The PS/2 Keyboard and Mouse Interface

Prof. Stephen A. Edwards
sedwards@cs.columbia.edu

Columbia University
Spring 2006
The IBM PC Keyboard

[Images of IBM PC keyboards]
The IBM PC Keyboard

Original keyboard connector: DIN-5
The PS/2 Mini-DIN 6 Connector

The PS/2 Keyboard and Mouse Interface – p. 4/?
Like RS-232, but with a clock.
Odd parity, one start, one stop.
Keyboard-to-host shown: keyboard initiates everything.
### Codes (Keyboard to Host)

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Error or buffer overflow</td>
</tr>
<tr>
<td>FF</td>
<td>Error or buffer overflow</td>
</tr>
<tr>
<td>F0</td>
<td>Key-up</td>
</tr>
<tr>
<td>FA</td>
<td>Acknowledge</td>
</tr>
<tr>
<td>EE</td>
<td>Echo response</td>
</tr>
<tr>
<td>FE</td>
<td>Resend</td>
</tr>
<tr>
<td>E0</td>
<td>Extended code coming</td>
</tr>
</tbody>
</table>

- **ESC** 7F
- **F1** 05
- **F2** 06
- **F3** 04
- **F4** 0C
- **F5** 03
- **F6** 63
- **F7** 0A
- **F8** 01
- **F9** 09
- **FA** 76
- **FB** 07
- **F0** Key-up
- **FA** Acknowledge
- **EE** Echo response
- **FE** Resend
- **E0** Extended code coming

---

The PS/2 Keyboard and Mouse Interface – p. 6/7
Host brings Clock low, then Data low to indicate transfer to keyboard, then releases Clock (rises).

Keyboard starts generating clock signals. Host supplies serial data, changing after each falling edge. After stop bit, host releases Data. Keyboard pulls Data low for one more clock signal to indicate it received the byte.
ED  LED control
        Caps lock  Num lock  Scroll lock

EE  Echo
    Keyboard will respond with EE

F0  Set scan code set
    Keyboard will respond with FA and wait for another byte 01–03. 00 leaves scan code unchanged.

F3  Set key repeat rate
    Keyboard responds with FA and waits for second byte, indicating repeat rate.
Commands (Host to Keyboard)

F4  Enable keyboard
    Responds with FA, clears buffer, enables scanning.

F5  Disable keyboard
    Responds with FA, disables keyboard.

FE  Resend
    Retransmit the last byte.

FF  Reset Keyboard
Three bytes sent every time mouse moves or button clicked:

<table>
<thead>
<tr>
<th>MSB</th>
<th>LSB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>X</td>
</tr>
<tr>
<td>Overflow</td>
<td>Sign</td>
</tr>
<tr>
<td>X movement</td>
<td></td>
</tr>
<tr>
<td>Y movement</td>
<td></td>
</tr>
</tbody>
</table>

Movement values are since last transmission: 9-bit two’s-complement (signed) numbers.

Many more variants, modes, and other junk.
Add lines in the .UCF file about PS2C and PS2D:

```ucf
NET "PS2D" LOC=m15;  # Data
NET "PS2C" LOC=m16;  # Clock
```

Add these ports in the “add cores” dialog. Make the clock pin a falling-edge-sensitive interrupt.

In the .MHS file, this appears as

```mhs
PORT PS2D = ps2io_GPIO_in, DIR = IN
PORT PS2C = PS2C, DIR = IN,
          SIGIS = INTERRUPT,
          SENSITIVITY = EDGE_FALLING
```
Add an instance of an “opb_gpio.” Connect it to the OPB bus. Configure it to be a single-bit input and connect GPIO_in to the data line.

In the .MHS file:

```
BEGIN opb_gpio
    PARAMETER INSTANCE = ps2io
    PARAMETER HW_VER = 3.01.a
    PARAMETER C_BASEADDR = 0x80200e00
    PARAMETER C_HIGHADDR = 0x80200fff
    PARAMETER C_ALL_INPUTS = 1
    PARAMETER C_GPIO_WIDTH = 1
    PARAMETER C_IS_BIDIR = 0
    BUS_INTERFACE SOPB = mb_opb
    PORT OPB_Clk = sys_clk_s
    PORT GPIO_in = ps2io_GPIO_in
END
```
Add the PS/2 Clock signal to the list of interrupts handled by the opb_intc (add cores dialog). In the .MHS:

BEGIN opb_intc
    PARAMETER INSTANCE = opb_intc_0
    PARAMETER HW_VER = 1.00.c
    PARAMETER C_BASEADDR = 0x80200100
    PARAMETER C_HIGHADDR = 0x802001ff
    BUS_INTERFACE SOPB = mb_opb
    PORT Irq = Interrupt
    PORT Intr = PS2C & RS232_INTERRUPT
END
Using a PS/2 Port

This makes a keyboard-to-host port. Interrupt signals a clock. Do the shift register in software.

Register the handler and enable interrupts:

```c
microblaze_enable_interrupts();

XIntc_RegisterHandler(
    XPAR_OPB_INTC_0_BASEADDR,
    XPAR_OPB_INTC_0_SYSTEM_PS2C_INTR,
    (XInterruptHandler)ps2_int_handler,
    (void *)0);

XIntc_mMasterEnable(XPAR_OPB_INTC_0_BASEADDR);

XIntc_mEnableIntr(
    XPAR_OPB_INTC_0_BASEADDR,
    XPAR_RS232_INTERRUPT_MASK | XPAR_SYSTEM_PS2C_MASK);
```

The PS/2 Keyboard and Mouse Interface – p. 14/7
#define SIZE 16
unsigned char buffer[SIZE];
int head = 0; int tail = 0;

unsigned int code = 0; unsigned int bit = 11;

void ps2_int_handler(void *baseaddr_p) {
    int next;
    code = (code >> 1) |
           (XGpio_mReadReg(XPAR_PS2IO_BASEADDR,
                           XPAR_PS2IO_BASEADDR, XPAR_PS2IO_BASEADDR, XPAR_PS2IO_BASEADDR, XPAR_PS2IO_BASEADDR, XPAR_PS2IO_BASEADDR)) << 9);

    if (--bit == 0) {
        next = (head + 1) & (SIZE - 1);
        if (next != tail) {
            buffer[head] = code;
            head = next;
        }
        bit = 11; code = 0;
    }
    XGpio_mWriteReg( /* Acknowledge interrupt */ XPAR_PS2IO_BASEADDR, XPAR_PS2IO_BASEADDR, XPAR_PS2IO_BASEADDR, XPAR_PS2IO_BASEADDR, XPAR_PS2IO_BASEADDR, XPAR_PS2IO_BASEADDR); }

The PS/2 Keyboard and Mouse Interface – p. 15/7
int character_available()
{
    int result;
    microblaze_disable_interrupts();
    result = (head != tail);
    microblaze_enable_interrupts();
    return result;
}

unsigned char get_character()
{
    unsigned char result;
    microblaze_disable_interrupts();
    result = buffer[tail];
    tail = (tail + 1) & (SIZE - 1);
    microblaze_enable_interrupts();
    return result;
}