Silicon Integrated

蜂鸟 E200 系列 Core 协处理器扩展

快速使用说明

Content

1 Preface

1.1 Revision History

Date	Version	Author	Change Summary
Sep 29, 2017	0.1	Bob Hu	Initial version

2 RISC-V 架构的可扩展性

RISC-V 架构的一个显著特性便是其开放的可扩展性，从而能够非常容易的在 RISC-V 的通用架构基础上实现 Domain Specific Accelerator，这也被誉为 RISC-V 架构相比 ARM 和 x86 等主流的商业架构的最大优点。RISC-V 的可扩展性体现在两个方面：

- 预留的指令编码空间
- 预定义的 Custom 指令

2.1 RISC-V 的预留指令编码空间

RISC-V 架构定义的标准指令集仅仅使用了少部分的指令编码空间，更多的指令编码空间被预留给用户作为扩展指令使用。由于 RISC-V 架构支持多种不同的指令长度，不同的指令长度均预留有不同的编码空间。以最常用的 32 比特和 16 比特长度指令为例，如表 2-1 所示，指令的低 7 位为 opcode，各种不同的 opcode 值的组合代表了不同的指令类型，譬如 AMO 指令，OP-FP（浮点）指令。用户可以从三个方面利用 RISC-V 预留的编码空间：

- 每个指令类型拥有剩余的 25 位的编码空间，其中除了用于寄存器操作数的索引之外，还剩余众多比特的编码空间，对于这些没有使用的编码空间，用户均可以加以利用。
- 另外对于某些特定的处理器实现，往往不会实现所有的指令类型，对于没有实现的指令类型的编码空间，用户也可以加以利用。
- 有一些没有定义的指令类型组，用户也可以加以利用。

表 2-1 RISC-V 架构 32 比特指令 opcode 表（inst[1:0]==11）

inst[4:2] inst[6:5]	000	001	010	011	100	101	110	111 ($> 32b$)
00	LOAD	LOAD-FP	*custom-0*	MISC-MEM	OP-IMM	AUIPC	OP-IMM-32	$48b$
01	STORE	STORE-FP	*custom-1*	AMO	OP	LUI	OP-32	$64b$
10	MADD	MSUB	NMSUB	NMADD	OP-FP	*reserved*	*custom-2/rv128*	$48b$
11	BRANCH	JALR	*reserved*	JAL	SYSTEM	*reserved*	*custom-3/rv128*	$\geq 80b$

表 2-2 RISC-V 架构 16 比特指令 opcode 表

inst[15:13] / inst[1:0]	000	001	010	011	100	101	110	111	
00	ADDI4SPN	FLD FLD LQ	LW	FLW LD LD	*Reserved*	FSD FSD SQ	SW	FSW SD SD	RV32 RV64 RV128
01	ADDI	JAL ADDIW ADDIW	LI	LUI/ADDI16SP	MISC-ALU	J	BEQZ	BNEZ	RV32 RV64 RV128
10	SLLI	FLDSP FLDSP LQ	LWSP	FLWSP LDSP LDSP	J[AL]R/MV/ADD	FSDSP FSDSP SQ	SWSP	FSWSP SDSP SDSP	RV32 RV64 RV128
11	>16b								

2.2 RISC-V 的预定义的 Custom 指令

为了便于用户对 RISC-V 进行扩展，RISC-V 架构甚至在 32 位长度的指令中预定义了四组 Custom 指令类型，每种 Custom 均有自己的 Opcode，如表 2-1 中所示，custom-0，custom-1，custom-2，custom-3 四条指令类型。用户可以利用这四种指令类型扩展成为自定义的协处理器指令。蜂鸟 E200 处理器核便使用 custom 指令扩展协处理器指令。

2.3 蜂鸟 E200 的协处理器接口 EAI

蜂鸟 E200 处理器的协处理器机制借鉴了开源 RISC-V 处理器 Rocket Core 的协处理器接口 RoCC（Rocket Custom Coprocessor），且接口信号定义非常类似，为了将其与原始的 RoCC 接口区分，命名为 EAI（E200 Extension Accelerator Interface），本节将以一实际案例来详细阐述如何使用 EAI 接口和 Custom 指令扩展蜂鸟 E200 协处理器。

2.4 EAI 指令的编码

32 位的 EAI 指令的编码格式如图 2-1 所示：

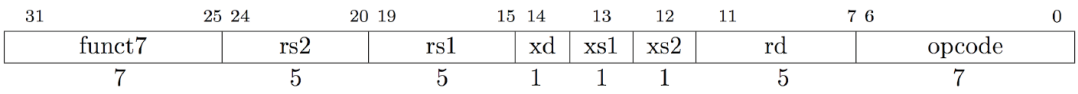

图 2-1 EAI 指令编码格式

- 指令的第 0 位至至第 6 位区间为 Opcode 编码段，根据表 2-1 中的编码规则可以编码 custom-0，custom-1，custom-2，和 custom-3 四种指令组。
- xs1，xs2，和 xd 比特分别用于控制是否需要读源寄存器和写目标寄存器 rd。如果 xs1 位的值为 1，则表示该指令需要读取由 rs1 比特位指示的通用寄存器作为源操作数 1，如果 xs1 位的值为 0，则表示该指令不需要源操作数 1；同理，如果 xs2 位的值为 1，则表示该指令需要读取由 rs2 比特位指示的通用寄存器作为源操作数 2，如果 xs2 位的值为 0，则表示该指令不需要源操作数 2；如果 xd 位的值为 1，则表示该指令需要写回结果至由 rd 比特位指示的目标寄存器，如果 xd 位的值为 0，则表示该指令需写回结果。
- 指令的第 25 位至第 31 位为 funct7 区间，可作为额外的编码空间，用于编码更多的指令，因此一种 custom 指令组可以使用 funct7 区间编码出 128 条指令，则四组 custom 指令共可以编码出 512 条两读一写（读取两个源寄存器，写回一个目标寄存器）协处理器指令。如果有的协处理器指令仅读取一个源寄存器，或者无需写回目标寄存器，则可以使用这些无用的比特位（譬如 rd 比特位）来编码出更多的协处理器指令。

2.5 EAI 接口信号

EAI 接口信号如表 2-3 所示。EAI 接口主要包含 4 个通道：

- 请求通道（Request Channel）
  Request Channel 主要用于主处理器在 EXU 级将指令信息和源操作数派发给协处理器。
- 返回通道（Response Channel）
  Response Channel 主要用于协处理器反馈主处理器告知其已经完成了该指令，并将结果写回主处理器。
- 存储器请求通道（Memory Request Channel）
  Memory Request Channel 主要用于协处理器向主处理器发起存储器读写请求。
- 存储器返回通道（Memory Response Channel）
  Memory Response Channel 主要用于主处理器向协处理器返回存储器读写结果。

表 2-3 EAI 接口信号

通道	方向	宽度	信号名	介绍
Request Channel	Output	1	eai_req_valid	主处理器向协处理器发送指令请求信号
	Input	1	eai_req_ready	协处理器向协处理器返回指令接受信号
	Output	32	eai_req_instr	该 Custom 指令的 32 比特完整编码
	Output	32	eai_req_rs1	源操作数 1 的值
	Output	32	eai_req_rs2	源操作数 2 的值
	Output	2	eai_req_itag	该指令的派发标号
Reponse Channel	Input	1	eai_rsp_valid	协处理器向主处理器发送反馈请求信号
	Output	1	eai_rsp_ready	主处理器向协处理器返回反馈接受信号
	Input	32	eai_rsp_wdat	返回计算结果的值
	Input	2	eai_rsp_itag	返回该指令的派发标号
	Input	1	eai_rsp_err	返回该指令的错误标志
Memory Request Channel	Input	1	eai_icb_cmd_valid	协处理器向主处理器发送存储器读写请求信号
	Output	1	eai_icb_cmd_ready	主处理器向协处理器返回存储器读写接受信号
	Input	32	eai_icb_cmd_addr	存储器读写地址
	Input	1	eai_icb_cmd_read	存储器读或是写的指示
	Input	32	eai_icb_cmd_wdata	写存储器的数据
	Input	4	eai_icb_cmd_wmask	写存储器的字节掩码
Memory Reponse Channel	Output	1	eai_icb_rsp_valid	主处理器向协处理器发送存储器读写反馈请求信号
	Input	1	eai_icb_rsp_ready	协处理器向协处理器返回存储器读写反馈接受信号
	Output	32	eai_icb_rsp_rdata	存储器读反馈的数据
	Output	1	eai_icb_rsp_err	存储器读写反馈的错误标志
Memory Holdup	Input	1	eai_mem_holdup	协处理器需要独占存储器访问通道的指示信号

2.6 EAI 流水线接口

基于 EAI 接口的协处理器在蜂鸟 E200 流水线中位置如图 2-2 所示。

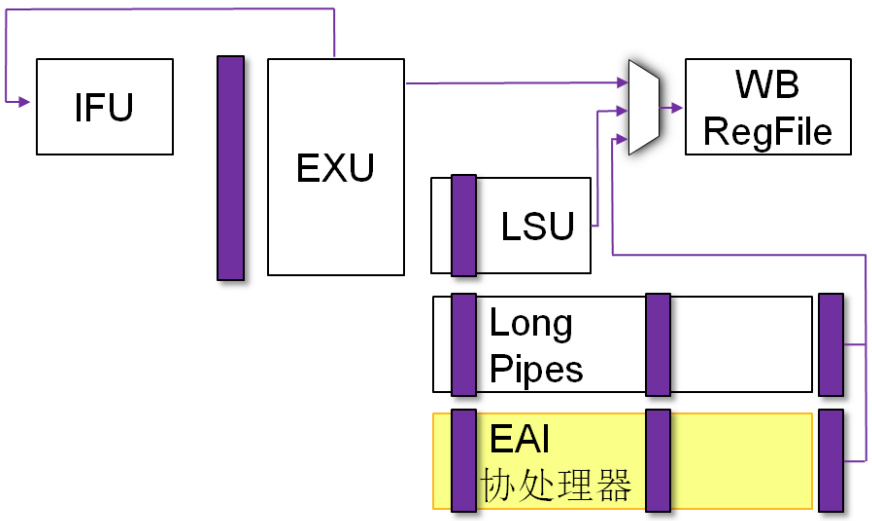

图 2-2EAI 接口协处理器在流水线中位置

蜂鸟 E200 作为一个两级 pipeline 的处理器，EXU 和 Commit 处于一个周期，均处于流水线的第二级。对于不在 EXU 写回结果的指令被称为是 Long-pipe Post Commit Write-Back 指令，简称 Long-Pipe 指令。整个 EAI 指令的执行过程如下：

- 主处理器的译码（Decode）单元在 EXU 级对指令的 Opcode 进行译码，判断其是否属于任意一种 Custom 指令组。
- 如果主处理器译码判断该指令属于 Custom 指令，则继续依据指令编码的 xs1 和 xs2 位判断是否需要读取源寄存器，如需要读取，则在 EXU 级读取寄存器文件（Regfile）读出源操作数。
- 主处理器会维护数据依赖的正确性，譬如如果该指令需要读取的源寄存器与之前正在执行的某条指令存在着先写后读（RAW）的依赖性，则处理器流水线会暂停直至该 RAW 解除。另外主处理器会依据指令编码的 xd 位判断该 Custom 指令是否需要写回结果至寄存器文件，如需要写回，则会将目标寄存器的索引信息存储在主处理器的流水线控制模块中直至写回完成，以便供后续的指令进行数据依赖性的判断。

- 经过上述的步骤之后，主处理器在EXU级通过EAI接口的请求通道（Request Channel）派发给外部的协处理器，派发的信息包括指令的编码信息，两个源操作数的值（由于蜂鸟E200是32位架构，则两个源操作数均为32位宽），和该指令的派发标号（Dispatch ITag）。蜂鸟 E200 处理器属于“乱序执行-顺序写回”， 因此该派发标号（ITag）主要用于追踪指令的派发顺序，协处理器收到 Tag 后需要携带该 Tag 直至写回结果并连同 ITag 返还给主处理器。
- 协处理器通过 Request Channel 接受指令之后，需对指令做进一步的译码，并进行实际的执行操作。协处理器可以当拍便返回结果，也可以多拍之后再返回结果。对于多拍返回结果的指令，其行为可以是阻塞式的，也可以是流水线式。由于协处理器和主处理器的 Request Channel 采取的是 Valid-Ready 方式的同步握手接口，只要协处理器能够接受指令，其可以将 Ready 信号拉高，因此只要协处理器能够先后连续接受多条指令，则主处理器可以先后连续发送多条指令至协处理器。
- 协处理器在完成执行后，将结果通过 EAI 接口的返回通道（Response Channel）将结果反馈给主处理器。如果协处理器接受并执行了多条指令，则其需要保证其在 Response Channel 反馈结果的顺序必须与其在 Request Channel 接受指令时的顺序一致，即按序写回。因此 Response Channel 需要包括该指令的 ITag 并遵循其顺序。如果是需要写回结果的指令，则 Response Channel 还需包含返回结果的值。
- 主处理器在收到 Response Channel 的反馈结果之后，则将此次指令从流水线中退役（Retire）并将结果写回 Regfile（如果有写回需求）。Custom 指令从被发送至协处理器到协处理器反馈结果并 Retire 之间的这段期间，我们称之为滞外指令（Oustanding Instructions），蜂鸟 E200 处理器最多仅能支持不超过 4 条以上的 Oustanding Instructions，因此如果协处理器是流水线执行方式，但是其需要超过4个周期以上才能反馈结果的话，那么流水线会出现空泡（Bubble），因为其仅能从主处理器收到 4 条指令。

2.7 EAI 存储器接口

协处理器访问存储器资源可以扩大协处理器的类型范围，而不仅仅是限于运算指令类型。在处理器的 LSU 模块中为 EAI 协处理器预留了专用的访问接口，如图 2-3 所示，因此基于 EAI 接口的协处理器可以访问处理器能够寻址的数据存储器资源，包括 ITCM，DTCM，系统存储总线，系统设备总线，快速 IO 接口等。

EAI 指令访问存储器资源的机制如下：

- 主处理器的 LSU 为 EAI 协处理器预留的专用访问通道基于 ICB 总线标准。ICB 总线分为存储器请求通道（Memory Request Channel）和存储器返回通道（Memory Response Channel）。
- 为了防止后续指令访问存储器和 EAI 协处理器访问存储器形成死锁。协处理器在接收到 EAI 的 Request Channel 发送过来的指令后进行译码，如果发现是需要访问存储器资源的协处理器指令，则需立即将存储器独占信号拉高（eai_mem_holdup），主处理器将会阻止后续的指令继续访问存储器资源。
- 协处理器需要访问存储器是通过 ICB 的 Memory Request Channel 向主处理器的 LSU 发起请求。Memory Request Channel 中的信息包括需要访问的存储器地址，访问是读或是写操作。如果是读操作，意味着对主处理器进行 32 位对齐的一次读操作；如果是写操作，则通过 Memory Request Channel 中的写数据（wdata）和字节掩码（wmask）控制写操作的数据和粒度。

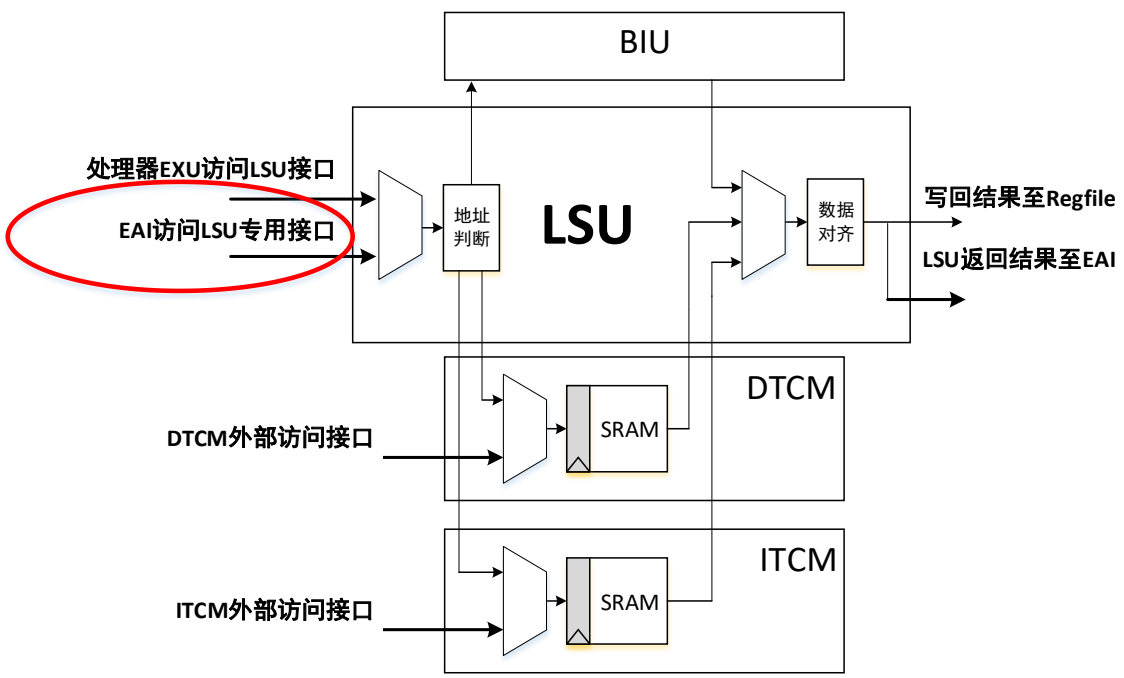

图 2-3 LSU 为 EAI 预留专用访问通道

- 主处理器的 LSU 在完成存储器读写操作后，通过 ICB 的 Memory Response Channel 向协处理器反馈。如果是读操作，Memory Response Channel 中的信息包括返回的读数据，和本次读操作是否发生了错误；如果是写操作，Memory Response Channel 中的信息仅包含本次写操作是否发生了错误。
- 由于协处理器和主处理器 LSU 接口的 ICB 总线采取的是 Valid-Ready 方式的同步握手接口，因此只要主处理器的 LSU 能够先后连续多次存储器访问操作，则协处理器可以先后连续发送多个存储器读写请求。
- 协处理器在完成对存储器的访存之后，需将存储器独占信号拉低（eai_mem_holdup），主处理器将会释放 LSU 允许后续的指令继续访问存储器资源。

2.8 EAI 接口时序

本节将描述 EAI 接口的若干典型时序。

- 主处理器向协处理器通过 EAI 的 Request Channel 派发指令，协处理器在同一个周期即返回结果，但是主处理器不能接收，下一周期才能收此结果，如图 2-4 所示。

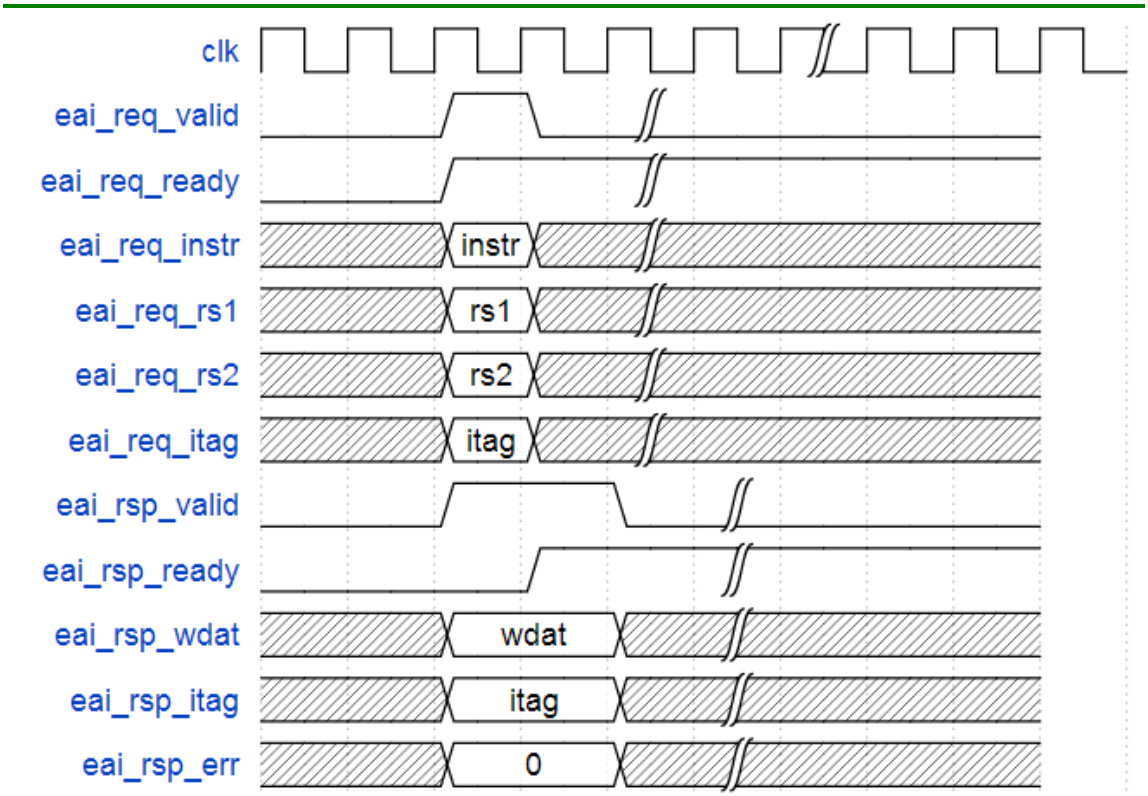

图 2-4 协处理器同一周期内返回结果

- 主处理器向协处理器通过 EAI 的 Request Channel 派发指令，协处理器需要多个周期才能返回结果，且协处理器是阻塞式的，因此其不能接受新的指令（将 eai_req_ready 拉低），直至其通过 EAI 的 Response Channel 返回计算结果且被主处理器接收，如图 2-5 所示。

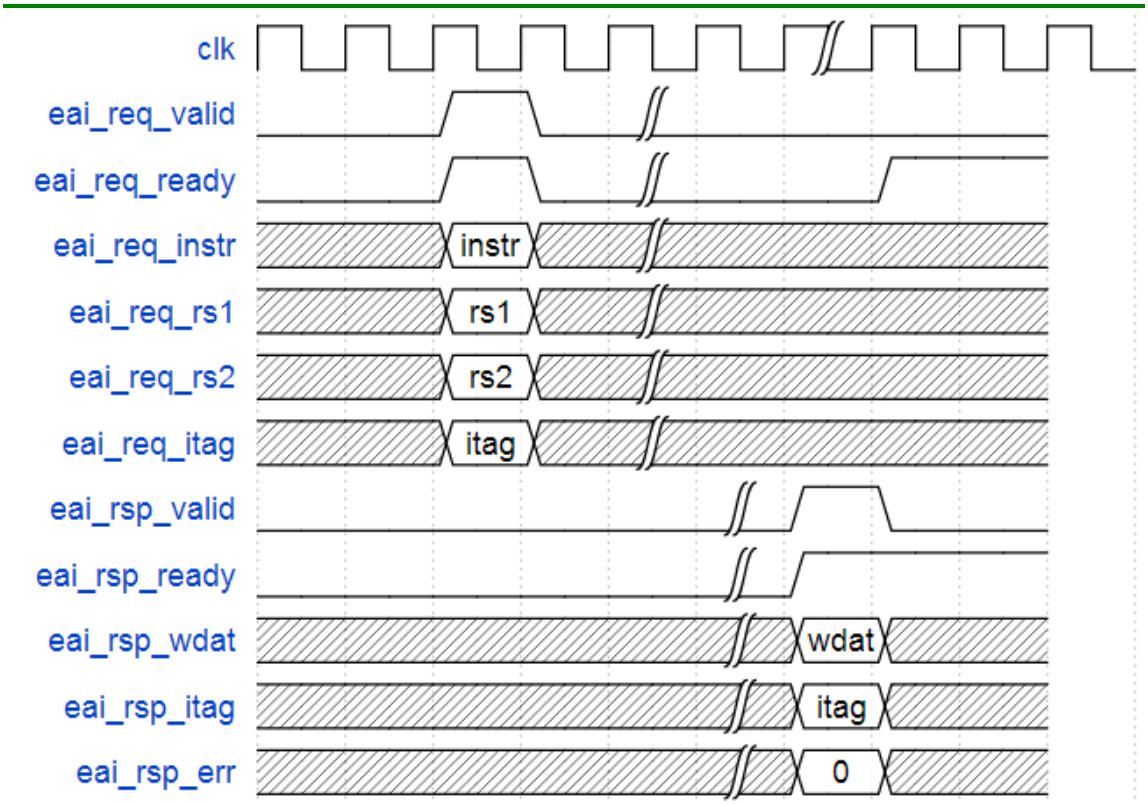

图 2-5 协处理器阻塞式多周期返回结果

- 主处理器向协处理器通过 EAI 的 Request Channel 派发指令，协处理器需要四个周期才能返回结果，但协处理器是非阻塞式的，因此能连续接受新的指令（将 eai_req_ready 拉高），直至其通过 EAI 的 Response Channel 返回计算结果且被主处理器接收，如图 2-6 所示，整个过程无空泡。

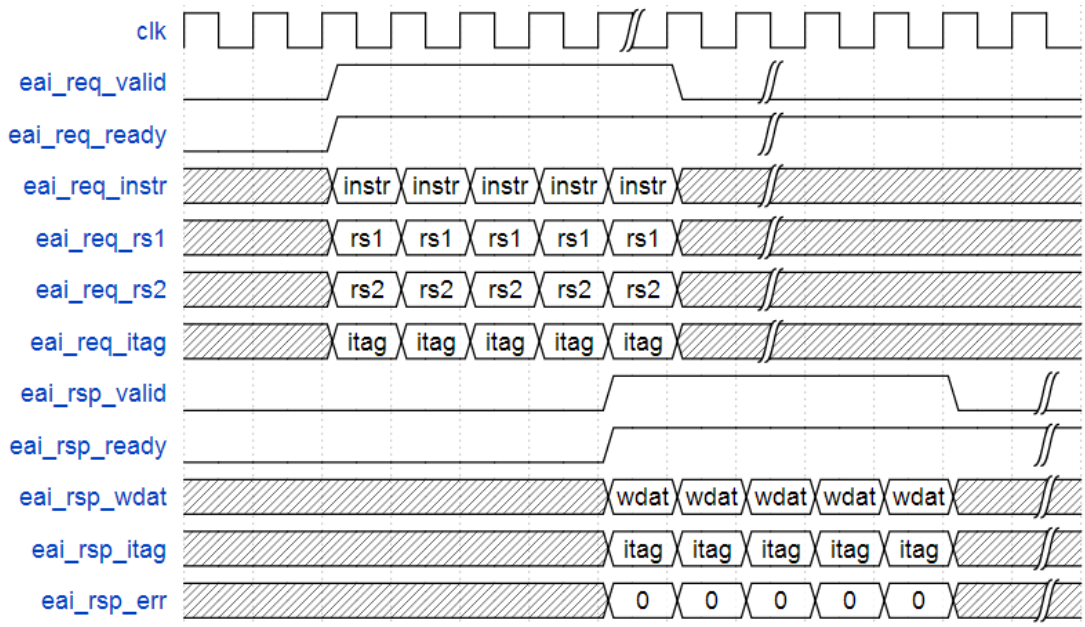

图 2-6 协处理器非阻塞式连续接收四条指令无空泡周期

- 主处理器向协处理器通过 EAI 的 Request Channel 派发指令，协处理器需要八个周期才能返回结果，但协处理器是非阻塞式的，因此能连续接受新的指令（将 eai_req_ready 拉高）。但是主处理器最多只能发送四条 Oustanding Instructions，因此

在 EAI 的 Response Channel 返回计算结果且被主处理器接收之前会出现四个空泡周期，如图 2-7 所示。

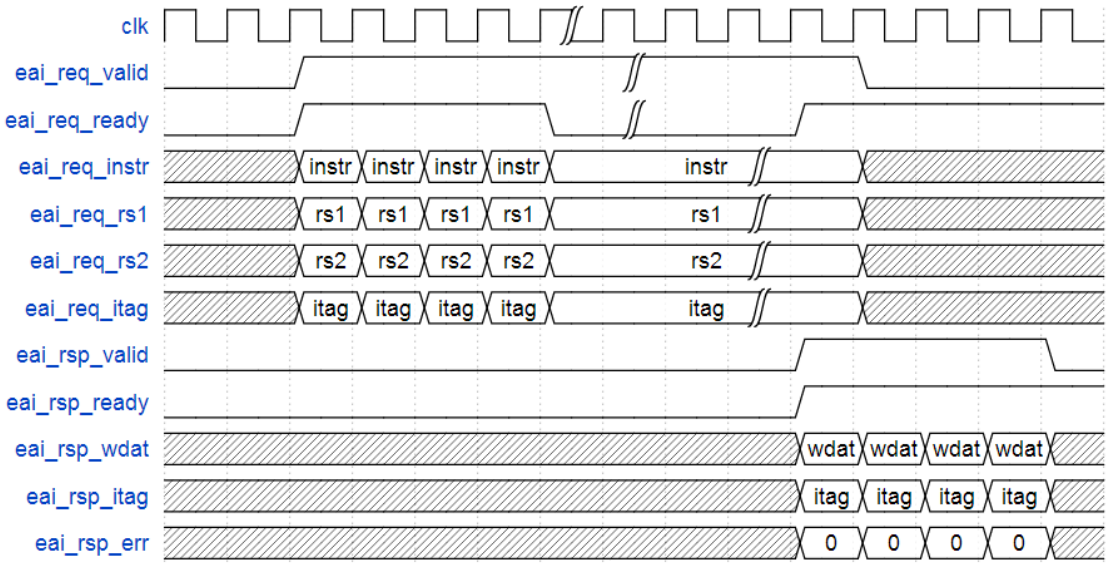

图 2-7 协处理器非阻塞式连续接收四条指令有空泡周期

- 主处理器向协处理器通过 EAI 的 Request Channel 派发指令，协处理器译码出该指令需要访问存储器，则将存储器独占信号拉高（eai_mem_holdup），协处理器通过

Memory Request Channel 向主处理器 LSU 发起读写请求，主处理器通过 Memory Response Channel 在下一个周期返回结果，协处理器继而将存储器独占信号拉低，如图 2-8 所示。

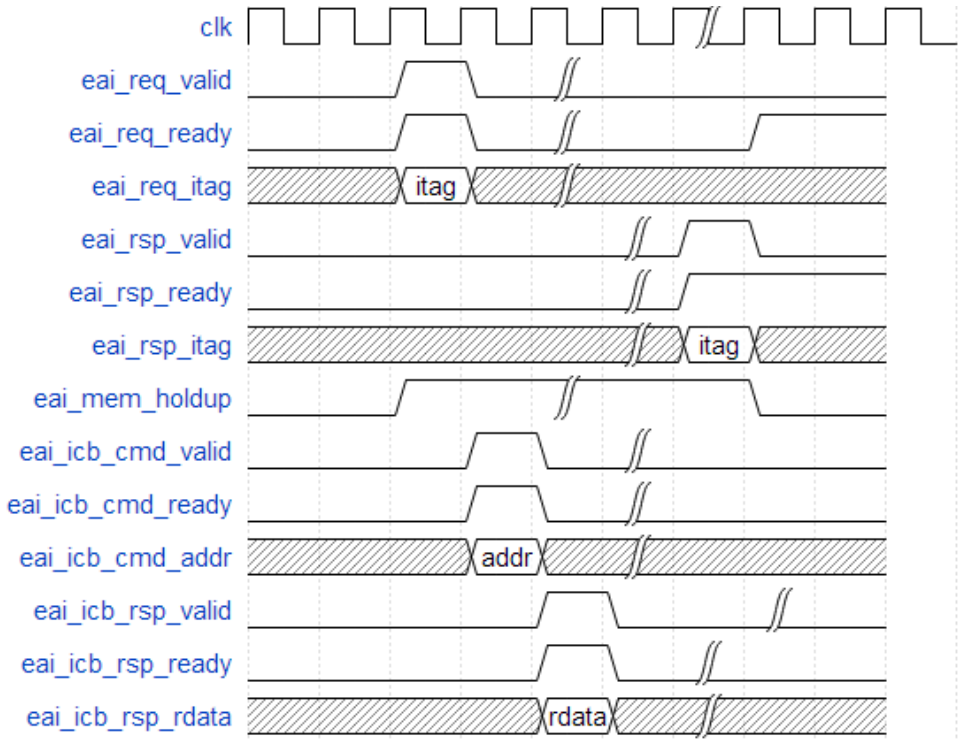

图 2-8 协处理器访问存储器一个周期返回结果

- 主处理器向协处理器通过 EAI 的 Request Channel 派发指令，协处理器译码出该指令需要访问存储器，则将存储器独占信号拉高（eai_mem_holdup），协处理器通过 Memory Request Channel 向主处理器 LSU 发起读写请求，主处理器通过 Memory Response Channel 在多个周期之后返回结果，协处理器继而将存储器独占信号拉低，如图 2-9 所示。

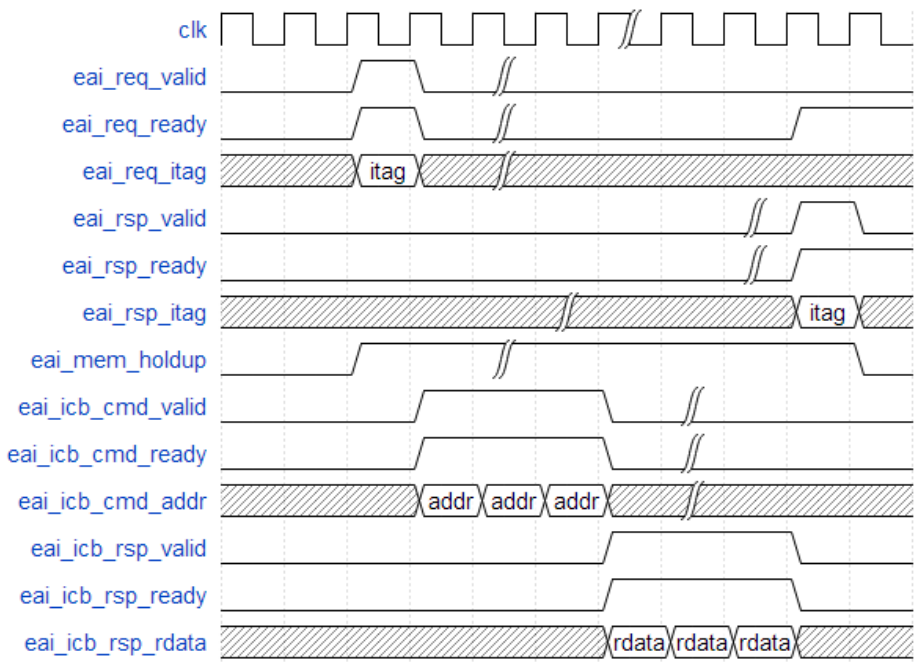

图 2-9 协处理器访问存储器多个周期返回结果

- 主处理器向协处理器通过 EAI 的 Request Channel 派发指令，协处理器译码出该指令需要访问存储器，则将存储器独占信号拉高（eai_mem_holdup），协处理器通过 Memory Request Channel 向主处理器 LSU 发起读写请求，主处理器通过 Memory Response Channel 在多个周期之后返回结果，但是结果指示读写存储器是发生了错误，协处理器继而将存储器独占信号拉低，如图 2-10 所示。

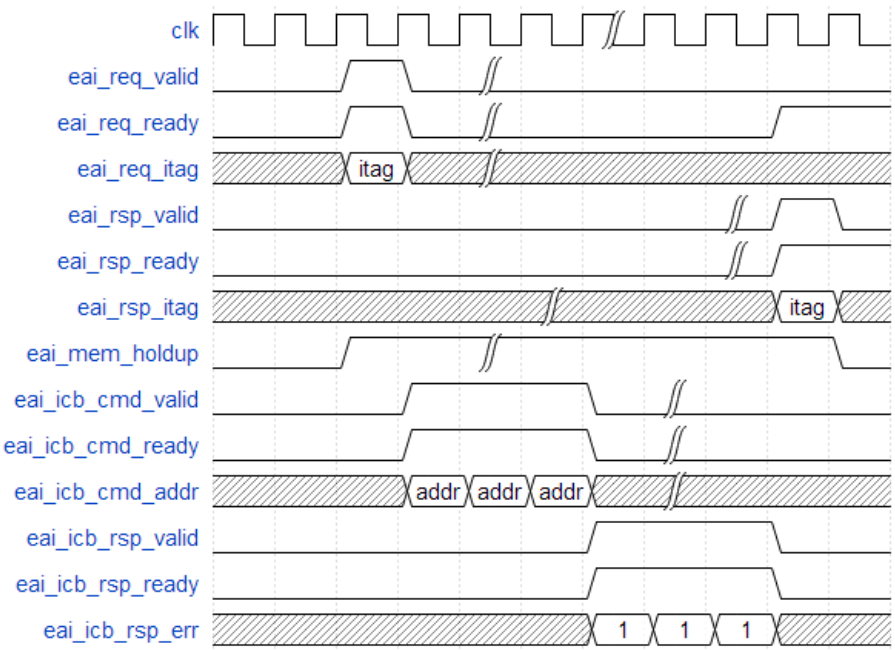

图 2-10 协处理器访问存储器多个周期返回错误标志

- 主处理器向协处理器通过 EAI 的 Request Channel 派发指令，协处理器译码出该指令是个非法指令，其通过 Response Channel 返回错误标志。如图 2-11 所示。

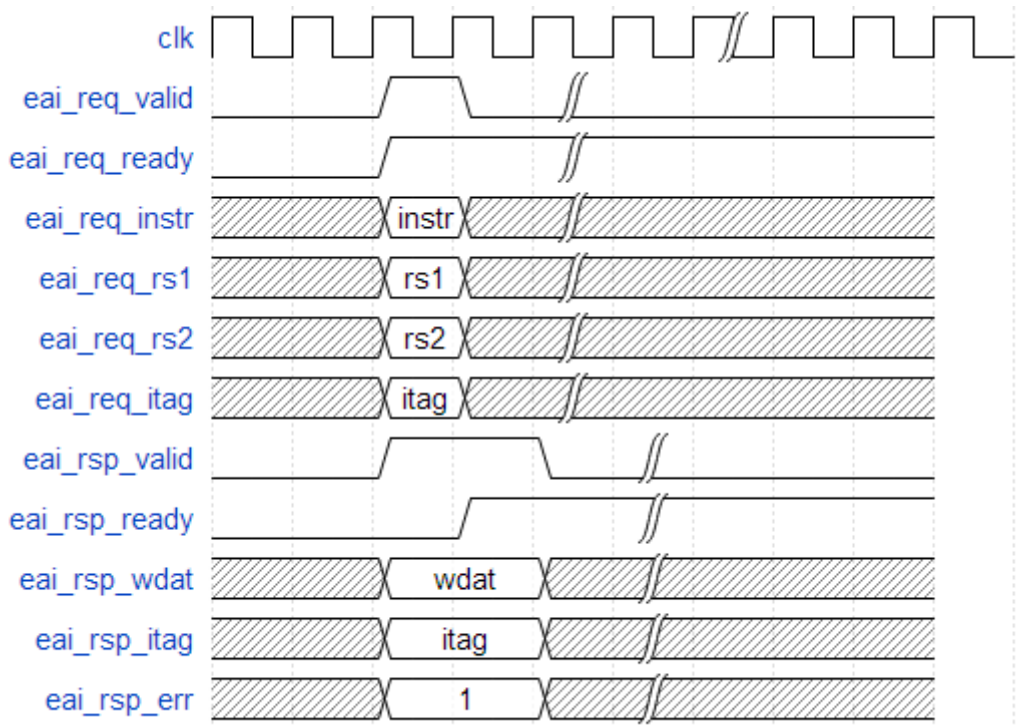

图 2-11 协处理器非法指令返回错误标志

3 蜂鸟 E200 的协处理器参考示例

本节将通过 Rocket Core 的协处理器接口 RoCC 的参考示例，阐述蜂鸟 E200 处理器如何使用 EAI 接口定义并实现一个协处理器。

3.1 示例协处理器需求

假设有一个 N 行 N 列的矩阵按顺序存储在存储器中，矩阵的每个元素都是 32 位的整数，如图 3-1 所示。需要对该矩阵进行如下操作：

- 计算逐行的累加和，由于有 N 行，因此可以得出四个累加的结果，分别是 Rowsum1，Rowsum2，Rowsum3，…，Rowsum<N>。
- 计算逐列的累加和，由于有 M 列，因此可以得出四个累加的结果，分别是 Colsum1，Colsum2，Colsum3，…，Colsum<N>。

1	2	3
4	5	6
7	8	9

图 3-1　3*3 矩阵

如果采用常规的程序进行计算，则需要采用循环方式，按行读取各个元素，然后将各个元素相加得到各行的累加和，然后再采取循环方式，按列读取各个元素，然后将各个元素相加得到各列的累加和，如图 3-2 所示。

```
Start_Timer();

for (i = 0; i < msize; i++){
  rawsum[i] = 0;
  for (j = 0; j < msize; j++){
    rawsum[i] = rawsum[i] + matrix[i][j];
  }
}

for (i = 0; i < msize; i++){
  colsum[i] = 0;
  for (j = 0; j < msize; j++){
    colsum[i] = colsum[i] + matrix[j][i];
  }
}

Stop_Timer();

User_Time = End_Time - Begin_Time;
```

图 3-2 C 语言程序计算矩阵行列累加和

这种程序转换成为汇编代码需要消耗较多的指令条数，如图 3-3 所示，理论上它需要完整的从存储器中读取矩阵元素两次，第一次用于计算行累加和，第二次用于计算列累加和，因此需要总共 N*N*2 次存储器读操作，此外程序还需要指令参与循环控制，累加计算等等。假设 N=3，N=3，则总计需要几十个指令周期才能完成。

```
80002150:   429c                lw  a5,0(a3)
80002152:   42c8                lw  a0,4(a3)
80002154:   468c                lw  a1,8(a3)
80002156:   06b1                addi    a3,a3,12
80002158:   97aa                add a5,a5,a0
8000215a:   97ae                add a5,a5,a1
8000215c:   c21c                sw  a5,0(a2)
8000215e:   0611                addi    a2,a2,4
80002160:   fed818e3        bne a6,a3,80002150 <main+0x68>
80002164:   474c                lw  a1,12(a4)
80002166:   4f10                lw  a2,24(a4)
80002168:   4785                li  a5,1
8000216a:   0040                addi    s0,sp,4
8000216c:   97ae                add a5,a5,a1
8000216e:   86a2                mv  a3,s0
80002170:   97b2                add a5,a5,a2
80002172:   c29c                sw  a5,0(a3)
80002174:   1028                addi    a0,sp,40
80002176:   0711                addi    a4,a4,4
80002178:   0691                addi    a3,a3,4
8000217a:   00e50c63        beq a0,a4,80002192 <main+0xaa>
8000217e:   431c                lw  a5,0(a4)
80002180:   474c                lw  a1,12(a4)
80002182:   4f10                lw  a2,24(a4)
80002184:   0711                addi    a4,a4,4
80002186:   97ae                add a5,a5,a1
80002188:   97b2                add a5,a5,a2
8000218a:   c29c                sw  a5,0(a3)
8000218c:   0691                addi    a3,a3,4
8000218e:   fee518e3        bne a0,a4,8000217e <main+0x96>
```

图 3-3 C 语言编译所得汇编程序

3.2 示例协处理器指令

为了提高性能和能效比，可以将此矩阵操作定义成为协处理器指令进行加速。如表 3-1 所示，定义了三条指令，分别是 SETUP，ROWSUM，和 COLSUM。

表 3-1 示例协处理器指令

协处理器指令	介绍	编码
SETUP	SETUP 指令定义矩	■ Opcode 指明使用 Custom0 指令组。

	阵的大小，并指明存储器中连续存储矩阵元素地址区间的第一个地址。	■ xd 位的值为 0，表示此指令并不需要写回任何结果。 ■ xs1 位的值为 1，表示此指令需要读取操作数 rs1。操作数 rs1 的值为连续存储矩阵元素地址区间的第一个地址。 ■ xs2 位的值为 1，表示此指令需要读取操作数 rs2。操作数 rs2 的值为指示矩阵的大小的 N 值。 ■ funct7 的值为 0，该值编码 SETUP 指令。
ROWSUM	ROWSUM指令指示协处理器计算指定行的累加值，并通过结果寄存器返回累加值，如果发生了溢出，则返回32位整数能表示的最大饱和值。	■ Opcode 指明使用 Custom0 指令组。 ■ xd 位的值为 1，表示此指令需要通过写回结果至 rd 寄存器。 ■ xs1 位的值为 1，表示此指令需要读取操作数 rs1。操作数 rs1 的值为指定行的行号（从 1 开始编号）。 ■ xs2 位的值为 0，表示此指令不需要读取操作数 rs2。 ■ funct7 的值为 1，该值编码 ROWSUM 指令。
COLSUM	COLSUM 指令指示协处理器计算指定列的累加值，并通过结果寄存器返回累加值，如果发生了溢出，则返回 32 位整数能表示的最大饱和值。	■ Opcode 指明使用 Custom0 指令组。 ■ xd 位的值为 1，表示此指令需要通过写回结果至 rd 寄存器。 ■ xs1 位的值为 1，表示此指令需要读取操作数 rs1。操作数 rs1 的值为指定行的列号（从 1 开始编号）。 ■ xs2 位的值为 0，表示此指令不需要读取操作数 rs2。 ■ funct7 的值为 2，该值编码 COLSUM 指令。
注意，以上的几条指令需要按照特定的顺序编写： ■ 首先使用SETUP指令指定矩阵的大小及相关参数。 ■ 然后使用多条ROWSUM指令分别计算各行的累加和。 ■ 然后使用多条COLSUM指令分别计算各列的累加和。		

3.3 示例协处理器实现

示例协处理器的硬件实现框图如图 3-4 所示，其主要由控制模块和累加模块组成。控制模块主要负责和主处理器的 EAI 接口交互，并调用累加器进行累加运算。

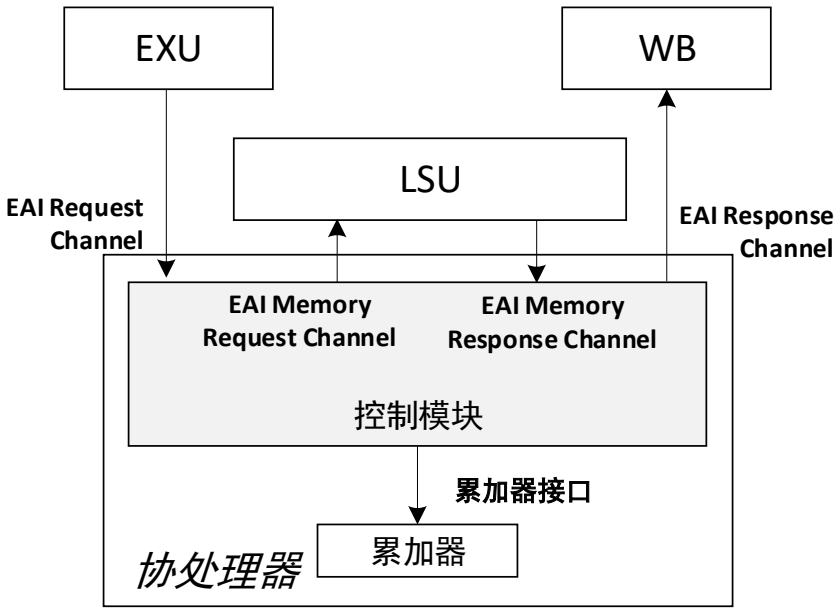

图 3-4 示例协处理器控制模块框图

累加器框图如图 3-5 所示，其主要负责累加运算。

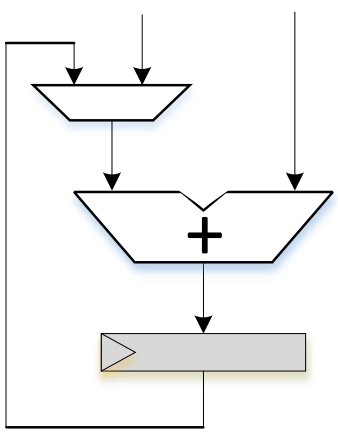

图 3-5 示例协处理器累加器模块框图

示例协处理器实现方案的处理过程如表 3-6 中所示

表 3-6 示例协处理器实现方案

步骤	指令行为	硬件行为
步骤 1	使用 SETUP 指令指定需要运算的矩阵参数。	■ 控制模块使用参数设置合适的行为
步骤 2	使用多条ROWSUM指令分别计算各行的累加和。	■ 为了能够复用读取的数据，在硬件模块中分配一个行大小的 buffer，称之为 buf_row，用于存储上一次处理的行的元素值。 ■ ROWSUM 指令将触发协处理器从存储器地址中逐个读取改行的元素值，且每次读取回来的数值被送给累加器进行累加，完全读取 N 个元素后的累加结果即为该行的累加和。 ■ 从第二行开始，每次读取回来的数值不仅需要使用累加器计算改行的累加值，还需使用新都取的数据和上一行的数据（存储在 buf_row 之中）累加，并且将结果继续暂存在 buf_row 之中。 ■ 经过 N 个 ROWSUM 指令之后，不仅仅每行的累加和已经被计算出，而且每一列的累加和也被计算出存储在 buf_row 之中。
步骤 3	使用多条COLSUM指令分别计算各列的累加和。	■ 由于在步骤 2 中已经计算出了各列的累加和存储在 buf_row 之中。因此本步骤无需任何的存储器访问和计算操作，直接从 buf_row 中取出所需的计算结果即可。
以图 3-1 中的矩阵实例为例子，整个计算过程的数值如表 3-7 所示。		

表 3-7 示例协处理器计算过程的数值

累加和	结果	存储器访问次数	列 1 部分和	列 2 部分和	列 3 部分和
ROWSUM 指令计算行 1 累加和	6	3	1	2	3
ROWSUM 指令计算行 2	15	3	5	7	9

累加和					
ROWSUM 指令计算行 3 累加和	24	3	12	15	18
COLSUM 指令计算列 1 累加和	12	0	12	15	18
COLSUM 指令计算列 2 累加和	15	0	12	15	18
COLSUM 指令计算列 3 累加和	18	0	12	15	18

3.4 示例协处理器性能

以图 3-1 中的矩阵实例为例子，分析示例协处理器的性能如下：

- SETUP 指令可以在一个周期内完成。
- 假设矩阵存储在 DTCM 区间内，可以通过一个周期从 DTCM 中读回一个矩阵元素，则计算第一行的累加和需要 3 个时钟周期访问 3 个存储器地址，采用流水线的方式连续读取和运算（读取一个元素后的累加计算和下一次读取操作重叠），则总共需要 4 个周期完成第一行的 ROWSUM 指令。
- 从第二行开始，不仅需要计算该行的累加和，还需要累加计算得到 row_buf 中每一列的部分累加和，因此需要额外的 3 个周期，则总共需要 7 个周期完成第二行和第三行的 ROWSUM 指令。
- 每一个 COLSUM 指令可以直接从 row_buf 中取出计算结果，可以在一个周期内完成。
- 综上，完成全部操作需要 7 条指令，1+4+7+7+1+1+1=22 个时钟周期完成。

与普通的 C 语言程序实现相比的性能对比如表 3-8 所示，在性能和能效比方面就能带来 3-5 倍的提升。

表 3-8 使用示例协处理器与不使用之性能对比

	普通的 C 语言程序	**使用示例协处理器**
动态执行指令数	66	7

动态执行周期数	80	22
读取 ITCM 取指令次数	66	7
读取 DTCM 取数据次数	18	9

4 蜂鸟 E200 的协处理器源代码

对于示例协处理器的代码包括两部分内容：

- 示例协处理器的 Verilog RTL 实现代码。
- 使用协处理器指令的软件代码。

由于蜂鸟 E200 的协处理器接口目前尚未开源。本文在此不予详述其代码，有兴趣的用户可以自行尝试实现或者联系 bobhu_riscv@163.com。