

# LPC related knowledge

原创

Anthony

Posted on 2021-12-06 16:10:29

Read 4.1k


Collection 16

Likes

Category Column: BIOS Basics

Article Tags: C LPC

Copyright CC 4.0 BY-SA

 BIOS Basics

This column includes this content

6 articles

Subscribe to our column

The LPC (Low Pin Count) interface is a low IO peripheral interface introduced by Intel, used to connect LPC devices such as SuperIO and Flash

Instructions:

Before using the LPC function, you need to configure the relevant PAD registers and configure the corresponding PAD to the corresponding LPC function before you can use the LPC function:

1. Determine the interface access type of the device you want to access, and then send the corresponding IO, FIRMWARE Memory, Memory, and DMA requests through the device type register of the APB interface address.
2. Use nu\_serirq\_config[31] to determine whether it is a 4-byte read or a single-byte read.

First, get the base address of LPC, such as the base address is 0x000\_20000000

Let's check the registers again, taking the following code as an example:

AI generated projects 登录复制

```
1 MmioWrite32(0x27ffffcc, 1);
2 MmioWrite32(0x27ffffe8, 0x80000000);
3 MmioWrite32(0x27ffffd8, 0);
4 MmioWrite32(0x27ffffd4, 0);
```

Write 1 to the first offset address. For details, please refer to the LPC protocol document.

Mainly BIT3-BIT0:

000 is IO read, 001 is IO write, 010 is memory read.....

The second offset address writes the value 0x80000000, which is the nu\_serirq\_config[31] mentioned above, and is described as follows:

Configuration register (bit31: enable flag for reading 4 bytes of data each time (1'b1: read 1 byte); bit1~0: start cycle configuration (2'b11: 8; 2'b10: 6; otherwise 4, default 4 clk), bit2: serial interrupt mode configuration default continuous mode default is continuous mode), bit3~4: number of serial interrupt devices supported (2'b01 represents 32 otherwise 16 default 16)

The third interrupt mask register and the fourth configuration start cycle register

After writing the above, we can perform IO read and write operations through LPC, taking EC and BIOS communication as an example

EC provides 256 bytes of RAM space that can be read and written by the system. EC resources are mapped in this RAM space. By accessing the corresponding offset (0x00~0xFF), the corresponding resources can be operated. Those who understand EC, especially ITE, are familiar with ports 62 and 66. If BIOS wants to obtain the value in the EC register, it needs to add 62 and 66 to the base address.

for example:

AI generated projects 登录复制

```
1 #define LPC_MEM_BASE    0x20000000
2 #define EC_COMMAND_PORT 0x66 | LPC_MEM_BASE
3 #define EC_DATA_PORT    0x62 | LPC_MEM_BASE
```

The subsequent EC read and write codes are as follows:

```
1 #include <Base.h>
2 #include <Library/BaseLib.h>
3 #include <Library/IoLib.h>
4 #include <Library/EcLib.h>
5 #include <Library/DebugLib.h>
6 #include <Library/TimerLib.h>
7
8
9 EFI_STATUS
10 EcIbFree ()
11 {
```

```

12 | EFI_STATUS   Status = EFI_SUCCESS;          13 | UINTN   Count = 10000;
14 |   UINT8      Data;
15 |
16 |   do {
17 |       Data = MmioRead8(EC_COMMAND_PORT);
18 |       if ((Data & EC_IBF) == 0) {
19 |           break;
20 |       }
21 |       MicroSecondDelay (1000);
22 |       Count --;
23 |   } while (Count);
24 |
25 |   if (Count == 0) {
26 |       Status = EFI_TIMEOUT;
27 |   }
28 |
29 |   return Status;
30 | }
31 |
32 | EFI_STATUS
33 | EcObFull()
34 | {
35 |     EFI_STATUS   Status = EFI_SUCCESS;
36 |     UINTN        Count = 10000;
37 |     UINT8        Data;
38 |
39 |     do {
40 |         Data = MmioRead8(EC_COMMAND_PORT);
41 |         if (Data & EC_OBF) {
42 |             break;
43 |         }
44 |         MicroSecondDelay (1000);
45 |         Count --;
46 |     } while (Count);
47 |
48 |     if (Count == 0) {
49 |         Status = EFI_TIMEOUT;
50 |     }
51 |
52 |     return Status;
53 | }
54 |
55 |
56 | EFI_STATUS
57 | EcWriteCmd (
58 |     UINT8      cmd
59 | )
60 | {
61 |     while((MmioRead8(EC_COMMAND_PORT)) & EC_OBF) {
62 |         MmioRead8(EC_DATA_PORT);
63 |     }
64 |
65 |     EcIbFree();
66 |     MmioWrite8(EC_COMMAND_PORT, cmd);
67 |     EcIbFree();
68 |     return EFI_SUCCESS;
69 | }
70 |
71 |
72 | EFI_STATUS
73 | EcWriteData (
74 |     UINT8      data
75 | )
76 | {
77 |     EcIbFree();
78 |     MmioWrite8(EC_DATA_PORT, data);
79 |     EcIbFree();
80 |     return EFI_SUCCESS;
81 | }
82 |
83 |
84 | EFI_STATUS
85 | EcReadData (
86 |     UINT8      *pData
87 | )
88 | {
89 |
90 |     if (EFI_ERROR(EcObFull()))
91 |         return EFI_DEVICE_ERROR;

```

```

92 |     93 | *pData=MmioRead8(EC_DATA_PORT);
94 | return EFI_SUCCESS;
95 | }
96 |
97 | EFI_STATUS
98 | EcReadMem (
99 |     UINT8 Index,
100 |     UINT8 *Data
101 | )
102 | {
103 |     UINT8 cmd = 0x80;
104 |     EcWriteCmd (cmd);
105 |     EcWriteData(Index);
106 |     EcReadData(Data);
107 |     return EFI_SUCCESS;
108 | }
109 |
110 | EFI_STATUS
111 | EcWriteMem (
112 |     UINT8 Index,
113 |     UINT8 Data
114 | )
115 | {
116 |     UINT8 cmd = 0x81;
117 |     EcWriteCmd (cmd);
118 |     EcWriteData(Index);
119 |     EcWriteData(Data);
120 |     return EFI_SUCCESS;
121 | }

```

收起 ^