**设计报告**

华中科技大学一队

张鑫、马翔、林力韬、梁瑾

# 一、设计简介

构建MIP32五级流水CPU，即分IF、ID、EX、MEM、WB五段处理。IF段读指令；ID段译指令；EX段执行指令；MEM段读写内存；WB段写回寄存器。对于特殊的指令，如跳转指令，提前到ID段执行。具有重定向、嵌套中断、异常处理功能， cache加速访存，搭建AXI总线，支持多种外设。

## （一）设计变更说明

无。

# 二、设计方案

## （一）总体设计思路

系统大致可以分成三部分：CPU、cache和总线。之所以这么划分，主要是根据CPU的工作方式来的：CPU的主要工作是，从存储器中读取指令，解析执行，再把执行结果写回到寄存器/存储器中。为了让CPU与存储器之间能够通信，我们需要用到总线，有AHB、AXI等多种总线可选，我们选用功能较为强大的AXI总线。由于总线的传输需要多个时钟周期，为了提升系统性能，必须减少总线访问，所以需要将指令和cache预取到CPU内部，这就是cache的由来，指令cache和数据cache分开。

CPU工作时，先从指令cache中取指令，如果程序地址对应的指令不在指令cache中，就需要将CPU停住，向总线发送读指令请求，经过若干个周期以后，总线送回需要的指令附近的一行指令，写入到cache的一行，若cache已满，则用随机策略替换其中一行。取到指令以后，在译码段解析该指令，生成控制信号。从寄存器文件中读取操作数，送到算数逻辑单元ALU，计算结果送访存段。如果是load型指令（lb等），在访存段需要读数据cache，如果发生缺失，则与指令cache的缺失一样，访问总线，调入数据，替换策略也为随机替换。如果是store型指令（sb等），则需要写数据cache。采用写回策略，比较节省时间。当数据cache需要发生替换时，要判断被替换的块是不是被写过，若是，则该数据是脏数据，需要通过总线写到RAM中，再进行替换。数据经过访存段以后，继续送往写回段，写到寄存器组中。并不是所有的指令都需要执行这五段，但是不论执行与否，都需要通过流水线传输，保证系统的规范性。

CPU是整个系统的主体，cache和总线都是为CPU服务的，但是从实现需要的工作量来说，三者的难度相当，所以下面分成三部分来进行介绍。

CPU：除了控制器、通用寄存器组、ALU等必需的部件以外，为了实现跳转，需要地址转移逻辑；为了处理指令间的数据冲突，需要冲突检测和数据重定向；为了实现异常处理，需要cp0寄存器组和异常处理逻辑；为了实现乘除法，需要HILO寄存器。按照五段进行划分，如所图 1示，CPU中主要的部件如下：

取值段：程序计数器PC、指令cache；

译码段：控制器、位扩展器、通用寄存器组、HILO寄存器、cp0寄存器组、分支跳转逻辑；

执行段：算数逻辑单元ALU、重定向（旁路）检测模块；

访存段：数据cache，异常处理模块；

写回段：通用寄存器、cp0寄存器、HILO寄存器（同译码段）。

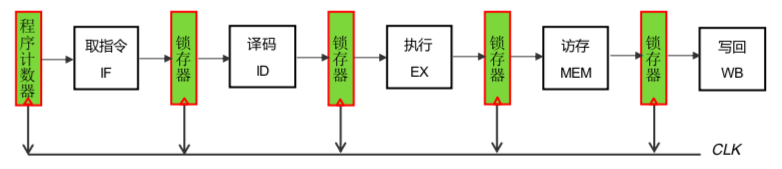


图 1 指令流水逻辑架构

Cache：根据用途分为iCache和dCache，分别负责指令缓存和数据缓存；

总线：实现AXI总线功能。

CPU实现指令集如表 1所示。

表 1 指令集

| **#** | **指令名称格式** | **简单功能描述** |
| --- | --- | --- |
| 1 | ADD rd,rs,rt | 加(可产生溢出例外) |
| 2 | ADDI rt,rs,immediate | 加立即数(可产生溢出例外) |
| 3 | ADDU rd,rs,rt | 加(不产生溢出例外) |
| 4 | ADDIU rt,rs,immediate | 加立即数(不产生溢出例外) |
| 5 | SUB rd,rs,rt | 减(可产生溢出例外) |
| 6 | SUBU rd,rs,rt | 减(不产生溢出例外) |
| 7 | SLT rd,rs,rt | 有符号小于置 1 |
| 8 | SLTI rt,rs,immediate | 有符号小于立即数设置 1 |
| 9 | SLTU rd,rs,rt | 无符号小于设置 1 |
| 10 | SLTIU rt,rs,immediate | 无符号小于立即数设置 1 |
| 11 | DIV rs,rt | 有符号字除 |
| 12 | DIVU rs,rt | 无符号字除 |
| 13 | MULT rs,rt | 有符号字乘 |
| 14 | MULTU rs,rt | 无符号字乘 |
| 15 | AND rd,rs,rt | 位与 |
| 16 | ANDI rt,rs,immediate | 立即数位与 |
| 17 | LUI rt,immediate | 寄存器高半部分置立即数 |
| 18 | NOR rd,rs,rt | 位或非 |
| 19 | OR rd,rs,rt | 位或 |
| 20 | ORI rt,rs,immediate | 立即数位或 |
| 21 | XOR rd,rs,rt | 位异或 |
| 22 | XORI rt,rs,immediate | 立即数位异或 |
| 23 | SLL rd,rs,sa | 立即数逻辑左移 |
| 24 | SLLV rd,rs,rt | 变量逻辑左移 |
| 25 | SRA rd,rs,rt | 立即数算术右移 |
| 26 | SRAV rd,rs,rt | 变量算术右移 |
| 27 | SRL rd,rt,sa | 立即数逻辑右移 |
| 28 | SRLV rd,rs,rt | 变量逻辑右移 |
| 29 | BEQ rs,rt,offset | 相等转移 |
| 30 | BNE rs,rt,offset | 不等转移 |
| 31 | BGEZ rs,offset | 大于等于 0 转移 |
| 32 | BGTZ rs,offset | 大于 0 转移 |
| 33 | BLEZ rs,offset | 小于等于 0 转移 |
| 34 | BLTZ rs,offset | 小于 0 转移 |
| 35 | BLTZAL rs,offset | 小于 0 调用子程序并保存返回地址 |
| 36 | BGEZAL rs,offset | 大于等于 0 调用子程序并保存返回地址 |
| 37 | J target | 无条件直接跳转 |
| 38 | JAL target | 无条件直接跳转至子程序并保存返回地址 |
| 39 | JR rs | 无条件寄存器跳转 |
| 40 | JALR rd,rs | 无条件寄存器跳转至子程序并保存返回地址下 |
| 41 | MFHI rd | HI 寄存器至通用寄存器 |
| 42 | MFLO rd | LO 寄存器至通用寄存器 |
| 43 | MTHI rs | 通用寄存器至 HI 寄存器 |
| 44 | MTLO rs | 通用寄存器至 LO 寄存器 |
| 45 | BREAK | 断点 |
| 46 | SYSCALL | 系统调用 |
| 47 | LB rt,offset(base) | 取字节有符号扩展 |
| 48 | LBU rt,offset(base) | 取字节无符号扩展 |
| 49 | LH rt,offset(base) | 取半字有符号扩展 |
| 50 | LHU rt,offset(base) | 取半字无符号扩展 |
| 51 | LW rt,offset(base) | 取字 |
| 52 | SB rt,offset(base) | 存字节 |
| 53 | SH rt,offset(base) | 存半字 |
| 54 | SW rt,offset(base) | 存字 |
| 55 | ERET | 从中断、例外处理返回 |
| 56 | MFC0 rt,rd,sel | 从协处理器0的寄存器取值 |
| 57 | MTC0 rt,rd,sel | 向协处理器0的寄存器存值 |

CPU内部部分数据通路设计如图 2所示。

## （二）异常处理模块设计

对模块内部设计方案进行更进一步描述。可以包含：模块的功能意图，模块的输入输出，模块内部的数据通路和控制逻辑，以及可能的软硬件交互机制。

在MIPS中，中断、陷阱、系统调用和任何可以中断程序正常执行流的情况都称作异常，并且由单独的机制处理。引起异常的事件有：

1. 外部事件：CPU核外部的事件，即中断。中断用于使CPU的注意力转向某外部事件。中断是唯一独立于CPU正常指令流的异常条件；
2. 内存翻译异常：当地址需要翻译，但是对于硬件，没有有效的地址翻译可用，或者写入一个写保护的页时，会发生这种情况。操作系统必须决定这样的异常是否错误。如果异常是应用程序访问了允许访问的地址空间以外的地方，那么这个问题必须通过终止应用程序以保护系统的其他部分来修正；
3. 其他不太常见的内核修正的程序条件：这种异常中最常见的就是浮点指令导致的异常，硬件不能处理一些困难的、很少见的操作和操作数的组合；
4. 程序或者硬件探测到的错误：包括不存在的指令，在用户特权级非法的指令，在禁用对应的SR标志的情况下执行协处理器指令，整数溢出，地址对齐错误，以及在用户模式下访问kuseg以外的地址；
5. 数据完整性问题：许多MIPS CPU持续地在总线上检查数据，或者检查缓存中数据的每字节奇偶错误或者字范围上的ECC纠错码。缓存或奇偶错误在支持数据检查的CPU上产生异常；
6. 系统调用和陷入：有些指令的目的就是产生可识别的异常，用于以一个安全的方式构建软件工具（系统调用、仔细的代码产生的条件陷入和断点）。

目前，CPU异常处理模块支持中断异常、地址错误异常、未定义指令异常、整数溢出异常、系统调用异常、断点异常。

1. 中断异常：启用中断时发生中断请求时发生中断异常。
2. 地址错误异常：一个地址错误异常发生在如下情况。

* 取指令字地址未对齐，即32位地址低两位非0；
* 加载/存储字指令lw、sw执行的地址未对齐，即32位地址低两位非0；
* 加载/存储半字指令lh、sh执行的地址未对齐，即32位地址最低位非0；
* 从用户模式或管理员模式引用内核地址空间；
* 从用户模式引用超级用户地址空间。

如果取指令未在字边界上对齐，则在检测到条件之前更新PC。 因此，EPC和BadVAddr都指向未对齐的指令地址。

1. 未定义指令异常：CPU暂不支持该指令。
2. 整数溢出异常：陷阱形式的整数运算指令导致溢出，则发生整数溢出异常。
3. 系统调用异常：syscall指令执行产生系统调用异常。
4. 断点异常：break指令执行产生系统调用异常。

除中断异常发生时，Cause寄存器IV位为1时异常入口向量为中断向量（偏移0x200），否则上述异常的异常入口向量为通用异常向量（偏移0x180）。

协处理寄存器Cause的ExcCode域存储着5位的编码，指示当前发生的异常为何种异常。具体情况如下表 2。

表 2 ExcCode值：不同异常类型

| ExcCode值 | 助记符 | 描述 |
| --- | --- | --- |
| 0 | Int | 中断 |
| 1 | Mod | 修改TLB表项标记为只读的页 |
| 2 | TLBL | TLB翻译失败（数据加载或者取指令），意味着TLB中没有与用到的程序地址对应的表项。当根本没有匹配的表项（甚至连无效的表项也没有），且CPU并没有处于异常模式，即SR(EXL)置位，这是一个TLB缺失。将由一个特殊的指向能流畅处理这种常见事件的异常入口点处理，一般由操作系统控制 |
| 3 | TLBS | TLB翻译失败（数据存储）， 异常处理同上 |
| 4 | AdEL | 地址错误（数据加载或取指令），是在用户态下试图访问kuseg以外的段，或者是试图读一个双字、字或者半字而地址没有相应的对齐 |
| 5 | AdES | 地址错误（数据存储），错误原因同上 |
| 6 | IBE | 总线错误（取指令）。外部硬件指示发生了某种错误，异常处理与系统相关。 |
| 7 | DBE | 总线错误（数据加载），错误原因同上。而存储数据操作发生总线错误时，只能间接地反应出来，表现为缓存读入写过的高速缓存块中的结果。 |
| 8 | Syscall | 由syscall指令执行 |
| 9 | Bp | 由break指令执行，被调试器使用 |
| 10 | RI | CPU没有定义的指令（或非法指令） |
| 11 | CpU | 使用协处理器指令，但对应的SR(CU3-0)没有使能 |
| 12 | Ov | 陷阱形式的整数运算指令导致溢出 |
| 13 | TRAP | 条件陷入指令teq等的某个条件被满足 |
| 14 | MSAFPE | MSA浮点异常 |
| 15 | FPE | 浮点异常 |
| 16-17 | - | 用户自定义的异常类型，依赖于具体实现 |
| 18 | C2E | 协处理器2异常 |
| 19 | TLBRI | TLB读禁止异常 |
| 20 | TLBXI | TLB执行禁止异常 |
| 21 | MSADis | MSA无效异常 |
| 22 | MDMX | 以前是MDMX不可用异常（MDMX ASE）。 MDMX已弃用修订版5。 |
| 23 | WATCH | 加载/存储的物理地址和WatchHi / WatchLo寄存器中的值匹配 |
| 24 | MCheck | 机器检测 |
| 25 | Thread | 线程相关异常 |
| 26 | DSP | DSP模块状态禁用异常 |
| 27 | GE | 虚拟客户异常 |
| 28-29 | - | 保留 |
| 30 | CacheErr | 缓存错误。 在正常模式下，高速缓存错误异常具有专用向量，并且不更新Cause寄存器。 如果在调试模式下实现EJTAG并发生缓存错误，则此代码将写入DebugDExcCode字段，以指示重新进入调试模式是由缓存错误引起的。 |
| 31 | - |  |

检测ALU、Controller等部件发出的信号，接入异常处理模块进行异常检测及处理。当一条指令同时满足多个例外触发条件时，CPU将按照表 3所示例外优先级，优先触发优先级高的例外。

表 3 异常优先级

| **异常** | **类型** |
| --- | --- |
| 中断 | 异步 |
| 地址错例外—取指 | 同步 |
| 保留指令例外 | 同步 |
| 整型溢出例外、陷阱例外、系统调用例外 | 同步 |
| 地址错例外—数据访问 | 同步 |

异常处理单元引脚与功能描述如表 4所示。

表 4 异常处理单元引脚与功能描述

| 引脚 | 输入/输出 | 位宽 | 功能描述 |
| --- | --- | --- | --- |
| invalid\_inst | 输入 | 1 | 未定义指令信号 |
| syscall | 输入 | 1 | 系统调用指令信号 |
| break\_inst | 输入 | 1 | break指令信号 |
| eret | 输入 | 1 | eret指令信号 |
| pc\_value | 输入 | 32 | 当前PC值 |
| in\_delayslot | 输入 | 1 | 在分支延迟槽发生异常则为高电平 |
| overflow | 输入 | 1 | 整数运算溢出信号，高电平有效 |
| interrupt\_flags | 输入 | 8 | 中断信号。硬件中断信号+软中断信号 |
| allow\_int | 输入 | 1 | 中断使能,{SR(ERL,EXL,IE)==001} |
| ebase\_in | 输入 | 20 | CP0寄存器组15.1号寄存器——EBase,产生exception\_base |
| epc\_in | 输入 | 32 | CP0寄存器组14号寄存器——EPC |
| special\_int\_vec | 输入 | 1 | CP0寄存器组13号寄存器——Cause(VI)[23],使用特殊中断入口 |
| boot\_exp\_vec | 输入 | 1 | 启动时异常向量：当boot\_exp\_vec==1时，CPU用ROM(kseg1)空间的异常入口。正常运行中的操作系统里，boot\_exp\_vec一般设置为0 |
| iaddr\_exp\_illegal | 输入 | 1 | 指令地址非法异常信号 |
| daddr\_exp\_illegal | 输入 | 1 | 数据地址非法异常信号 |
| mem\_data\_vaddr | 输入 | 32 | MEM段数据虚拟地址 |
| mem\_data\_we | 输入 | 1 | MEM段数据写信号 |
| flush | 输出 | 1 | 异常的停机信号 |
| cp0\_wr\_exp | 输出 | 1 | 送CP0异常信号，高电平有效 |
| cp0\_clean\_exl | 输出 | 1 | 写往SR[EXL] |
| exp\_epc | 输出 | 32 | 异常后应记录的EPC |
| exp\_code | 输出 | 5 | 异常的编号 |
| exp\_bad\_vaddr | 输出 | 32 | 发生异常的地址，写往CP0寄存器组第8号寄存器BadVaddr |
| cp0\_badv\_we | 输出 | 1 | 地址相关异常的信号，高电平有效 |
| exception\_new\_pc | 输出 | 32 | 异常后的新PC |
| exp\_asid | 输出 | 8 | 地址空间重填的值，ENTRYHi(ASID)与使能 |
| cp0\_exp\_asid\_we | 输出 | 1 | 地址空间重填信号，高电平有效，一般为0 |

检测是否发生中断异常、地址错误异常、未定义指令异常、整数溢出异常、系统调用异常和断点异常，若有上述异常发生，则cp0\_wr\_exp置1向CP0报告例外发生,需要修改EXL位，并置cp0\_clean\_exl为0，屏蔽其他的中断和例外，陷入内核。发出异常停机信号flush。

更新EPC，当异常发生在延迟槽时回退一条指令到分支，否则取当前段的PC值。异常后的PC先指向其他异常,默认是32'h(BFC00200+180)。

eret指令信号到来，清除异常编号，清除异常信号，异常级别SR[EXL]在这条指令后恢复,离开内核态，恢复到CP0中保存的EPC位置执行。

遇到无法识别的异常予以清除，并清除异常的停机信号，清楚无法识别的异常编号。

## （三）控制器模块设计

控制器负责解析指令，产生流水线控制信号。

表 5 控制器单元引脚与功能描述

| 引脚 | 输入/输出 | 位宽 | 功能描述 |
| --- | --- | --- | --- |
| op | 输入 | 6 | MIPS32指令op段，ins[31:26] |
| func | 输入 | 6 | MIPS32指令func段，ins[5:0] |
| rs | 输入 | 5 | MIPS32指令rs段，ins[25:21] |
| rt | 输入 | 5 | MIPS32指令rt段，ins[20:16] |
| shamt | 输入 | 5 | MIPS32指令shamt段，ins[10:6] |
| pc\_id\_byte | 输入 | 2 | ID段PC值的低两位 |
| control\_bus | 输出 | 61 | 控制器产生的信号总线 |
| branch\_jump | 输出 | 10 | 所有跳转指令的编码，送跳转逻辑 |
| in\_delayslot | 输出 | 1 | 延迟槽，送IF-ID流水段，延迟一个周期 |

控制器产生的信号有：有符号加法信号（用于溢出判断）、有符号减法信号（用于溢出判断）、load\_store信号（用于识别所有访存指令，使用统一编码）、寄存器指令编码信号（用于通用寄存器重定向数据识别，寄存器重定向到寄存器、寄存器重定向到lo、寄存器重定向到hi、寄存器重定向到cp0）、nop指令信号（高电平有效）、未定义指令信号（高电平有效）、eret指令信号（高电平有效）、break指令信号（高电平有效）、syscall指令信号（高电平有效）、hilo寄存器写模式（写寄存器hi、寄存器lo使能）、memory写使能信号、memory加载信号、寄存器r1值使用信号、寄存器r2值使用信号、ALU输入A选择信号、ALU输入B选择信号、立即数扩展选择信号、cp0仅存起写使能信号、寄存器写数据选择信号、寄存器写编号选择信号、寄存器写使能信号、寄存器文件读编号选择信号、ALU操作码信号。

数据通路所需信号编码如图 2所示。



图 2 CPU内部数据通路信号编码

## （四）ALU运算器模块设计

构建一个 32 位运算器，可支持算术加、减、乘、除，逻辑与、 或、非、异或运算、逻辑左移、逻辑右移，算术右移运算，支持常用程序状态标志(溢出overflow，除法进行is\_diving)，运算器输入输出引脚以及功能见下表表 6、表 7。

表 6 ALU运算器单元引脚与功能描述

| 引脚 | 输入/输出 | 位宽 | 功能描述 |
| --- | --- | --- | --- |
| clk | 输入 | 1 | 时钟，服务于除法器 |
| rst | 输入 | 1 | 复位信号，服务于除法器 |
| X | 输入 | 32 | 操作数X |
| Y | 输入 | 32 | 操作数Y |
| S | 输入 | 4 | 运算器功能码，具体功能见下表 |
| add\_sub | 输入 | 2 | 有符号加减法信号编码 |
| flush | 输入 | 1 | 异常的停机信号，服务于除法器 |
| Result1 | 输出 | 32 | ALU运算结果 |
| Result2 | 输出 | 32 | ALU运算结果第二部分，用于乘法指令结果高位或除法指令的余数位 |
| overflow | 输出 | 1 | 溢出信号，高电平有效 |
| is\_diving | 输出 | 1 | 正在进行除法信号，高电平有效 |

表 7 ALU运算器功能码

| ALU OP | 十进制 | 运算功能 |
| --- | --- | --- |
| 0000 | 0 | Result = X << Y 逻辑左移 (Y 取低五位) Result2=0 |
| 0001 | 1 | Result = X >>>Y 算术右移 (Y 取低五位) Result2=0 |
| 0010 | 2 | Result = X >> Y 逻辑右移 (Y 取低五位) Result2=0 |
| 0011 | 3 | Result = (X \* Y)[31:0]; Result2 = (X \* Y)[63:32] 有符号 |
| 0100 | 4 | Result = X/Y; Result2 = X%Y 无符号 |
| 0101 | 5 | Result = X + Y (Set overflow) |
| 0110 | 6 | Result = X - Y (Set overflow) |
| 0111 | 7 | Result = X & Y 按位与 |
| 1000 | 8 | Result = X | Y 按位或 |
| 1001 | 9 | Result = X⊕Y 按位异或 |
| 1010 | 10 | Result = ~(X |Y) 按位或非 |
| 1011 | 11 | Result=(X<Y)?1:0 符号比 |
| 1100 | 12 | Result=(X<Y)?1:0 无符号比较 |

ALU运算器除法采用32周期除法，除法运算时流水线暂停，除法运算结束流水线重新开始工作。

ALU运算器乘法采用FPGA内置乘法器，单周期运算。

对于不同的指令，控制器生成不同的ALU运算器功能码，具体对应关系见图 3。

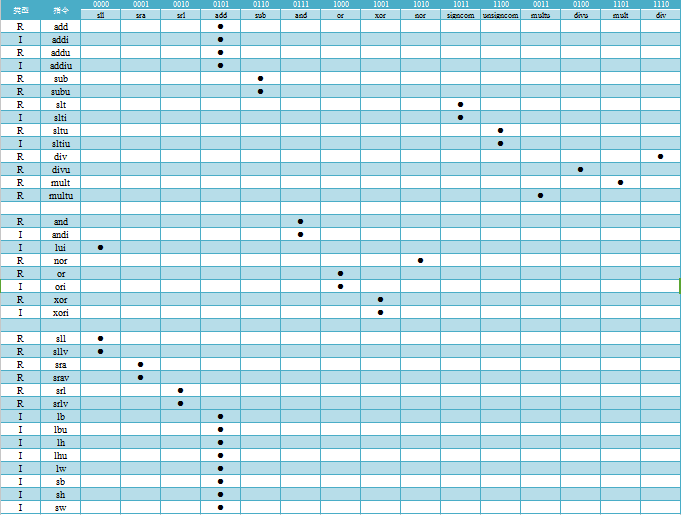


图 3 ALU\_OP与指令的对应关系

## （五）冲突检测模块设计

流水线中存在指令相关，指令间存在依赖关系。指令相关包括数据相关、结构相关、控制相关，指令相关会导致流水线冲突/冒险(Hazzard)。流水线冲突是指由于指令相关的存在，导致指令流水线出现“断流”或“阻塞”，下一条指令 不能在预期的时钟周期加载到流水线中。流水线冲突包括数据冲突、结构冲突、控制冲突。

流水线冲突常规处理除load-use相关（访存指令）外，可采用重定向逻辑解决。

表 8 冲突检测单元引脚与功能描述

| 引脚 | 输入/输出 | 位宽 | 功能描述 |
| --- | --- | --- | --- |
| r1\_r\_id | 输入 | 1 | ID段r1寄存器使用信号 |
| r2\_r\_id | 输入 | 1 | ID段r2寄存器使用信号 |
| r1\_id | 输入 | 5 | ID段r1寄存器编号 |
| r2\_id | 输入 | 5 | ID段r2寄存器编号 |
| rw\_ex | 输入 | 5 | EX段写寄存器编号 |
| rw\_mem | 输入 | 5 | MEM段写寄存器编号 |
| load\_ex | 输入 | 1 | EX段load指令信号，高电平有效 |
| load\_mem | 输入 | 1 | MEM段load指令信号，高电平有效 |
| conflict\_stall | 输出 | 1 | 检测到冲突，高电平有效 |

ID段与EX段发生load-use相关，寄存器r1与写寄存器为同一寄存器且r1寄存器被使用，或寄存器r2与写寄存器为同一寄存器且r2寄存器被使用，此时EX段为load指令、寄存器文件写使能有效。

ID段与MEM段发生load-use相关同理。

load-use相关通过插入气泡消除如图 4所示。

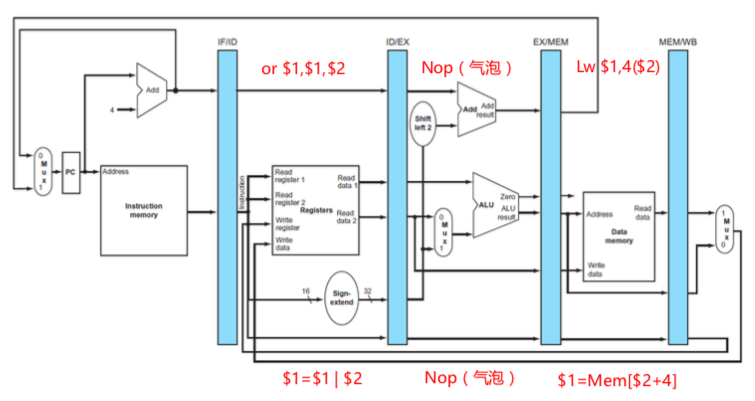


图 4 插入气泡消除load-use相关

## （六）通用寄存器读值重定向模块设计

重定向逻辑可解决除load-use相关外的数据相关。

由于分支跳转逻辑在ID段实现且使用通用寄存器，因此通用寄存器读值重定向亦在ID段实现。

表 9 通用寄存器读值重定向单元引脚与功能描述

| 引脚 | 输入/输出 | 位宽 | 功能描述 |
| --- | --- | --- | --- |
| rdata1\_id | 输入 | 32 | ID段r1寄存器值 |
| rdata2\_id | 输入 | 32 | ID段r2寄存器值 |
| alu\_r1\_ex | 输入 | 32 | EX段ALU Result1值 |
| alu\_r1\_mem | 输入 | 32 | MEM段ALU Result1值 |
| hilo\_ex | 输入 | 64 | EX段hilo寄存器值 |
| hilo\_mem | 输入 | 64 | MEM段hilo寄存器值 |
| cp0\_data\_ex | 输入 | 32 | EX段读的CP0寄存器值 |
| cp0\_data\_mem | 输入 | 32 | MEM段读的CP0寄存器值 |
| pc\_ex | 输入 | 32 | EX段PC值 |
| pc\_mem | 输入 | 32 | MEM段PC值 |
| r1\_r\_id | 输入 | 1 | ID段r1寄存器使用信号 |
| r2\_r\_id | 输入 | 1 | ID段r2寄存器使用信号 |
| r1\_id | 输入 | 5 | ID段r1寄存器编号 |
| r2\_id | 输入 | 5 | ID段r2寄存器编号 |
| din\_sel\_ex | 输入 | 3 | EX段通用寄存器写数据来源编号信号 |
| din\_sel\_mem | 输入 | 3 | MEM段通用寄存器写数据来源编号信号 |
| rw\_ex | 输入 | 5 | EX段写寄存器编码 |
| rw\_mem | 输入 | 5 | MEM段写寄存器编码 |
| real\_rdata1\_id | 输出 | 32 | 重定向r1寄存器值 |
| real\_rdata2\_id | 输出 | 32 | 重定向r2寄存器值 |

检测ID段与EX、MEM段存在数据相关，根据din\_sel信号区分EX、MEM段指令有效数据，并重定向到数据相关的r1或r2。

## （七）hilo寄存器重定向模块设计

hi、lo寄存器重定向逻辑同通用寄存器重定向在实现ID段实现。

表 10 hilo寄存器重定向单元引脚与功能描述

| 引脚 | 输入/输出 | 位宽 | 功能描述 |
| --- | --- | --- | --- |
| hilo\_id | 输入 | 64 | ID段hilo寄存器值 |
| alu\_r1\_ex | 输入 | 32 | EX段ALU Result1值 |
| alu\_r2\_ex | 输入 | 32 | EX段ALU Result2值 |
| alu\_r1\_mem | 输入 | 32 | MEM段ALU Result1值 |
| alu\_r2\_mem | 输入 | 32 | MEM段ALU Result2值 |
| rdata1\_ex | 输入 | 32 | ID段r1寄存器值 |
| rdata1\_mem | 输入 | 32 | ID段r2寄存器值 |
| hilo\_mode\_ex | 输入 | 2 | EX段hilo写模式编号 |
| hilo\_mode\_mem | 输入 | 2 | MEM段hilo写模式编号 |
| real\_hilo\_id | 输出 | 64 | ID段hilo重定向值 |

根据EX段、MEM段的hilo\_mode信号区分EX、MEM段指令有效数据，并重定向到数据相关的hi、lo寄存器。

## （八）分支跳转模块设计

ID段实现分支跳转逻辑，尽量减少发生分支冲突时的流水线性能损失。

分支指令跳转在ID段执行，分支指令执行时需要修改程序计数器 PC 的值，运算的分支目标地址的结果经多路选择器传送给PC，下一个时钟上跳沿到来时锁存进入PC。MIPS流水线中采用了分支延迟槽技术，也就是分支指令后面的一条指令无论跳转成功与否均会被执行。

表 11 分支跳转单元引脚与功能描述

| 引脚 | 输入/输出 | 位宽 | 功能描述 |
| --- | --- | --- | --- |
| bj\_type\_ID | 输入 | 10 | ID段所有跳转指令的编码 |
| num\_a\_ID | 输入 | 32 | ID 段重定向r1寄存器值 |
| num\_b\_ID | 输入 | 32 | ID 段重定向r2寄存器值 |
| imm\_b\_ID | 输入 | 32 | ID段B跳转指令立即数扩展 |
| imm\_j\_ID | 输入 | 26 | ID段J跳转指令立即数 |
| JR\_addr\_ID | 输入 | 32 | ID段jr指令跳转地址 |
| PC\_ID | 输入 | 32 | ID段PC值 |
| Branch\_Jump | 输出 | 1 | 跳转信号，高电平有效 |
| BJ\_address | 输出 | 32 | 跳转地址 |

根据bj\_type区分分支指令，不同的分支指令跳转地址计算公式有所差异。由于分支跳转模块的r1、r2寄存器值为重定向，此时即可判断跳转是否成功，跳转成功置Branch\_Jump为1，否则为0。

## （九）Cache模块设计

本设计的cache是支持uncache访问的多路组相联设计，工作在写回模式下，对外使用AXI协议接口，支持多次连续突发传输。Cache的设计采用全参数化设计，可以任意设置参数调整cache行的字长、cache的大小、组相联的路数等等。

根据需要，我们将cache行设置为了16字每行，在进行AXI总线访问时一次读写一个cache行，如果没有延迟将耗费23个周期的时间。Cache的访存使用物理地址，虚拟地址到物理地址的转换在CPU的数据通路内完成。我们将cache设为2路组相联，因为这一数量既满足命中率的要求又不会浪费空间；我们将DCache的容量定为8K，ICache的容量定为8K，我们将cache容量定为8k，无论ICache、DCache均采用相同的参数，主要是在命中率、资源耗费、综合布线难度中取折中。

Cache的设计分为4个层次，分别是cache行层次、cache组层次、cache/uncache逻辑层次、dcache/icache对外封装层次。每个层次完成不同的功能，并向更高的层次提供接口。Cache行是真正的存储体，其包含了有效位、脏位、tag、数据等存储体，主要负责根据4位的使能来进行字节级别的读写操作；cache组是多个cache行的封装，其主要进行组内的tag比对，给出命中、不命中、已满等信息，并执行随机替换算法；逻辑层次包含两个模块， cache模块是对cache组的再一次封装，主要实现接受处理CPU访存请求，对外的AXI状态机等等；uncache模块则是不包含存储体的模块，仅仅将CPU的请求转为AXI协议的信号时序；封装层次是区分dcache和icache而设计，icache仅包含cache的一个实例，dcache包含cache和uncache两个实例，并处理了地址异常的情况。

表 12 DCache/ICache单元引脚与功能描述

| 引脚 | 输入/输出 | 位宽 | 功能描述 |
| --- | --- | --- | --- |
| rst | 输入 | 1 | 复位信号，低电平有效 |
| clk | 输入 | 1 | 时钟 |
| AXI\_rd\_dready | 输入 | 1 | 读访问，总线上数据就绪可写Cache的控制信号，高电平有效 |
| AXI\_rd\_last | 输入 | 1 | 读访问,标识总线上当前数据是最后一个数据字的控制信号，高电平有效 |
| AXI\_rd\_data | 输入 | 32 | 读访问总线给出的数据 |
| AXI\_rd\_addr\_clear | 输入 | 1 | 读请求地址已经被响应信号，高电平有效 |
| AXI\_wr\_next | 输入 | 1 | 写访问，总线允许送下一个数据的控制信号，高电平有效 |
| AXI\_wr\_ok | 输入 | 1 | 写访问，总线标识收到从设备最后一个ACK的控制信号，高电平有效 |
| AXI\_wr\_addr\_clear | 输入 | 1 | 写请求地址已经被响应信号，高电平有效 |
| cpu\_addr | 输入 | 32 | CPU访问地址 |
| cpu\_byteenable | 输入 | 4 | CPU访问模式 |
| cpu\_read | 输入 | 1 | CPU读命令 |
| cpu\_write | 输入 | 1 | CPU写命令 |
| cpu\_hitwriteback | 输入 | 1 | CPU强制写穿命令 |
| cpu\_hitinvalidate | 输入 | 1 | CPU强制失效命令 |
| cpu\_wrdata | 输入 | 32 | CPU写数据 |
| cpu\_addr\_illegal | 输入 | 1 | CPU访问地址非法信号，高电平有效 |
| new\_lw\_ins\_tocache | 输入 | 1 | CPU执行新指令信号，高电平有效 |
| AXI\_addr | 输出 | 32 | 送总线地址 |
| AXI\_addr\_valid | 输出 | 1 | 送总线地址有效的控制信号 |
| AXI\_we | 输出 | 1 | 送总线标记访问是读还是写的控制信号 |
| AXI\_size | 输出 | 3 | 访问，送总线size |
| AXI\_lens | 输出 | 8 | 访问，送总线length |
| AXI\_rd\_rready | 输出 | 1 | 送总线，主设备就绪读数据的控制信号 |
| AXI\_wr\_data | 输出 | 32 | 写访问，送总线的数据 |
| AXI\_wr\_dready | 输出 | 1 | 写访问，送总线一个字数据就绪的控制信号 |
| AXI\_byte\_enable | 输出 | 4 | 写访问，送总线写字节使能的控制信号 |
| AXI\_wr\_last | 输出 | 1 | 写访问，送总线表示当前数据是最后一个数据字的控制信号 |
| AXI\_response\_rready | 输出 | 1 | 写响应就绪信号 |
| cpu\_rddata | 输出 | 32 | 发往CPU读到的数据 |
| cpu\_stall | 输出 | 1 | 发往CPU停机等待信号 |

表 13 UnCache单元引脚与功能描述

| 引脚 | 输入/输出 | 位宽 | 功能描述 |
| --- | --- | --- | --- |
| rst | 输入 | 1 | 复位信号，低电平有效 |
| clk | 输入 | 1 | 时钟 |
| AXI\_rd\_dready | 输入 | 1 | 读访问，总线上数据就绪可写Cache的控制信号，高电平有效 |
| AXI\_rd\_last | 输入 | 1 | 读访问,标识总线上当前数据是最后一个数据字的控制信号，高电平有效 |
| AXI\_rd\_data | 输入 | 32 | 读访问总线给出的数据 |
| AXI\_rd\_addr\_clear | 输入 | 1 | 读请求地址已经被响应信号，高电平有效 |
| AXI\_wr\_next | 输入 | 1 | 写访问，总线允许送下一个数据的控制信号，高电平有效 |
| AXI\_wr\_ok | 输入 | 1 | 写访问，总线标识收到从设备最后一个ACK的控制信号，高电平有效 |
| AXI\_wr\_addr\_clear | 输入 | 1 | 写请求地址已经被响应信号，高电平有效 |
| cpu\_addr | 输入 | 32 | CPU访问地址 |
| cpu\_byteenable | 输入 | 4 | CPU访问模式 |
| cpu\_read | 输入 | 1 | CPU读命令 |
| cpu\_write | 输入 | 1 | CPU写命令 |
| cpu\_hitwriteback | 输入 | 1 | CPU强制写穿命令 |
| cpu\_hitinvalidate | 输入 | 1 | CPU强制失效命令 |
| cpu\_wrdata | 输入 | 32 | CPU写数据 |
| cpu\_new\_ins | 输入 | 1 | CPU执行新指令信号，高电平有效 |
| AXI\_addr | 输出 | 32 | 送总线地址 |
| AXI\_addr\_valid | 输出 | 1 | 送总线地址有效的控制信号 |
| AXI\_we | 输出 | 1 | 送总线标记访问是读还是写的控制信号 |
| AXI\_size | 输出 | 3 | 访问，送总线size |
| AXI\_lens | 输出 | 8 | 访问，送总线length |
| AXI\_rd\_rready | 输出 | 1 | 送总线，主设备就绪读数据的控制信号 |
| AXI\_wr\_data | 输出 | 32 | 写访问，送总线的数据 |
| AXI\_wr\_dready | 输出 | 1 | 写访问，送总线一个字数据就绪的控制信号 |
| AXI\_byte\_enable | 输出 | 4 | 写访问，送总线写字节使能的控制信号 |
| AXI\_wr\_last | 输出 | 1 | 写访问，送总线表示当前数据是最后一个数据字的控制信号 |
| AXI\_response\_rready | 输出 | 1 | 写响应就绪信号 |
| cpu\_rddata | 输出 | 32 | 发往CPU读到的数据 |
| cpu\_stall | 输出 | 1 | 发往CPU停机等待信号 |

Cache状态机如图 5所示。

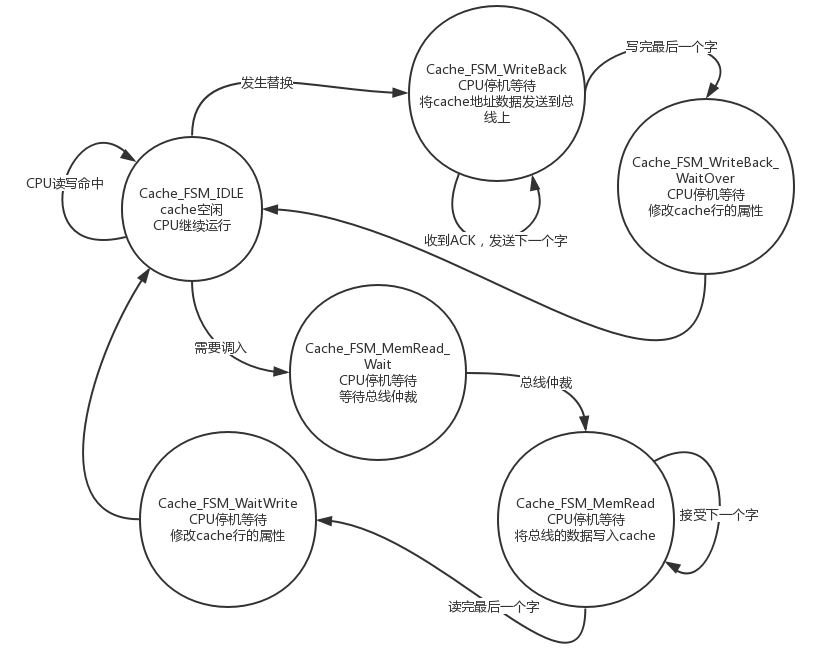


图 5 Cache状态机

下面将讨论一个典型的访存例子，如果CPU需要进行一次写访问，此时cache中没有需要写的字，而其地址对应的组已经全满，而且都已经被修改。这时响应CPU的请求分为淘汰、调入、写操作三步完成。

首先，向CPU发送停机信号，命令CPU停机等待访存。接着根据访存地址查找到对应的cache组，使用随机淘汰算法决定淘汰的cache行，将其地址发送到AXI总线上；

第二步，等待总线空闲和仲裁流程，总线响应后开始将cache行的连续16个字写到总线上，写一个字需要收到一个ACK确认，直到写完最后一个字，发送response响应。

第三步，根据CPU访存地址发送访存请求到AXI总线上，等待总线空闲和仲裁流程。总线响应后将总线数据写入刚刚淘汰的空的cache行，写最后一个字的同时将cache行修改为有效，脏位为0。

第四步，结束访存后，使用一个上升沿，在写入CPU需要写入的数据的同时，撤除停机信号，CPU在下一个时钟开始重新运转。

在性能测试带有固定延时的框架下，写回共需要26个周期，调入共需要48个周期，响应CPU请求1个周期，共计75个时钟周期，这也是耗时最长的一种访存情况。

## （十）AXI接口模块设计

AXI协议是基于burst的传输，并且定义了以下5个独立的传输通道：读地址通道、读数据通道、写地址通道、写数据通道、写响应通道。

地址通道携带控制消息用于描述被传输的数据属性，数据传输使用写通道来实现“主”到“从”的传输，“从”使用写响应通道来完成一次写传输；读通道用来实现数据从“从”到“主”的传输。

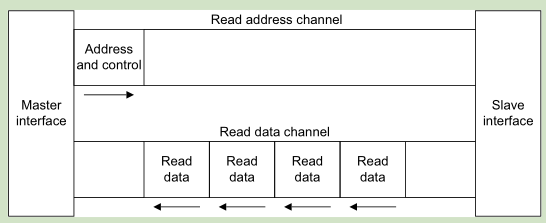


图 6 AXI读架构

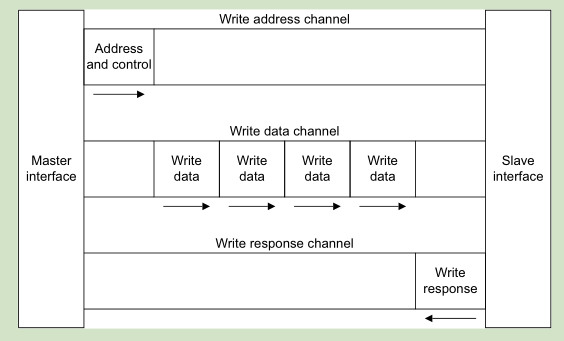


图 7 AXI写架构

AXI是基于VALID/READY的握手机制数据传输协议，传输源端使用VALID表明地址/控制信号、数据是有效的，目的端使用READY表明自己能够接受信息。

读/写地址通道：读、写传输每个都有自己的地址通道，对应的地址通道承载着对应传输的地址控制信息。

读数据通道：读数据通道承载着读数据和读响应信号包括数据总线（8/16/32/64/128/256/512/1024bit）和指示读传输完成的读响应信号。

写数据通道：写数据通道的数据信息被认为是缓冲（buffered）了的，“主”无需等待“从”对上次写传输的确认即可发起一次新的写传输。写通道包括数据总线（8/16...1024bit）和字节线（用于指示8bit 数据信号的有效性）。

写响应通道：“从”使用写响应通道对写传输进行响应。所有的写传输需要写响应通道的完成信号。

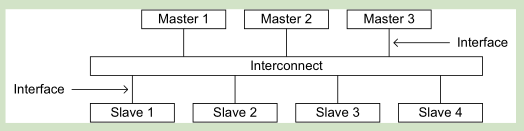


图 8 AXI接口与互联

AXI协议提供单一的接口定义，能用在下述三种接口之间：master/interconnect、slave/interconnect、master/slave。

可以使用以下几种典型的系统拓扑架构。

* 共享地址与数据总线
* 共享地址总线，多数据总线
* multilayer多层，多地址总线，多数据总线

在大多数系统中，地址通道的带宽要求没有数据通道高，因此可以使用共享地址总线，多数据总线结构来对系统性能和互联复杂度进行平衡。

表 14 AXI接口单元引脚与功能描述

| 引脚 | 输入/输出 | 位宽 | 功能描述 |
| --- | --- | --- | --- |
| rset | 输入 | 1 | 复位信号，低电平有效 |
| clk | 输入 | 1 | 时钟 |
| i\_addr | 输入 | 32 | 读访问，总线上数据就绪可写Cache的控制信号，高电平有效 |
| i\_addr\_valid | 输入 | 1 | 读访问,标识总线上当前数据是最后一个数据字的控制信号，高电平有效 |
| i\_we | 输入 | 1 | 读访问总线给出的数据 |
| i\_size | 输入 | 3 | 读请求地址已经被响应信号，高电平有效 |
| i\_lens | 输入 | 8 | 写访问，总线允许送下一个数据的控制信号，高电平有效 |
| i\_rready | 输入 | 1 | 写访问，总线标识收到从设备最后一个ACK的控制信号，高电平有效 |
| d\_addr | 输入 | 32 | 写请求地址已经被响应信号，高电平有效 |
| d\_addr\_valid | 输入 | 1 | CPU访问地址 |
| d\_we | 输入 | 1 | CPU访问模式 |
| d\_size | 输入 | 3 | CPU读命令 |
| d\_lens | 输入 | 8 | CPU写命令 |
| d\_rready | 输入 | 1 | CPU强制写穿命令 |
| d\_wr\_data | 输入 | 32 | CPU强制失效命令 |
| d\_wr\_valid | 输入 | 1 | CPU写数据 |
| d\_byte\_enable | 输入 | 4 | CPU执行新指令信号，高电平有效 |
| d\_resp\_ready | 输入 | 1 | 送总线地址 |
| d\_wr\_wlast | 输入 | 1 | 送总线地址有效的控制信号 |
| axi\_arready | 输入 | 1 | 送总线标记访问是读还是写的控制信号 |
| axi\_awready | 输入 | 1 | 访问，送总线size |
| axi\_bid | 输入 | 4 | 访问，送总线length |
| axi\_bresp | 输入 | 2 | 送总线，主设备就绪读数据的控制信号 |
| axi\_bvalid | 输入 | 1 | 写访问，送总线的数据 |
| axi\_rdata | 输入 | 32 | 写访问，送总线一个字数据就绪的控制信号 |
| axi\_rid | 输入 | 4 | 写访问，送总线写字节使能的控制信号 |
| axi\_rlast | 输入 | 1 | 写访问，送总线表示当前数据是最后一个数据字的控制信号 |
| axi\_rresp | 输入 | 2 | 写响应就绪信号 |
| axi\_rvalid | 输入 | 1 | 发往CPU读到的数据 |
| axi\_wready | 输入 | 1 | 发往CPU停机等待信号 |

仲裁：

针对cache制定的仲裁机制，由于cache和uncache的处理在cache模块中以及进行了处理，所以在接口模块中主要考虑的情况则是指令cache和数据cache 的同时读以及指令cache的读和数据cache的写，采用固定优先级的方式进行仲裁，数据cache优先执行，同时由于AXI协议是五通道读写独立传输，因此可以实现指令cache读和数据cache写同时进行的处理。

最后根据仲裁结果产生锁信号，同时生成对应通道数据传输的相关参数。

读操作：

依据读操作涉及的通道信号依赖关系搭建状态机，如图 9所示。

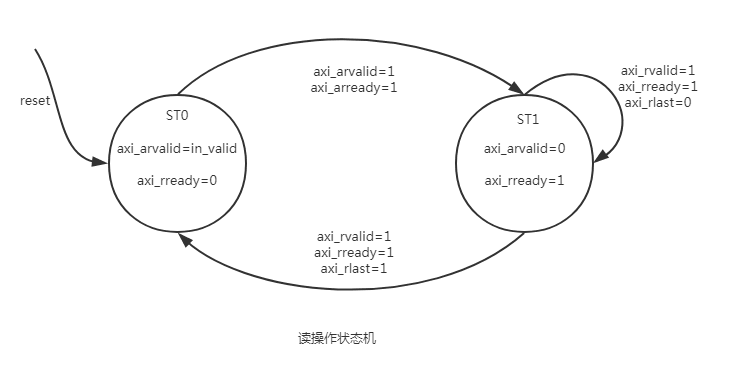


图 9 AXI总线读操作状态机

State0：读地址握手阶段

等待地址握手，地址握手成功则撤销axi\_arvalid信号，并切换状态至State1，并将axi\_rready置为1。

State1：读数据传输阶段

等待数据传输，数据握手成功则发送一次cache写入信号，将读取数据载入cache，在最后一次传输axi\_rlast后切换状态至State0，并将仲裁锁信号撤销。

写操作：

依据写操作涉及的三通道信号依赖关系搭建状态机如图 10所示。

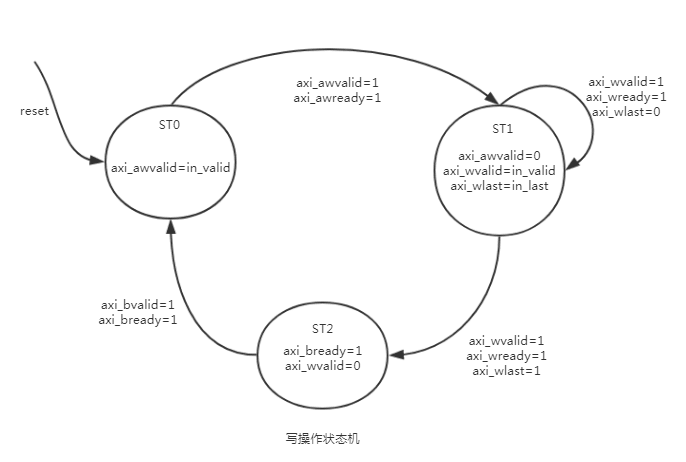


图 10 AXI总线写操作状态机

State0：地址握手阶段

等待地址握手，地址握手成功则撤销axi\_awvalid信号，同时切换状态置State1。

State1：写数据阶段

等待cache提供的写请求，数据握手成功后，提醒cache进行下一次数据写操作，直到接收到最后一次写请求，握手成功后切换状态至State2，同时提醒cache撤销stall停机信号。

State2：写响应阶段

等待写响应，响应握手成功后才能撤销仲裁的写锁，进而等待下一次写操作。

# 三、设计结果

## （一）设计交付物说明

目录如下：

|--HUST\_1\_zhangxin/

|--score.xls

|--report.pdf

|--soc\_axi\_func/

|--rtl/

|--CONFREG/

|--confreg.v

|--myCPU/ 目录：myCPU功能源文件

|--alu\_select.v

|--ALU.v

|--AXI\_interface.v

|--Branch\_Junmp.v

|--cache\_axi.v

|--conflict.v

|--controller.v

|--cp0\_reg.v

|--datapath.v

|--DataRAM.v

|--dcache\_with\_uncache.v

|--decoder.v

|--exception.v

|--extend.v

|--pc.v

|--pipline\_stage.v

|--redirect\_id.v

|--register\_select.v

|--register.v

|--ram\_wrap/

|--axi\_wrap\_ram.v

|--soc\_axi\_lite\_top.v

|--xilinx\_ip/

|--axi\_crossbar\_1X2/

|--axi\_crossbar\_1X2.xci

|--axi\_ram/

|--axi\_ram.xci

|--clk\_pll/

|--clk\_pll.xci

|--run\_vivado

|--mycpu\_prj1/

|--func.bit 功能测试比特流

|--memoty.bit 记忆游戏比特流

|--mycpu.xpr

|--soc\_lite.xdc

|--testbench

|--soc\_axi\_perf/

|--rtl/

|--CONFREG/

|--confreg.v

|--myCPU/ 目录：myCPU性能源文件

|--alu\_select.v

|--ALU.v

|--AXI\_interface.v

|--Branch\_Junmp.v

|--cache\_axi.v

|--conflict.v

|--controller.v

|--cp0\_reg.v

|--datapath.v

|--DataRAM.v

|--dcache\_with\_uncache.v

|--decoder.v

|--exception.v

|--extend.v

|--pc.v

|--pipline\_stage.v

|--redirect\_id.v

|--register\_select.v

|--register.v

|--ram\_wrap/

|--axi\_wrap\_ram.v

|--soc\_axi\_lite\_top.v

|--xilinx\_ip/

|--axi\_crossbar\_1X2/

|--axi\_crossbar\_1X2.xci

|--axi\_ram/

|--axi\_ram.xci

|--clk\_pll/

|--clk\_pll.xci

|--run\_vivado/

|--mycpu\_prj1/

|--mycpu.xpr

|--pref.bit 性能测试比特流

|--run\_allbench.tcl

|--soc\_lite.xdc

|--testbench/

|--mycpu\_tb.v

|--soft/

|--func/

…

|--memory\_game/

…

|--perf\_func/

…

## （二）设计演示结果

CPU最终设计RTL如图 11所示。

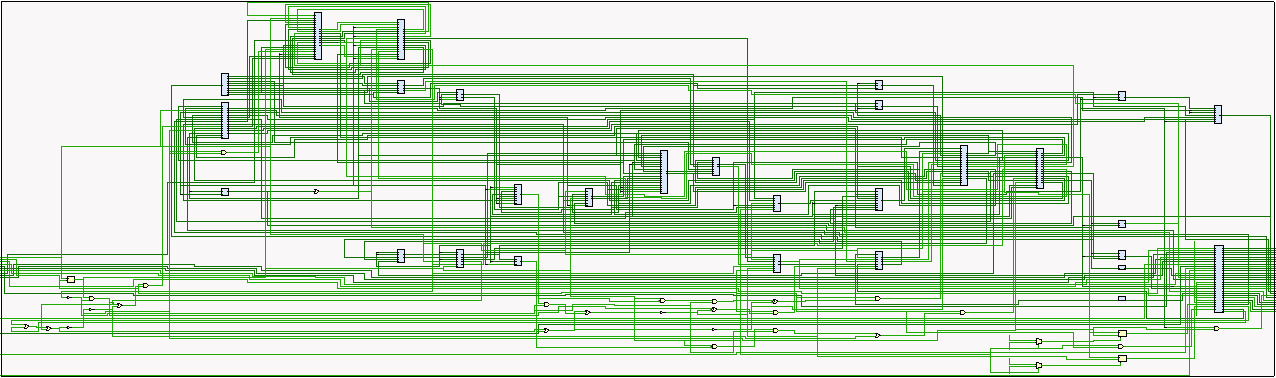


图 11 CPU——RTL结构

1. **axi接口功能测试**

**仿真：**功能测试一共89项，仿真结果见图 12。

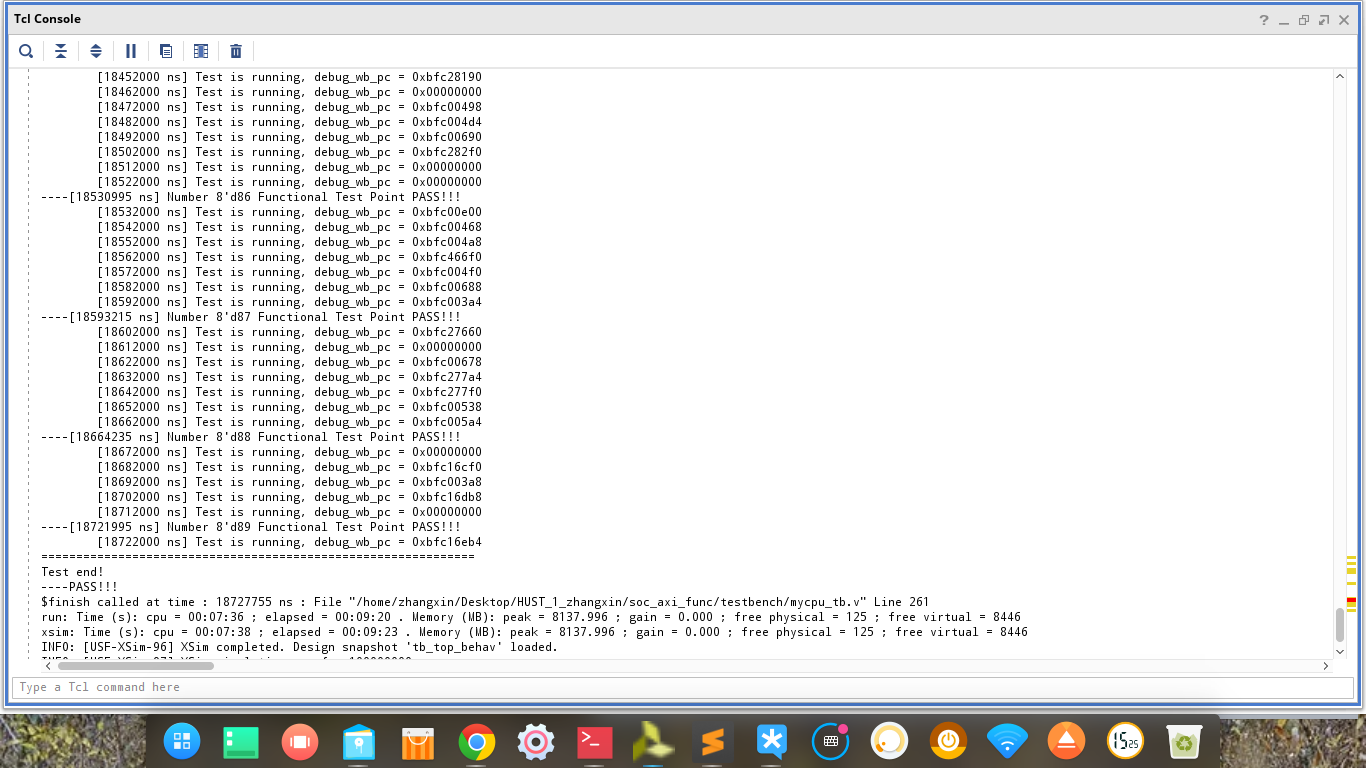


图 12 AXI接口功能仿真

**上板：**不同的随机数种子的上板结果见图 13 axi接口功能测试上板——随机数1、图 14 axi接口功能测试上板——随机数2、图 15 axi接口功能测试上板——随机数3、图 16 axi接口功能测试上板——随机数4、图 17 axi接口功能测试上板——随机数5、图 18 axi接口功能测试上板——随机数6、图 19 axi接口功能测试上板——随机数7。

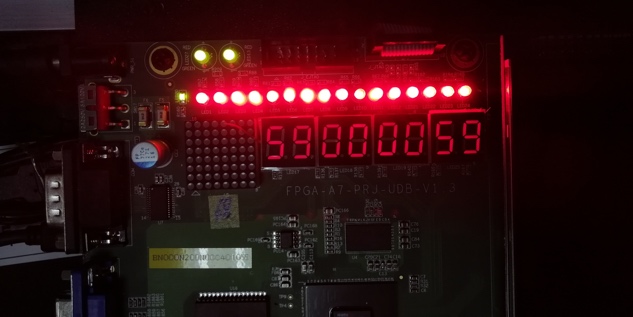
****

图 13 axi接口功能测试上板——随机数1

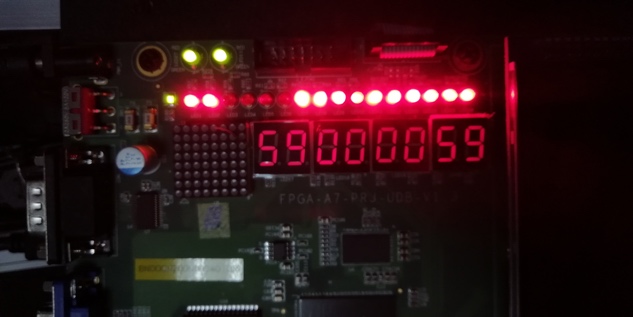
****

图 14 axi接口功能测试上板——随机数2

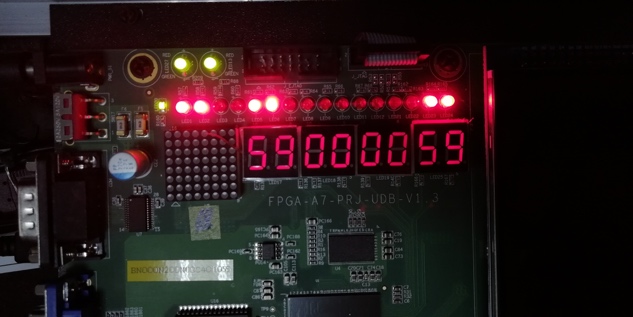
****

图 15 axi接口功能测试上板——随机数3

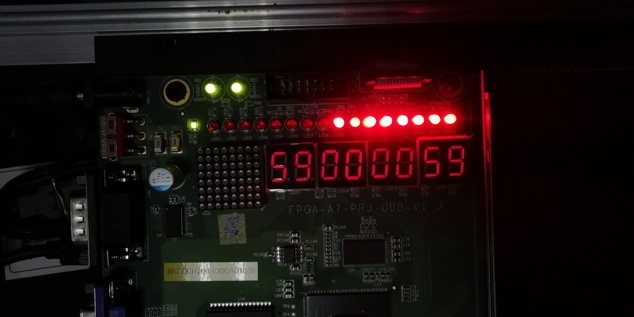
****

图 16 axi接口功能测试上板——随机数4

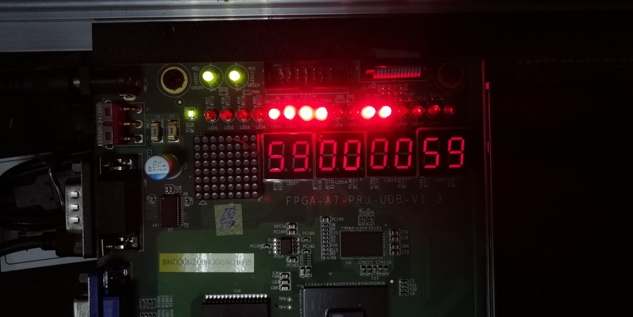
****

图 17 axi接口功能测试上板——随机数5

****

图 18 axi接口功能测试上板——随机数6

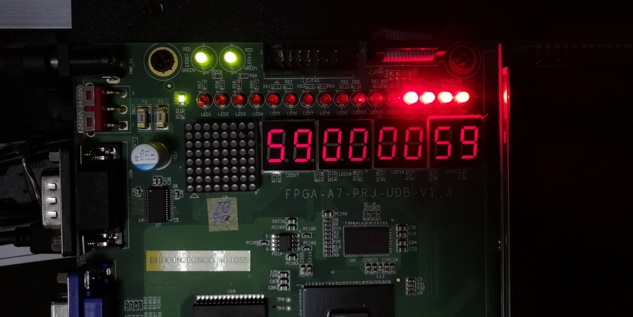
****

图 19 axi接口功能测试上板——随机数7

1. **性能测试**

**仿真：**性能测试分stream copy、bitcount、bubble sort、coremark、crc32、dhrystone、quick sort、select sort、sha、string search十个性能测试部分。仿真结果见下图 20 stream copy仿真、图 21 bitcount仿真、图 22 bubble sort仿真、图 23 coremark仿真(a)、图 24 coremark仿真(b)、图 25 crc32(a)、图 26 crc32仿真(b)、图 27 dhrystone仿真(a)、图 28 dhrystone仿真(b)、图 29 string search仿真(a)、图 30 string search仿真(b)、图 31 select sort仿真、图 32 quick sort仿真、图 33 sha仿真。

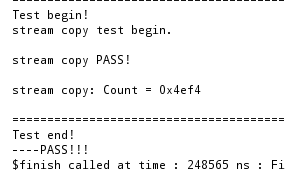


图 20 stream copy仿真

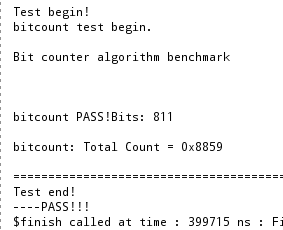


图 21 bitcount仿真

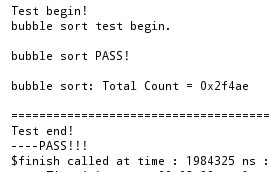


图 22 bubble sort仿真

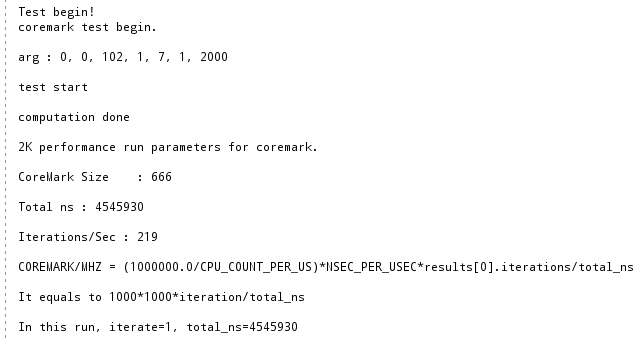


图 23 coremark仿真(a)

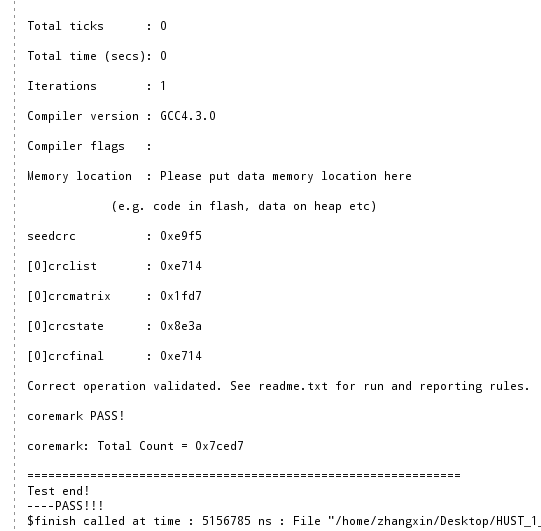


图 24 coremark仿真(b)

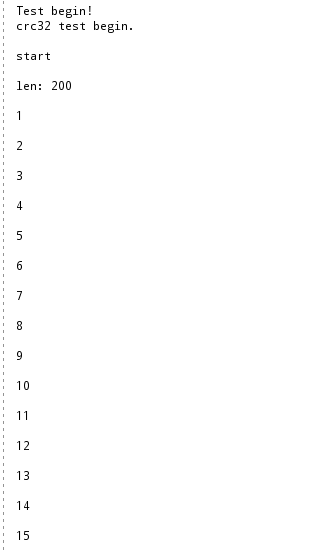


图 25 crc32(a)

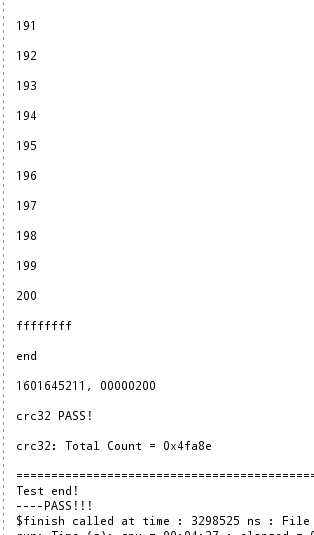


图 26 crc32仿真(b)

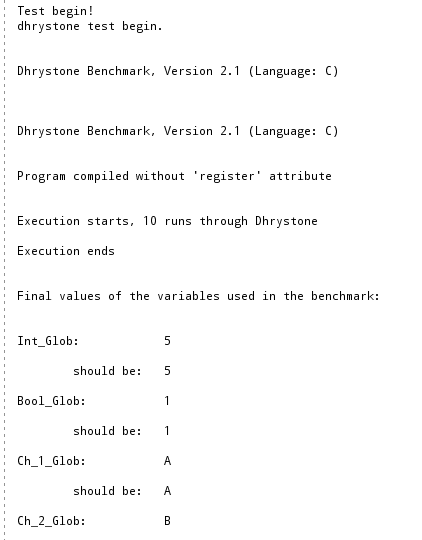


图 27 dhrystone仿真(a)

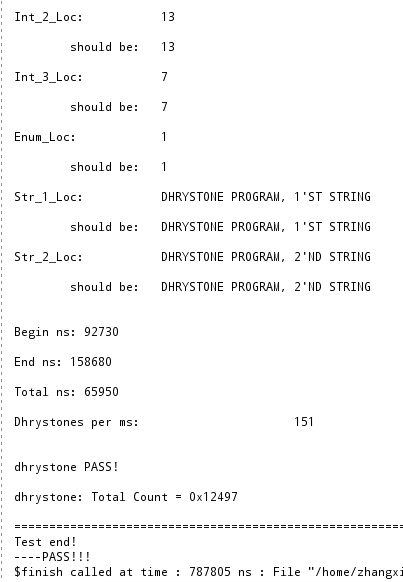


图 28 dhrystone仿真(b)

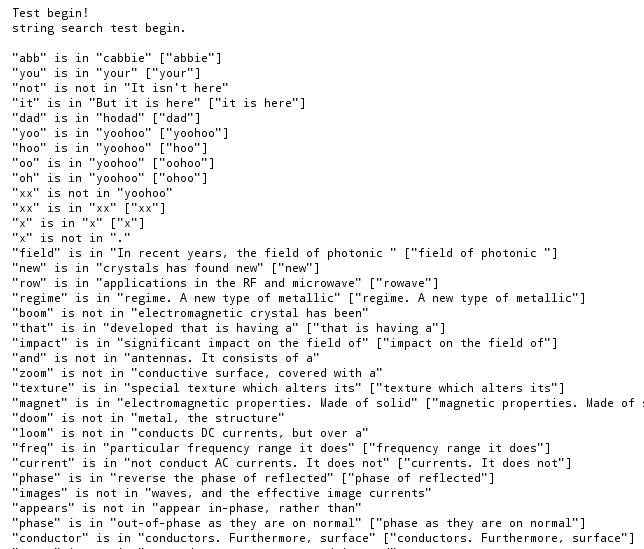


图 29 string search仿真(a)



图 30 string search仿真(b)

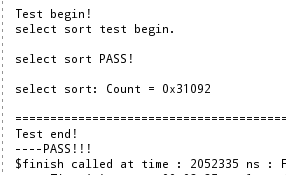


图 31 select sort仿真

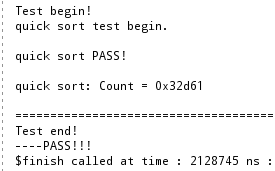


图 32 quick sort仿真

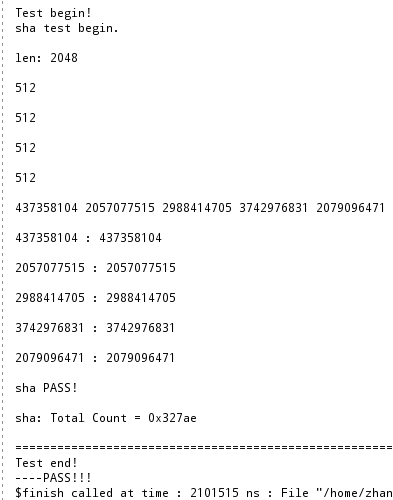


图 33 sha仿真

**上板：**性能测试stream copy、bitcount、bubble sort、coremark、crc32、dhrystone、quick sort、select sort、sha、string search十个性能测试部分，上板结果见图 34 stream copy上板、图 35 bitcount上板、图 36 bubble sort上板、图 37 coremark上板、图 38 crc32上板、图 39 dhrystone上板、图 40 quick sort上板、图 41 select sort上板、图 42 sha上板、图 43 string search上板。

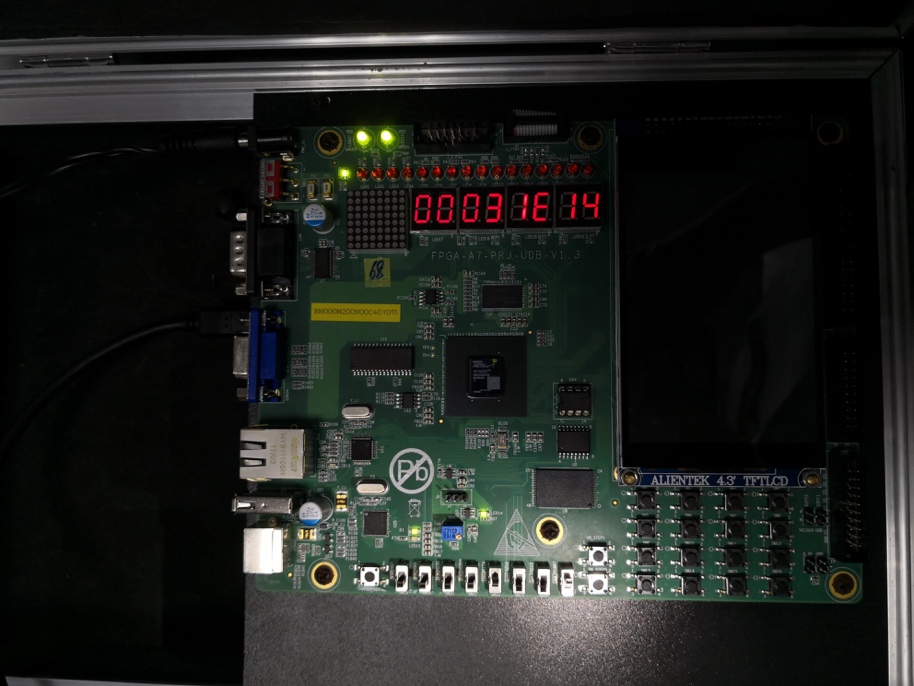
****

图 34 stream copy上板

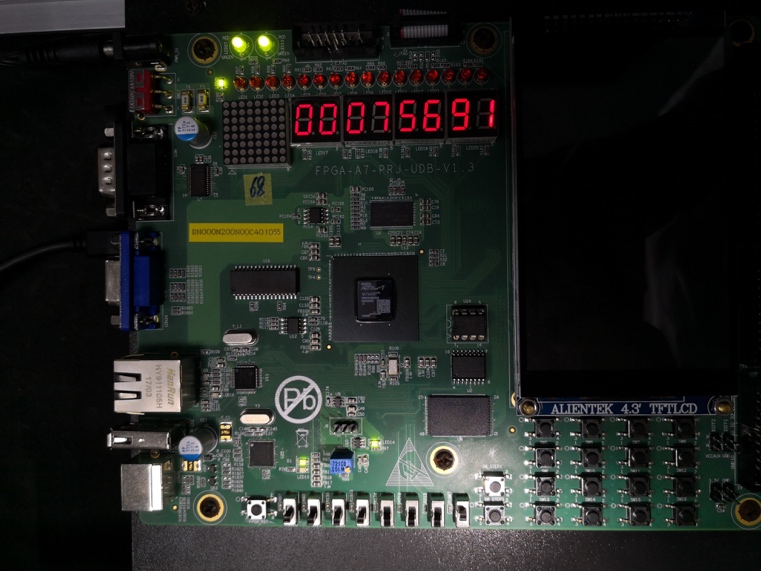
****

图 35 bitcount上板

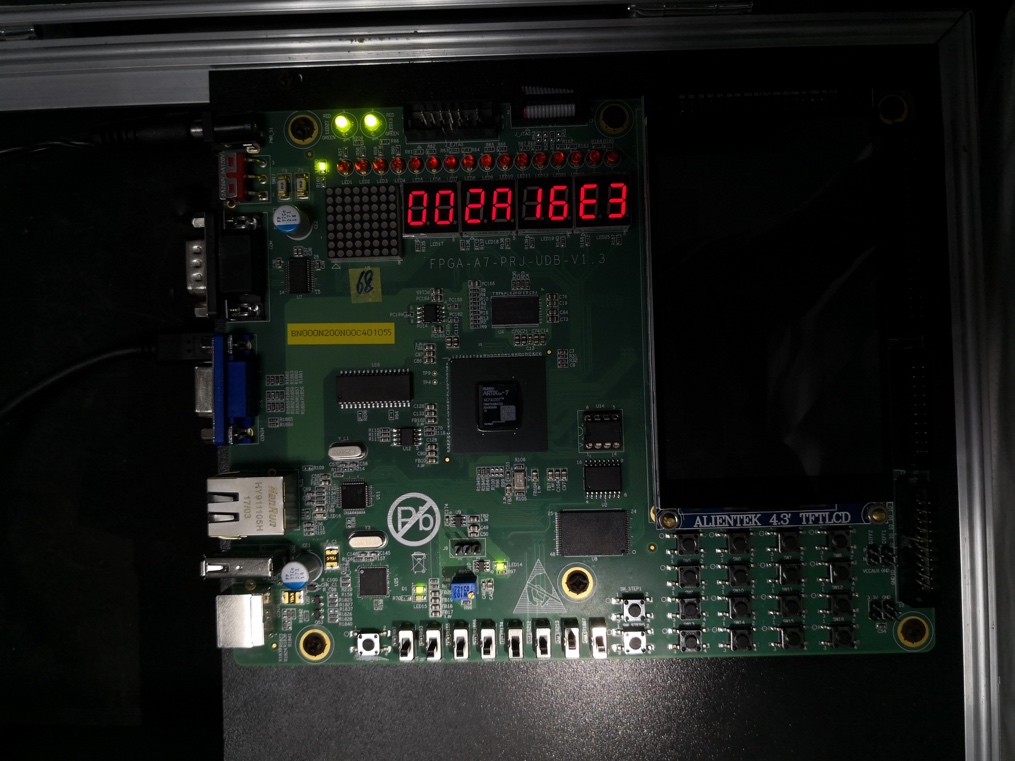
****

图 36 bubble sort上板

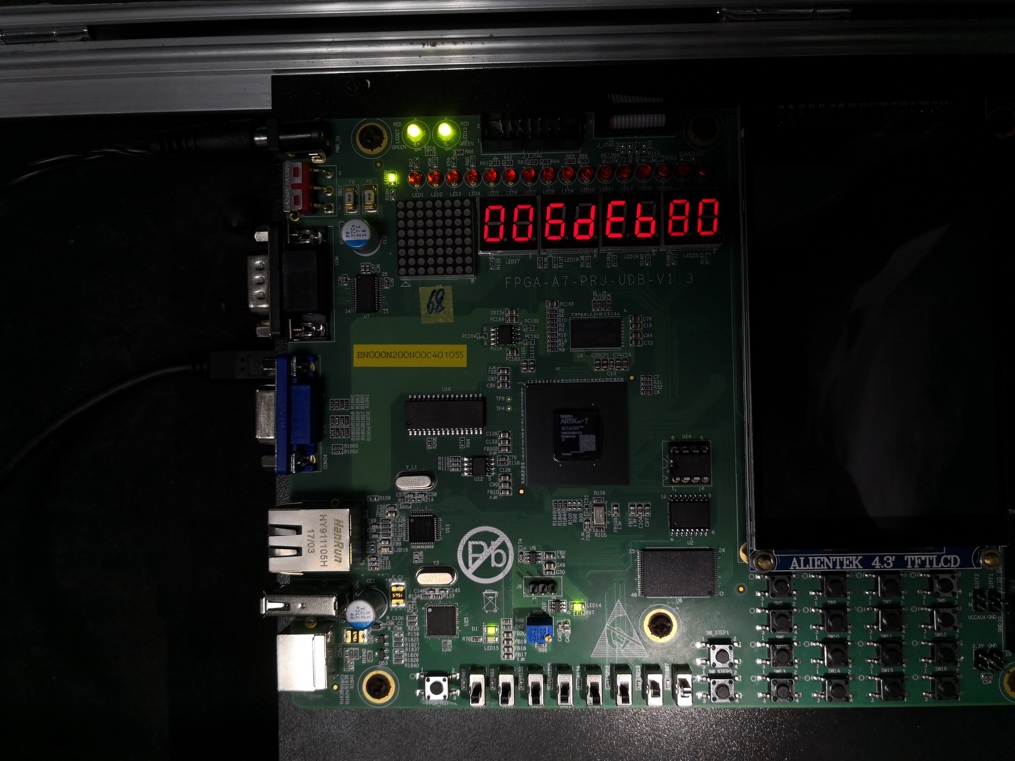
****

图 37 coremark上板

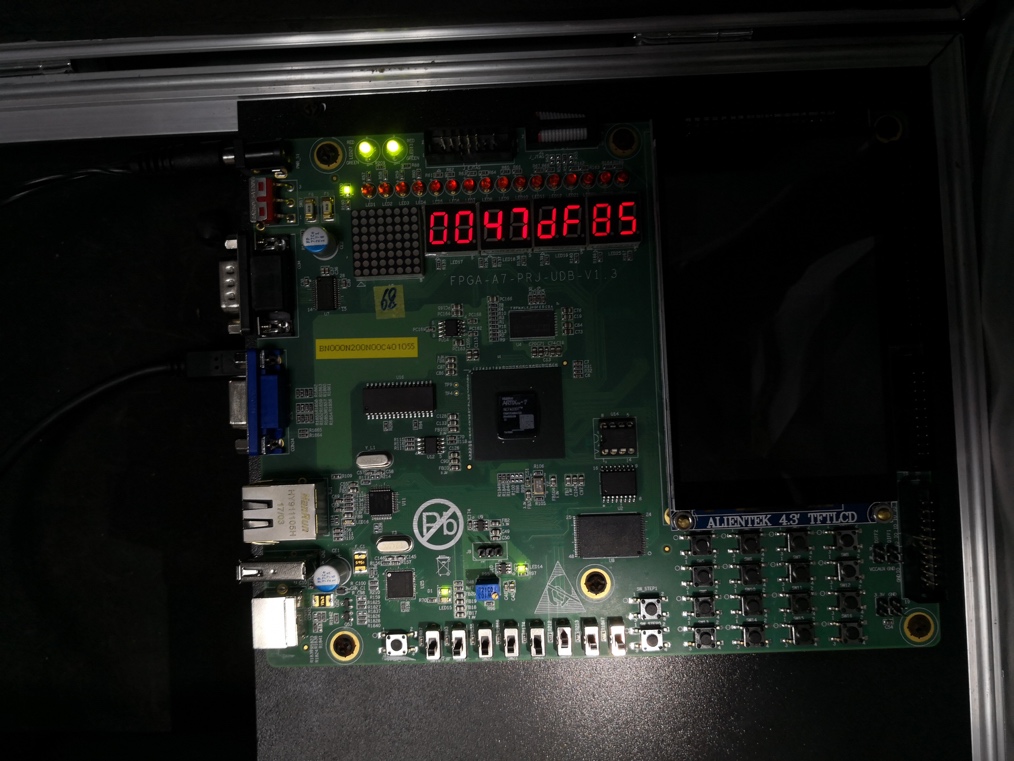
****

图 38 crc32上板

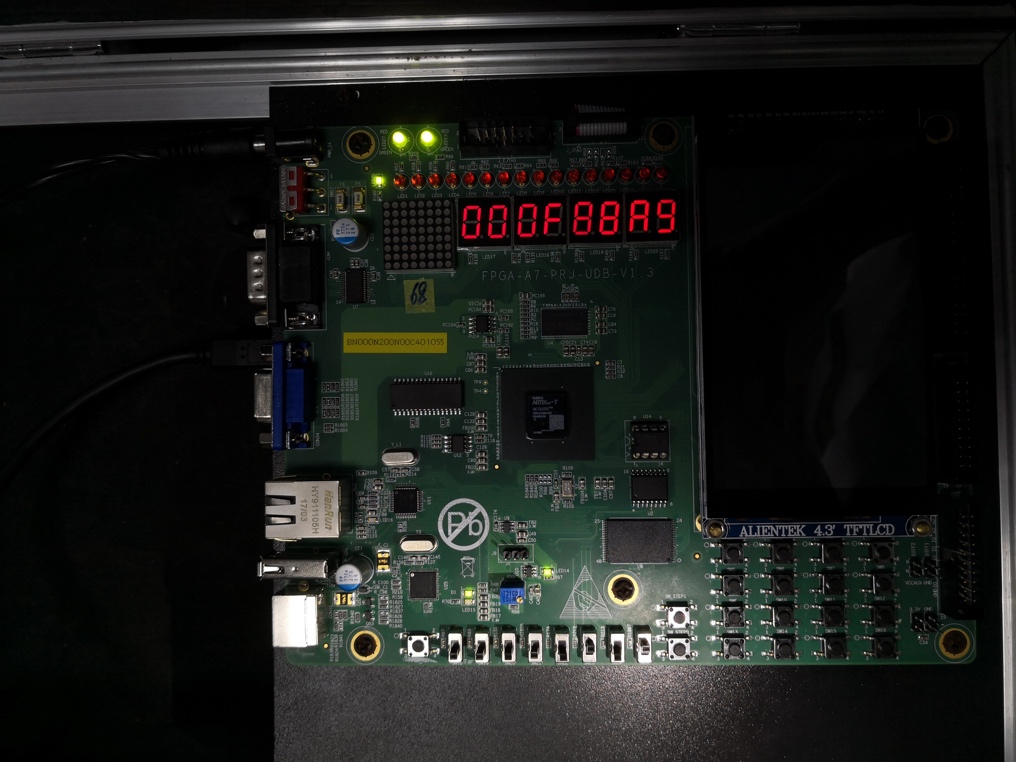
****

图 39 dhrystone上板

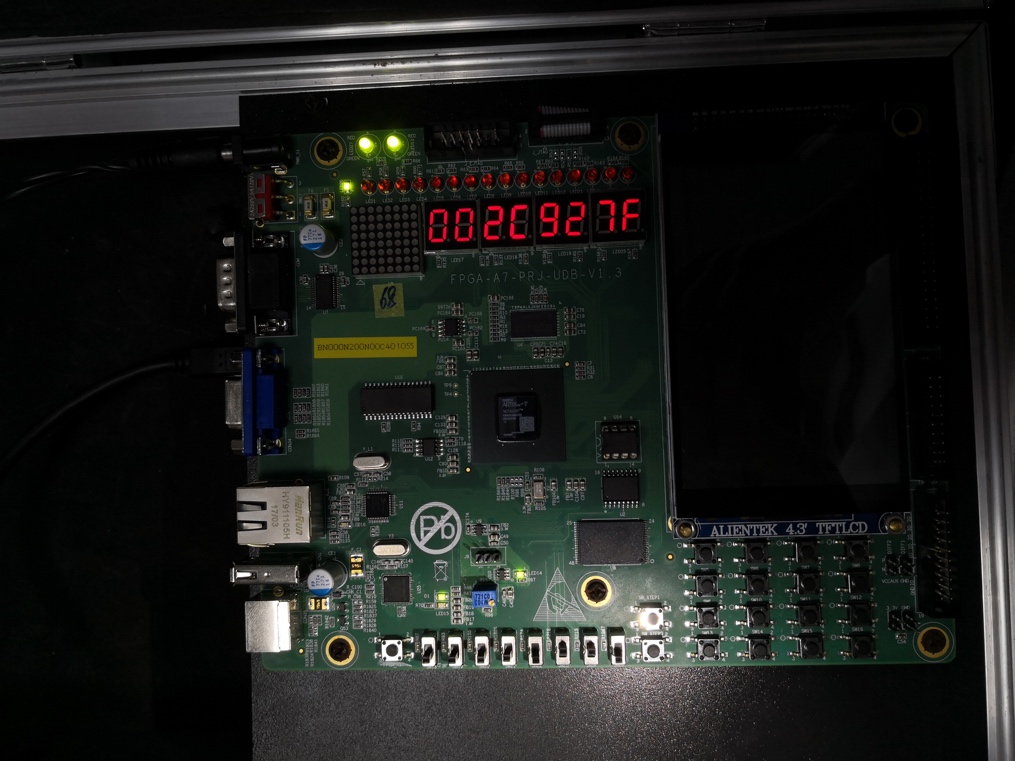
****

图 40 quick sort上板

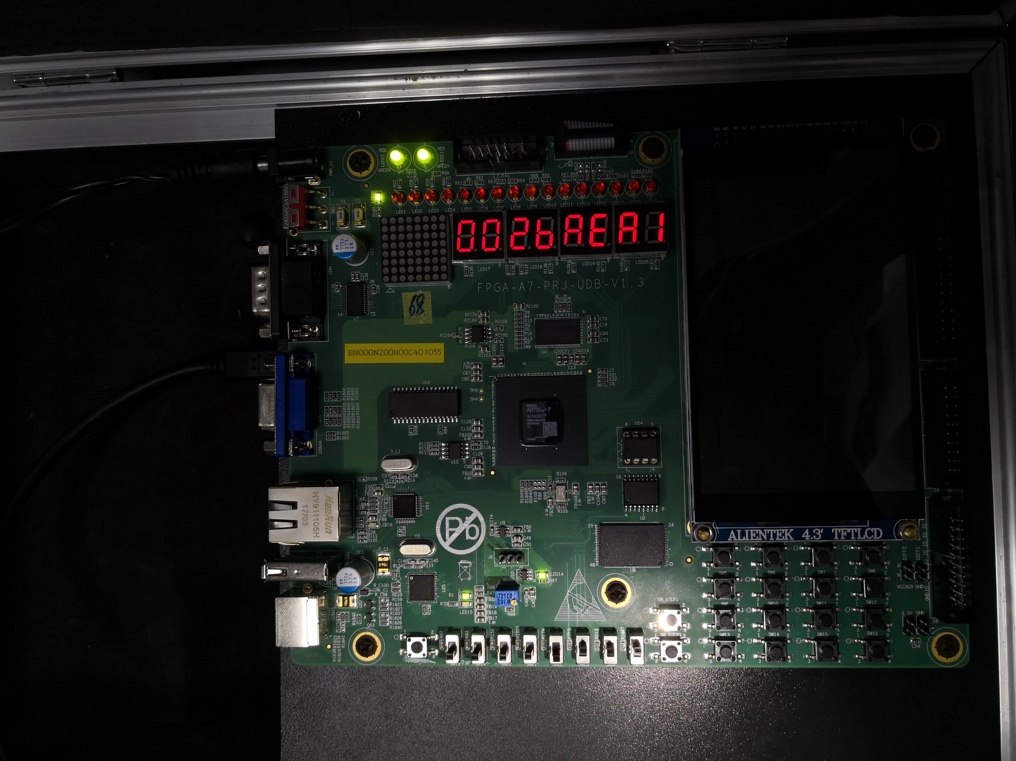
****

图 41 select sort上板

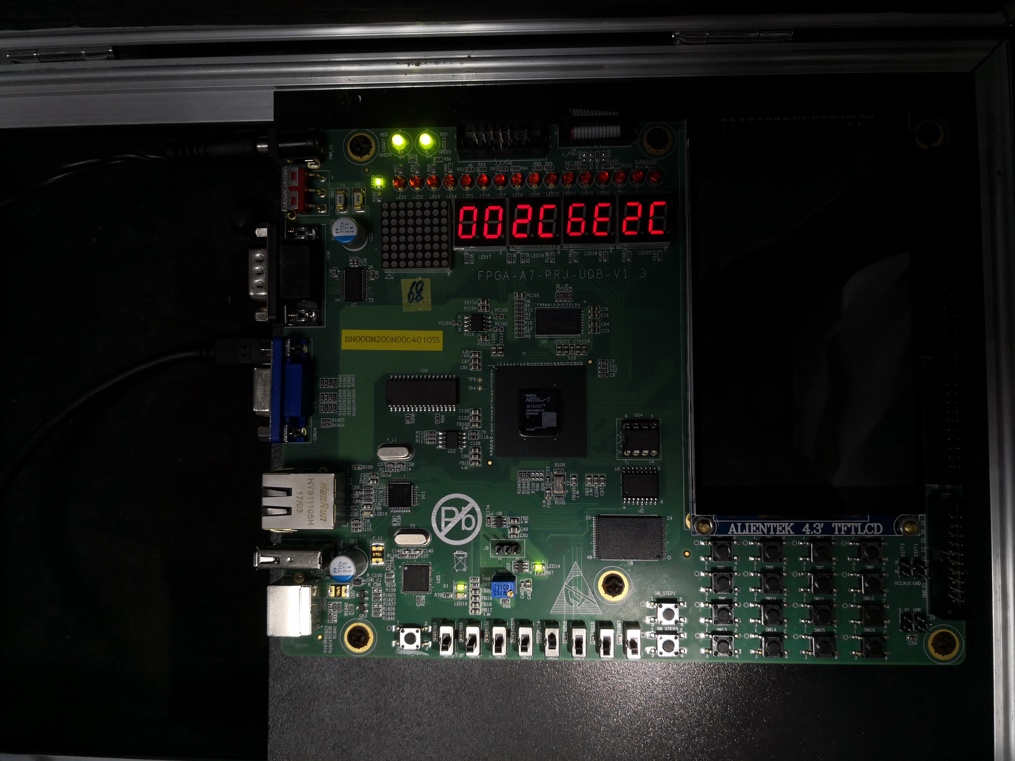
****

图 42 sha上板

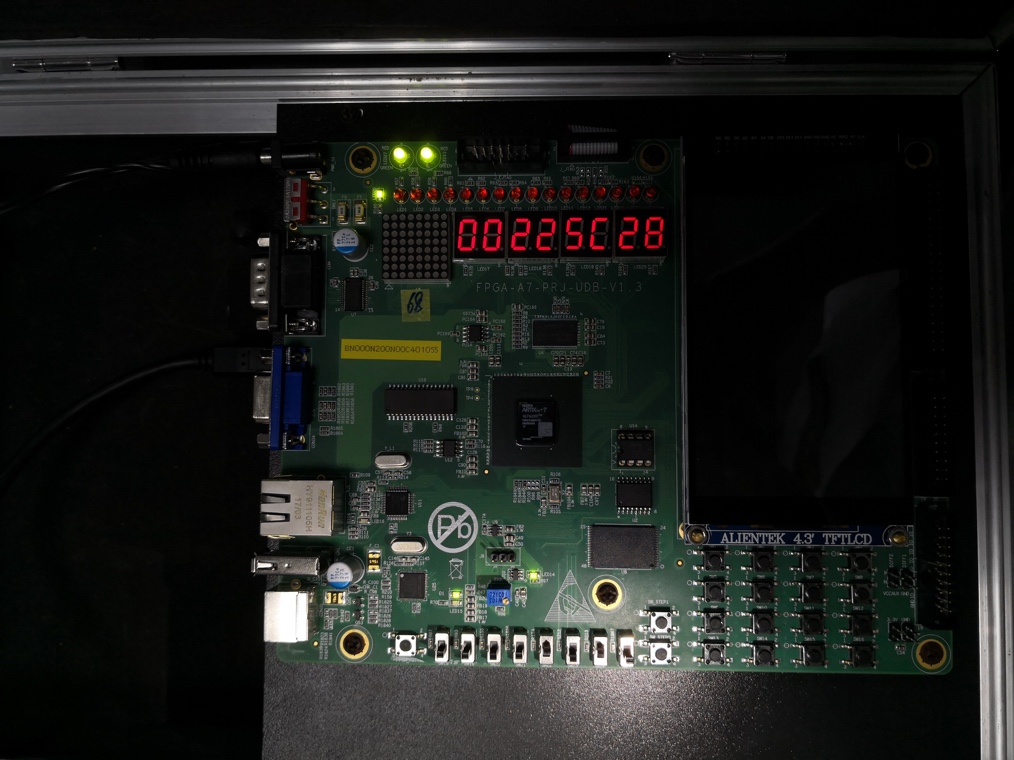
****

图 43 string search上板

功能、性能测试最终得分如下图 44，由于计时器的影响，结果与上板截图结果有合理的差异。



图 44 功能、性能测试得分

# 四、参考设计说明

1. CP0模块参考NaiveMIPS CP0部分结构；
2. 多周期除法设计参考NaiveMIPS除法部分；
3. 异常处理模块设计参考NaiveMIPS exception模块；
4. 通用寄存器结构设计参考NaiveMIPS；
5. AXI模块关键信号参考NaiveMIPS confreg模块；

# 五、参考文献

A.专著：

[序号] 主要责任者. 文献题名[M]. 出版地: 出版者, 出版年: 起止页码(可选项).

[1] 陈进才、刘乐善.《微型计算机接口技术》[M].北京: 人民邮电出版社, 2015.

[2] 秦磊华、吴非、莫正坤. 《计算机组成原理》[M].北京: 清华大学出版社,2011: P110-P188.

[3] 张晨曦、王志英等. 《计算机系统结构教程(第2版)》[M].北京: 清华大学出版社,2009: P189-P239.