Implement BMC KCS interface code in C language

原创 @If you bloom Modified on 2025-04-16 16:31:34 Read 8.8k Collection 43 Likes 9

Category Column: KCS Article Tags: C language C++ linux server Program Life

KCS This column includes this content 1 article Subscribe to our column

摘要 This article introduces a method to implement the BMCKCS interface using C language, and explains in detail the working principle of the KCS interface, including the use of status and command registers, and data input and output registers. It also shows how to interact with the BMC through the KCS interface through specific code examples, including the process of writing and reading data.

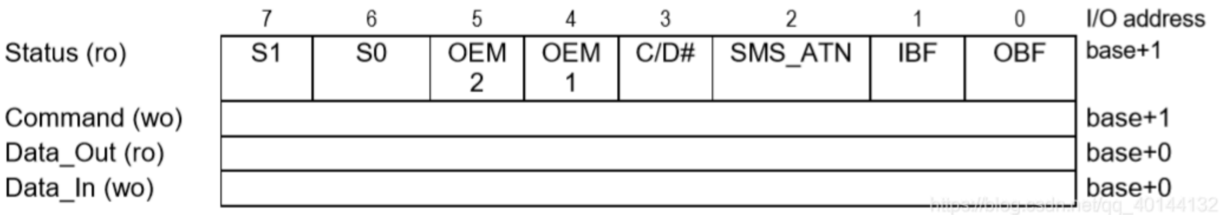
The summary is generated in C Know , supported by DeepSeek-R1 full version, go to experience>

C language implementation of BMC KCS interface code

KCS interface is essentially a set of I/O addresses mapped to the system I/O space. This set of addresses contains a continuous 4-byte storage space, as shown in the figure below. 2 bytes are provided to the KCS status and command registers , and 2 bytes are provided to the data input and output registers. The default base address of KCS, that is, the data register address, is CA2, and the status and command register address is base address+1 (CA3). CA2 and CA3 are essentially an 8-bit I/O port register.

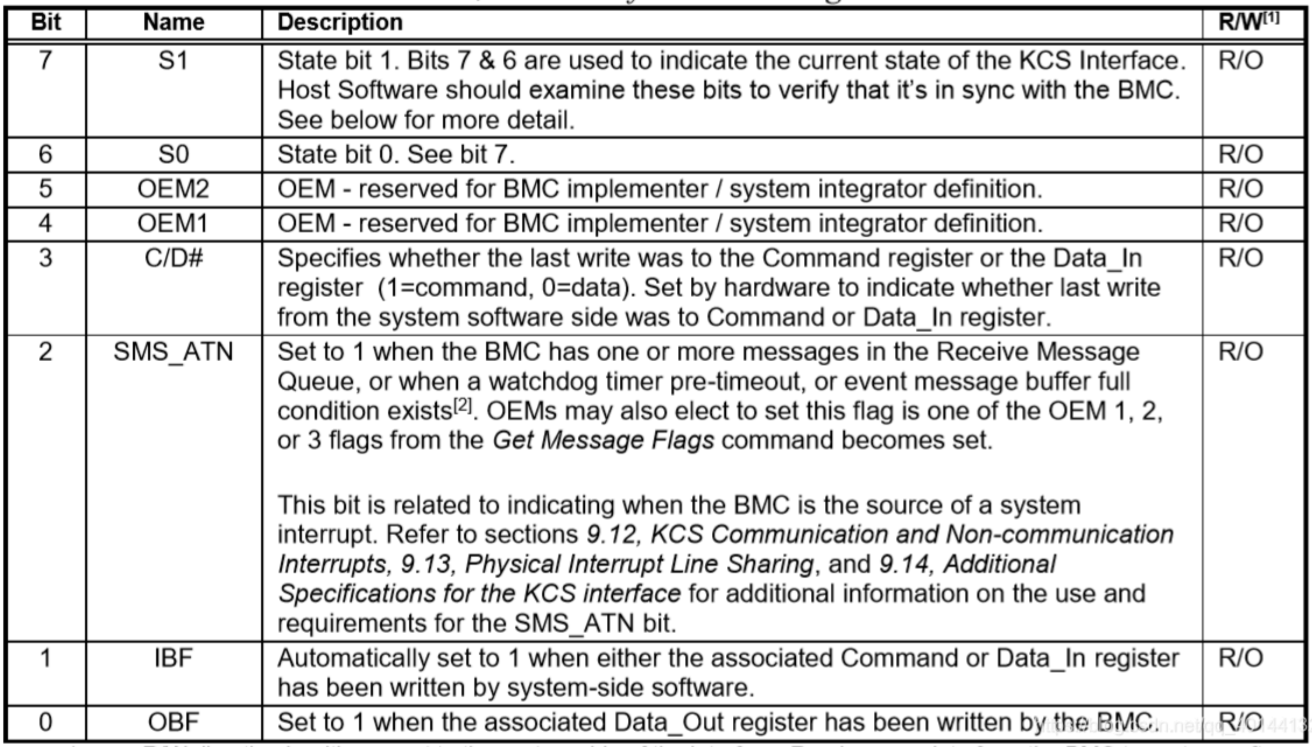
The status register is relatively complex. The first two high bits (S1, S0) are used to identify the status of the KCS interface (idle state 0x00, read state 0x40, write state 0x80, error state 0xc0).

Figure 9-5, KCS Interface Registers



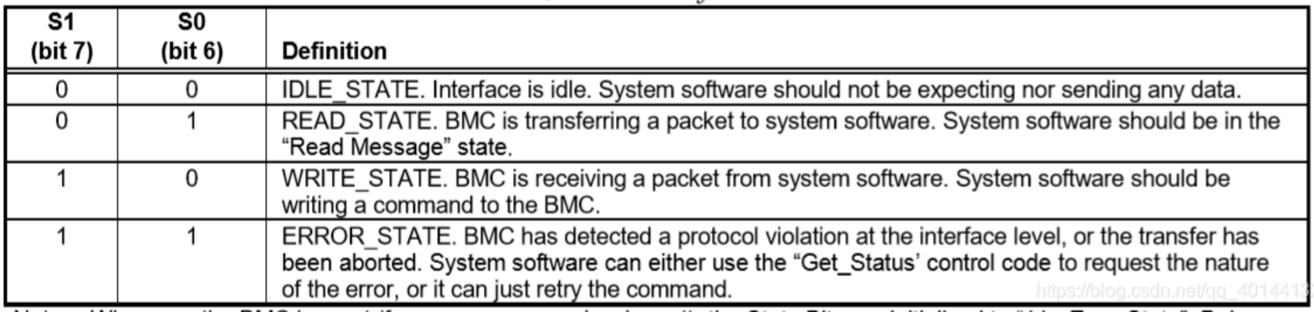
	7	6	5	4	3	2	1	0	I/O address
Status (ro)	S1	S0	OEM 2	OEM 1	C/D#	SMS_ATN	IBF	OBF	base+1
Command (wo)									base+1
Data_Out (ro)									base+0
Data_In (wo)									base+0

Table 9-1, KCS Interface Status Register Bits



Bit	Name	Description	R/W[1]
7	S1	State bit 1. Bits 7 & 6 are used to indicate the current state of the KCS Interface. Host Software should examine these bits to verify that it's in sync with the BMC. See below for more detail.	R/O
6	S0	State bit 0. See bit 7.	R/O
5	OEM2	OEM - reserved for BMC implementer / system integrator definition.	R/O
4	OEM1	OEM - reserved for BMC implementer / system integrator definition.	R/O
3	C/D#	Specifies whether the last write was to the Command register or the Data_In register (1=command, 0=data). Set by hardware to indicate whether last write from the system software side was to Command or Data_In register.	R/O
2	SMS_ATN	Set to 1 when the BMC has one or more messages in the Receive Message Queue, or when a watchdog timer pre-timeout, or event message buffer full condition exists[2]. OEMs may also elect to set this flag is one of the OEM 1, 2, or 3 flags from the *Get Message Flags* command becomes set. This bit is related to indicating when the BMC is the source of a system interrupt. Refer to sections *9.12, KCS Communication and Non-communication Interrupts*, *9.13, Physical Interrupt Line Sharing*, and *9.14, Additional Specifications for the KCS interface* for additional information on the use and requirements for the SMS_ATN bit.	R/O
1	IBF	Automatically set to 1 when either the associated Command or Data_In register has been written by system-side software.	R/O
0	OBF	Set to 1 when the associated Data_Out register has been written by the BMC.	R/O

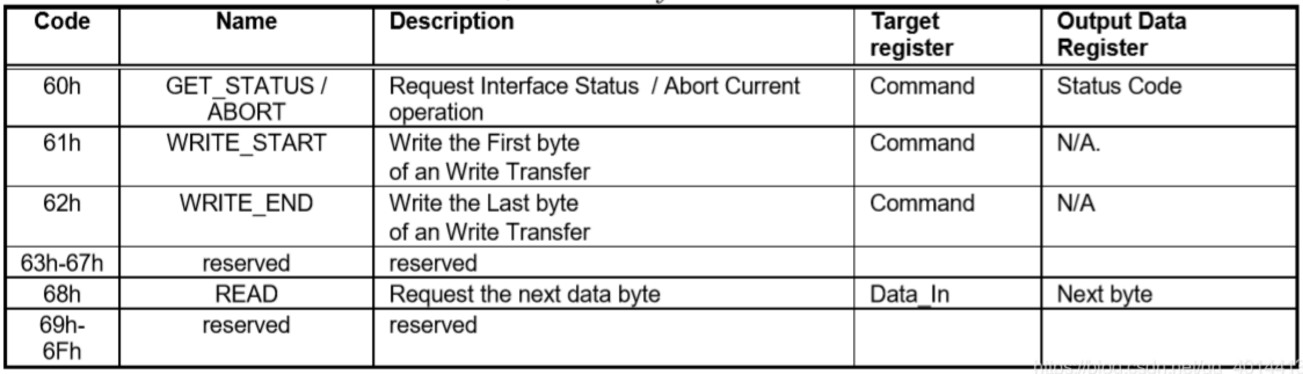
Table 9-2, KCS Interface State Bits



S1 (bit 7)	S0 (bit 6)	Definition
0	0	IDLE_STATE. Interface is idle. System software should not be expecting nor sending any data.
0	1	READ_STATE. BMC is transferring a packet to system software. System software should be in the "Read Message" state.
1	0	WRITE_STATE. BMC is receiving a packet from system software. System software should be writing a command to the BMC.
1	1	ERROR_STATE. BMC has detected a protocol violation at the interface level, or the transfer has been aborted. System software can either use the "Get_Status' control code to request the nature of the error, or it can just retry the command.

The values of the above status registers are all set by the BMC. The system side only needs to read them to obtain the current state of KCS. If the system side needs to control the state of KCS, it needs to send a control code to KCS. After receiving it, the BMC side sets KCS to the corresponding state until a command is executed .

Table 9-3, KCS Interface Control Codes



Code	Name	Description	Target register	Output Data Register
60h	GET_STATUS / ABORT	Request Interface Status / Abort Current operation	Command	Status Code
61h	WRITE_START	Write the First byte of an Write Transfer	Command	N/A.
62h	WRITE_END	Write the Last byte of an Write Transfer	Command	N/A
63h-67h	reserved	reserved		
68h	READ	Request the next data byte	Data_In	Next byte
69h-6Fh	reserved	reserved		

The C file is named KCS-rawdata.c

```c
//加入代码实现需要的C标准库和文件
#include <stdio.h>
#include <stdlib.h>
#include <sys/io.h>
#include <string.h>
#include "linux/capability.h"

//KCS接口定义了一组I/O映射的通信寄存器
#define KCS_CMD_REG     0xCA3   //cmd/status寄存器
#define KCS_DATA_REG    0xCA2   //data_in/data_out寄存器

//定义KCS控制码
#define GET_STATUS      0x60    //发送0x60到cmd/status寄存器获取KCS寄存器状态
#define WRITE_START     0x61    //发送0x61到cmd/status寄存器开始写入数据
#define WRITE_END       0x62    //发送0x62到cmd/status寄存器结束写入数据
#define READ            0x68    //发送0x68到data_in/data_out寄存器读取数据

//定义KCS状态码
#define IDLE_STATE      0       //KCS闲置状态寄存器读值
#define READ_STATE      0x40    //读取数据时状态寄存器的读值
#define WRITE_STATE     0x80    //写入数据时状态寄存器的读值
#define ERROR_STATE     0xC0    //出现错误是状态寄存器的读值

//延时函数
void timedelay(int ms)
{
    int x,y,k=0;
    for(x=100;x>0;x--)
    for(y=ms;y>0;y--)
    k++;
}

//等待IBF被清空
void WaitIBFClear()
{
    int IBFStatus;
    do{
        timedelay(100);
        IBFStatus = inb(KCS_CMD_REG);   //从命令寄存器读取数据直到IBF为1
    }while ((IBFStatus & 0x02) == 1);
}

```

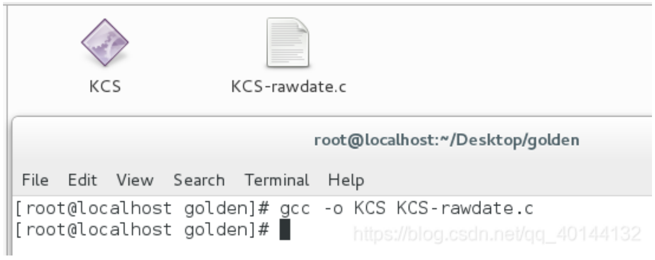
```
//主动清空OBF
void ClearOBF()
{
    inb(KCS_DATA_REG); //BMC读取数据寄存器时，OBF就会被清空
}

//等待OBF被设置
void WaitOBFSet()
{
    int OBFStatus;
    do{
        timedelay(100);
        OBFStatus = inb(KCS_CMD_REG); //从命令寄存器读取数据直到OBF为0
    }while ((OBFStatus & 0x01) == 0);
}

//获取KCS接口状态
KCS_State()
{
    int KCSState;
    KCSState = inb(KCS_CMD_REG);
    //return KCSState;
    return (KCSState & 0xC0);
}

//读取数据
void Read()
{
    int i = 0;
    int state;
    int responseArray[100]; //设置足够长的接收数据的数组
    printf(" ");
    do
    {
        state = KCS_State();    //判断当前KCS接口状态，执行不同操作
        if(state == READ_STATE)
        {
            WaitOBFSet();   //如果是读状态，等待OBF被设置
            responseArray[i] = inb(KCS_DATA_REG); //获取数据寄存器的值
            state = KCS_State();
            if((i>1) && (state == READ_STATE)) //如果完成码不是0x00，表示输入的命令错误或者不支持该命令
            {
                if((responseArray[2]) & 0xff != 0)
                {
                    printf("%02x\n ",responseArray[2]);
                    printf("the command you input is not correct, please check it carefully!\n");
                    exit(1);
                }
                printf("%02x ",responseArray[i]);
            }
            outb(READ,KCS_DATA_REG); //每次只能读取一个字节的数据，每次都需要发送READ控制码请求读取更多数据，直到数据接收完成
            i++;
        }
        else if((state | IDLE_STATE) == IDLE_STATE) //读取完所有数据后，KCS接口则为空闲状态
            {
                printf("\n");
                WaitOBFSet(); //等待OBF被设置
                responseArray[i] = inb(KCS_DATA_REG);   //Read dummy data byte from 数据寄存器
                exit(1);
            }
            else
            {
                exit(0);
            }
    } while (1);
}

//主函数，程序运行的入口
int main(int argc,char *argv[])
{
    int i=0,res;
    unsigned char ReqCmdData = 0;
    int state;
    if (argc<3){    //KCS的命令最少需要3组
        return printf("Not KCS-to-BMC request message!\n");
    }
    res=iopl(3);    //设置当前进程的I/O级别为最大
    if (res < 0)
    {
      perror("iopl set error!");
```

```
        return -1;
    }

    WaitIBFClear();
    ClearOBF();
    outb(WRITE_START,KCS_CMD_REG);  //发送WRITE_START控制码到命令寄存器开始写入操作
    WaitIBFClear();
    state = KCS_State();
    if ((state == WRITE_STATE)|(state == ERROR_STATE))
    {
        ClearOBF();
        ReqCmdData = strtoul(argv[1],0,0);  //将输入的第一个数据转化为十六进制字符串
        ReqCmdData = ReqCmdData << 2;   //向左偏移两位
        outb(ReqCmdData,KCS_DATA_REG);  //将得到的数据写入数据寄存器
        WaitIBFClear();
        ClearOBF();

        if(argc-1 == 2) //如果输入的数据只有两个，接着就需要将WRITE_END控制码发送到命令寄存器准备结束写操作
        {
            outb(WRITE_END,KCS_CMD_REG);    //将WRITE_END控制码发送到命令寄存器
            WaitIBFClear();
            state = KCS_State();
            if(state = WRITE_STATE)
            {
                ClearOBF();
                ReqCmdData = strtoul(argv[2],0,0);
                outb(ReqCmdData,KCS_DATA_REG); //写入最后1byte数据
                WaitIBFClear();
                Read(); //开始读取BMC返回的数据
            }
            else
            {
                return 0;
            }
        }
        else
        {
            for (i=2; i<argc-1; i++)    //如果输入的数据大于2byte,可以循环接收数据直到最后1byte
            {
                ReqCmdData = strtoul(argv[i],0,0);
                outb(ReqCmdData,KCS_DATA_REG);
                WaitIBFClear();
                ClearOBF();
            }
            outb(WRITE_END,KCS_CMD_REG); //写完倒数第二byte是，将WRITE_END控制码发送到命令寄存器
            WaitIBFClear();
            state = KCS_State();
            if(state = WRITE_STATE)
            {
                ClearOBF();
                ReqCmdData = strtoul(argv[argc-1],0,0);
                outb(ReqCmdData,KCS_DATA_REG);  //接收最后1byte数据
                WaitIBFClear();
                Read(); //开始写操作
            }
            else
            {
                return 0;
            }
        }
    }
    else
    {
         printf("BMC is not prepared to receiving this message!\n");
         exit(0);
    }
}
```

收起 ^

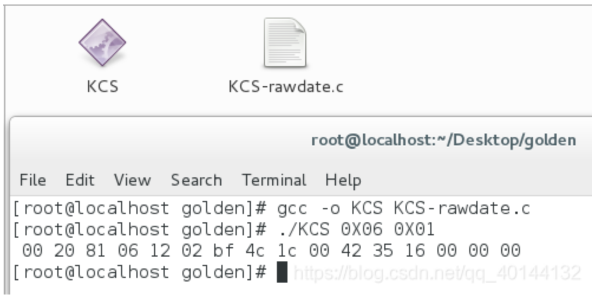
Compile KCS-rawdata.c to generate an executable file named KCS.

gcc -o KCS -c KCS-rawdata.c


KCS
KCS-rawdate.c
root@localhost:~/Desktop/golden
File Edit View Search Terminal Help
https://blog.csdn.net/qq_40144132

```
[root@localhost golden]# gcc -o KCS KCS-rawdate.c
[root@localhost golden]# 
```

Execute the command
./KCS 0x06 0x01


KCS
KCS-rawdate.c
root@localhost:~/Desktop/golden
File Edit View Search Terminal Help
https://blog.csdn.net/qq_40144132

```
[root@localhost golden]# gcc -o KCS KCS-rawdate.c
[root@localhost golden]# ./KCS 0X06 0X01
 00 20 81 06 12 02 bf 4c 1c 00 42 35 16 00 00 00
[root@localhost golden]# 
```

to realize the interaction between the system and BMC through the KCS interface. (This function is the same as the process of sending raw data by ipmitool)