SensorBoard CMD设计

2014年7月2日
11:34

目录

存储位置

存储在SensorBoard上的EEPROM中，偏移位置为0x8

Tips:EEPROM是有Page的概念的，但是为了通用性考虑，这里忽略Page概念，不考虑是否跨页。page由ROMDriver管理。
同时，也有可能更改存储芯片，所以这里的偏移位置只是针对存储器的开始来讲。具体实现都由ROMDriver来封装。

分隔符

- 命令集分割符
  - 使用 ||区分。

所谓命令集，是指attachCMD或者captureCMD或者detachCMD。每个CMD可以包含很多命令。而命令集分隔符是用来区分这些CMD集的。
每个命令集必须由||包围，即命令集开头和结尾必须是||。

- 命令分隔符
  - 使用;(分号)区分。

每条命令必须包含命令分隔符;，即使命令集中仅有一条命令。

- 参数分隔符
  - 使用空格分隔。

example：||I2C 1 W 0x10 0x3;||GPIO 2 O H;||;||

命令存储格式

- 命令采用字符格式存储；
- 区分大小写。

命令返回

有可能一条命令需要使用上一条命令的返回结果。目前限制，每条命令只能引用上一条命令的结果，无法引用上一条之前的命令的结果。

※ 返回结果分类

- 数据型。返回结果为char或者int。
- 指针型。返回结果为一个缓冲区指针。

※ 返回结果

? 返回结果长度固定为4字节？还是可变？

可变长度比较难管理，每条命令的返回结果具体有多长，需要参照手册，比较麻烦。

不如固定为4字节。考虑如下：

- 对于只返回一个结果的命令，一般情况下，4字节足够盛放。
- 对于返回多于一个结果的命令，此时可以使用缓冲区返回，返回结果为指向该缓冲区的指针，在32位机的情况下，正好是4字节。

返回结果如何存储？

- √ 位于ROM固定位置，每条命令执行完后，将结果都放于此位置。
  - 实现：只需定义一个全局变量即可。也不算浪费空间。
- × 动态从堆上分配空间，并将返回结果存入。
  - 每条命令都分配，使用完后释放。
    - 浪费又麻烦
  - 模块初始化时分配，模块卸载时再释放。
    - 与第一种情况类似，区别是不占用ROM，而是RAM。为了方便，还是使用第一种方法吧。不用malloc。

指针型返回结果的长度

数据型返回结果固定为4字节，但是对于返回缓冲区指针的指针型返回结果，如何确定其所指向的缓冲区的长度以及数据类型(char*、int*)？

数据类型

由引用符号决定，定义如下：

- **$P1** 表示缓冲区中的数据是以一个字节为单位的，例如char*类型
- **$P2** 表示缓冲区中的数据是以两个字节为单位的，例如int16*类型
- **$p4** 表示缓冲区中的数据是以四个字节为单位的，例如int32*类型
  - ! Tips：暂不支持结构体类型

缓冲区长度

- 由命令决定。每条命令具体返回多少数据，在命令手册中指定，然后引用者必须确保不会超过缓冲区长度。
  - × 很容易出错。
- 在缓冲区第一个字节指定缓冲区的大小。
  - √ 虽然会浪费一个字节，但是不容易出错，比如缓冲区越界访问。

返回结果的引用

在命令中用 **$** 字符代表上一条命令的返回结果。

$P 表示以指针的方式来引用返回结果，后面跟数字表示指针类型，具体参考指针引用数据类型

$D 表示以数据的方式来引用返回结果，即将返回结果强制转换为数据，如果上一条命令的返回结果为指针的话。

命令列表

- **GPIO**
  - **GPIO R GPIONumber**
  - **GPIO W GPIONumber State**

参数说明：

- R/W 表示读或者写GPIO端口，R表示Read，W表示Write
- GPIONumber GPIO端口号。0~31，但是有些端口用途已固定，无法使用GPIO。具体哪些可以端口请参考引脚资源图
- State: 只在输出有效，表示GPIO输出状态，0表示低，1表示高电平

返回结果：

Read返回当前输入状态，Write返回当前输出状态

- **I2C**
  - **I2C busNum R/W slvAddr subAddr data.....**

参数说明：

- busNum ：表示I2C总线号，目前支持两路I2C，所以busNum的取值为0,1
- R/W ：表示读取或者写入
- slvAddr ：I2C设备从地址
- subAddress：设备子地址
- Data ：要写入的数据。当为Read时，表示连续读取多少次；当为Write时，表示要写入的数据

- **数学运算**
  - 支持常用的数学运算符，+、-、*、/、|、&、~、^（求反）
    - 需要注意的是，运算操作类似forth语言。请参考forth语言编程指南。
- **逻辑运算**

- && || ！ 与C语言一致

- **流程控制语句**
  - **IF $D expression：**根据上一条语句的执行结果，判断本条语句是否执行。当为True时，才执行。
  - **LOOP n expression**
    循环执行本条语句。n表示执行次数，n为0表示不执行。expression表示要执行的语句