EE445M/EE360L.6
Embedded and Real-Time Systems/
Real-Time Operating Systems

Lecture 8:
Secure Digital Card, DMA,
Filesystems

Secure Digital Card (SDC)

• Memory card standard
  – Upwards-compatible to multi-media card (MMC)
  – Reduced-size variants (miniSD, microSD)
  – Embedded micro-controller
  – Block based access (512 bytes/block)
  – Usually FAT file system

Source: http://elm-chan.org/docs/mmc/mmc_e.html
Other references: http://www.sdcard.org/home
http://www.ece.utexas.edu/~valvano/EE345M/SD_Physical_Layer_Spec.pdf
SD Card Interfacing

- Native SD/MMC mode or SPI
  - 4-bit and 1-bit native modes
  - 9 SDC pads (3 for power supply, 6 effective)
  - 2.7 to 3.6V power supply
  - Up to 15mA standby
  - Up to 50-100mA in write mode

ST7735 SDC Connector

- Using TM4C123 SPI interface
  - Two SSI0 slaves (TFT & Card via chip select)
Serial Peripheral Interface (SPI)

- Serial on-board, inter-IC connection
  - Motorola (Freescale)
  - Similar to I²C (Philips)
  - 3-4 wires, up to 20Mbps

SPI Physical Layer Protocol

- Synchronous (shared clock) protocol
  - Shift and latch on opposite clock edges
  - Four operating modes
    - Clock polarity (CPOL/SPO) and phase (CPHA/SPH)
    - SDC uses Mode 0 (CPOL=0, CPHA=0)

![SPI Diagram](image-url)
### SPI Command & Response

- Serial, byte-oriented communication
  - Fixed length (6 bytes) command frame packet
    - Host to device: CMD(1 byte), Arg(4 byte), CRC(1 byte)
  - Up to 8 bytes command response time (NCR)
    - Host continues to read & send (0xFF) bytes
- 1 or more bytes response (R1, R2, or R3/R7)

![Figure 6.13. SD command frame.](image)

**Table 6.6. SD commands.**

<table>
<thead>
<tr>
<th>Command Index</th>
<th>Argument Description</th>
<th>Response</th>
<th>Data Description</th>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMD0</td>
<td>None(0)</td>
<td>R1</td>
<td>No</td>
<td>IDLE_STATE</td>
<td>Software reset.</td>
</tr>
<tr>
<td>CMD1</td>
<td>None(0)</td>
<td>R1</td>
<td>No</td>
<td>SEND_OP_COND</td>
<td>Initiate initialization process.</td>
</tr>
<tr>
<td>ACMD41(*)</td>
<td>P2</td>
<td>R1</td>
<td>No</td>
<td>APP_SEND_OP_COND</td>
<td>For only SDC. Initiate initialization process.</td>
</tr>
<tr>
<td>CMD8</td>
<td>P3</td>
<td>R7</td>
<td>No</td>
<td>SEND_IF_COND</td>
<td>For only SDC V2. Check voltage range.</td>
</tr>
<tr>
<td>CMD9</td>
<td>None(0)</td>
<td>R1</td>
<td>Yes</td>
<td>SEND_CSD</td>
<td>Read CSD register.</td>
</tr>
<tr>
<td>CMD10</td>
<td>None(0)</td>
<td>R1b</td>
<td>No</td>
<td>SEND_CID</td>
<td>Read CID register.</td>
</tr>
<tr>
<td>CMD12</td>
<td>None(0)</td>
<td>R1b</td>
<td>No</td>
<td>STOP TRANSMISSION</td>
<td>Stop to read data.</td>
</tr>
<tr>
<td>CMD16</td>
<td>Block length[31:0]</td>
<td>R1</td>
<td>No</td>
<td>SET_BLOCKLEN</td>
<td>Change R/W block size.</td>
</tr>
<tr>
<td>CMD17</td>
<td>Address[31:0]</td>
<td>R1</td>
<td>Yes</td>
<td>READ_SINGLE_BLOCK</td>
<td>Read a block.</td>
</tr>
<tr>
<td>CMD18</td>
<td>Address[31:0]</td>
<td>R1</td>
<td>Yes</td>
<td>READ_MULTIPLE_BLOCK</td>
<td>Read multiple blocks.</td>
</tr>
<tr>
<td>CMD23</td>
<td>Number of block[15:0]</td>
<td>R1</td>
<td>No</td>
<td>SET_BLOCK_COUNT</td>
<td>For only MMC. Define number of blocks to transfer with next multi-block read/write command.</td>
</tr>
<tr>
<td>ACMD23(*)</td>
<td>Number of block[22:0]</td>
<td>R1</td>
<td>No</td>
<td>SET_WR_BLOCK_ERASE_COUNT</td>
<td>For only SDC. Define number of blocks to pre-erase with next multi-block write command.</td>
</tr>
<tr>
<td>CMD24</td>
<td>Address[31:0]</td>
<td>R1</td>
<td>Yes</td>
<td>WRITE_BLOCK</td>
<td>Write a block.</td>
</tr>
<tr>
<td>CMD25</td>
<td>Address[31:0]</td>
<td>R1</td>
<td>Yes</td>
<td>WRITE_MULTIPLE_BLOCK</td>
<td>Write multiple blocks.</td>
</tr>
<tr>
<td>CMD55(*)</td>
<td>None(0)</td>
<td>R1</td>
<td>No</td>
<td>APP_CMD</td>
<td>Leading command of ACMD&lt;n&gt; command.</td>
</tr>
<tr>
<td>CMD58</td>
<td>None(0)</td>
<td>R3</td>
<td>No</td>
<td>READ_OCR</td>
<td>Read Operations Condition Register (OCR). Indicates supported working voltage range.</td>
</tr>
</tbody>
</table>

*1: ACMD<n> means a command sequence of CMD55-CMD<n>.
*2: Rsv(0)[31], BCS[30], Rsv(0)[29].
*3: Rsv(0)[31:12], Supply Voltage[11:8], Check Pattern[0xAA][7:0].

---

**Lecture 8 J. Valvano, A. Gerstlauer**

**EE445M/EE380L.6**
### SDC Initialization Procedure

- **To put SDC into SPI mode**
  1. **Power ON (Insertion)**
     - After Vcc > 2.2V, wait for > 1ms
     - Set clock between 100 and 400 kHz
     - Set DI and CS high, send 74 or more clock pulses
  2. **Software reset (Set to SPI mode)**
     - Send CMD0 with CS low (and proper CRC)
     - Card enters SPI and responds with R1 idle state (0x01)
  3. **Initialization (CMD0, CMD1/ACMD41, CMD58)**
     - Send ACMD41 (SDCv1) or CMD1 (MMC)
     - Repeat until R1 response changes to 0x00 (100s of ms)
     - Increase clock rate (25MHz or more)
Data Transfer (1)

• Data packet and data response
  – Sent/received after command response
    • Data packet with token, data block, CRC
      – Token $FE$ for read/single-write, $FC$/F$D$ for multi-write

Data Packet

<table>
<thead>
<tr>
<th>Data Token</th>
<th>Data Block</th>
<th>CRC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 byte</td>
<td>1 - 2048 bytes</td>
<td>2 bytes</td>
</tr>
</tbody>
</table>

Data Token

1 1 1 1 1 1 0  Data token for CMD17/18/24
1 1 1 1 1 1 0  Data token for CMD25
1 1 1 1 1 0 1  Stop Trans token for CMD25

Data Response

X X X 0  Status 1

0 1 0  — Data accepted
1 0 1  — Data rejected due to a CRC error
1 1 0  — Data rejected due to a write error

Error Token

0 0 0  Flags

- Error
- CC error
- Card ECC failed
- Out of range
- Card is locked

Data Transfer (2)

• Single block read

It took 300 $\mu$s to setup and 535 $\mu$s to read one 512 byte block (logic analyzer resolution too slow for SCLK)
Data Transfer (3)

- Single block write

![Image of logic analyzer showing data transfer](image1)

It took 272μs to write one 512 byte block (logic analyzer resolution too slow for SCLK)

Data Transfer (4)

- Multi block read
- Multi block write

![Image of logic analyzer showing multi-block transfer](image2)
SD Card Driver

// DSTATUS of type BYTE (8 bits)
// STA_NOINIT  0x01  Drive not initialized
// STA_NODISK  0x02  No medium in the drive
// STA_PROTECT 0x04  Write protected
DSTATUS eDisk_Init(BYTE drv);

// DRESULT of type BYTE (8 bits)
// RES_OK       0: Successful
// RES_ERROR    1: R/W Error
// RES_WRPRT    2: Write Protected
// RES_NOTRDY   3: Not Ready
// RES_PARERR   4: Invalid Parameter
DRESULT eDisk_Status(BYTE drv);

//*************** eDisk_Read ***********
// Read 1 block of 512 bytes from the SD (write to RAM)
// Inputs: pointer to an empty RAM buffer
//         sector number of SD card to read: 0,1,2,...
DRESULT eDisk_ReadBlock (    BYTE *buff,  /* Pointer to buffer to store data */
                          DWORD sector /* Start sector number (LBA) */
)    

//*************** eDisk_Write ***********
// Write 1 block of 512 bytes of data to the SD card
// Inputs: pointer to RAM buffer with information
//         sector number of SD card to write: 0,1,2,...
DRESULT eDisk_WriteBlock (    const BYTE *buff, /* Pointer to data to be written */
                          DWORD sector /* Start sector number (LBA) */
)    

Read performance (i=0..100):

GPIOB->ODR |= 0x1000; // bit 12 on LED
result = disk_write(0,testBuff,i,1);
GPIOB->ODR &= ~0x1000; // bit 12 off LED

Write performance (i=0..100):

GPIOB->ODR |= 0x1000; // bit 12 on LED
result = disk_write(0,testBuff,i,1);
GPIOB->ODR &= ~0x1000; // bit 12 off LED

Kingston 2GB SD Memory Card: SD-M02G
Write block time: 2300 μs/block, 200 kB/s
Read block time: 552 μs/block, 1 MB/s

Kingston 4GB SD Memory Card: SD-K04G
Write block time: 4000 μs/block, 128 kB/s
Read block time: 665 μs/block, 0.77 MB/s

Mixed read-after-write:
result = disk_initialize(0);
if(result) printf("SD error=%u\n",result);
while(result==0){
    for(i=0;i<100;i++){
        GPIOB->ODR |= 0x1000;    // bit 12 on LED
        result = disk_write(0,testBuff,i,1);
        GPIOB->ODR &= ~0x1000;   // bit 12 off LED
        if(result) printf("SD write error=%u block=%u\n",result,i);
        GPIOB->ODR |= 0x8000;    // bit 15 on LED
        result = disk_read(0,buffer,i,1);
        GPIOB->ODR &= ~0x8000;   // bit 15 off LED
        if(result) printf("SD read error=%u block=%u\n",result,i);
    }
}

Kingston 2GB SD Memory Card: SD-M02G
Write block time: 2300 μs/block, 200 kB/s
Read block time: 272 μs/block, 1.9 MB/s
High-Speed Interfacing

- **Bandwidth**
  - Average or peak bytes transferred per second
- **Latency**
  - Interface latency
  - Device latency

<table>
<thead>
<tr>
<th>The time a need arises</th>
<th>The time the need is satisfied</th>
</tr>
</thead>
<tbody>
<tr>
<td>new input is available</td>
<td>the input data is read</td>
</tr>
<tr>
<td>new input is available</td>
<td>the input data is processed</td>
</tr>
<tr>
<td>output device is idle</td>
<td>new output data is written</td>
</tr>
<tr>
<td>sample time occurs</td>
<td>ADC is triggered, input data</td>
</tr>
<tr>
<td>periodic time occurs</td>
<td>output data, DAC is triggered</td>
</tr>
<tr>
<td>control point occurs</td>
<td>control system executed</td>
</tr>
</tbody>
</table>

- **Device latency**

High-Speed Applications (1)

- **Mass storage**
  1. Position head, wait for physical location (seek time)
  2. Transfer data

- Mass storage

  - **7200 RPM hard drive:** 70 MB/s
  - **SATA:** up to 300 MB/sec
  - **Original CD:** 150 kB/s
  - **52x CD:** 7.8 MB/s
  - **1x DVD:** 1.44 MB/s (9x CD)
  - **16x DVD:** 22 MB/s (144x CD)
  - **Class 2 SDC:** 2 MB/s (13x CD)
  - **Class 4 SDC:** 4 MB/s (26x CD)
High-Speed Applications (2)

- High speed data acquisition
  
  CD-quality sound
  16-bit, 44kHz, stereo: 176 kB/s

  Digital scope/signal generator
  8-bit, 1GHz: 1 MB/s

- High-speed signal generation

High-Speed Applications (3)

- Video displays
  
  VGA display
  256 colors (8-bit), 640x480, 60Hz: 18 MB/s

- Network communications

Ethernet
10Mbs: 1.2 MB/s
High-Speed Interfaces (1)

- **Hardware FIFO**
  - Software satisfies average bandwidth, but not peak guarantees (max. latency)

![Diagram of STM32F103 USART and 16550 UART](image)

High-Speed Interfaces (2)

- **Dual-port memory**
  - Shared memory between hardware & software
    - Framebuffer in video/graphics cards
  - Arbitrate between simultaneous accesses

![Diagram of memory and graphics hardware](image)
High-Speed Interfaces (3)

• Bank-switched memory
  – Double-buffering
    • Share memory, but avoid conflicts
  – Alternate between different banks or buffers
    • Hardware accesses bank A/B
    • Software accesses bank B/A
    • Switch banks (M=0/1)

Direct Memory Access (DMA)

• Transfer data directly
  – RAM/ROM <-> device
• Does not involve software/processor
  – Frees up CPU to do other tasks
• At speed of device/memory
DMA Initiation

- DMA Controller (DMAC)
  - System bus master to handle DMA transactions
  - Software programmed (memory-mapped registers)
- Software initiated DMA
  - Software to setup DMA controller
  - Software triggers DMA transfer
  - Software to check for completion (poll/interrupt)
- Hardware initiated DMA
  - Software to setup DMA controller
  - Hardware triggers DMA transfer
  - Software to check for completion (poll/interrupt)

Burst vs. Cycle Steal DMA

Figure 6.6. An input block is transferred all at once during burst mode DMA.

Figure 6.7. Each time an input byte is ready it is transferred to memory using single cycle DMA.
Single Address DMA

Figure 6.8. Block diagram showing the modules involved in a disk read.

Figure 6.9. Timing diagram of a single address DMA-controlled floppy disk read.

Dual Address DMA

Figure 6.10. Block diagram showing the modules involved in a SPI read.

Figure 6.11. Timing diagram of a dual address DMA-controlled SPI read.
TM4C123 DMA Programming

DMA Channels

Table 9-1: μDMA Channel Assignments (DMACHMAPn, High/Low Priority via DMAPRISET/DMAPRICLR)

<table>
<thead>
<tr>
<th>Channel</th>
<th>Peripheral Type</th>
<th>Peripheral</th>
<th>Channel</th>
<th>Peripheral Type</th>
<th>Peripheral</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>USB2 EP2 RX</td>
<td>Software</td>
<td>1</td>
<td>USB2 EP2 TX</td>
<td>Software</td>
</tr>
<tr>
<td>2</td>
<td>USB2 EP1 RX</td>
<td>Software</td>
<td>3</td>
<td>USB2 EP1 TX</td>
<td>Software</td>
</tr>
<tr>
<td>4</td>
<td>USB2 EPS RX</td>
<td>Software</td>
<td>5</td>
<td>USB2 EPS TX</td>
<td>Software</td>
</tr>
<tr>
<td>6</td>
<td>Software</td>
<td>Software</td>
<td>7</td>
<td>Software</td>
<td>Software</td>
</tr>
<tr>
<td>8</td>
<td>UAR10 RX</td>
<td>Software</td>
<td>9</td>
<td>UAR10 TX</td>
<td>Software</td>
</tr>
<tr>
<td>10</td>
<td>UAR10 TX</td>
<td>Software</td>
<td>11</td>
<td>DDS0 TX</td>
<td>Software</td>
</tr>
<tr>
<td>12</td>
<td>Software</td>
<td>Software</td>
<td>13</td>
<td>Software</td>
<td>Software</td>
</tr>
<tr>
<td>14</td>
<td>ADC0 SS0</td>
<td>Software</td>
<td>15</td>
<td>ADC0 SS1</td>
<td>Software</td>
</tr>
<tr>
<td>16</td>
<td>ADC0 SS2</td>
<td>Software</td>
<td>17</td>
<td>ADC0 SS3</td>
<td>Software</td>
</tr>
<tr>
<td>18</td>
<td>ADC0 SS2</td>
<td>Software</td>
<td>19</td>
<td>ADC0 SS3</td>
<td>Software</td>
</tr>
<tr>
<td>20</td>
<td>ADC0 SS1</td>
<td>Software</td>
<td>21</td>
<td>ADC0 SS0</td>
<td>Software</td>
</tr>
<tr>
<td>22</td>
<td>ADC0 SS0</td>
<td>Software</td>
<td>23</td>
<td>ADC0 SS1</td>
<td>Software</td>
</tr>
<tr>
<td>24</td>
<td>ADC0 SS2</td>
<td>Software</td>
<td>25</td>
<td>ADC0 SS3</td>
<td>Software</td>
</tr>
<tr>
<td>26</td>
<td>ADC0 SS3</td>
<td>Software</td>
<td>27</td>
<td>ADC0 SS1</td>
<td>Software</td>
</tr>
<tr>
<td>28</td>
<td>ADC0 SS0</td>
<td>Software</td>
<td>29</td>
<td>ADC0 SS2</td>
<td>Software</td>
</tr>
<tr>
<td>30</td>
<td>ADC0 SS3</td>
<td>Software</td>
<td>31</td>
<td>Reserved</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

Transfer Buffers Used by μDMA

System Memory

DMA Channels (DMACHMAPn, High/Low Priority via DMAPRISET/DMAPRICLR)
DMA Channel Control Structure

- Two per channel in main memory
  - Primary array (DMACTLBASE)
  - Alternate (DMAALTBASE) for continuous ping-pong

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source address</td>
<td>Address of the module (memory or input) that generates the data</td>
</tr>
<tr>
<td>Destination address</td>
<td>Address of the module (memory or output) that accepts the data</td>
</tr>
<tr>
<td>DSTINC</td>
<td>Automatically +1/2/4/0 the destination address after each transfer</td>
</tr>
<tr>
<td>DSTSIZE</td>
<td>Destination data size (byte/halfword/word)</td>
</tr>
<tr>
<td>SRCINC</td>
<td>Automatically +1/2/4/0 the source address after each transfer</td>
</tr>
<tr>
<td>SRCSIZE</td>
<td>Source data size (byte/halfword/word)</td>
</tr>
<tr>
<td>ARBSIZE</td>
<td>Size of bursts between bus arbitrations (powers of 2)</td>
</tr>
<tr>
<td>XFERSIZE</td>
<td>Number of items to transfer</td>
</tr>
<tr>
<td>NXTUSE</td>
<td>Next use burst for scatter-gather transfers</td>
</tr>
<tr>
<td>XFERMODE</td>
<td>Transfer mode (auto-request, ping-pong, etc.) and transfer status</td>
</tr>
</tbody>
</table>

Software-Triggered Memory-to-Memory DMA

```c
// The ucControlTable table must be aligned to a 1024 byte boundary.
uint32_t ucControlTable[256] __attribute__((aligned(1024)));
#define CH30 (30*4)     // channel 30 for SW-triggered
#define BIT30 0x40000000 // ************DMA_Init*****************
// Initialize the memory to memory transfer
// This needs to be called once before requesting a transfer
void DMA_Init(void){
  volatile uint32_t delay;
  SYSCTL_RCGCDMA_R = 0x01;   // µDMA Module Run Mode Clock Gating Control
  delay = SYSCTL_RCGCDMA_R;  // allow time to finish
  UDMA_CFG_R = 0x01;         // MASTEN Controller Master Enable
  UDMA_CTLBASE_R = (uint32_t)ucControlTable;
  UDMA_PRIOCLR_R = BIT30;    // default, not high priority
  UDMA_ALTCLR_R = BIT30;     // use primary control
  UDMA_USEBURSTCLR_R = BIT30; // responds to both burst and single
  UDMA_REQMASKCLR_R = BIT30;  // allow controller to recognize requests
}

// ************DMA_Xfr *****************
// Called to transfer words from source to destination
// Inputs: src is a pointer to the first element of the original data
//         dest is a pointer to a place to put the copy
//         cnt is the number of words to transfer (max is 1024 words)
void DMA_Xfr(uint32_t *src, uint32_t *dest, uint32_t cnt){
  ucControlTable[CH30] = (uint32_t)src+cnt*4-1;   // last address
  ucControlTable[CH30+1] = (uint32_t)dest+cnt*4-1;  // last address
  ucControlTable[CH30+2] = 0xAA00C002+((cnt-1)<<4);      // Control Word
  /* DMACHCTL          Bits    Value Description
   * DSTINC            31:30   2     32-bit destination address increment
   * DSTSIZE           29:28   2     32-bit destination data size
   * SRCINC            27:26   2     32-bit source address increment
   * SRCSIZE           25:24   2     32-bit source data size
   * ARBSIZE           17:14   3     Arbitrates after 8 transfers
   * XFERSIZE          13:4  cnt-1   Transfer cnt items
   * NXTUSEBURST       3      0     N/A for this transfer type
   * XFERMODE          2:0     2     Use Auto-request transfer mode
   */
  UDMA_ENASET_R = BIT30;  // µDMA Channel 30 is enabled
  UDMA_ENASYS_E = BIT30; // software start, do not wait for completion
}
```
Lab 5 File System

• Layered software architecture
  – SSI <-> SDC
  – eDisk <-> physical blocks
    • Optional DMA for transfers
  – eFile <-> logical data

Know Your problem

• Read access
  – Sequential versus random access
• Write access
  – Sequential versus random access
  – Insert/Append/Remove
  – Write once (data logger, flight recorder)
• Size, bandwidth, response time
• Reliability
• Security (fail-safe)
Know your disk

- Block size
- Disk size
- Read/write speed
- Types and chances of error
  - Wear leveling
  - Conditional probability

File System Responsibilities

- Logical to physical translation
  - Byte number to block number
- Directory
  - File name to physical translation
- Free space
  - Used
  - Free
  - Damaged
File System Performance

- File size
- Disk size
- Number of files
- Speed
  - Time to create, open, close
  - Write bandwidth
  - Read bandwidth
- Fragmentation
  - External if max file size < total free space

File System Allocation (1)

- Contiguous allocation
  - First fit
  - Best fit
  - Worst fit

Good for sequential write, never erase
Fast random read access

Internal fragmentation: on average, each file wastes 1/2 block
External fragmentation: largest file size to allocate < free space
File System Allocation (2)

- **Linked allocation**

  Each block has a link and a size
  
  Good for erase, append, delete
  Slow for random access
  Internal fragmentation: on average, each file wastes 1/2 block
  No external fragmentation

Free Space Management

- **Linked allocation of free space**

  To handle wear-leveling
  free to one end of list
  allocate from other end

  What if bad block?
File System Allocation (3)

• Indexed allocation

Good for erase, append, delete
Fast for random access
No external fragmentation

Directory

• Name, Type, Date, Size, How to access

Internal fragmentation

How many files?

2 bytes for link to next block
2 bytes for size

Disk smaller than 32Meb
Directory

- Name, Type, Date, Size, How to access

```
<table>
<thead>
<tr>
<th>Directory in Block 0</th>
<th>Block1</th>
<th>Block2</th>
<th>Block3</th>
<th>Block4</th>
<th>Block5</th>
<th>Block6</th>
</tr>
</thead>
<tbody>
<tr>
<td>'jv1'</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>508</td>
<td>20</td>
<td>508</td>
<td>24</td>
<td>508</td>
<td></td>
</tr>
<tr>
<td>'tree'</td>
<td>Free</td>
<td>'jv1'</td>
<td>Data</td>
<td>'Jon'</td>
<td>'Jon'</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>519</td>
<td>20</td>
<td>11</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>1040</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

How many files?
- 4 bytes for link to next block
- 2 bytes for size
- Disc larger than 32Meb

Free Space Management

- Linked
- Bit vector (7.5)

```
<table>
<thead>
<tr>
<th>directory in block 0</th>
<th>block1</th>
<th>block2</th>
<th>block3</th>
<th>block4</th>
<th>block5</th>
<th>block6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0, - , -</td>
<td>free</td>
<td>free</td>
<td>free</td>
<td>free</td>
<td>free</td>
<td>free</td>
</tr>
<tr>
<td>0, - , -</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0, - , -</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0, - , -</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0, - , -</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0, - , -</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0, - , -</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

What percentage of the disk is wasted using a) linked; b) bit vector?
File Allocation Table (FAT)

Derive a relation between FAT size and disk size.

File Allocation Table Disk

Derive a relation between FAT size and disk size.

External fragmentation?

Internal fragmentation?

What if bad block?

Why cluster?
## File System Summary

- **Internal fragmentation**
- **External fragmentation**
- **Speed**
  - Random versus sequential
  - Read versus write
- **Reliability, recover from errors**
  - Error detection
  - Redundant Array of Independent Disks
  - Wear-leveling
- **Clustering**
- **Size**
- **Number of files**
- **Legacy**
- **Low voltage**

Lecture 8  
J. Valvano, A. Gerstlauer  
EE445M/EE380L.6