Open BMC Development Series (VIII) IPMI KCS Channel

原创 dayuli Posted on 2021-12-09 18:43:22 Read 1.7w Collection 91 Likes 44

Category Column: BMC Article Tags: bmc

摘要 This article explains in detail how the LPC interface of the AST2500 chip supports the KCS module, and introduces the KCS implementation of IPMI, involving KCS initialization, interaction between BIOS and BMC, and IPMI specifications. The focus is on the data transmission process of the KCS interface between BIOS and BMC, as well as the various system interface applications of IPMI.

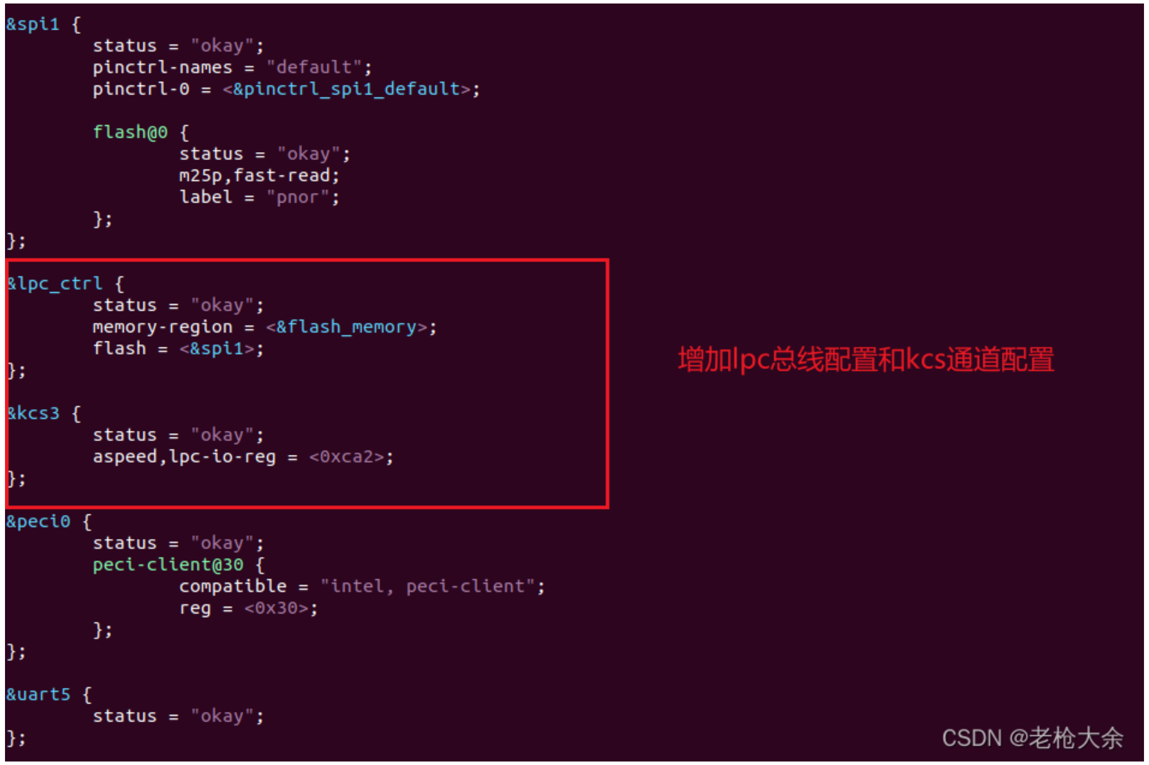
The summary is generated in C Know , supported by DeepSeek-R1 full version, go to experience>

1. Official Documentation

I use the ast2500 chip, so I checked the 2500 chip manual and searched the contents of kcs, and mainly found two pieces of content.

1. PIC- Express 2.0 Bus supports kcs devices.

PCI-Express 2.0 Bus Device Controller Support optional BMC KCS device
2. LPC interface support for KCS
LPC Bus Interface
– Slave mode: designed for BMC functions (I/O and memory or fifirmware read/write cycles)
• Support Serial IRQ (reduce polling time)
• Support port 80H/81H (programmable address) snooping registers with interrupt options
• Support 1 virtual UART for Serial-over-Lan (SOL) application
• Support up to 4 sets of KCS mode registers and 1 BT mode registers (IPMI 2.0 Complaint), or 3 sets of KCS mode registers and 1+1 BT mode registers
LPC Controller Features
Compliant with IPMI version 2.0 KCS mode and BT mode
– Channel #1 supports KCS interface
– Channel #2 supports KCS interface
– Channel #3 supports KCS or BT (H8S/2168 compliant) interface
– Channel #4 supports KCS interface
– Channel #5 supports iBT (IPMI compliant) interface
I mainly focus on the functions of IPMI, so we can see that kcs is a function hanging under the LPC interface. If the LPC controller supports IPMI 2.0, there are mainly two modes, one is kcs mode and the other is BT mode.
Among the 5 LPC channels, 3 of them can only be set as kcs, 1 can only be set as a BT channel using the IPMI protocol, and 1 can be set as a BT channel using the H8S/2168 protocol, and can also be set as kcs.
We usually configure the kcs3 channel specifically for the device tree configuration. The reason is here. kcs3 is an optional channel.



```
&spi1 {
        status = "okay";
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_spi1_default>;

        flash@0 {
                status = "okay";
                m25p,fast-read;
                label = "pnor";
        };
};

&lpc_ctrl {
        status = "okay";
        memory-region = <&flash_memory>;
        flash = <&spi1>;
};

&kcs3 {
        status = "okay";
        aspeed,lpc-io-reg = <0xca2>;
};
&peci0 {
        status = "okay";
        peci-client@30 {
                compatible = "intel, peci-client";
                reg = <0x30>;
        };
};

&uart5 {
        status = "okay";
};
```

The following is a reprint of a useful KCS blog post:

2. KCS Description

BMC and BIOS communication interface
BMC and UEFI communication is mainly connected through the LPC bus. The LPC controller of AST2500 is connected to the LPC controller of our bridge chip. The LPC controller of the bridge chip is the master and the BMC end is the slave. In this way, the bios can access the registers of the device under the BMC end LPC through the address of the LPC bus domain. The 7a bridge chip maps the address of the LPC bus to the IO space of the bridge chip, so that we can access the device registers under the BMC LPC by accessing the IO space. In this way, communication is possible.

Initialization of KCS module
KCS module is a submodule under BMC LPC controller. The LPC controller interface of AST2500 chip is very rich, and KCS is just one of them. The initialization of this module, including channel selection, clock enable, register mapping address (0xCA2 by default), is completed by the BMC driver. That is, after the KCS module is initialized, the KCS module can work normally, and the BIOS can access it at this time. Note that the initialization work is completed by the driver in the BMC kernel. The details of how to initialize and which registers have been configured are not introduced in detail here.

The main points to note on the BIOS side
(1) The address of the KCS register mapping
This address is set in Address reg. The default value of this register is 0xCA2. You can also modify it yourself. We did not modify it here. This address determines the address of the register used to read and write data. In fact, there are only two registers used later, namely command/data/.
On our platform, the final virtual address of these registers is the base address of the IO space plus the above offset. That is,
LPC_IO_BASE + 0xCA2 is the command register, and LPC_IO_BASE + 0xCA3 is the data register
And the LPCIO base address is 0x90000efdfc000000.

(2) How to complete data transmission
The main thing is that the driver must comply with the specifications of Chapter 9 Keyboard Controller Style (KCS) Interface of IPMI, and read and write data according to the status of the registers in the protocol specifications. There is a reference code under the kernel.
In the interface, each ipmi command contains the following parts: NetFn, Lun, Cmd data...
The figure below shows the data format of the command sent by the bios to the BMC, and the data format of the information returned by the BMC to the bios.
Note: The implementation of each ipmi command is different. Some commands do not require the data part, some do, some commands do not return data, and some do. Therefore, you need to read the IPMI specifications in detail to see the corresponding format of each command. The NetFn, Lun, and Cmd of each command are defined by the protocol, which also provides commands defined by OEMs.

Note that in the response data, there are three bytes that must be present, namely CompletionCode NetFn and Cmd, which are the contents that each command must return. The first byte is the CompletionCode, which indicates the status of the command sent to the BMC. If the command is received successfully, the completion code is 0, otherwise there is a problem.

Note: The format of ipmi -raw [netfn] [lun] [cmd][data1–datan]
is as above, where netfn lun cmd is required, and some commands in the data part are not required.

3. KCS Implementation of IPMI

IPMI provides two mechanisms for describing information about the management capabilities of a platform:

(1) capabilities commands, IPMI command

(2) Sensor Data Records (SDR), command about sensor

IPMI defines three standard system interfaces to transmit information to the BMC.

(1) Keyboard Controller Style (KCS),

Intel 8742 Universal Peripheral Interface microcontroller

(2) System Management Interface Chip (SMIC),

When the BMC does not establish a KCS interface, SMIC provides an optional option.

(3) Block Transfer (BT),

The UDP port provides a more efficient interface. Unlike the previous two, it uses a per-block handshake for data transmission.

In addition to the above basic interfaces, there are also the serial port and LAN port that we commonly use.

The serial port supports 3 connection modes

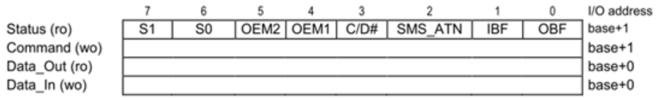
Basic Mode PPP Mode Terminal Mode:

SOL, Serial Over LAN,

It is used for asynchronous serial-based OS and pre-OS communication with the BMC.

BMC Transfer is generally implemented through the KCS interface in our code.

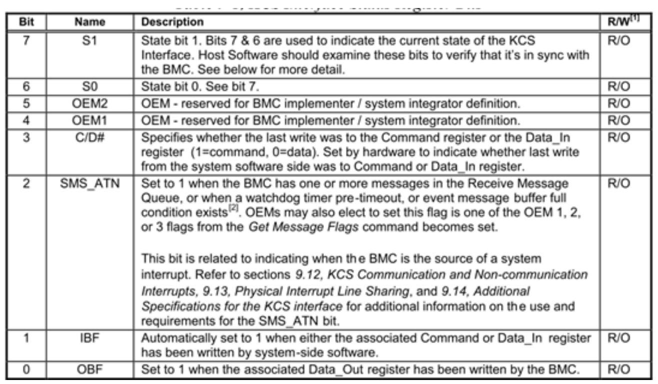
Before understanding the implementation process, you must first understand KCS Interface Registers



	7	6	5	4	3	2	1	0	I/O address
Status (ro)	S1	S0	OEM2	OEM1	C/D#	SMS_ATN	IBF	OBF	base+1
Command (wo)									base+1
Data_Out (ro)									base+0
Data_In (wo)									base+0

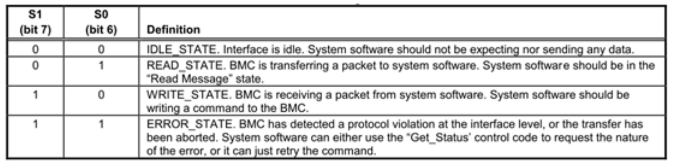
1. Status Register

There will always be some information transmission in the system. Before transmission, it is necessary to determine the status of the device. For the KCS interface, this is done through its Status Register. The information represented by each bit of the Status Register is shown in the figure:



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

The two status bits Bit6 and Bit7 indicate a total of four states that the interface can be in.



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.

2. Command Register

When the IBF is cleared, commands can be written to it.

3. Data Registers

BMC is used to transmit data.

How to write data to BMC through KCS interface in CODE:

(1) Read the Status Register information and obtain the IBF value. When the IBF value reaches 0, continue execution.

(2) Clear OBF

(3) Check KCS status

(4) Send data to BMC in byte units

(5) Check the KCS status again

Read data from BMC:

(1) Check the KCS status. If it is IDLE_STATE, return EFI_SUCCESS.

(2) Read data from DATA_OUT.

(3) Write the data read from DATA_OUT to DATA_IN

Note that the KCS code in the PEI and DXE stages is different.

The memory has been initialized in the DXE stage, so the use of KCS in the DXE stage needs to determine whether it is SMM mode.

Finally: one-click triple click, muah!

Like is a virtue,

Attention is fate,

Collection is for sure.

Reward as you like,

Your encouragement is part of the good in my world, love you guys!