL_S_DQ[14]	MEMCTL_S_DQ[13]	MEMCTL_S_DQ[12]
K	UART_RX	SPI_MASTER_CSNO_0	SPI_MASTER_SD00_0	VSS	VDD33	VSS	VDD	VDD	VDD	VDD	VSS	VSSD33	VDD	EMMC_CDATA[0]	VSS	MEMCTL_S_DQ[15]
L	SPI_MASTER_SDIO_I	VSS	SELECT_C0_I	VDD	VSSD33	VSS	VSS	VSS	VSS	VSS	VSS	VDD33	VSS	EMMC_CARD_DETECT_N	EMMC_CDATA[3]	EMMC_CDATA[1]
M	SPI_CLK_I	SPI_SD00_0	SPI_SD10_I	VSS	VDD33	VSSD33	VDD33	VSSD33	VDD33	VSSD33	VDD33	VSSD33	VDD	EMMC_CARD_WRITE_PRT	VSS	EMMC_CDATA[2]
N	SPI_CS_I	VSS	TMS_I	VDD	VSS	VDD	VSS	VDD	VSS	VDD	VSS	VDD	VSS	PIN11_GPIO4_CAMD_ATA2	SDIO_CARD_INT_N	EMMC_CCM_D
P	TRSTN_I	TDI_I	VSS	TCK_I	CO_UART_TX	CO_PIN0_SPIMCLK_SCL	CO_PIN3_SPI_MSDI0_GPIO1	CO_PIN6_PWM2_GPIO4	SPIM1CLK	PIN4_SPI_M1SDIO_V_SYNC	PIN7_PWM2_CAMDAT_A6	PIN8_PWM3_CAMDAT_A5	PCLK	VSS	MMC_4_4_RST_N	EMMC_CCLK_OUT
R	VSS	TD0_0	CO_TESTMODE_I	CO_FETCH_ENABLE_I	CO_UART_RX	VSS	CO_PIN4_PWM0_GPIO2	VSS	PIN1_SDA_UART1RX	VSS	PIN5_PWM0_HREF	VSS	PIN9_GPIO2_CAMDATA4	PIN13_GPIO6_CAMDATA0	PIN12_GPIO5_CAMDATA1	VSS
T	VSS	VSS	CO_SPI_CS_I	VSS	CO_PIN1_SPIMCSN_SDA	CO_PIN2_SPIMSDO_GPIO0	CO_PIN5_PWM1_GPIO3	CO_PIN7_PWM3_GPIO5	PIN0_SCL_UART1TX	PIN2_SPI_M1SDIO_GPIO0	PIN3_SPI_M1CSNO_GPIO1	PIN6_PWM1_CAMDAT_A7	VSS	PIN10_GPIO3_CAMDATA3	VSS	VSS

图 10.1 引脚定义

11 存储映射

表 11-1 系统地址分配

0x0000_0000	64KB
0x0000_FFFF	(i-sram)
0x0001_0000	512B
	(boot_rom)
0x000F_FFFF	
0x0FFF_FFFF	
0x2000_0000	64kB
	(d-sram)
0x2000_FFFF	0x2001_0000
	0x21FF_FFFF
	0x2200_0000
	0x23FF_FFFF
SDRAM	
(128MB-64KB)	
	0x2400_0000
	0x25FF_FFFF
	0x2600_0000
	0x27FF_FFFF
0x2800_0000	memctrl
0x2800_1000	cam_if
0x2800_2000	aNN
0x2800_3000	sdio
0x2800_3FFF	4KB each
0x2800_FFFF	

0x1000_0000	4KB each
0x1A10_0000	Uart0
0x1A10_1000	gpio0
0x1A10_2000	Spi_master0
0x1A10_3000	timer
0x1A10_4000	event
0x1A10_5000	I2c0
0x1A10_6000	Soc_ctrl
0x1A10_7000	Uart1
0x1A10_8000	Spi_master1
0x1A10_9000	I2c1
0x1A11_0000	
0x1A11_7FFF	debug
0x1A11_8000	pwm
0x1A11_8FFF	