

The PS/2 Keyboard and Mouse Interface

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

Columbia University
Spring 2006

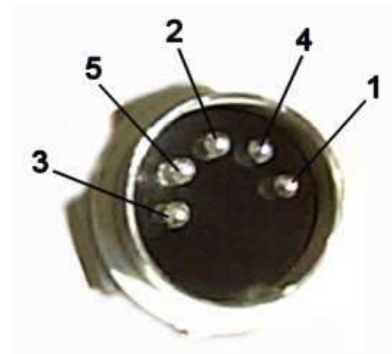
The IBM PC Keyboard



The IBM PC Keyboard



Original keyboard connector: DIN-5



The PS/2 Mini-DIN 6 Connector



VCC = 4

6

5 = Clk

3 = GND

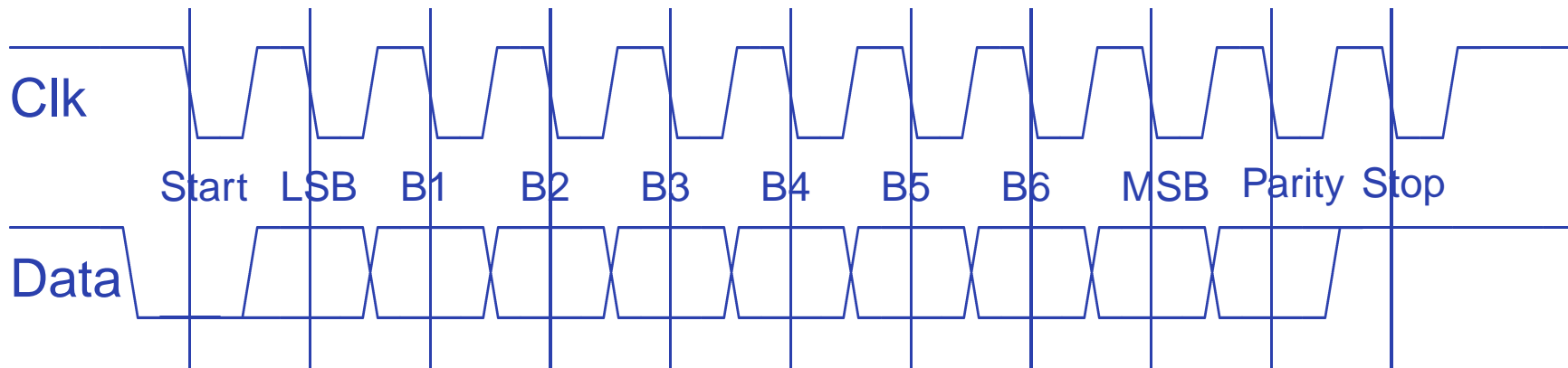


2

1 = Data

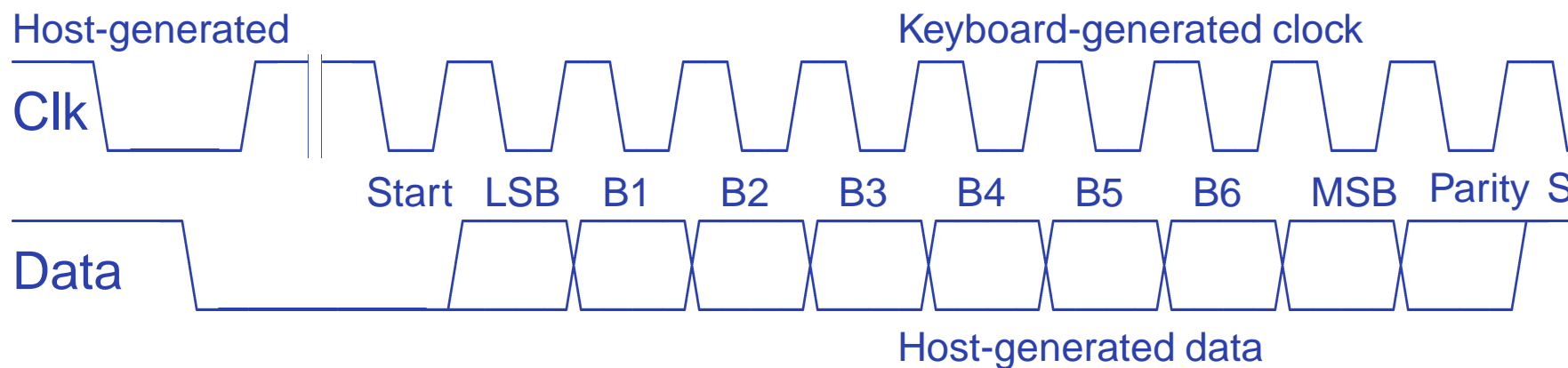
Female Socket

Synchronous Serial Interface



Like RS-232, but with a clock.
Odd parity, one start, one stop.
Keyboard-to-host shown: keyboard initiates everything.

Communicating to the Keyboard



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

Commands (Host to Keyboard)

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

PS/2 Mouse Protocol

Three bytes sent every time mouse moves or button clicked:

MSB				LSB			
Y	X	Y	X	1	Middle	Right	Left
Overflow		Sign			Buttons		
X movement							
Y movement							

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

Many more variants, modes, and other junk.

Using a PS/2 Port

Add lines in the .UCF file about PS2C and PS2D:

```
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

```
PORT PS2D = ps2io_GPIO_in, DIR = IN
PORT PS2C = PS2C, DIR = IN,
      SIGIS = INTERRUPT,
      SENSITIVITY = EDGE_FALLING
```

Using a PS/2 Port

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
```

Using a PS/2 Port

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:

```
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);
```

Interrupt Handler

```
#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,
                       XGPIO_DATA_OFFSET) << 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, XGPIO_ISR_OFFSET, 1);
}
```

Buffer Management Routines

```
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;
}
```