Revision 1.0
19 November 2020

Technical Note TN_SD_00

SD/microSD Cards Health Status Monitoring: S.M.A.R.T. Command

About ATP

ATP Electronics Taiwan Inc.(“ATP”) is a leading solutions provider of high-performance, high-quality and durable NAND flash and DRAM memory modules.

For over 25 years, we have been transforming the industrial and enterprise computing landscape with our high-performance and high-endurance NAND flash storage products and DRAM memory modules. Acutely aware of the rigors of data-hungry applications, we design, test and manufacture products according to the most meticulous standards.

The ATP name is synonymous with absolute dedication to uncompromising product and service quality, and this is why companies around the world entrust their storage and memory needs to us. ATP products are designed and built to meet the exacting demands of modern computing and to accomplish mission-critical tasks even under the toughest operating conditions.

Advanced packaging techniques like System-in-Package (SiP), and unique technologies such as PowerProtector, Secure Erase, and more, demonstrate our astute technical capabilities. Testing and validation processes such as Test During Burn-In and Automatic Test Equipment systems showcase our exceptional manufacturing and testing capabilities as well as our resolute commitment to deliver best-in-class products.

The ATP brand continues to grow through industrial OEM sales channels with offices in the USA, Europe, and Asia offering worldwide engineering and sales support.

Contents

Introduction

Quite often, in many applications that adopt managed NAND as storage media, specific reasons, such as equipment preventive maintenance or devices fault tolerance, lead to the need of predicting the future lifespan and in general to have parameters, which can account for the health status of those media. In many cases, it is even required to monitor the health status parameters in real time.

In general, some key figures, such as “Remaining Life %,” “Average Erase Count,” and “Later Bad Block Count” are widely adopted in the industry to account for the storage media health status, which is mainly related to the NAND flash array wearout due to the electrical stress induced on the cells tunnel oxide during program/erase (P/E) operations.

To meet the above-mentioned lifespan prediction and health status monitoring requirements, ATP offers software (SW) libraries and SW tools suitable for several operating systems, to support its customers in the development and integration, in their application SW of microSD and SD health status real-time monitoring. In particular, ATP can provide the application programming interface (API) for Windows XP/7/8/10 and also Linux libraries for Ubuntu, Fedora and CentOS distros, which provide functions to retrieve a proprietary set of health status parameters, like for SSD Self-Monitoring, Analysis and Reporting Technology (S.M.A.R.T.) attributes.

As an additional option, ATP makes available to its customers a solution to retrieve the microSD and SD S.M.A.R.T. attributes by sending a specific vendor command — CMD56. This method is tailored to specific microSD and SD product families and is suitable only on cards plugged on native SDIO slots (not on USB adapters).

ATP recognizes the value of open source compatibility and the importance of supporting the open source community. The sample source code in this document is aimed to exclusively act as a guidance for Linux and non-Linux operating system software development.

For additional support to enable the S.M.A.R.T. attributes command for ATP microSD and SD cards, contact your ATP representative.

Disclaimer

All information provided in this technical note is preliminary and will be subject to change at any time without notice. The terms and conditions stated in this technical note may also be changed at any time without notice.

By using the information contained herein, the receiving party or the reader of this technical note understands and acknowledges that ATP will not be liable for any provided information and any changes, nor for the usage of the provided information.

Command Format

The ATP device S.M.A.R.T. protocol, with its specific command argument and response format, extends the SD card generic command and allows host access to device S.M.A.R.T. data. The host can issue the S.M.A.R.T. command (CMD56) several times during the same power-on cycle. Furthermore, irrespective of the command sent, in case of an invalid parameter in the command block, the device will abort it without altering the SD.

The tables below provide a description of the general CMD56 command format.

Table 1: CMD56 Format

Bit Position	47	46	[45:40]	[39:8]	[7:1]	0
Width	1	1	6	32	7	1
Value	´0´	´1´	x	x	X	´1´
Description	Start	Transmission	Command Index	Argument	CRC7	End Bit

Table 2: CMD56 Description

CMD Index	Type	Argument	Response	Abbreviation	Command Description
CMD56	adtc	[31:1] stuff bits. [0]: RD/WR	R1	GEN_CMD	Used either to transfer a data block to the card or to get a data block from the card for general purpose/application-specific commands. In the case of Standard Capacity SD Memory Card, the size of the data block shall be set by the SET_BLOCK_LEN command. In the case of High Capacity SD Memory Cards, the size of the data block is fixed to 512 bytes. The host sets RD/WR=1 for reading data from the card and sets to 0 for writing data to the card.

For details about the specific command argument and read data block format (CMD56_ATP_ARG and BAO definition in the source file “mmc_cmds.c”), for retrieving S.M.A.R.T. attributes, please contact your ATP representative and request the “ATP SD SMART Command Guideline”. Please consider that the granting of such Guidelines requires the signing of a Non-Disclosure Agreement (NDA).

Linux Environment Setup

To support the customers in implementing the S.M.A.R.T. command, ATP developed and made available a sample source code based on mmc-utils (MMC/SD ioctl device interface), which is an open source test tool for MMC/SD devices.

As preparatory step, before implementing the Linux porting C code contained in this document, the SW developer must perform the following installation and compilation actions to set up a proper working environment under Linux.

1. Download the latest “mmc_utils” distribution from:

   ```
   $> sudo git clone
   https://kernel.googlesource.com/pub/scm/linux/kernel/git/cjb/
   ```
   in the development directory.

2. By running the “Make” build utility, build the binary, compiling the sources with appropriate cross-compiler.

3. Check the usage and availability of “mmc utilities” tool by:

   ```
   $> ./mmc -h
   ```

S.M.A.R.T. Command Porting in “mmc-utils” Tool

Table 3: “mmc-utils” Changes

File in /mmc-utils	Content	Changes
mmc.h	Global Variable definition for mmc.c	Include the definitions for the S.M.A.R.T. command support, listed in **Table 4**, in the mmc.h file.
mmc.c	Command Line user Interface parser.	Include the command structure for the S.M.A.R.T. command, listed in **Table 5**, in the mmc.c file.
mmc_cmds.c	CLI command wrappers for the functions declared in mmc_cmds.h.	Include in mmc_cmds.c the CLI command wrappers (declared for each function in mmc_cmds.h) listed in **Table 7**.
mmc_cmds.h	Set of CLI command wrappers declarations. The purpose of CLI command wrappers is to enclose specific command functions from the lower level mmc ioctl system calls. These functions send custom commands to the card by using ioctl (fd, MMC_IOC_CMD, (struct mmc_ioc_cmd*) &ioctl_data) with fd pointing to the proper mmcblkn device.	From mmc_cmds.c, include the following corresponding function declarations (listed in **Table 6**) in the mmc_cmds.h file:
		Function 1 (to implement the S.M.A.R.T. command): **int do_ATPHS(int nargs, char **argv);**
		Function 2: **int CMD56_data_rd(int fd, int cmd56_arg, char lba_block_data);**
		Function 3: **void dump_whole_block(char *lba_block_data);**

Table 4: mmc.h File

```
/* mmc.c global variable definition for ATP S.M.A.R.T.  command */
#define OPC_CMD56  56                    /* adtc, R1 */
#define SD_BLOCK_SIZE 512   /* Data block size for CMD56 */
#define MMC_RSP_R2 (MMC_RSP_PRESENT|MMC_RSP_136|MMC_RSP_CRC)
#define MMC_CMD_BCR (3 << 5)
```

Table 5: mmc.c File

```
/* ATPHS  CLI command definition */
{ do_ATPHS, -3,
  "atphs read", "<dump ctrl> <dev_type> <pe_cycles> <device>\n"
  "Usage: mmc atphs read <-d> <dev_type> <pe_cycles> <device>\n"
  "-d\tdump data block",
  NULL
},
```

Table 6: mmc_cmds.h File

```
/* ATPHS wrapper definitions */
int do_ATPHS(int nargs, char **argv);                                   /* Function 1 */
int CMD56_data_rd(int fd, int cmd56_arg, char *lba_block_data);         /* Function 2 */
void dump_whole_block(char *lba_block_data);                            /* Function 3 */
```

Table 7: mmc_cmds.c File

```
/* ====================== ATPHS functions BEGIN========================== */
#define UC (unsigned char)
#define CMD56_ATP_ARG 0x11000001   // WARNING!!! The correct argument value can be found in the “SMART command guide”
#define BAO 00                     // WARNING!!! The correct offset address value can be found in the “SMART command guide”


int do_ATPHS(int nargs, char **argv)
{
  int cmd56_arg = CMD56_ATP_ARG;    //ATP CMD56 argument
  char data_in[SD_BLOCK_SIZE];
  int fd, ret, RemLife;
  char *device;
  static char SpeedClass[5] = {0,2,4,6,10};
  static char UHSSpeed[4][9] = {"<10MB/s", ">10MB/s", "Reserved", ">30MB/s"};
  unsigned long Conversion;

  if (!((nargs == 4)||(nargs == 5))) {
     printf("Usage: mmc atphs read <-d> <dev_type> <pe_cycles> <device>\n");
     exit(1);
  }
  device = argv[nargs-1];
  fd = open(device, O_RDWR);
  if (fd < 0) {
    perror("open");
    exit(1);
  }

  //execute CMD56 and get one 512-byte data block
  ret = CMD56_data_rd(fd, cmd56_arg, data_in);
  if (ret) {
    fprintf(stderr, "CMD56 CALL FAILED, %s\n", device);
    exit(1);
  }

  if (!strcmp("-d", argv[1]))
    dump_whole_block(data_in); //data block dumping

  switch (atoi(argv[nargs-3])) {
   case 2:                          //SM2702 controller
    printf("Bus Width: %d\n", UC data_in[BAO]>>2);
    printf("Speed Class: %d\n", SpeedClass[UC data_in[BAO+2]]);
    printf("UHS Speed Grade: %d\n", UC data_in[BAO+3]);
    Conversion= (UC data_in[BAO+4]<< 24)+ (UC data_in[BAO+5]<< 16)+(UC data_in[BAO+6]<< 8)+(UC data_in[BAO+7]);
    printf("Protect. Area Size: %lu\n", Conversion);
    printf("Spare Block Count: %d\n", UC data_in[BAO+8]);
    printf("Later Bad Blok Count: %d\n", UC data_in[BAO+10]);
    Conversion= (UC data_in[BAO+12]<< 24)+(UC data_in[BAO+13]<< 16)+(UC data_in[BAO+14]<< 8)+(UC data_in[BAO+15]);
    printf("Sudden Power OFF Reset count: %lu\n", Conversion);
    Conversion= (UC data_in[BAO+16]<< 24)+(UC data_in[BAO+17]<< 16)+(UC data_in[BAO+18]<< 8)+(UC data_in[BAO+19]);
    printf("Min. Erase Count: %lu\n", Conversion);
    Conversion= (UC data_in[BAO+20]<< 24)+(UC data_in[BAO+21]<< 16)+(UC data_in[BAO+22]<< 8)+(UC data_in[BAO+23]);
    printf("Max. Erase Count: %lu\n", Conversion);
    Conversion= (UC data_in[BAO+24]<< 24)+(UC data_in[BAO+25]<< 16)+(UC data_in[BAO+26]<< 8)+(UC data_in[BAO+27]);
    printf("Total Erase Count: %lu\n", Conversion);
    Conversion= (UC data_in[BAO+28]<< 24)+(UC data_in[BAO+29]<< 16)+(UC data_in[BAO+30]<< 8)+(UC data_in[BAO+31]);
    printf("Avg. Erase Count: %lu\n", Conversion);
    if (atoi(argv[nargs-2])==0) {
        printf("pe_cycles argument cannot be 0");
       exit(1);
     }
```

```
    RemLife = round(100*(1-Conversion/atoi(argv[nargs-2])));
    printf("Remaining Life: %d\n", RemLife);
   break;
   case 7:                          ////SM2707 controller
    printf("Speed Class: %d\n", SpeedClass[UC data_in[BAO+2]]);
    printf("UHS Speed Grade: %s\n", UHSSpeed[UC data_in[BAO+3]]);
    Conversion= (UC data_in[BAO+8]<< 8)+(UC data_in[BAO+9]);
    printf("Original Bad Block Count: %lu\n", Conversion);
    printf("Run-time Bad Block Count: %d\n", UC data_in[BAO+10]);
    printf("Spare Block Count: %d\n", UC data_in[BAO+11]);
    Conversion= (UC data_in[BAO+16]<< 24)+(UC data_in[BAO+17]<< 16)+(UC data_in[BAO+18]<< 8)+(UC data_in[BAO+19]);
    printf("Min. Erase Count MLC_TLC: %lu\n", Conversion);
    Conversion= (UC data_in[BAO+20]<< 24)+(UC data_in[BAO+21]<< 16)+(UC data_in[BAO+22]<< 8)+(UC data_in[BAO+23]);
    printf("Max. Erase Count MLC_TLC: %lu\n", Conversion);
    Conversion= (UC data_in[BAO+24]<< 24)+(UC data_in[BAO+25]<< 16)+(UC data_in[BAO+26]<< 8)+(UC data_in[BAO+27]);
    printf("Total Erase Count MLC_TLC: %lu\n", Conversion);
    Conversion= (UC data_in[BAO+28]<< 24)+(UC data_in[BAO+29]<< 16)+(UC data_in[BAO+30]<< 8)+(UC data_in[BAO+31]);
    printf("Avg. Erase Count MLC_TLC: %lu\n", Conversion);
    Conversion= (UC data_in[BAO+32]<< 24)+(UC data_in[BAO+33]<< 16)+(UC data_in[BAO+34]<< 8)+(UC data_in[BAO+35]);
    printf("Min. Erase Count SLC: %lu\n", Conversion);
    Conversion= (UC data_in[BAO+36]<< 24)+(UC data_in[BAO+37]<< 16)+(UC data_in[BAO+38]<< 8)+(UC data_in[BAO+39]);
    printf("Max. Erase Count SLC: %lu\n", Conversion);
    Conversion= (UC data_in[BAO+40]<< 24)+(UC data_in[BAO+41]<< 16)+(UC data_in[BAO+42]<< 8)+(UC data_in[BAO+43]);
    printf("Total Erase Count SLC: %lu\n", Conversion);
    Conversion= (UC data_in[BAO+44]<< 24)+(UC data_in[BAO+45]<< 16)+(UC data_in[BAO+46]<< 8)+(UC data_in[BAO+47]);
    printf("Avg. Erase Count SLC: %lu\n", Conversion);
    Conversion= (UC data_in[BAO+48]<< 24)+(UC data_in[BAO+49]<< 16)+(UC data_in[BAO+50]<< 8)+(UC data_in[BAO+51]);
    printf("Raw Capacity (MB): %lu\n", Conversion);
    Conversion= (UC data_in[BAO+52]<< 8)+(UC data_in[BAO+53]);
    printf("NAND P_E Cycles: %lu\n", (100*Conversion));
    printf("Card Life: %d\n", UC data_in[BAO+54]);
    Conversion= (UC data_in[BAO+60]<< 24)+(UC data_in[BAO+61]<< 16)+(UC data_in[BAO+62]<< 8)+(UC data_in[BAO+63]);
    printf("Power ON_OFF Count: %lu\n", Conversion);
   break;
  }
  return ret;
}

//CMD56 implementation
int CMD56_data_rd(int fd, int cmd56_arg, char *lba_block_data)
{
    int ret = 0;
    struct mmc_ioc_cmd idata;
    memset(&idata, 0, sizeof(idata));
    memset(lba_block_data, 0, sizeof(__u8) * 512);
    idata.write_flag = 0;
    idata.opcode = OPC_CMD56;
    idata.arg = cmd56_arg;
    idata.flags = MMC_RSP_SPI_R1 | MMC_RSP_R1 | MMC_CMD_ADTC;
    idata.blksz = SD_BLOCK_SIZE;
    idata.blocks = 1;
    mmc_ioc_cmd_set_data(idata, lba_block_data);
    ret = ioctl(fd, MMC_IOC_CMD, &idata);
    if (ret)
    perror("ioctl");
    return ret;
}

void dump_whole_block(char *lba_block_data)
{
    int count=0;
    printf("CMD56 data block dumping:");
    while( count < SD_BLOCK_SIZE) {
    if(count % 16 == 0)
    printf("\n%03d: ", count);
    printf("%02x ", (unsigned char) lba_block_data[count]);
    //printf("Add: %d  /  Data:%02x\n", count, (unsigned char) lba_block_data[count]);
    count++;
    }
    printf("\n");
    return;
}
/* ====================== ATP ATPHS  functions END  ====================== */
```

Usage Examples

The execution of mmc command with “atphs read” option has the format:

mmc atphs read [-d] <SD family identifier > <n. of max erase cycles for the given SD PN> <device>

-d

This optional switch allows dumping of the full data block

<SD family identifier>

This switch can assume only values 2 and 7, depending by the SD part number.

<n. of max erase cycles for the given SD PN>

integer value >0, specific for each SD part number

```
$> ./mmc atphs read 7 10000 /dev/mmcblk0
Speed Class: 10
UHS Speed Grade: >30MB/s
Original Bad Block Count: 11
Run-time Bad Block Count: 0
Spare Block Count: 46
Min. Erase Count MLC_TLC: 0
Max. Erase Count MLC_TLC: 0
Total Erase Count MLC_TLC: 0
Avg. Erase Count MLC_TLC: 0
Min. Erase Count SLC: 0
Max. Erase Count SLC: 1
Total Erase Count SLC: 12
Avg. Erase Count SLC: 0
Raw Capacity (MB): 120580
NAND P_E Cycles: 10000
Card Life: 100
Power ON_OFF Count: 2
```

```
$> ./mmc atphs read -d 7 10000 /dev/mmcblk0
CMD56 data block dumping:
000: 09 41 50 00 00 00 00 00 00 00 00 00 00 00 00 00
016: 10 00 04 03 08 00 00 00 00 0b 00 2e 00 00 00 00
032: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
048: 00 00 00 00 00 00 00 01 00 00 00 0c 00 00 00 00
064: 00 01 d7 04 00 64 64 00 00 00 00 00 00 00 00 02
080: 2c a4 08 32 a1 00 09 00 53 4d 32 37 30 37 45 4e
096: 00 00 00 00 00 00 00 00 07 d0 00 00 00 00 00 18
112: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
128: 32 37 30 37 45 4e 41 42 32 30 31 39 30 37 33 30
144: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
160: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
176: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
192: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
208: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
224: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
240: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
256: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
272: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
288: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
304: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
320: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
336: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
352: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
368: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
384: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
400: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
416: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
432: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
448: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
464: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
480: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
496: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Speed Class: 10
UHS Speed Grade: >30MB/s
Original Bad Block Count: 11
Run-time Bad Block Count: 0
Spare Block Count: 46
Min. Erase Count MLC_TLC: 0
Max. Erase Count MLC_TLC: 0
Total Erase Count MLC_TLC: 0
Avg. Erase Count MLC_TLC: 0
Min. Erase Count SLC: 0
Max. Erase Count SLC: 1
Total Erase Count SLC: 12
Avg. Erase Count SLC: 0
Raw Capacity (MB): 120580
NAND P_E Cycles: 10000
Card Life: 100
Power ON_OFF Count: 2
```

For further details about the definition and meaning of the SD/microSD S.M.A.R.T. attributes, please refer to the documents listed in the **References** section of this technical note.

References

- “SD Specifications, Physical Layer Specification, version 3.01.”
- “ATP SD SMART Command Guideline_V2 181123_SD command 2707”
- “ATP SD SMART Command Guideline_V1.1 20201111_command 2702”
- “TN-SD-02: Enabling Memory Card Health Monitor System” from Micron Technology Inc.

Revision History

Date	Revision	Changes Compared to Previous
November 2020	1.0	

Certifications and Compliances

To meet the strictest quality standards and regulations expected by its demanding customers and industries, ATP spent an inordinate amount of efforts to ensure compliance with leading global certifications.

In addition, ATP has extensive product validation experience in industry-specific standards, including:

- AEC-Q100
- SNIA
- JESD219
- IEC 60529
- IP6X
- ATIS
- JESD22-A110
- MIL-STD-883
- IEC 61000-4-2:2008
- JESD78B
- UL94-v0

ATP TAIWAN (HQ)
TEL: +886-2-2659-6368
FAX: +886-2-2659-4982
sales-apac@atpinc.com

ATP USA
TEL: +1-408-732-5000
FAX: +1-408-732-5055
sales@atpinc.com

ATP JAPAN
TEL: +81-3-6260-0797
FAX: +81-3-6260-0798
sales-japan@atpinc.com

ATP EUROPE
TEL: +49-89-3749999-0
FAX: +49-89-3749999-29
sales-europe@atpinc.com

ATP CHINA
TEL: +86-21-5080-2220
FAX: +86-21-9687-0000-026
sales@cn.atpinc.com