# “SINO”——面向P8的mips微系统软件规范

游子诺 17373321

目录

[“SINO”——面向P8的mips微系统软件规范 1](#_Toc534124320)

[一、 系统概览 1](#_Toc534124321)

[二、 系统架构与任务 2](#_Toc534124322)

[三、 通用寄存器使用规范 7](#_Toc534124323)

[四、 外部设备 8](#_Toc534124324)

[五、 系统开发建议与提示 10](#_Toc534124325)

## 系统概览

SINO是基于MIPS-C5指令集而开发的系统，硬件使用5级流水线CPU（带有CP0处理器且支持部分异常与外部中断）。

内部设备有支持持续中断和断点中断的定时器。外部设备有9位8段LED数码显示管、32位LED灯、8位用户按键、2组32位微动开关和8位UART串口通信。

**SINO系统软件规范具有以下特点**：

* 1. **模块化：**软件功能模块化，各个任务写权限划分明显，使用了寄存器冻结减少外部中断这种“不可预知”的行为对正在运行中的主程序稳定性的干扰，维护更新方便。
  2. **异常锁死与显示**：系统不支持出现异常后的正常功能运行**（不是CPU不支持中断异常）**，当出现内部异常后将产生锁死并显示相关信息，只有Reset信号能够解除，此特点旨在帮助开发者精确定位异常代码，快速修复。
  3. **权限清晰，更多规范：**系统基于通用mips寄存器操作规范上，结合系统构建难度，规范了如异常寄存器、外设状态寄存器等寄存器使用规则，使得各个子程序写权限更明确。与此同时，由于规范增加，开发使用的自由度将下降。**为了功能正常，请务必遵守对应的限制与规范**。
  4. **调试方便：**得益于对中断和异常的寄存器中介的引入，handler与其他程序的数据交流被“严格地限制”在几个部件中。因此，通过更改这几个部件的值，可实现外部信号模拟调试的全覆盖，从而使得在handler执行正确的前提下，在Mars软件中实现对处理程序的正确性调试（而无需硬件仿真）。

## 系统架构与任务

SINO系统任务如下图所示，分为主控程序（Main）、初始化（Init）、调试（Debug）、功能（Function）、中断异常处理（Handler）、异常锁死（Error）6个部分，并使用macro语句定义了对Switch、UserKey的读操作，对Display、LED、UART的写操作以对简化重复性高频操作。

1. Main主控程序

图 1 系统程序框架

Handler

Main

Init

Debug Set

Function

Error

Main主控程序是程序的入口，也是主干线，所有正常任务执行完毕后都将回到主控程序，而后进入下一个任务。大体执行流程：初始化-（调试设置）-功能任务-异常检查-若无异常则再次匹配功能任务，否则锁死。

此外，主控程序还起到“轮询”用户按键并跳转对应处理函数的功能。其写权限使用白名单模式：

表格 1 Main写权限白名单

|  |  |  |
| --- | --- | --- |
| 状态 | 对象 | 描述 |
| 允许 | s0 | 更新并保存User Keys的值 |
| 允许 | t0-t7 | 跳转与预处理辅助（随用随弃） |
| 允许 | ra | 跳转至对应功能时使用 |
| 允许 | t8-t9 | 提醒，仅macro可用（外设写权限不包括） |

1. Init初始化程序

Init初始化程序是程序首次进入或用户按键改变时（暂未支持）触发的功能，其具体内容视情况而定，因而权限约束不确定。

当前支持的初始化有：CP0的SR寄存器中断权限设置、计时器初始化、显示器\*2归零、$gp, $fp归零。

1. Debug-Set调试设置程序

Debug-Set调试设置程序旨在使用Mars模拟外部设备状态，从而验证MIPS软件功能的正确性，支持的调试设置包含但不限于以下内容：

* 1. 设置User Keys，Switch状态。
  2. 预设中断状态寄存器和异常寄存器。
  3. 预设显示内容。

1. Function功能程序

Function功能程序是系统的核心，其执行具体的真实功能，每种情况下都是不同的，其写权限约束采用黑名单方式。

表格 2 Function写权限黑名单

|  |  |  |
| --- | --- | --- |
| 状态 | 对象 | 描述 |
| 禁止 | s0 | 仅能读取和反应 |
| 禁止 | s2 | 异常状态寄存器，不可写 |
| 禁止 | k0-k1 | 仅handler可用 |
| 禁止 | gp、fp、at | 用户禁止使用 |

表格 3 Function写权限提醒

|  |  |  |
| --- | --- | --- |
| 状态 | 对象 | 描述 |
| 提醒 | ra | 允许，在调用子函数时使用，但同t0-t7寄存器一样，请注意进行备份和恢复。 |
| 提醒 | t8-t9 | 允许，仅macro使用（外设写权限不包括） |

1. Error内部异常检查程序

Error内部异常检查程序读取异常寄存器（s2）的内容判断是否有内部异常，若出现异常将进入锁死——死循环模式，在Display部件上显示异常寄存器的值（包括异常使能、异常代码和EPC值），其写权限采用白名单模式：

表格 4 Error写权限白名单

|  |  |  |
| --- | --- | --- |
| 状态 | 对象 | 描述 |
| 允许 | t0 | 辅助寄存器 |
| 允许 | Display | 显示异常状态 |
| 允许 | t8-t9 | 提醒，仅macro可用（外设写权限不包括） |

表格 5 P8内部异常识别支持总表



1. Handler中断异常处理程序

Handler是通用的中断异常处理程序，适用规定异常、计时器中断和外部设备中断三类。处理过程为：保存现场-判断中断异常类型-执行处理-恢复现场并跳回，

* 1. 对于内部异常：异常寄存器异常位将置位，记录ExcCode异常代码和EPC值，并跳回异常指令的下一条指令。
  2. 对于计时器中断：将关闭计时使能和中断使能，中断寄存器（s1）“计时器中断位”置位，并跳回受害指令。（请注意！Handler只有对计时器停止的权限，而对计时器的后续处理交给Function程序。）
  3. 对于UART中断：将读取UART的LSR寄存器判断读取数据的有效性，若有效将会把8位数据写入中断寄存器（s1）中，将中断寄存器（s1）的“UART中断”置位，并跳回受害指令。（请注意！Handler对UART没有写入数据的权限，对UART后续处理交给Function程序。）

对于Handler的写权限，采用白名单模式：

表格 6 Handler写权限白名单

|  |  |  |
| --- | --- | --- |
| 状态 | 对象 | 描述 |
| 允许 | t0-t7 | 功能不限，但需要负责现场的保存和恢复 |
| 允许 | s1 | 中断状态寄存器，用于中断设备和数据记录 |
| 允许 | s2 | 异常状态寄存器，EPC、ExcCode和异常状态置位 |
| 允许 | sp | 用于保存和恢复现场 |
| 允许 | k0-k1 | handler操作专用寄存器，用于读取CP0寄存器 |
| 允许 | EPC | 选择跳回指令 |
| 允许 | Timer | 仅限停用计时器（中断使能和计数使能置0） |
| 允许 | t8-t9 | 提醒，仅macro可用（外设写权限不包括） |

1. Macro定义

Macro语句在SINO系统中用于对常用的外部设备的读写操作进行简化，除了对应外部设备的写权限，Macro语句对寄存器的**写权限仅有t8和t9**，目前系统支持的Macro语句及其功能描述、调用者如下表所示：

表格 7 Macro语句格式与功能

|  |  |  |
| --- | --- | --- |
| 语句格式 | 功能描述 | 调用者 |
| read\_switch %a %b | 将当前A、B两组switch开关的值分别赋给%a %b | Function |
| read\_key %a | 将通用用户控制按键的值以最低字节赋给%a | Main |
| write\_uart %t | 将%t的最低字节值传送给UART串口驱动以便传输。  （写入时，Macro会检查UART当前输出数据位空闲状态，若不空闲将循环等待。） | Function |
| write\_display %a %b | 将%a的32位数据显示于display的数值位，将%b的低4位显示于display的符号位。  （对于符号位的显示，%b低4位中的最高位若为1则将显示F表示错误，否则低4位中最低位若为1则将显示负号，其他情况不显示。） | Function |
| write\_led %t | 将%t的32位值写入LED32位灯上。 | Function |

## 通用寄存器使用规范

在mips通用使用规划的基础上，SINO系统“面向P8需求”增加了相关寄存器的规范和限制，以下表格为对应整体设置：

表格 8 寄存器功能总表



在功能存在限制的寄存器中，有两个较为特殊的寄存器：s1和s2寄存器，参考Timer、UART等设备寄存器的设计风格，SINO系统中将两个寄存器分别作为外部设备状态寄存器和异常记录寄存器，该规范旨在划分系统各部分处理程序的权限，力求仅更改Function部分的代码即可实现几乎所有的定制化功能，利用寄存器中介方便对外部中断的调试；同时，32位拆分使得寄存器空间更充分利用。

下表为外部设备状态寄存器规范：

表格 9 外部设备状态寄存器（$s1）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Bit Name | Bit No | Description | R/W | VAR |
| Timer | 0 | 当handler收到timer中断后置1  当function响应timer中断后置0 | R/W | 0 |
| UART | 1 | 当handler收到uart有效数据后置1  当function响应uart有效数据后置0 | R/W | 0 |
| uart\_byte | 15:8 | 保存uart有效数据 | R/W | 0 |
| Reserved | 31:16 | 保留 | - | 0 |

下表为异常记录寄存器规范：

表格 10 异常记录寄存器（$s2）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Bit Name | Bit No | Description | R/W | VAR |
| Error\_EPC | 15:0 | 保存内部异常指令的地址 | R/W | 0 |
| Reserved | 25:16 | 保留 | - | 0 |
| Error\_Code | 30:26 | 保存内部异常代码 | R/W | 0 |
| Error | 31 | 内部异常出现时置1 | R/W | 0 |

## 外部设备

SINO系统目前共有5个外部设备，1个内部设备，功能概述如下：

1. Timer：计时器，中断访问，支持持续中断和断点中断。
2. UART：串口通信，中断访问，8 Bits位宽，波特率为9600，不支持奇偶校验。
3. Switch\*2：两组32位共64位微动开关，负信号有效（CPU内部已转为正信号有效）。
4. User Key：8位通用按键开关，常用于状态控制与切换，负信号有效（CPU内部已转为正信号有效）。
5. Display：9位8段数码管，共阳极接入，由1（符号位）+4（数码管组1）+4（数码管组2）组成。
6. LED：32位led灯，共阳极接入。

具体各设备驱动地址的划分、功能和规范划分如下表：

表格 11 外部设备驱动地址规划与功能总表

|  |  |  |  |
| --- | --- | --- | --- |
| Device | Addr[15:0] | Description | R/W |
| Timer | 7F00 | 控制寄存器  Bit0: 1-允许计数，0-停止计数  Bit1-2: 0-模式0，1-模式1  Bit2: 1-允许中断，0-禁止中断  其余位保留，恒为0，写无效 | R/W |
| 7F04 | 初始值寄存器 | R/W |
| 7F08 | 计数器寄存器 | R |
| UART | 7F10 | 数据寄存器（低8位有效）  当WE\_I信号无效时，表明读取UART数据  当WE\_I信号有效时，表明写入UART数据  其余位保留，恒为0，写无效 | R/W |
| 7F14-7F1C | 保留，恒为0，写无效 | - |
| 7F20 | 线路状态寄存器  Bit0: 1-接受数据有效，0-接受数据无效  Bit5: 1-发送保持器空，0-发送保持器非空  其余位保留，恒为0 | R |
| 7F24 | 接受除数寄存器，具体设置参见MiniUart设计文档 | R/W |
| 7F28 | 发送除数寄存器，具体设置参见MiniUart设计文档 | R/W |
| Switch | 7F2C | 微动开关（正有效）数据：第0-3组 | R |
| 7F30 | 微动开关（正有效）数据：第4-7组 | R |
| LED | 7F34 | LED显示驱动信号（共阴级形式） | R/W |
| Display | 7F38 | 数码管数值位显示驱动信号（共阴极形式） | R/W |
| 7F3C | 数码管符号位显示驱动信号（共阴极形式）  Bit0: 1-非异常时显示负号，0-非异常时不显示  Bit3: 1-显示F，0-显示由Bit0决定  其余位保留，恒为0，写无效 | R/W |
| User Key | 7F40 | 通用用户按键数据（低8位）  其余位保留，恒为0 | R |

## 系统开发建议与提示

1. 当新增自定义功能后，请不要急于仿真和综合，在条件允许的情况下应首先测试汇编软件的执行流程和写入数据及位数的正确性。就目前需求，所有的外部输入设备单周期效果均能够的通过更改主存和寄存器的值加以实现，若需增加动态多周期模拟，还需要进行变化。
2. 在实现自定义功能的过程中，应最大限度地通过只更改Main和Function部分来实现，一为保证其他部分代码的正确性，二为减少~~“触发”~~更多的硬件bug。强烈不建议更改Handler中的处理程序（尤其是UART相关代码），因为中断调试无法在Mars中实现，需要通过仿真甚至板载进行，耗时很长，收效甚微。
3. 当汇编软件测试成功后，应使用仿真进行执行语句的大体跟踪，避免硬件bug。
4. 外部中断的数据应该在主程序的一个生命周期中保持稳定，为此需要使用额外的寄存器或内存资源进行保存。同时，通常情况中，外部中断传入的数据也只应该在程序中“一个完整的生命周期”（假设任务是一周期能处理完成的）。即，对于不需要外部中断数据的系统状态，中断数据应及时予以清理。同时，对于不同状态中断需求，也可以通过更改CP0中断使能实现。