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

Lecture 8:

Secure Digital Card, DMA, Filesystems

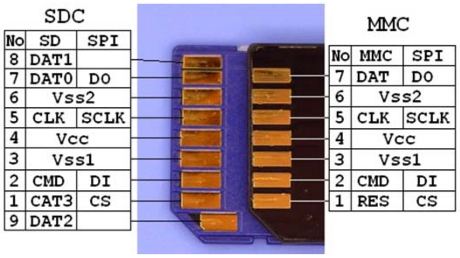
Secure Digital Card (SDC)

- 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



SDC

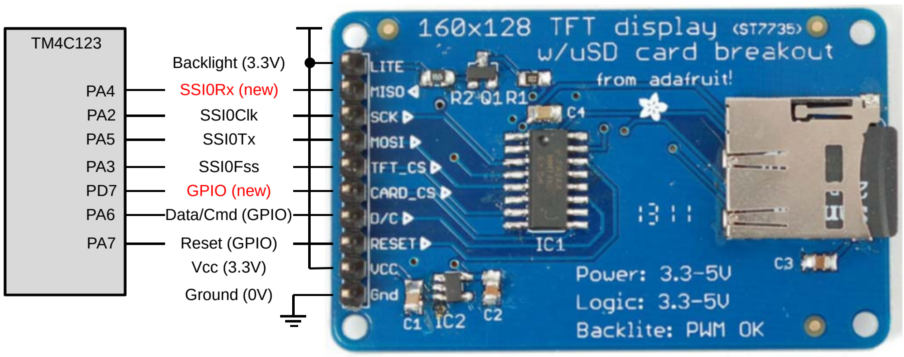
No	SD	SPI
8	DAT1	
7	DAT0	DO
6	Vss2	
5	CLK	SCLK
4	Vcc	
3	Vss1	
2	CMD	DI
1	CAT3	CS
9	DAT2	

MMC

No	MMC	SPI
7	DAT	DO
6	Vss2	
5	CLK	SCLK
4	Vcc	
3	Vss1	
2	CMD	DI
1	RES	CS

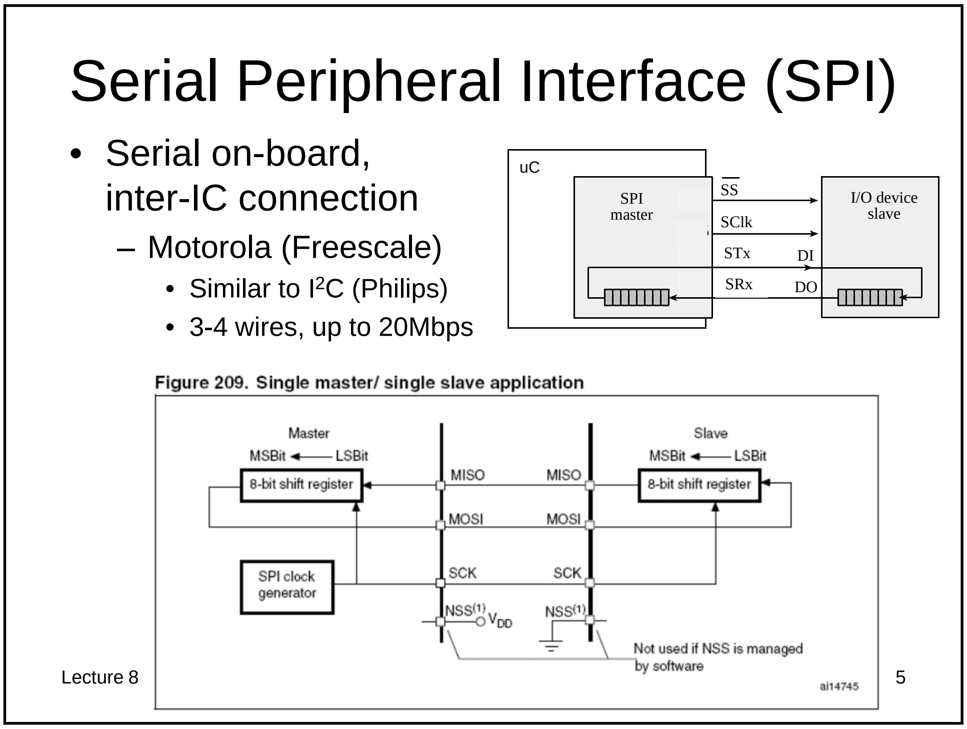
ST7735 SDC Connector

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



TM4C123

TM4C123	Signal
	Backlight (3.3V)
PA4	SSI0Rx (new)
PA2	SSI0Clk
PA5	SSI0Tx
PA3	SSI0Fss
PD7	GPIO (new)
PA6	Data/Cmd (GPIO)
PA7	Reset (GPIO)
	Vcc (3.3V)
	Ground (0V)

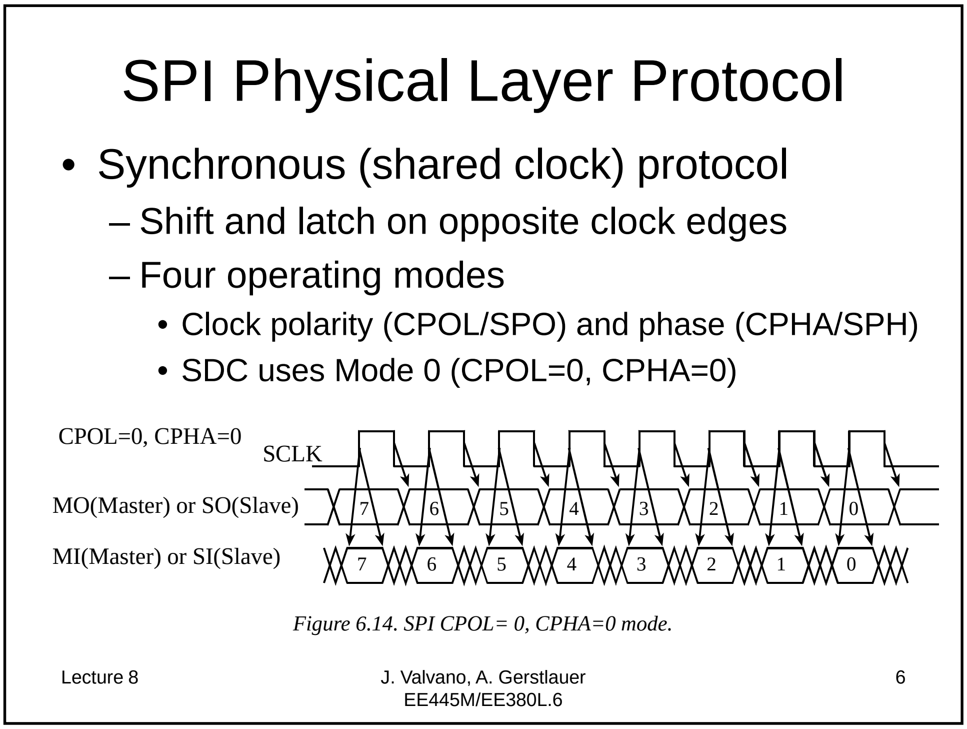


Serial Peripheral Interface (SPI)

- Serial on-board, inter-IC connection
  - Motorola (Freescale)
    - Similar to I^{2}C (Philips)
    - 3-4 wires, up to 20Mbps

Figure 209. Single master/ single slave application

Lecture 8 5

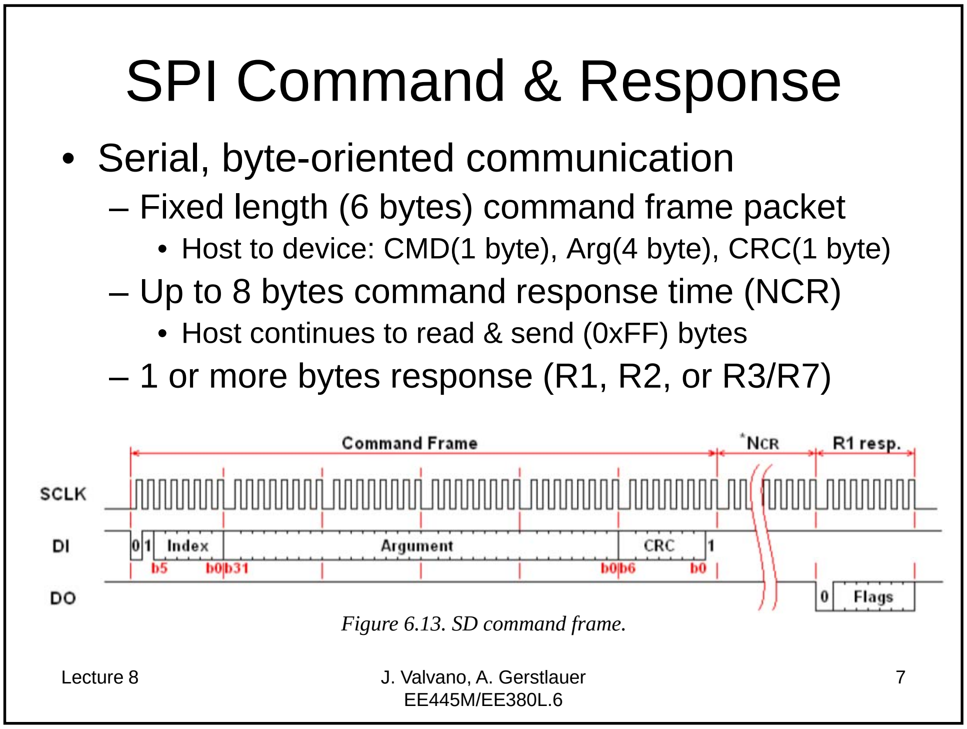


SPI Physical Layer Protocol

- 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)

Figure 6.14. SPI CPOL= 0, CPHA=0 mode.

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

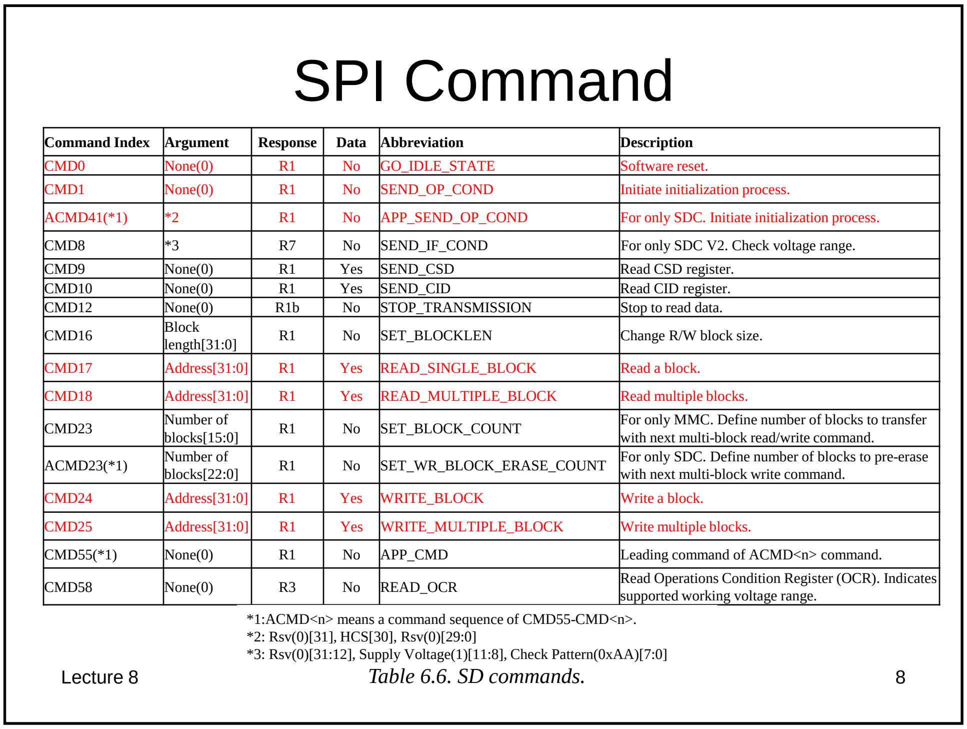


SPI Command & Response

- 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.

Lecture 8 J. Valvano, A. Gerstlauer EE445M/EE380L.6 7



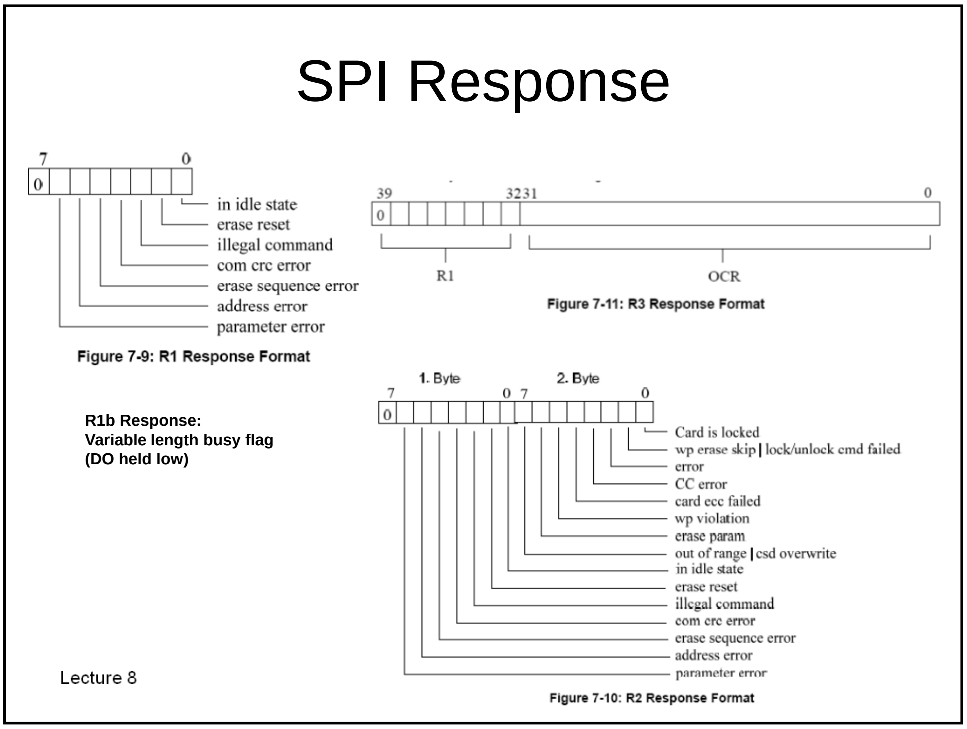
SPI Command

Command Index	Argument	Response	Data	Abbreviation	Description
CMD0	None(0)	R1	No	GO_IDLE_STATE	Software reset.
CMD1	None(0)	R1	No	SEND_OP_COND	Initiate initialization process.
ACMD41(*1)	*2	R1	No	APP_SEND_OP_COND	For only SDC. Initiate initialization process.
CMD8	*3	R7	No	SEND_IF_COND	For only SDC V2. Check voltage range.
CMD9	None(0)	R1	Yes	SEND_CSD	Read CSD register.
CMD10	None(0)	R1	Yes	SEND_CID	Read CID register.
CMD12	None(0)	R1b	No	STOP_TRANSMISSION	Stop to read data.
CMD16	Block length[31:0]	R1	No	SET_BLOCKLEN	Change R/W block size.
CMD17	Address[31:0]	R1	Yes	READ_SINGLE_BLOCK	Read a block.
CMD18	Address[31:0]	R1	Yes	READ_MULTIPLE_BLOCK	Read multiple blocks.
CMD23	Number of blocks[15:0]	R1	No	SET_BLOCK_COUNT	For only MMC. Define number of blocks to transfer with next multi-block read/write command.
ACMD23(*1)	Number of blocks[22:0]	R1	No	SET_WR_BLOCK_ERASE_COUNT	For only SDC. Define number of blocks to pre-erase with next multi-block write command.
CMD24	Address[31:0]	R1	Yes	WRITE_BLOCK	Write a block.
CMD25	Address[31:0]	R1	Yes	WRITE_MULTIPLE_BLOCK	Write multiple blocks.
CMD55(*1)	None(0)	R1	No	APP_CMD	Leading command of ACMD<n> command.
CMD58	None(0)	R3	No	READ_OCR	Read Operations Condition Register (OCR). Indicates supported working voltage range.

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

Table 6.6. SD commands.

Lecture 8 8



SPI Response

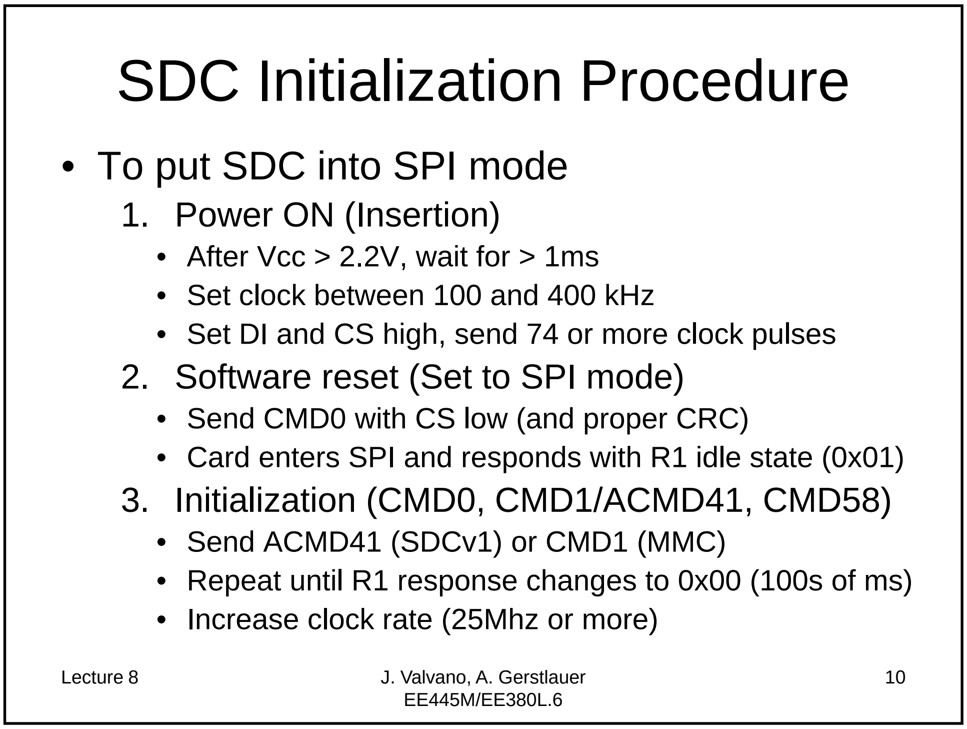
Figure 7-9: R1 Response Format

Figure 7-11: R3 Response Format

R1b Response:
Variable length busy flag
(DO held low)

Figure 7-10: R2 Response Format

Lecture 8



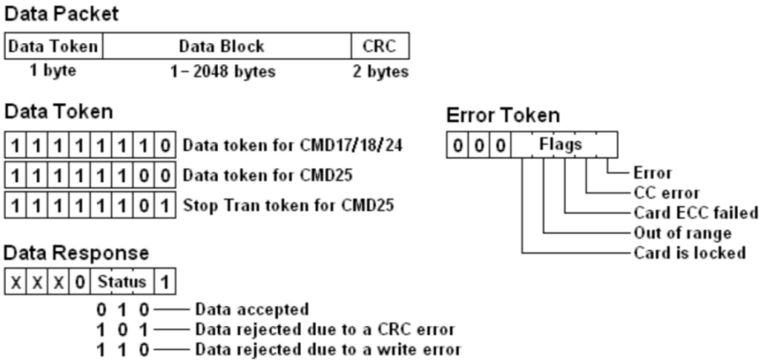
SDC Initialization Procedure

- 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)

Lecture 8 J. Valvano, A. Gerstlauer
EE445M/EE380L.6 10

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, $FC/$FD for multi-write



Data Packet

Data Token	Data Block	CRC
1 byte	1 – 2048 bytes	2 bytes

Data Token

1 1 1 1 1 1 1 0 Data token for CMD17/18/24

1 1 1 1 1 1 0 0 Data token for CMD25

1 1 1 1 1 1 0 1 Stop Tran token for CMD25

Error Token

0 0 0 Flags

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

Data Response

X X X 0 Status 1

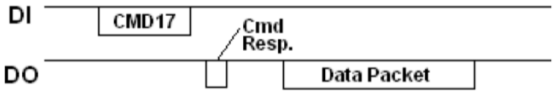
0 1 0 — Data accepted

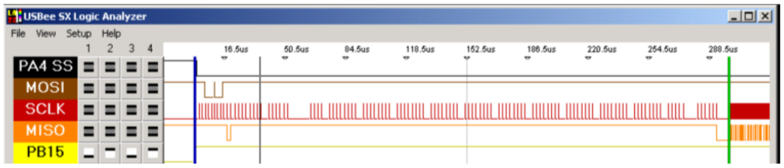
1 0 1 — Data rejected due to a CRC error

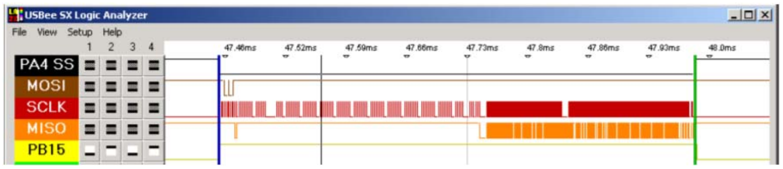
1 1 0 — Data rejected due to a write error

Data Transfer (2)

- Single block read



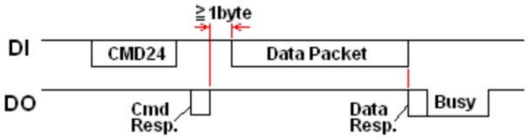


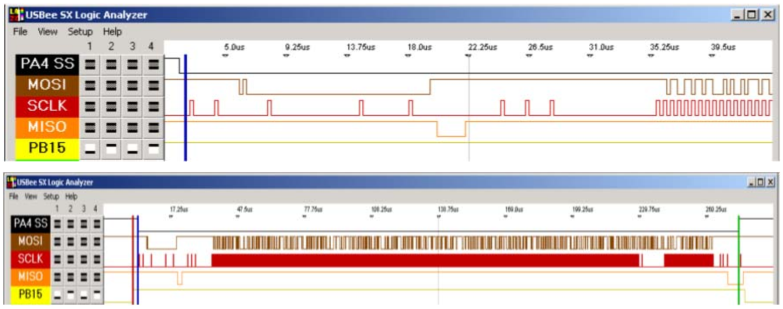


It took 300μs to setup and 535μs to read one 512 byte block
(logic analyzer resolution too slow for SCLK)

Data Transfer (3)

- Single block write

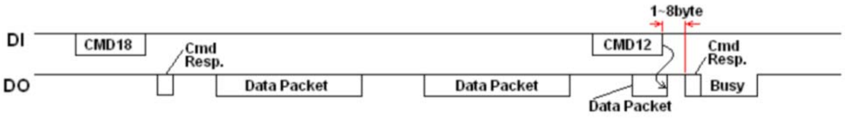




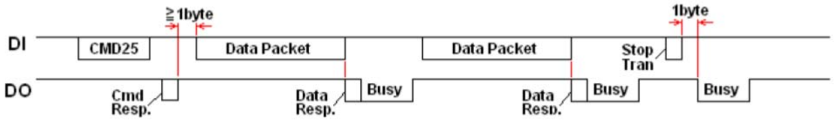
It took 272µs to write one 512 byte block
(logic analyzer resolution too slow for SCLK)

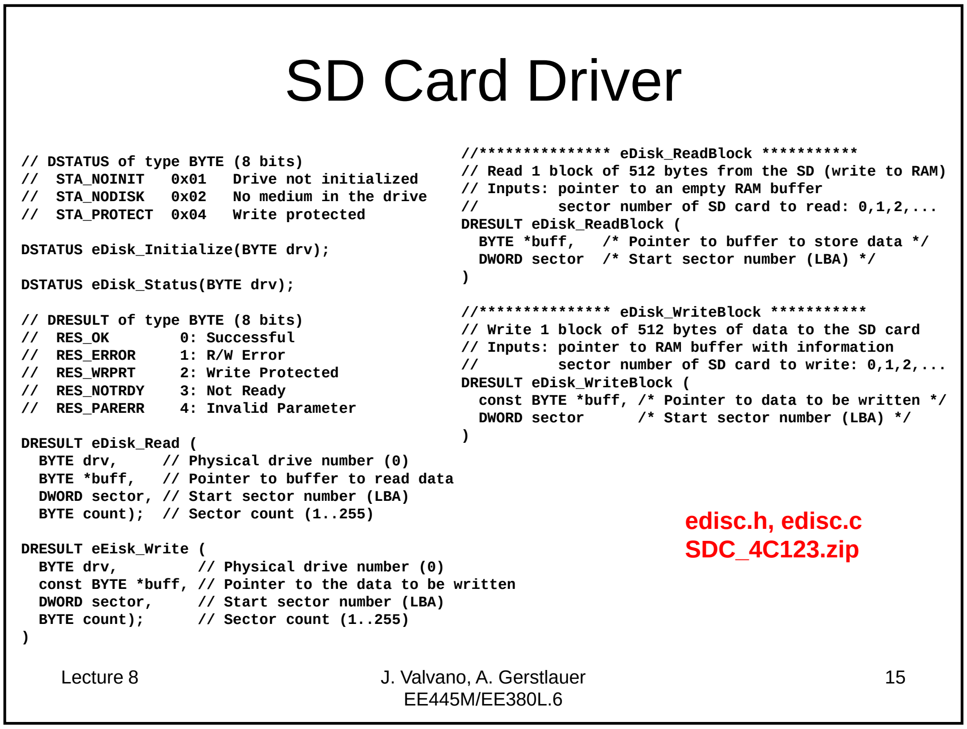
Data Transfer (4)

- Multi block read



- Multi block write





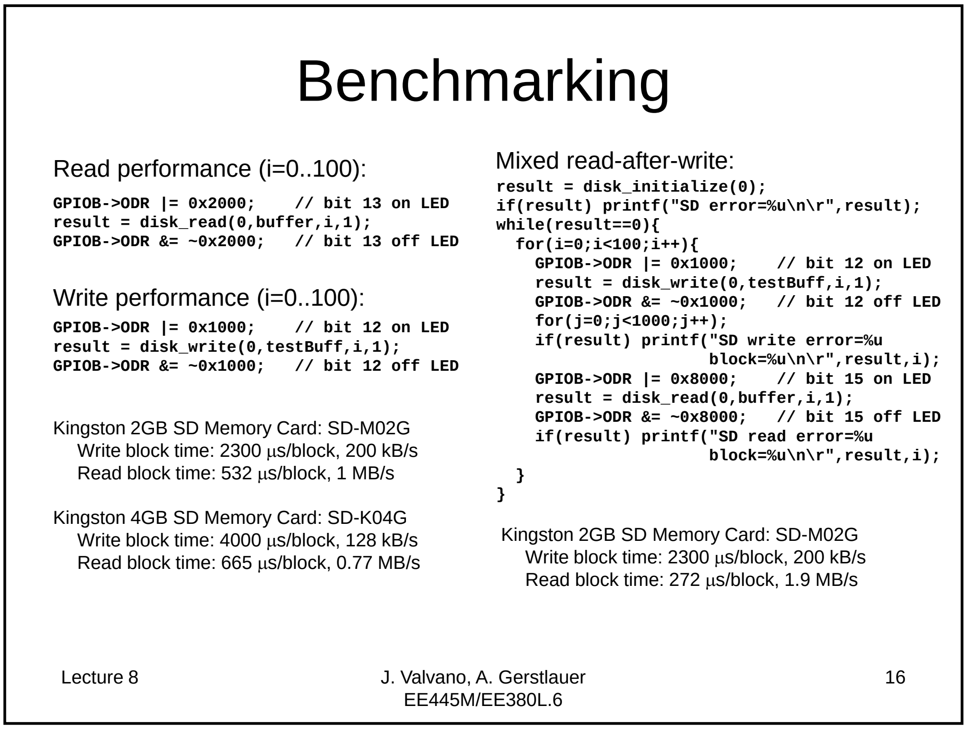
SD Card Driver

```
// 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_Initialize(BYTE drv);

DSTATUS eDisk_Status(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_Read (
  BYTE drv,     // Physical drive number (0)
  BYTE *buff,   // Pointer to buffer to read data
  DWORD sector, // Start sector number (LBA)
  BYTE count);  // Sector count (1..255)

DRESULT eEisk_Write (
  BYTE drv,         // Physical drive number (0)
  const BYTE *buff, // Pointer to the data to be written
  DWORD sector,     // Start sector number (LBA)
  BYTE count);      // Sector count (1..255)
)
```

```
//*************** eDisk_ReadBlock ***********
// 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_WriteBlock ***********
// 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) */
)
```

edisc.h, edisc.c
SDC_4C123.zip

Lecture 8 J. Valvano, A. Gerstlauer EE445M/EE380L.6 15



Benchmarking

Read performance (i=0..100):

```
GPIOB->ODR |= 0x2000;    // bit 13 on LED
result = disk_read(0,buffer,i,1);
GPIOB->ODR &= ~0x2000;   // bit 13 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: 532 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\r",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
    for(j=0;j<1000;j++);
    if(result) printf("SD write error=%u
                  block=%u\n\r",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\r",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

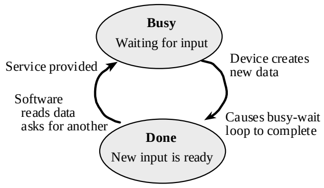
Lecture 8 J. Valvano, A. Gerstlauer EE445M/EE380L.6 16

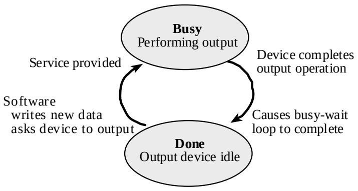
High-Speed Interfacing

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

The time a need arises	The time the need is satisfied
new input is available	the input data is read
new input is available	the input data is processed
output device is idle	new output data is written
sample time occurs	ADC is triggered, input data
periodic time occurs	output data, DAC is triggered
control point occurs	control system executed

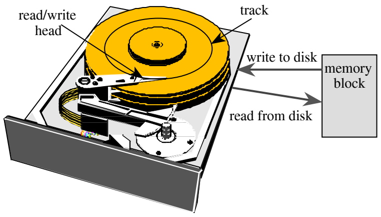
  - Device latency



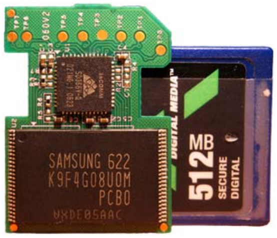


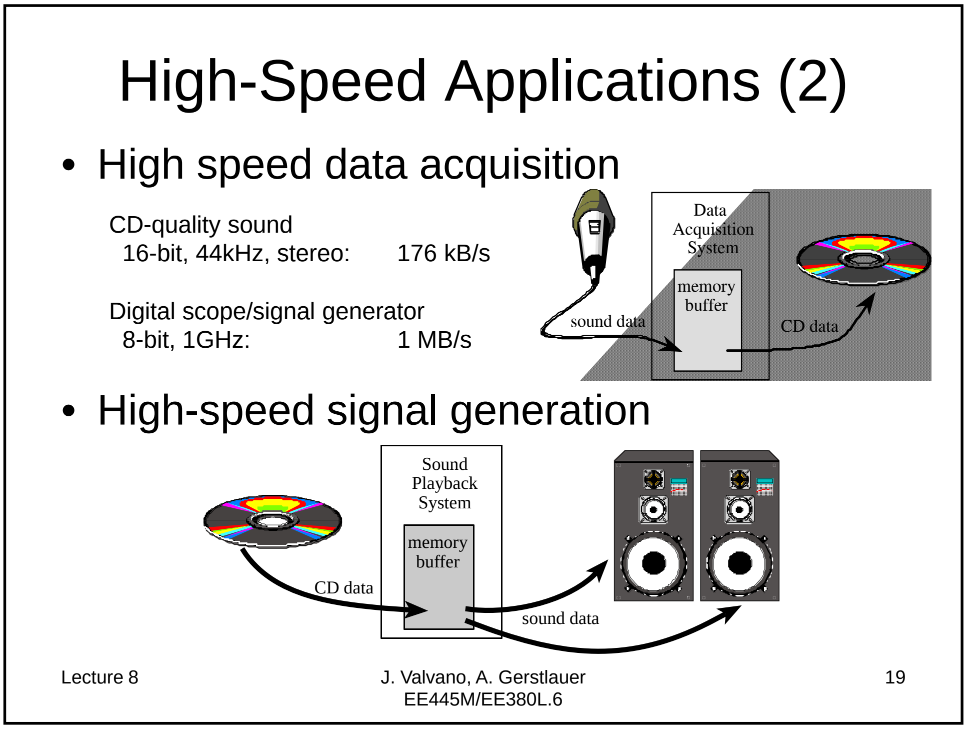
High-Speed Applications (1)

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



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: 1.4MB/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)





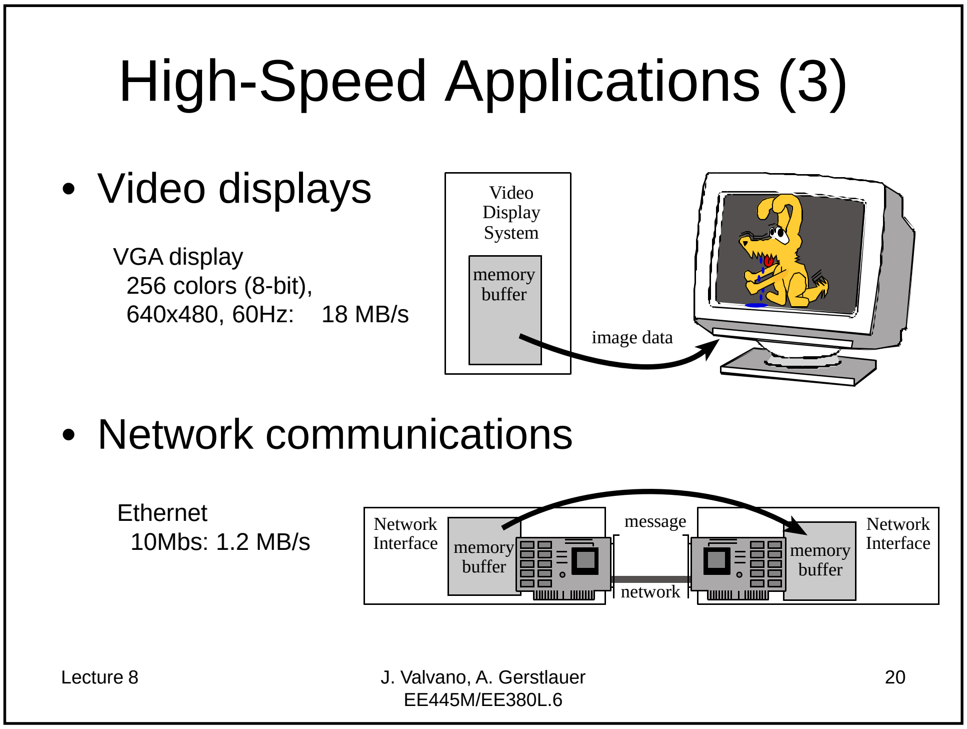
High-Speed Applications (2)

- 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

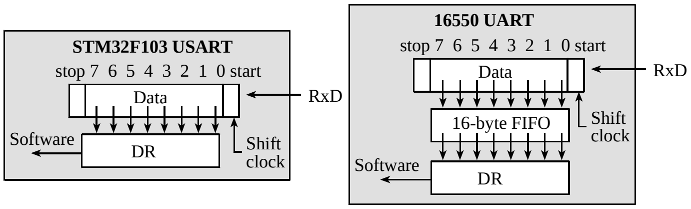
VGA display
256 colors (8-bit),
640x480, 60Hz: 18 MB/s

- Network communications

Ethernet
10Mbs: 1.2 MB/s

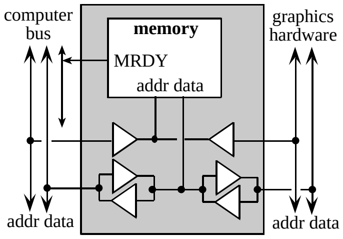
High-Speed Interfaces (1)

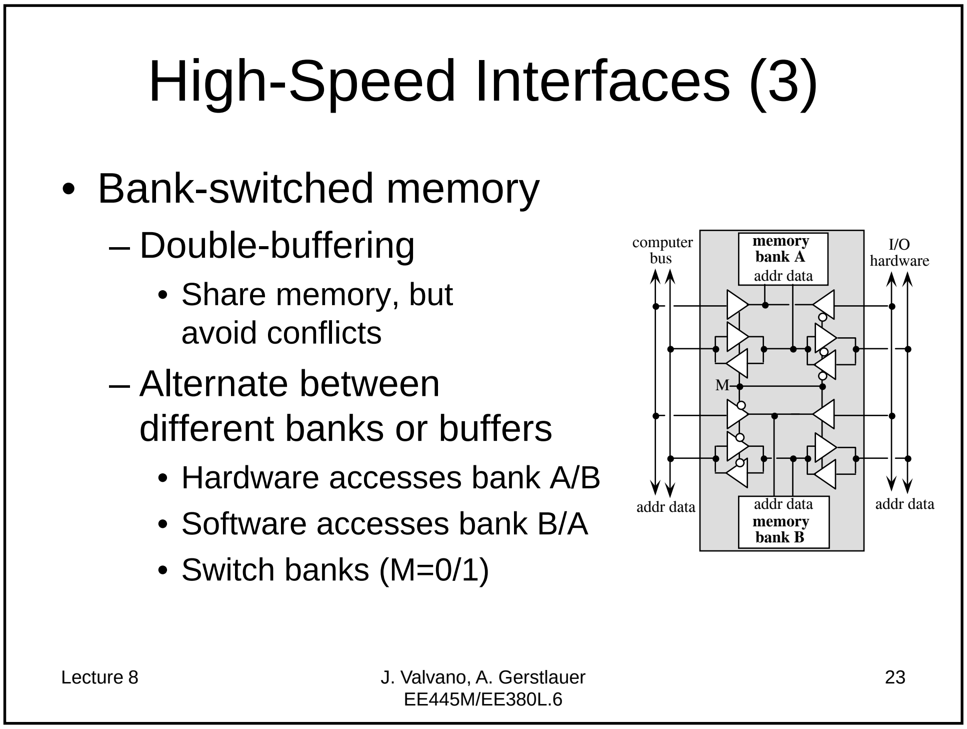
- Hardware FIFO
  - Software satisfies average bandwidth, but not peak guarantees (max. latency)



High-Speed Interfaces (2)

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

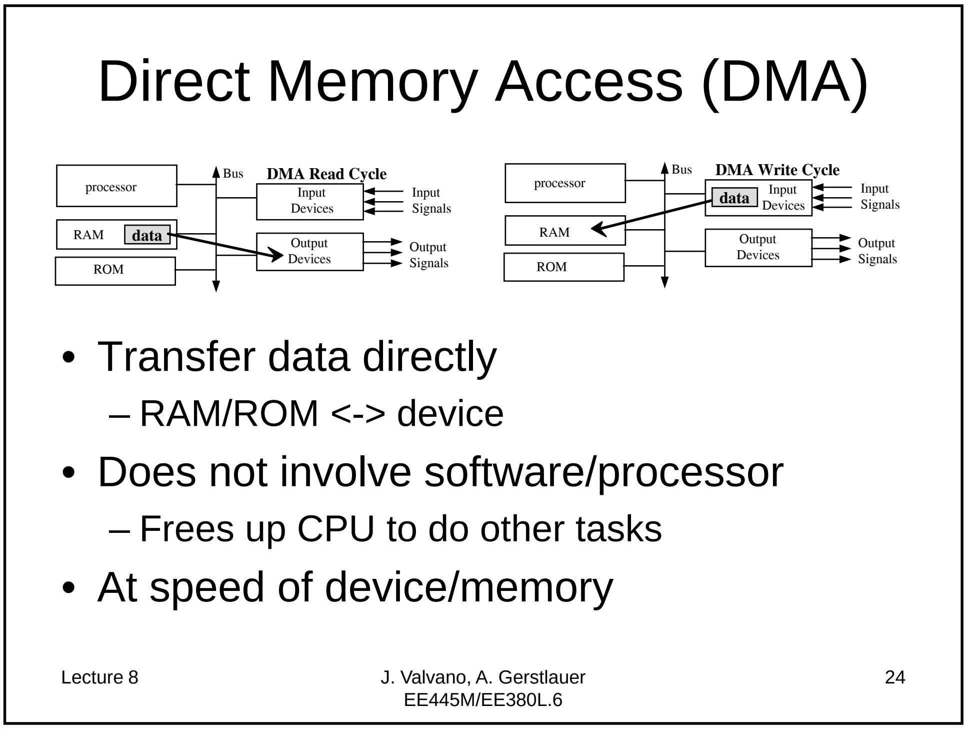




High-Speed Interfaces (3)

- 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)

computer bus
memory bank A
addr data
I/O hardware
M
addr data
addr data
memory bank B
addr data



Direct Memory Access (DMA)

processor
Bus
DMA Read Cycle
Input Devices
Input Signals
RAM
data
Output Devices
Output Signals
ROM
processor
Bus
DMA Write Cycle
data
Input Devices
Input Signals
RAM
Output Devices
Output Signals
ROM

- 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

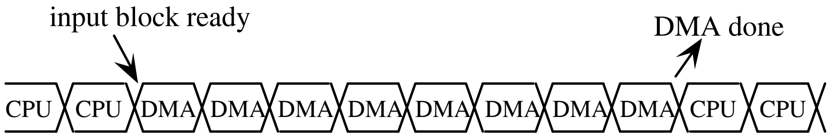


Figure 6.6. An input block is transferred all at once during burst mode DMA.

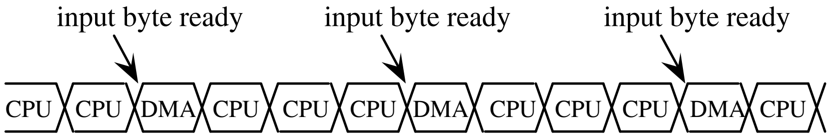
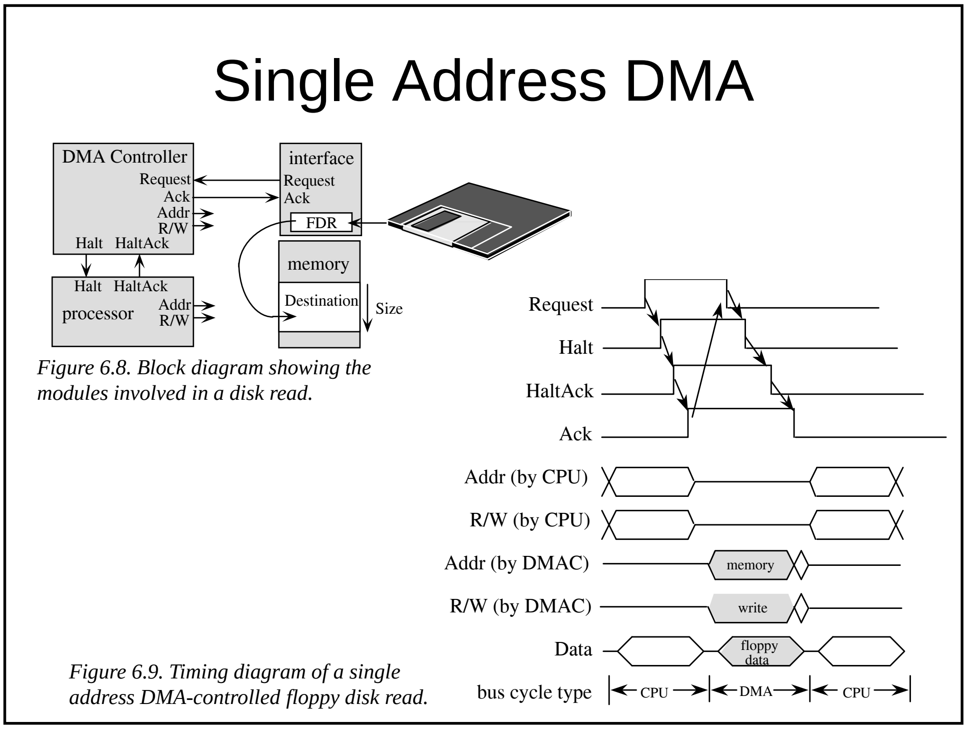


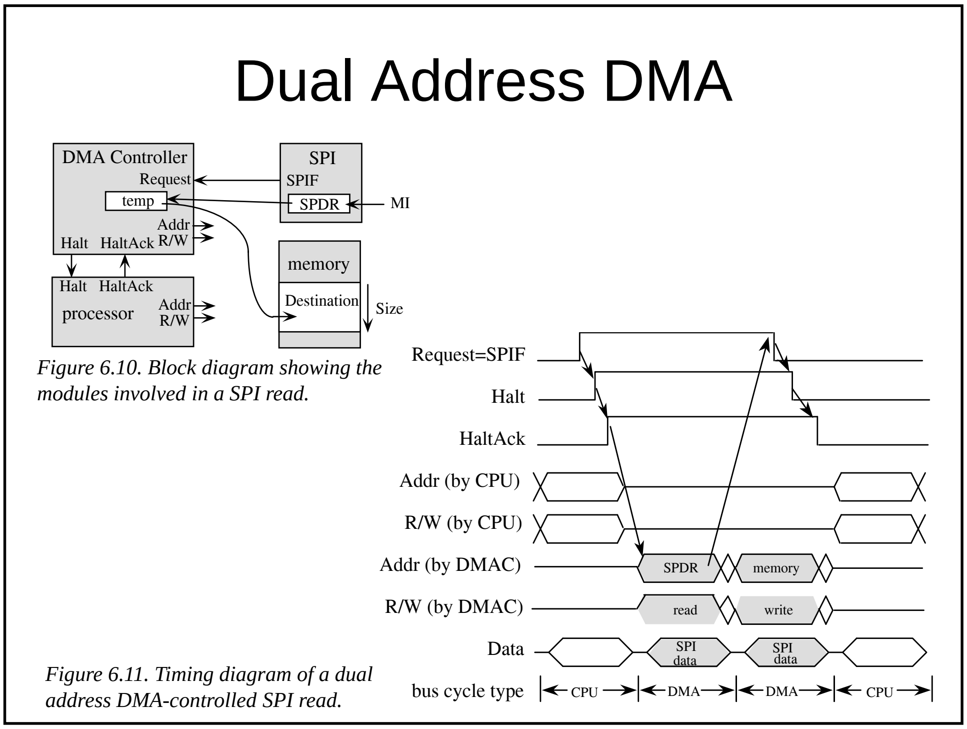
Figure 6.7. Each time an input byte is ready it is transferred to memory using single cycle DMA.



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.



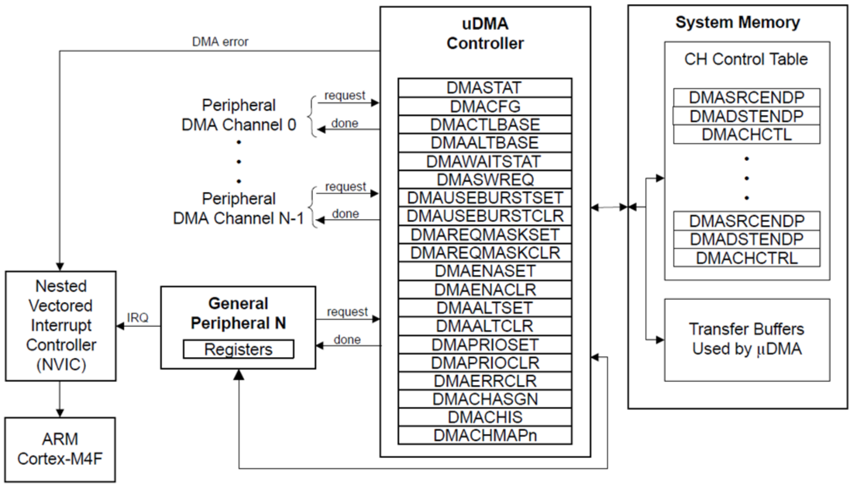
Dual Address DMA

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.

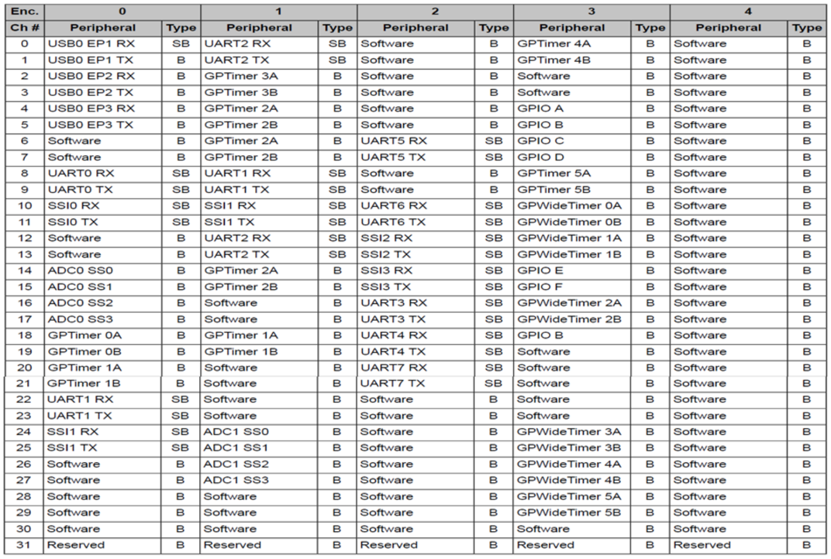
TM4C123 DMA Programming

Figure 9-1. µDMA Block Diagram

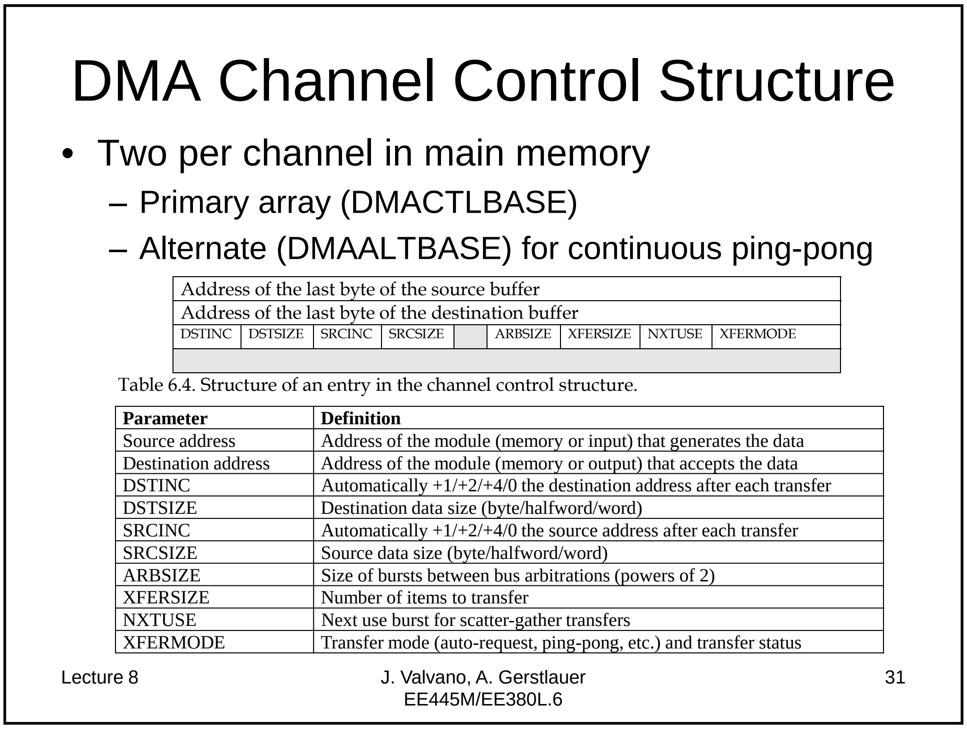


DMA Channels

Table 9-1. µDMA Channel Assignments **(DMACHMAP*n*, High/Low Priority via DMAPRIOSET/DMAPRIOCLR)**



Enc.	0		1		2		3		4	
Ch #	Peripheral	Type	Peripheral	Type	Peripheral	Type	Peripheral	Type	Peripheral	Type
0	USB0 EP1 RX	SB	UART2 RX	SB	Software	B	GPTimer 4A	B	Software	B
1	USB0 EP1 TX	B	UART2 TX	SB	Software	B	GPTimer 4B	B	Software	B
2	USB0 EP2 RX	B	GPTimer 3A	B	Software	B	Software	B	Software	B
3	USB0 EP2 TX	B	GPTimer 3B	B	Software	B	Software	B	Software	B
4	USB0 EP3 RX	B	GPTimer 2A	B	Software	B	GPIO A	B	Software	B
5	USB0 EP3 TX	B	GPTimer 2B	B	Software	B	GPIO B	B	Software	B
6	Software	B	GPTimer 2A	B	UART5 RX	SB	GPIO C	B	Software	B
7	Software	B	GPTimer 2B	B	UART5 TX	SB	GPIO D	B	Software	B
8	UART0 RX	SB	UART1 RX	SB	Software	B	GPTimer 5A	B	Software	B
9	UART0 TX	SB	UART1 TX	SB	Software	B	GPTimer 5B	B	Software	B
10	SSI0 RX	SB	SSI1 RX	SB	UART6 RX	SB	GPWideTimer 0A	B	Software	B
11	SSI0 TX	SB	SSI1 TX	SB	UART6 TX	SB	GPWideTimer 0B	B	Software	B
12	Software	B	UART2 RX	SB	SSI2 RX	SB	GPWideTimer 1A	B	Software	B
13	Software	B	UART2 TX	SB	SSI2 TX	SB	GPWideTimer 1B	B	Software	B
14	ADC0 SS0	B	GPTimer 2A	B	SSI3 RX	SB	GPIO E	B	Software	B
15	ADC0 SS1	B	GPTimer 2B	B	SSI3 TX	SB	GPIO F	B	Software	B
16	ADC0 SS2	B	Software	B	UART3 RX	SB	GPWideTimer 2A	B	Software	B
17	ADC0 SS3	B	Software	B	UART3 TX	SB	GPWideTimer 2B	B	Software	B
18	GPTimer 0A	B	GPTimer 1A	B	UART4 RX	SB	GPIO B	B	Software	B
19	GPTimer 0B	B	GPTimer 1B	B	UART4 TX	SB	Software	B	Software	B
20	GPTimer 1A	B	Software	B	UART7 RX	SB	Software	B	Software	B
21	GPTimer 1B	B	Software	B	UART7 TX	SB	Software	B	Software	B
22	UART1 RX	SB	Software	B	Software	B	Software	B	Software	B
23	UART1 TX	SB	Software	B	Software	B	Software	B	Software	B
24	SSI1 RX	SB	ADC1 SS0	B	Software	B	GPWideTimer 3A	B	Software	B
25	SSI1 TX	SB	ADC1 SS1	B	Software	B	GPWideTimer 3B	B	Software	B
26	Software	B	ADC1 SS2	B	Software	B	GPWideTimer 4A	B	Software	B
27	Software	B	ADC1 SS3	B	Software	B	GPWideTimer 4B	B	Software	B
28	Software	B	Software	B	Software	B	GPWideTimer 5A	B	Software	B
29	Software	B	Software	B	Software	B	GPWideTimer 5B	B	Software	B
30	Software	B	Software	B	Software	B	Software	B	Software	B
31	Reserved	B	Reserved	B	Reserved	B	Reserved	B	Reserved	B



DMA Channel Control Structure

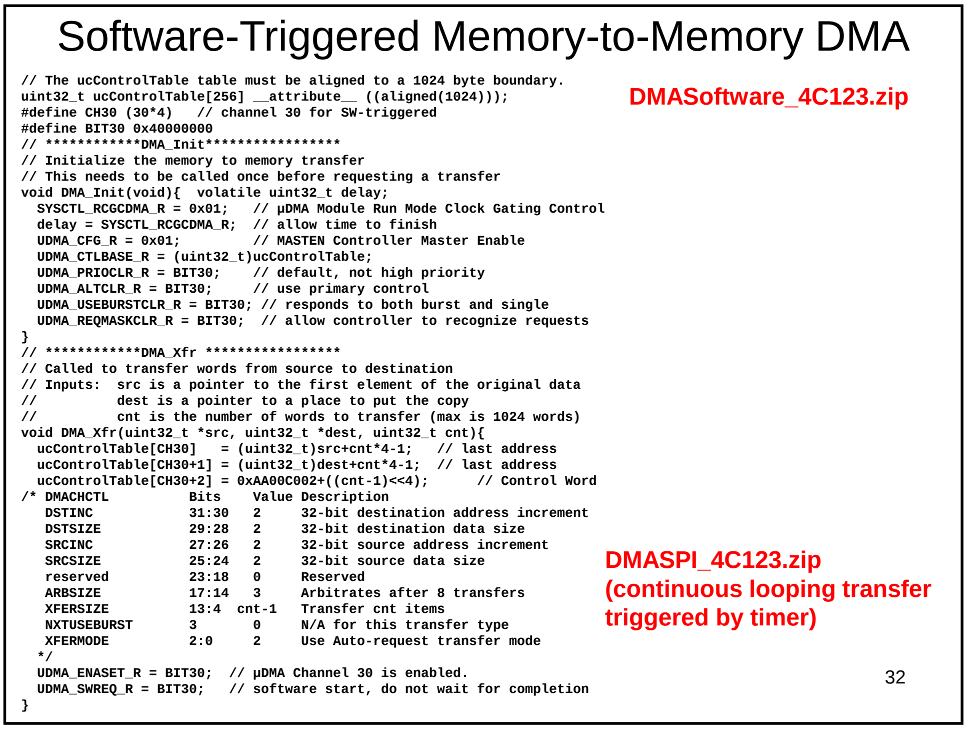
- Two per channel in main memory
  - Primary array (DMACTLBASE)
  - Alternate (DMAALTBASE) for continuous ping-pong

Address of the last byte of the source buffer								
Address of the last byte of the destination buffer								
DSTINC	DSTSIZE	SRCINC	SRCSIZE		ARBSIZE	XFERSIZE	NXTUSE	XFERMODE

Table 6.4. Structure of an entry in the channel control structure.

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

Lecture 8 J. Valvano, A. Gerstlauer
EE445M/EE380L.6 31



Software-Triggered Memory-to-Memory DMA

```
// 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
   reserved          23:18   0     Reserved
   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_SWREQ_R = BIT30;   // software start, do not wait for completion
}
```

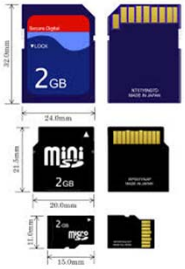
DMASoftware_4C123.zip

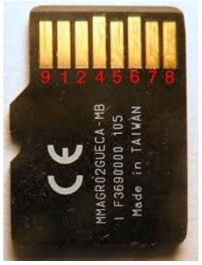
DMASPI_4C123.zip
(continuous looping transfer
triggered by timer)

32

Lab 5 File System

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





Reference EE445M book, Chapter 7

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

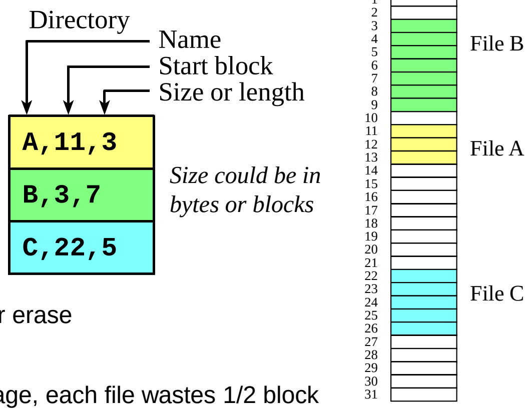
- Logical to physical translation
  - Byte number to block number
- Directory
  - File name to physical translation
- Free space
  - Used
  - Free
  - Damaged

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

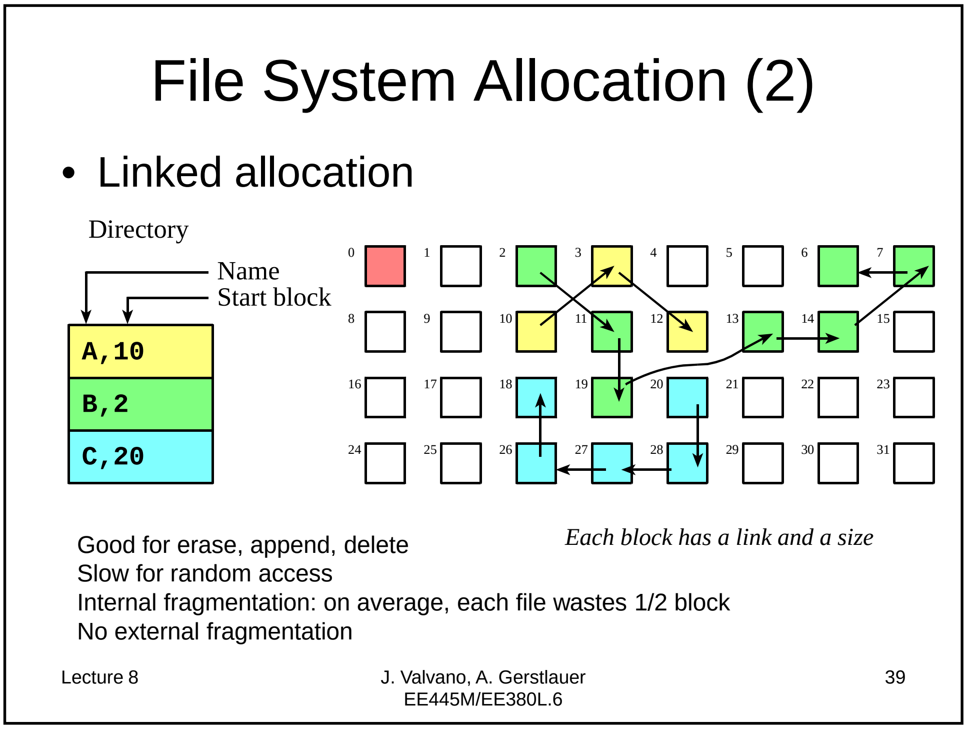


A,11,3
B,3,7
C,22,5

Size could be in bytes or blocks

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

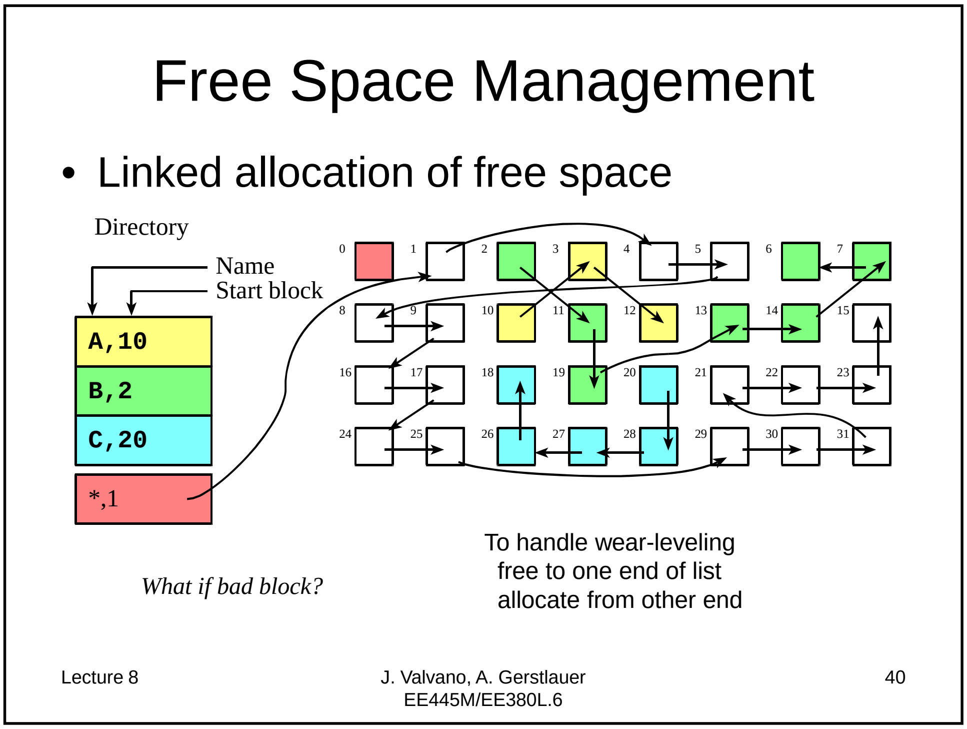


File System Allocation (2)

- 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

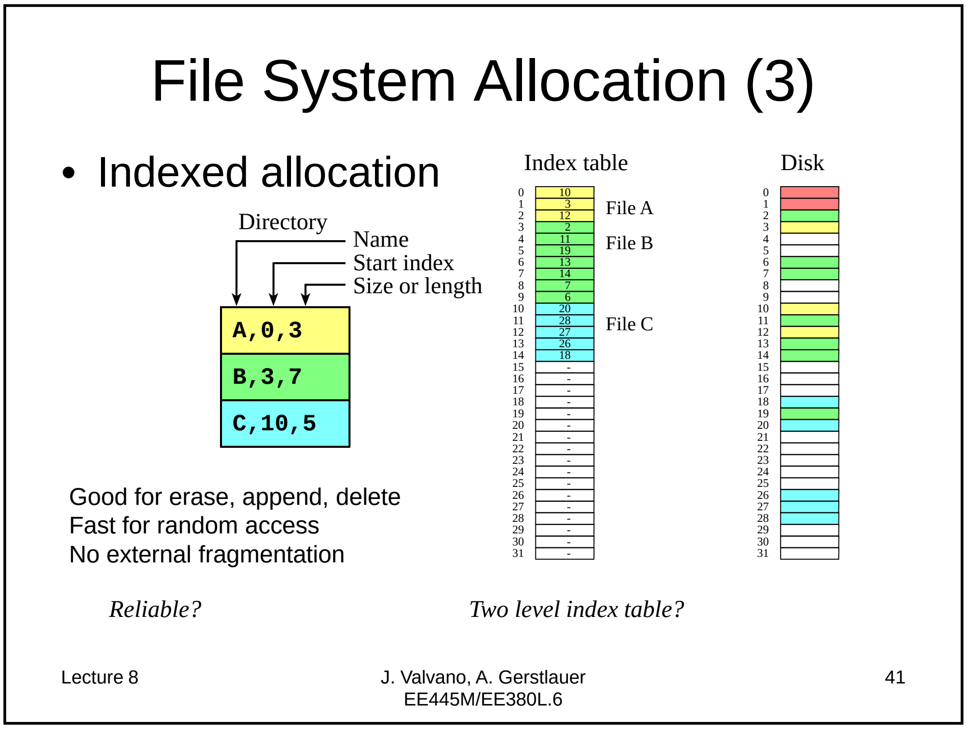


Free Space Management

- Linked allocation of free space

What if bad block?

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



File System Allocation (3)

- Indexed allocation

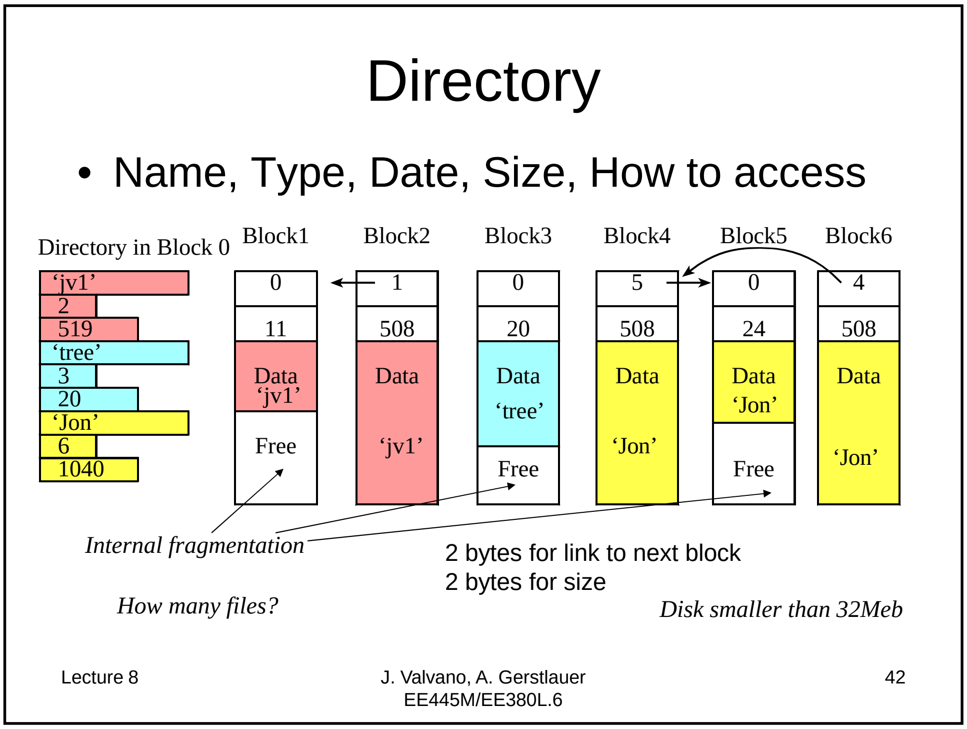
Directory
A,0,3
B,3,7
C,10,5

Index table	
0	10
1	3
2	12
3	2
4	11
5	19
6	13
7	14
8	7
9	6
10	20
11	28
12	27
13	26
14	18
15	-
16	-
17	-
18	-
19	-
20	-
21	-
22	-
23	-
24	-
25	-
26	-
27	-
28	-
29	-
30	-
31	-

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

Reliable? *Two level index table?*

Lecture 8 J. Valvano, A. Gerstlauer EE445M/EE380L.6 41



Directory

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

Directory in Block 0
‘jv1’
2
519
‘tree’
3
20
‘Jon’
6
1040

Block1	Block2	Block3	Block4	Block5	Block6
0	1	0	5	0	4
11	508	20	508	24	508
Data ‘jv1’	Data ‘jv1’	Data ‘tree’	Data ‘Jon’	Data ‘Jon’	Data ‘Jon’
Free		Free		Free	

Internal fragmentation

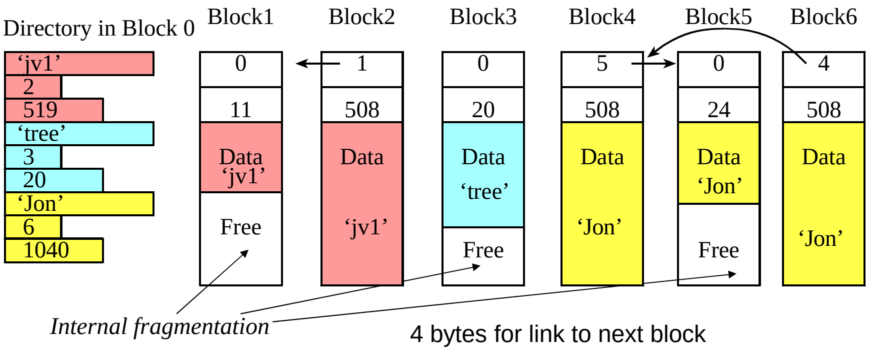
2 bytes for link to next block
2 bytes for size

How many files? *Disk smaller than 32Meb*

Lecture 8 J. Valvano, A. Gerstlauer EE445M/EE380L.6 42

Directory

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

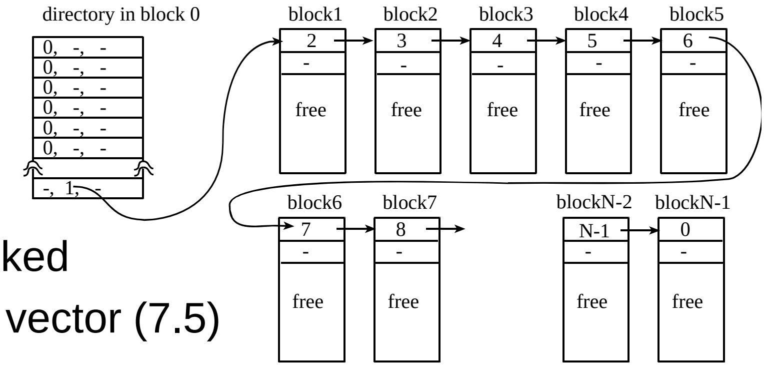


4 bytes for link to next block
2 bytes for size

How many files?

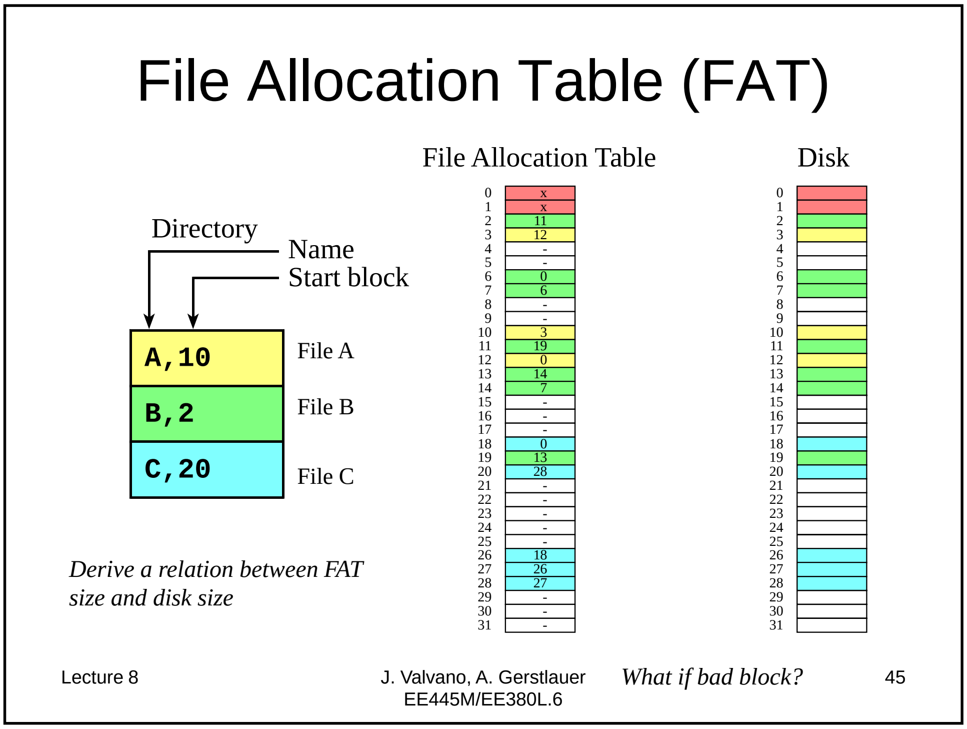
Disc larger than 32Meb

Free Space Management



- Linked
- Bit vector (7.5)

What percentage of the disk is wasted using a) linked; b) bit vector?



File Allocation Table (FAT)

Directory: Name, Start block

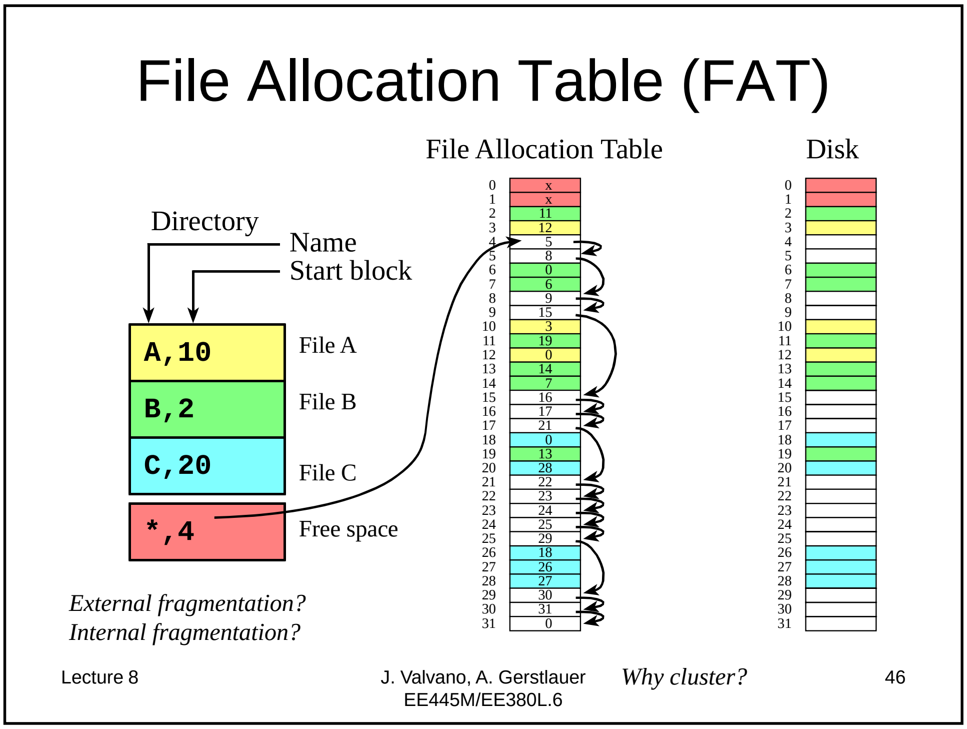
A,10	File A
B,2	File B
C,20	File C

	File Allocation Table	Disk
0	x	🟥
1	x	🟥
2	11	🟩
3	12	🟨
4	-	⬜
5	-	⬜
6	0	🟩
7	6	🟩
8	-	⬜
9	-	⬜
10	3	🟨
11	19	🟩
12	0	🟨
13	14	🟩
14	7	🟩
15	-	⬜
16	-	⬜
17	-	⬜
18	0	🟦
19	13	🟩
20	28	🟦
21	-	⬜
22	-	⬜
23	-	⬜
24	-	⬜
25	-	⬜
26	18	🟦
27	26	🟦
28	27	🟦
29	-	⬜
30	-	⬜
31	-	⬜

Derive a relation between FAT size and disk size

What if bad block?

Lecture 8 J. Valvano, A. Gerstlauer EE445M/EE380L.6 45



File Allocation Table (FAT)

Directory: Name, Start block

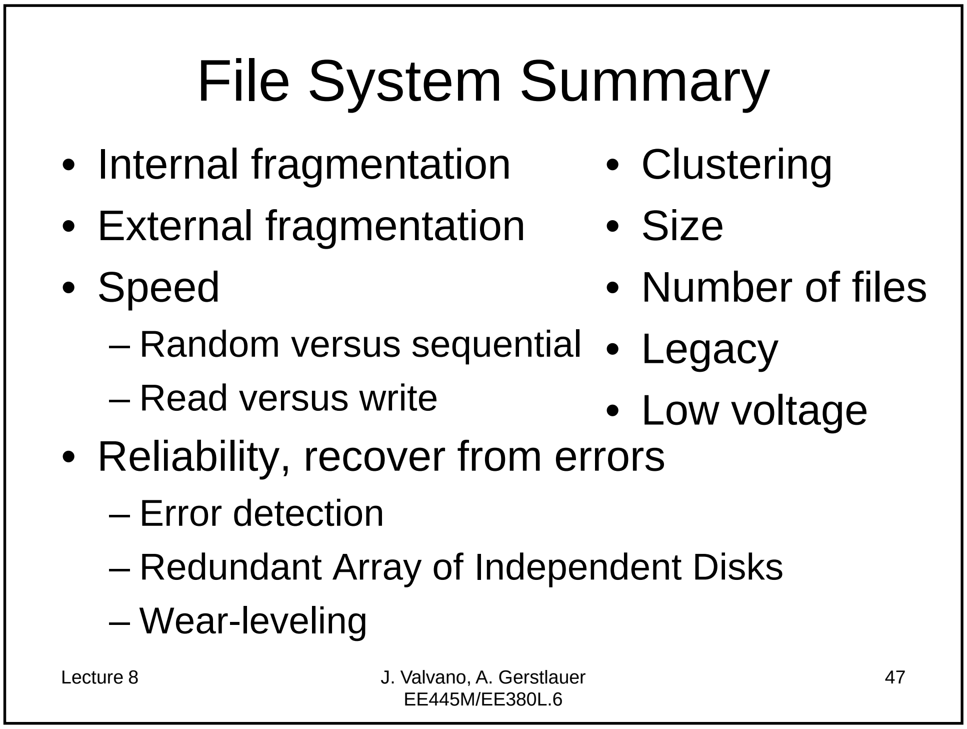
A,10	File A
B,2	File B
C,20	File C
*,4	Free space

	File Allocation Table	Disk
0	x	🟥
1	x	🟥
2	11	🟩
3	12	🟨
4	5	⬜
5	8	⬜
6	0	🟩
7	6	🟩
8	9	⬜
9	15	⬜
10	3	🟨
11	19	🟩
12	0	🟨
13	14	🟩
14	7	🟩
15	16	⬜
16	17	⬜
17	21	⬜
18	0	🟦
19	13	🟩
20	28	🟦
21	22	⬜
22	23	⬜
23	24	⬜
24	25	⬜
25	29	⬜
26	18	🟦
27	26	🟦
28	27	🟦
29	30	⬜
30	31	⬜
31	0	⬜

External fragmentation?
Internal fragmentation?

Why cluster?

Lecture 8 J. Valvano, A. Gerstlauer EE445M/EE380L.6 46



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 47