Computer Labs: The PC Keyboard

2º MIEIC

Pedro F. Souto (pfs@fe.up.pt)

October 12, 2014

Contents

Lab 3: The PC's Keyboard - Part 1

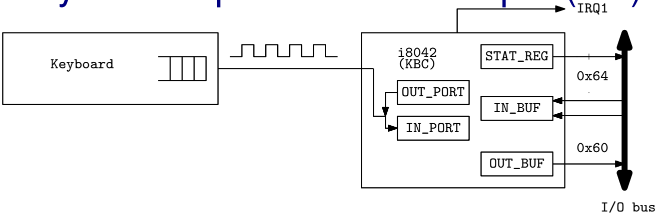
- Write functions:

```
int kbd_test_scan(unsigned short assembly)
int kbd_test_leds(unsigned short n, unsigned short *toggle)
```

  that require programming the PC's keyboard controller
- These functions are not the kind of functions that you can reuse later in your project
  - The idea is that you design the lower level functions (with the final project in mind).
- What's new?
  - Program the KBC controller (i8042)
  - In part 2:
    - Mix C with assembly programming
    - Handle interrupts from more than one device

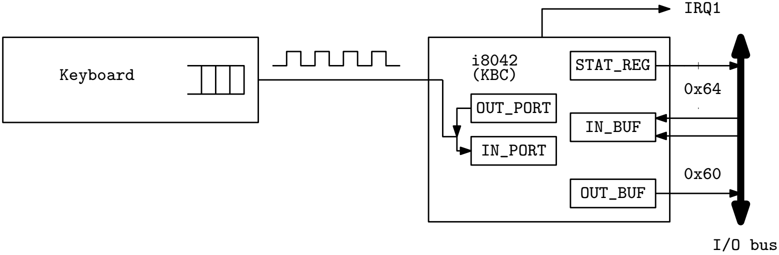
Contents

PC Keyboard Operation: Data Input (1/2)



- The keyboard has its own controller chip (not shown): the controller@KBD (C@KBD)
- When a key is pressed the C@KBD generates a **scancode (make code)** and puts it in a buffer for sending to the PC
  - **Usually, a scancode is one byte long**
- The same happens when a key is released
  - Usually, the scancode when a key is released (**break code**) is the make code of that key with the MSB set to 1
- The communication between the C@KBD and the PC is via a serial line
  - I.e. the bits in a byte are sent one after the other over a pair of wires

PC Keyboard Operation: Data Input (2/2)



- On the PC side this communication is managed by the keyboard controller (KBC)
  - In modern PCs, the KBC is integrated in the motherboard chipset
- When **OUT_BUF** is empty:
  1. The KBC signals that via the serial bus
  2. The C@KBD sends the byte at the head of its buffer to the KBC
  3. The KBC puts it in the OUT_BUF
  4. The KBC generates an interrupt by raising IRQ1

Lab 3: kbd_test_scan (1/2)

What Prints the scancodes, both the **makecode** and the **breakcode**, read from the KBC

- Should terminate when it reads the **breakcode** of the ESC key: 0x81
- The first byte of two byte scancodes is usualy 0xE0
  - This applies to both make and break codes

How Need to subscribe the KBC interrupts

- Upon an interrupt, read the scancode from the OUT_BUF

Note There is no need to configure the KBC

- It is already initialized by Minix

Issue Minix already has an IH installed

- Must be disabled to prevent it from reading the OUT_BUF before your handler does it

Solution Use not only the IRQ_REENABLE but also the IRQ_EXCLUSIVE policy in sys_irqsetpolicy()

Lab 3: `kbd_test_scan` (2/2)

KBC interrupt subscription in exclusive mode;

`driver_receive()` loop (similar to that of lab 2)

Interrupt handler reads the bytes from the KBC's `OUT_BUF`

- Should read only one byte per interrupt
  - The communication between the keyboard and the KBC is too slow
- Later, you may think about including the code that maps the scancodes to a character code
  - IH in Minix are usually out of the critical path
    - They are executed with interrupts enabled and after issuing the EOI command to the PIC
  - In many systems this may not be appropriate. For example, in Linux most DD break interrupt handling in two:

    Top half which is in the critical path, and therefore does minimal processing

    Bottom half which is not in the critical path, and therefore may do additional processing
- Should not print the scancodes

Minix 3 Notes: `driver_receive()` is not Polling

`driver_receive()` is a blocking call. If the process's "IPC queue" is empty:

- The OS will move it to the WAIT state
- The state will be changed to READY, only when a message (or notification) is sent to the process

```
while( 1 ) { /* You may want to use a different condition
    /* Get a request message. */
    if ( driver_receive(ANY, &msg, &ipc_status) != 0 ) {
        printf("driver_receive failed with: %d", r);
        continue;
    }
    if (is_ipc_notify(ipc_status)) { /* received notificat
        switch (_ENDPOINT_P(msg.m_source)) {
        case HARDWARE: /* hardware interrupt notification
            if (msg.NOTIFY_ARG & irq_set) { /* subscribed
                ... /* process it */
            }
            break;
        default:
            break; /* no other notifications expected: do
        }
```

Contents

Keyboard Commands (1/2)

- In the early PC models, interface with the keyboard used a very simple IC at port `0x60`
- For compatibility, the KBC provides two registers at that port:

  `IN_BUF` i.e. Input Buffer

  `OUT_BUF` i.e. Output Buffer

  and emulates the old interface:

  1. The KBC forwards bytes (commands) written in the `IN_BUF` to the C@KBD
  2. The C@KBD responds with one of 3 values: `0xFA` (ACK), `0xFE` (Resend) or `0xFC` (Error)
  3. The KBC puts the response in the `OUT_BUF` and raises IRQ1

Note The names of the registers `IN_BUF`/`OUT_BUF` are from the point of view of the KBC. The processor:

- Writes to the `IN_BUF`
- Read from the `OUT_BUF`

Keyboard Commands (2/2)

Command	Meaning	Args
0xFF	Reset KBD	
0xF6	Set default values and enable KBD	
0xF5	Disable KBD (set default values and stop scanning)	
0xF4	Clear buffer and enable KBD	
0xF3	Change KBD repetition rate/delay	bits 0-4 rate bits 5-6 delay
0xED	Switch on/off KBD LEDs	bits 0-2

Note The arguments of commands that require them have to be written to the IN_BUF too, and are also acknowledged

- The C@KBD responds with one of 3 values as described above.

Thus issuing such a command, requires 4 steps:

1. Write command to the IN_BUF
2. Read KBD response from the OUT_BUF
3. Write argument to the IN_BUF
4. Read KBD response from the OUT_BUF

If the KBD response is:

Resend (0xFE) the latest byte should be written again
Error (0xFC) the entire sequence should be restarted

Command `0xF3` (Configure Typematic Parameters)

- **Typematic** is an operating mode in which the keyboard generates a stream of scancodes when the user holds a key down
- The KBD uses two parameters for configuring this mode:

  Delay for entering typematic mode, counted from the moment the user presses down the key;

  Rate at which scancodes are generated, once the keyboard switches to typematic mode.

Command `0xED` (Set KBD LEDs)

Bit 2 Caps Lock indicator

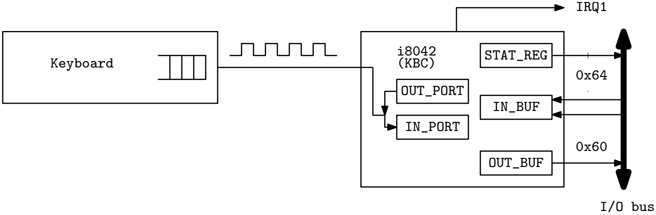
Bit 1 Numeric Lock indicator

Bit 0 Scroll lock indicator

- There is no way to read the value of these LEDs
  - The code that changes them should remember their state

Contents

The KBC Commands (of the PC-AT)



- The KBC added a few commands, the **KBC commands**, and two new registers at port `0x64`

  Status Register for reading the KBC state
  Not named for writing KBC commands

  - Apparently, this is not different from the `IN_BUF` at port `0x60`
  - The value of input line `A2` is used by the KBC to distinguish KBC commands from KBD commands
  - That is: the KBC has **only one** writable register, the `IN_BUF`

Status Register

- Input from/output to KBC requires reading the status register

Bit	Name	Meaning (if set)
7	Parity	Parity error - invalid data
6	Timeout	Timeout error - invalid data
5	Aux	Mouse data
4	INH	Inhibit flag: 0 if keyboard is inhibited
3	A2	A2 input line: 0 data byte 1 command byte
2	SYS	System flag: 0 if system in power-on reset, 1 if system already initialized
1	IBF	Input buffer full don't write commands or arguments
0	OBF	Output buffer full - data available for reading

- Bits 7 and 6 signal an error in the serial communication line between the keyboard and the KBC
- Do not write to the `IN_BUF`, if bit 1, i.e. the `IBF`, is set.

Keyboard-Related KBC Commands for PC-AT/PS2

- These commands must be written using address `0x64`
  - Arguments, if any, must be passed using address `0x60`
  - Return values, if any, are passed in the `OUT_BUF`

Command	Meaning	Args (A)/ Return (R)
`0x20`	Read Command Byte	Returns Command Byte
`0x60`	Write Command Byte	Takes A: Command Byte
`0xAA`	Check KBC (Self-test)	Returns 0x55, if OK Returns 0xFC, if error
`0xAB`	Check Keyboard Interface	Returns 0, if OK
`0xAD`	Disable KBD Interface	
`0xAE`	Enable KBD Interface	

KBD Interface is the serial interface between the keyboard and the KBC

- Disabling of the KBD interface is achieved by driving the clock line low.

- There are several others related to the mouse

(KBC "Command Byte")

7	6	5	4	3	2	1	0
–	–	DIS2	DIS	–	–	INT2	INT

`DIS2` 1: disable mouse

`DIS` 1: disable keyboard interface

`INT2` 1: enable interrupt on OBF, from mouse;

`INT` 1: enable interrupt on OBF, from keyboard

`-` : Either not used or not relevant for Lab

Read Use KBC command 0x20, which must be written to `0x64`

- But the value of the "command byte" must be read from `0x60`

Write Use KBC command 0x60, which must be written to `0x64`

- But the new value of the "command byte" must be written to `0x60`

Contents

Keyboard Programming/Configuration

Status Register @ address `0x64`

- Read the KBC state

Input Buffer @ either address `0x64` or address `0x60`. Can be used to **write**:

- **Commands to the KBC** access via address `0x64`;
- **Arguments of KBC commands** access via address `0x60`
- **Commands to the keyboard** access via address `0x60`
- **Arguments of keyboard commands** access via address `0x60`

Output Buffer @ address `0x60`. Can be used to **read**:

- **Scandcodes** both make and break, received from the keyboard;
- **Return values** from KBC commands;
- **Return values** from keyboard commands;
- **Confirmation protocol messages** `ACK`, `Resend` and `Error`

Note These addresses belong to the I/O address space

- Need to use `IN/OUT` assembly instructions or the library functions `sys_inb()`/`sys_outb()` of the kernel API

Issuing a Command to the KBC

```
#define STAT_REG    0x64
#define KBC_CMD_REG 0x64

    while( 1 ) {
        sys_inb(STAT_REG, &stat); /* assuming it returns OK */
        /* loop while 8042 input buffer is not empty */
        if( (stat & IBF) == 0 ) {
            sys_outb(KBC_CMD_REG, cmd); /* no args command */
            return 0;
        }
        delay(WAIT_KBC);
    }
```

Note 1 Cannot output to the `0x64` while the input buffer is full

Note 2 Code leaves the loop only when it succeeds to output the data to the `0x64`

- To make your code resilient to failures in the KBC/keyboard, it should give up after "enough time" for the KBC to send a previous command/data to the KBD.

Reading Return Value/Data from the KBC

```
#define OUT_BUF 0x60

    while( 1 ) {
        sys_inb(STAT_REG, &stat); /* assuming it returns OK */
        /* loop while 8042 output buffer is empty */
        if( stat & OBF ) {
            sys_inb(OUT_BUF, &data); /* assuming it returns OK

            if ( (stat &(PAR_ERR | TO_ERR)) == 0 )
                return data;
            else
                return -1;
        }
        delay(WAIT_KBC);
    }
```

Note 1 Code leaves the loop only upon some input from the `OUT_BUF`.

- It is not robust against failures in the KBC/keyboard

Note 2 Must mask IRQ1, otherwise the keyboard IH may run before we are able to read the `OUT_BUF`

KBC Programming Issues

Interrupts If the command has a response, and interrupts are enabled, the IH will “steal” them away from other code

- The simplest approach is just to disable interrupts.

Timing KBD/KBC responses are not immediate.

- Code needs to wait for long enough, but not indefinitely

Concurrent Execution The C@KBD continuously scans the KBD and may send scancodes, while your code is writing commands to the KBC:

- How can you prevent accepting a scancode as a response to a command?
  - It is easier to solve this for KBC commands than for KBD commands.
  - Assume that all scancode bytes generated by the KBD are different from the KBD responses

Contents

Lab 3: `kbd_test_leds()`

What? Toggle the keyboard LEDs – some portable computers do not have all, or even any of the LEDs

How? Use **keyboard command** `0xED` (set keyboard LEDs)

- Note that this command has one argument, which is the value with which the LEDs must be set.

Hint Try to design a solution based on layers that allows you to issue any keyboard or KBC command, not just command `0xED`

Bottom layer Functions that read/write the KBC registers. Deals with the details of the KBC HW interface. E.g.:

- Checks the `IBF` flag before writing
- Waits for the acks to the bytes of a KBD command

Top layer Functions to issue either KBC commands or KBD commands

- Knows about the commands and the protocol, writing parameters as necessary and waiting for responses

Further Reading

- IBM's Functional Specification of the 8042 Keyboard Controller (IBM PC Technical Reference Manual)
- W83C42 Data Sheet, Data sheet of an 8042-compatible KBC
- Andries Brouwer's The AT keyboard controller, Ch. 11 of Keyboard scancodes
- Andries Brouwer's Keyboard commands, Ch. 12 of Keyboard scancodes
- Randal Hyde's The PC Keyboard, Ch. 20 of the Art of Assembly Language