



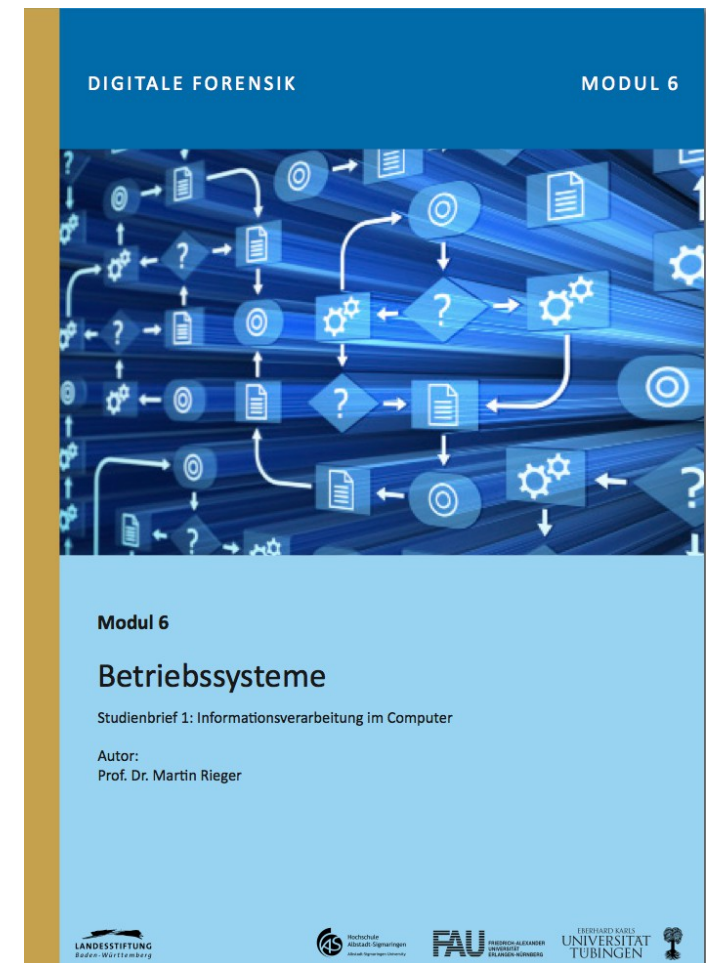
Hochschule
Albstadt-Sigmaringen
Albstadt-Sigmaringen University

Betriebssysteme

SS 2012

Hans-Georg Eßer
Dipl.-Math., Dipl.-Inform.

SB 3 (14.06.2012)
Eingabe/Ausgabe





- Kommunikation mit Hardware
- I/O-Ports mit Registern:
 - Status-Register
 - Control-Register
 - Daten-Register (lesen, schreiben)
- Zugriff über Prozessorbefehle `in`, `out`

- Beispiel: Disketten-Controller

```
enum FloppyRegisters
{
    STATUS_REGISTER_A           = 0x3F0, // read-only
    STATUS_REGISTER_B           = 0x3F1, // read-only
    DIGITAL_OUTPUT_REGISTER     = 0x3F2,
    TAPE_DRIVE_REGISTER         = 0x3F3,
    MAIN_STATUS_REGISTER        = 0x3F4, // read-only
    DATARATE_SELECT_REGISTER    = 0x3F4, // write-only
    DATA_FIFO                   = 0x3F5,
    DIGITAL_INPUT_REGISTER      = 0x3F7, // read-only
    CONFIGURATION_CONTROL_REGISTER = 0x3F7 // write-only
};
```

- Aufbau des Main Status Register:

7	6	5	4	3	2	1	0
MRQ	DIO	NDMA	BUSY	ACTD	ACTC	ACTB	ACTA

MRQ (Main Request)

1 = Data Register bereit
0 = nicht bereit

DIO (Data Input / Output)

1 = Controller → CPU
0 = CPU → Controller

NDMA (Non-DMA Mode)

1 = Controller nicht im DMA-Modus
0 = Controller im DMA-Modus

BUSY

1 = Instruktion wird gerade ausgeführt
0 = keine aktive Instruktion

ACTA-ACTD

(Laufwerk A, B, C, D Seek)

1 = aktiv
0 = nicht aktiv

Quelle: http://viralpatel.net/taj/tutorial/programming_fdc.php



- Allgemeine Vorgehensweise:
 - mit IN Statusregister auslesen und interpretieren
 - falls bereit: mit OUT zunächst Datenregister belegen und dann Befehl an Gerät schicken
 - entweder auf Interrupt vom Controller warten oder mit Polling Statusregister wiederholt auslesen (IN)
 - mit IN Datenregister auslesen



Polling vs. Interrupts

- Polling i. A. keine gute Idee, weil CPU-Zeit nicht sinnvoll genutzt wird
- bei BS-Start: Polling sinnvoll, wenn Multitasking noch nicht aktiviert ist

- Klassische Übertragung von Daten zwischen Speicher und Gerät: byte-/wort-weise, über einzelne IN-/OUT-Befehle
- bei großen Datenmengen langsam, umständlich
- Alternative: **DMA** (Direct Memory Access)
- Kommando an Controller enthält RAM-Adresse
 - Controller liest selbständig Daten aus dem RAM
 - oder schreibt selbständig in das RAM
 - Transfer benötigt keine CPU-Zeit, Interrupt nach Fertigstellung

- DMA arbeitet immer mit physikalischen Adressen (nicht mit virtuellen Adressen)
- denn: nur CPU/MMU „verstehen“ virtuelle Adressen
- zwei DMA-Arten im PC:
 - ISA-DMA (im Wesentlichen: Floppy)
 - PCI-Busmastering-DMA (z. B. Festplatten)

- Keyboard-Controller erzeugt bei jedem Tastendruck Interrupt 1
- In BS: Interrupt-Handler für IRQ 1 installieren
- Register IDT mit Adresse der Interrupt-Handler-Tabelle laden
- Handler liest I/O-Port 0x60
`byte = inb (0x60);`
- Rückgabewert ist ein Scancode, nach Konvertieren in ASCII-Zeichen Eintragen in Keyboard-Buffer



Keyboard-Interrupt-Handler

```
char system_kbd[BUFLLEN];    /* globaler Puffer */
int system_kbd_pos = 0;     /* aktuelle Position */

void keyboard_handler (struct regs *r) {
    /* Scancode auslesen ... */
    scancode = inb (0x60);
    c = convert_to_char (scancode);
    /* ... und in Puffer schreiben */
    system_kbd[system_kbd_pos] = c;
    system_kbd_pos = (system_kbd_pos + 1) % BUFLLEN;
    system_kbd_count++;
};

void keyboard_install () {
    /* Handler in Int.-Handler-Tabelle eintragen */
    irq_install_handler(1, keyboard_handler);
}

void irq_install_handler (int irq,
                          void (*handler)(struct regs *r)) {
    irq_routines[irq] = handler;
}
```

Keyboard ohne Interrupts

- Keine Interrupts? → Tastatur pollen
- Port 0x64: Keyboard-Status-Register (ro)

kbRead:

```
WaitLoop:    in     al, 64h      ; Read Status byte
              and    al, 10b     ; Test IBF flag (Status<1>)
              jz     WaitLoop    ; Wait for IBF = 1
              in     al, 60h     ; Read input buffer
```

Flag IBF = Input Buffer Full

Quelle: <http://www.computer-engineering.org/ps2keyboard/>