A USB BARCODE READER/SCANNER IMPLEMENTATION WITH PS/2-TO-USB CONVERSION

Paul Yang & Eric Lu
APIC, Philips Semiconductors

OVERVIEW

A barcode reader/scanner is typically employed in a point-of-sale (POS) class application in that the scanned bar code data can be transferred to the host party through the USB. The USB HID DWG has developed a HID usage table specification particularly for POS class products and has also released revision 1.0 for barcode usage table.

Barcode product vendors can design the new barcode reader/scanner to fully comply with the barcode specification. Nevertheless, an intermediate product that can convert PS/2 interface to USB will still find relevance in the current market. The reasons are:

- To design a new barcode product that is fully compliant with USB HID specification and USB barcode specification, vendors must invest time in training their engineers to be familiar with the specifications.
- To date, Win 98 has not integrated compatible minidrivers to support the USB barcode reader/scanner. This may also be true of Win2000. Hence, vendors have to develop their own minidrivers for such barcode products.
- Some of the USB HID compliant devices such as keyboard and mouse have drivers that are already mature under Win 98. By providing a converter that can expertly transform a traditional barcode product into such USB HID compliant devices, no special minidrivers will be necessary for the barcode product. Vendors are required only to convert the formatted data from PS/2 protocol to USB protocol in the converter firmware. For barcode vendors, the additional cost is very low, yet they get all the benefits of a new technology, i.e., USB, for their mature products. For existing customers of barcode products, they can protect their investment because they do not need to buy a new barcode reader or device driver.

To install the converter, a microcontroller and a USB interface IC are required. PDIUSBD11 (D11) of Philips Semiconductors is a USB interface IC that interfaces with the microcontroller through I^2C. It has some additional advantages, most importantly, serving as a good link and possessing enough physical endpoints.

In this engineering manual, we use D11 and 80C51 in implementing the PS/2-to-USB conversion.

HARDWARE SCHEME

Figure 1 shows the schematic of the proposed converter. The device is bus-powered. As D11 works under 3.3V of power supply, hence, a 3.3V power supply regulator is required to convert the 5V into 3.3V to drive the D11. A 22Ω terminator for D+/D- is required as mentioned in the D11 datasheet. In addition, the pins connected to the microcontroller must be pulled up because they are all open-drain gates.

The I^2C interface between D11 and the microcontroller, and the PS/2 interface between the PS/2 connector and the microcontroller are:

PS/2-SDA: P0.2
PS/2-SCL: $\overline{\text{INT0}}$ (P3.2)
I^2C -SCL: P0.0
I^2C -SDA: P0.1

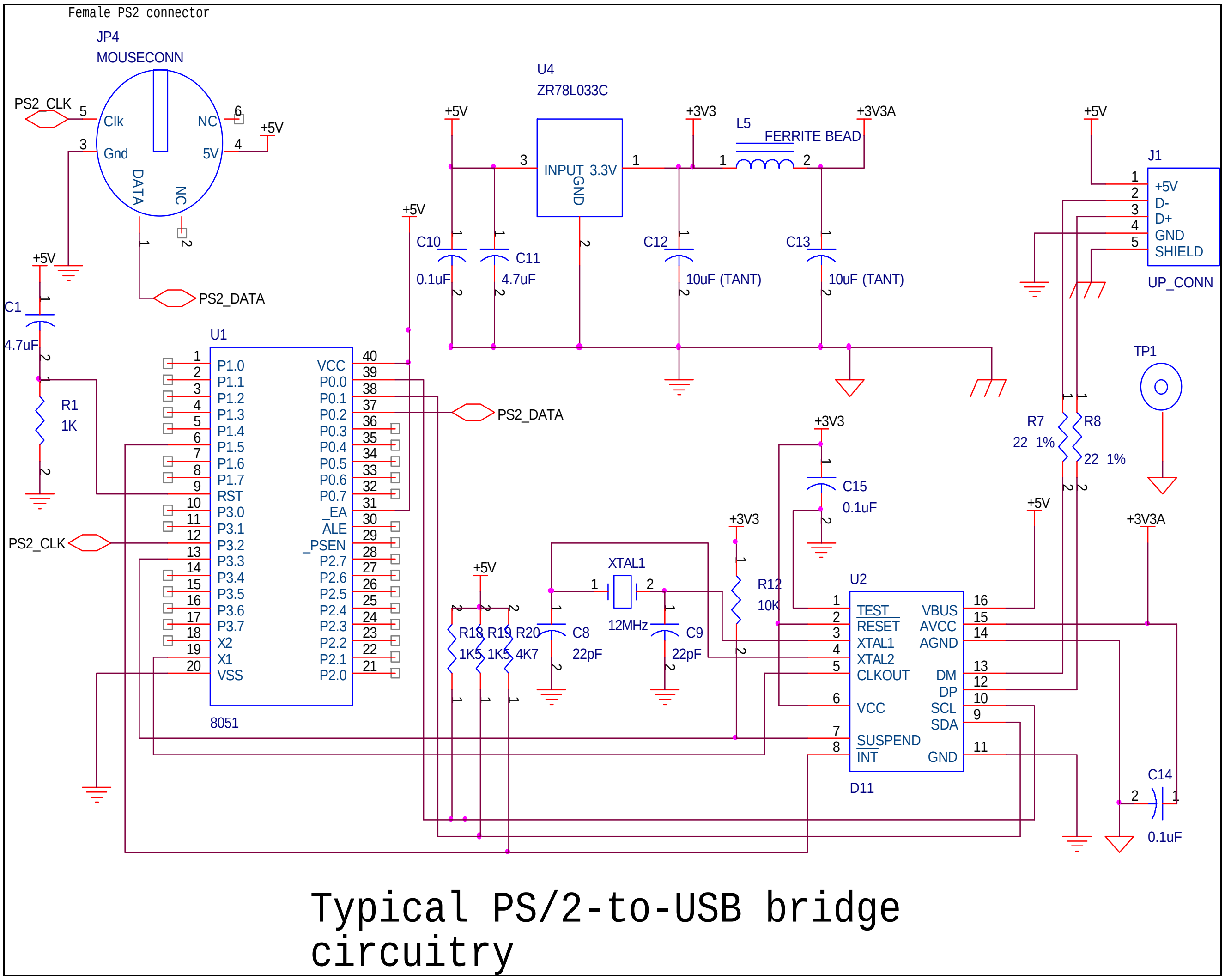

Figure 1. Schematic for PS/2-to-USB converter

All interface signals, except PS/2, employ IO pins of the microcontroller unit (MCU). The PS/2 clock signal employs SCL. The PS/2 interface is similar to the I^{2}C interface. They both employ two lines—clock and data—as the data transmission media. They both use “Line AND” logic, that is, the low logic of any device on a PS/2 bus will prohibit the active state of the other devices on the same bus. However, PS/2 protocol specifies that the moment the PS/2 clock becomes low, the data on the data line should already be stable. Hence, PS/2 data should be read from the data line at this moment.

The actual operation of a barcode reader is usually ON-Command, i.e.; a one-stroke button is built-up on the surface of a barcode reader. Once the button is pressed, the circuits inside the barcode reader begin to scan the bars and output the HEX code through its interface. In such a situation, the microcontroller of the converter is just passively reading data from the barcode reader: it will not know when data come and end. Hence, two methods—polling and interruption—may be used to sense the upcoming PS/2 clock or the data occurrence. Figure 2 shows a typical barcode data sequence.

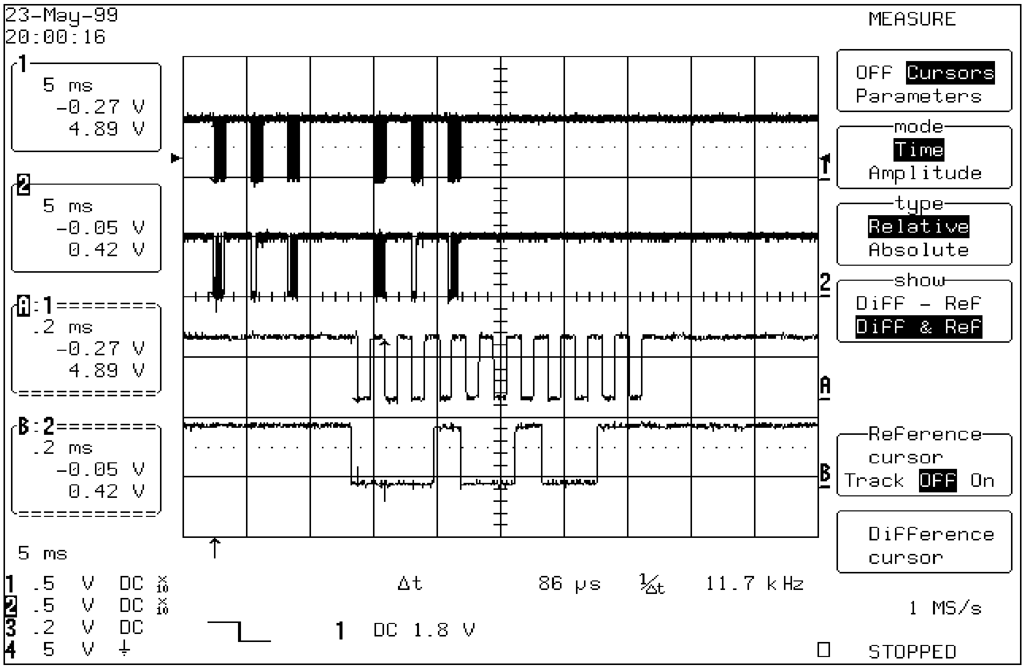

Figure 2. PS/2 data format

In Figure 2, Curve 1 is the PS/2 clock and Curve2 is the PS/2 data. Curve A is the zoom-in of the first clock in Curve 1 and Curve B is the zoom-in of the first PS/2 data in the data line. On each falling edge of the PS/2 clock, the level on the data line must be read as a bit value. In Figure 1 we see that the bit stream of a one-byte stream contains 11 bits. PS/2 protocol specifies that the first bit of any PS/2 data bit sequence must be low at the beginning, the last bit must be as high as the end of the one-byte stream, and the second last bit an odd parity bit for data byte. The rest of the 8 bits form one byte.

Because the PS/2 data rate is very low (11.7 Kbps as shown in Figure 1), polling will consume too much CPU time and hence deprive other USB devices of the USB bandwidth. Hence, only interruption is practical for sensing the upcoming PS/2 clock or data occurrence in this situation.

Nevertheless, both interruption and polling are used in PS/2 data acquisition. This will be explained in the firmware part.

DRIVER SUPPORTING ARCHITECTURE

Our PS/2-to-USB converter is intended to emulate any existing HID-compliant device that can be directly connected to a USB downstream port to work. This bypasses any need to develop any additional minidrivers for barcode readers. In today’s supporting architecture of Windows 98, the keyboard and mouse minidrivers above the HID driver stack are indeed mature. In this implementation, only the USB keyboard is emulated; the driver stacks are organized as Figure 3.

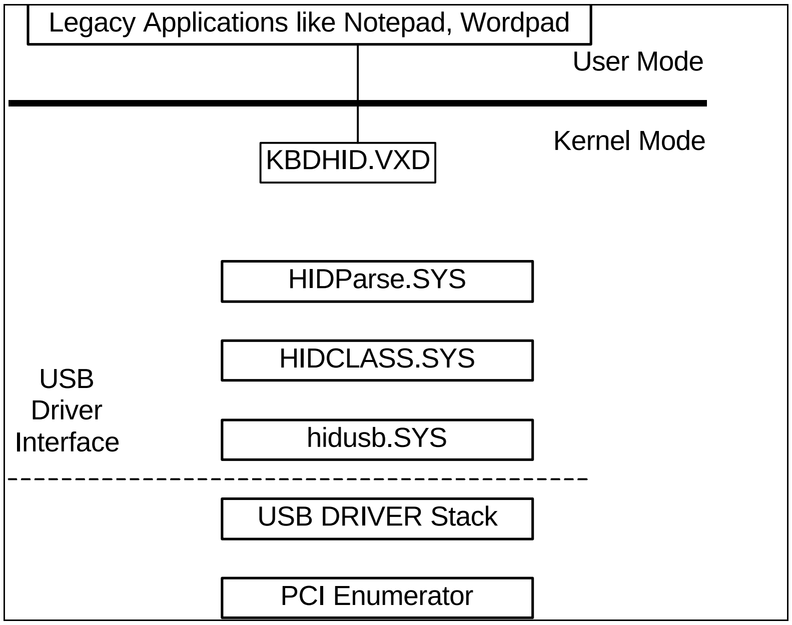

Figure 3. USB HID class keyboard drivers architecture

To see a keyboard entry in the Windows Device Manager, there must be a HID-compliant keyboard device connected to the computer. Under that device entry, the driver must contain KBDHID.VXD. In addition, there must be one USB Human Interface Device under Human Interface Device item; the drivers for standard USB HID device are HISUSB.SYS, HIDPARSE.SYS and HIDCLASS.SYS.

To emulate a HID keyboard, we must use the USB HID keyboard device report descriptor as the converter report descriptor. Appendix A lists the barcode reader report descriptor which is almost identical to the standard HID keyboard report descriptor but with the FEATURE tags removed as the barcode reader is an input-only device.

FIRMWARE ARCHITECTURE AND SCHEME

The firmware architecture for D11 is the same as the firmware for H11A/H12. Because the code for data acquisition is time-critical, we prefer assembly language to C language for coding the firmware.

The whole firmware for D11 can be separated into three parts:
- initialization and main loop
- Chap9 processing subroutines
- functional processing subroutine

The procedure is as follows:

1. Initialize the microcontroller and D11.
2. Set up the necessary RAMs for the I^2C buffer and the PS/2 data buffer.
3. Use the external interrupt $\overline{\text{INT0}}$ to sense the coming of PS/2 clock. Initialize it as follows:

```
      CLR    EA                  ;disable all interrupt
      SETB   EX0                 ;enable external interrupt INT0
      MOV    TCON,#00000001B     ;enable interrupt triggered with the falling edge
```

The interrupt $\overline{\text{INT0}}$ is always disabled until the USB device has been configured during enumeration. So the main loop looks like this:

```
MAIN:
      ACALL USB_SUBROUTINE
      JNB   EMB_CONFIG,MAIN     ;stop_ps2
      SETB  EA                  ;enable interrupt INT0 only after configuration
      AJMP  MAIN
```

Thus, data will not be acquired until the USB device is already enumerated.

The second part is handling the standard device and class-specific requests, which we will not cover.

The third part is responsible for PS/2 data acquisition, data format conversion, and sending the converted data to host. As in any USB HID device, the USB data is transferred through the interrupt pipe of the HID device. Of these subsections, the data acquisition is the most important.

Data acquisition

Because the data acquisition is handled in interrupt service subroutine, while the data transfer is handled by polling (depending on the interval the host requires to access the interrupt endpoint, and how long the microcontroller finishes the data acquisition and microcontroller machine cycle). So, it is one of the CPU's task to ensure an orderly transfer of data. In this application, the data generation is very slow (only when people press down the stroke button of the barcode reader), and we can manage the CPU not to acquire data before it finishes data transfer. That can be achieved by controlling the interrupt disable/enable bit EX0 of TCON.

Similarly, because of the low data transfer speed (normally 25ms) of interrupt pipe, the CPU doesn't necessarily always read each PS/2 bit by entering the interrupt service subroutine. On one hand, reading one bit needs some time (at worst case, it needs $100\mu s$) which may excel one bit-time (one PS/2 clock cycle), hence causing bit lost. On the other hand, entering interrupt service subroutine requires too many stack buffers. So, the trade-off between PS/2 clock sensing (requiring interruption) and data reading (needing some time) is that the interrupt is used to sense the coming of each data sequence, while inside the interrupt service subroutine, polling is used to read the bit stream of this data.

As a reminder, before exiting from interrupt service subroutine, the CPU should clear the interrupt flag bit of TCON. Otherwise, once it leaves the interrupt service subroutine, it would enter into another interrupt service subroutine again because each falling edge of PS/2 clock causes this flag bit set.

Figure 4 plots the flowchart for data acquisition. There are some very important implications in this flowchart.

Data conversion and transfer:

Both data conversion and transfer are handled in the interrupt pipe subroutine. The data transfer is very simple **-** just send data into FIFO of D11.

The data conversion is actually a lookup table because the PS/2 code is different from HID keyboard. What we have is the PS/2 code, but we have to find out the HID code for a PS/2 code in lookup table. Table 1 lists the PS/2 code of one key, its position in standard keyboard, and the corresponding HID code.

Another point to note is the PS/2 code format. It is usually configured as:

xxH, 0F0h, xxH, 5AH, 0F0H, 5AH for uppercase keys
12H, xxH, 0F0H, xxH, 0F0h, 12H, 5AH, 0F0H, 5AH for lowercase key.

The true key value is the number 0, number 3 and so on. The last key value is always the ENTER key.

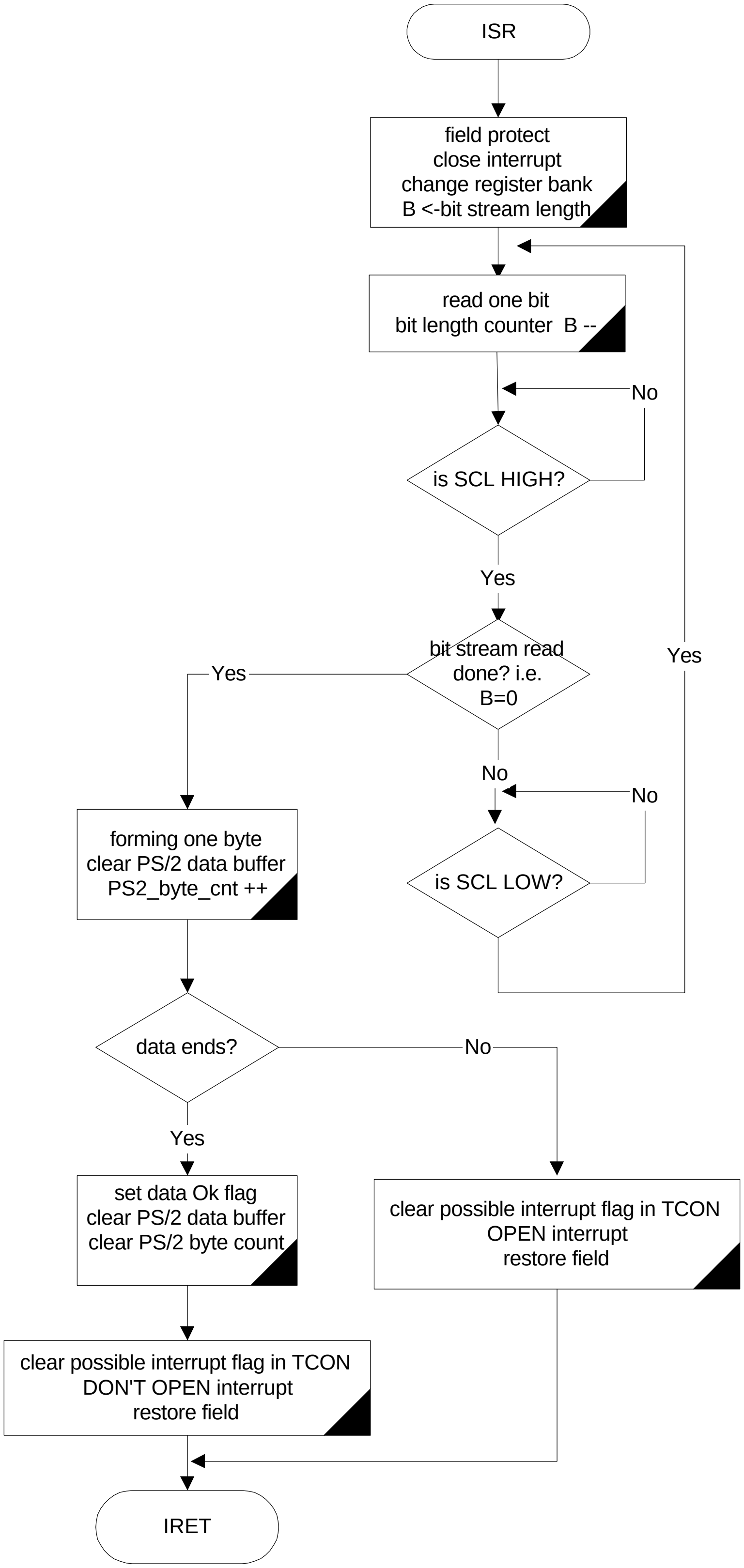

Figure 4. Flowchart for data acquisition

From the preceding data format, the PS/2 barcode assumes the default status of the keyboard which is Caps Lock (0x39). If it is the first time in transfering any key value, 39H must first be transferred to the host.

TABLE 1 Conversion Lookup Table from PS/2 code to HID keyboard code

Ps2 key code (HEX)	Traditional position in keyboard	HID keyboard code (HEX)
00	N/A	00
01	74	00
02	N/A	00
03	72	00
04	71	00
05	70	00
06	65	00
07	N/A	00
08	N/A	00
09	69	00
0A	68	00
0B	67	00
0C	66	00
0D	16	2B
0E	1	35
0F	N/A	00
10	N/A	00
11	58	E0
12	44	02
13	N/A	00
14	30	39
15	17	14
16	2	1E
17	N/A	00
18	N/A	00
19	N/A	00
1A	46	1D
1B	32	16
1C	31	04
1D	18	1A
1E	3	1F
1F	N/A	00
20	N/A	00
21	48	06
22	47	1B
23	33	07
24	19	08
25	5	21
26	4	20
27	N/A	00
28	N/A	00
29	61	2C
2A	49	19
2B	34	09
2C	21	17
2D	20	15
2E	6	22
2F	N/A	00
30	N/A	00
31	51	11
32	50	05
33	36	0B
34	35	0A
35	22	1C
36	7	23
37	N/A	00
38	N/A	00
39	N/A	00
3A	52	10
3B	37	0D
3C	23	18
3D	8	24
3E	9	25
3F	N/A	00
40	N/A	00
41	36	36
42	38	0E
43	24	0C
44	25	12
45	11	27
46	10	26
47	N/A	00
48	N/A	00
49	54	37
4A	55	38
4B	39	0F
4C	40	33
4D	26	13
4E	12	2D
4F	N/A	00
50	N/A	00
51	N/A	00
52	41	34
53	N/A	00
54	27	2F
55	13	2E
56	N/A	00
57	N/A	00
58	64	E4
59	57	E5
5A	43	28
5B	28	30
5C	N/A	00
5D	14	31
5E	N/A	00
5F	N/A	00
60	N/A	00
61	N/A	00
62	N/A	00
63	N/A	00
64	N/A	00
65	N/A	00
66	15	2A
67	N/A	00
68	N/A	00
69	93	59
6A	N/A	00
6B	92	5C
6C	91	5F
6D	N/A	00
6E	N/A	00
6F	N/A	00
70	99	62
71	104	63
72	98	5A
73	97	5D
74	102	5E
75	96	60
76	90	53
77	95	54
78	N/A	00
79	108	58
7A	103	5B
7B	107	85
7C	106	57
7D	101	61
7E	100	55
7F	N/A	00
80	N/A	00
81	N/A	00
82	N/A	00
83	73	00
84	105	56

CONCLUSION

The PS/2-to-USB converter for a barcode reader/scanner is a trade-off between SUB barcode fully compliant to USB barcode specification and legacy barcode. It results in reducing the total cost of upgrading legacy barcode device to USB device.

It is envisioned that in not too far a future, the USB barcode which is fully compliant to USB barcode specification will be developed because the minidriver will eventually be made available, and fully compliant solutions can generate savings through reducing hardware cost.

Appendix

A: report descriptor for barcode reader

```
;
;************************************************************************
;* HID device (Barcode reader with Keyboard protocol) report descriptor
;************************************************************************
RptDsc:
      db    5h,1h       ; USAGE_PAGE (Generic Desktop)
      db    9h,6h       ; USAGE (Keyboard)
      db    0a1h,1h     ; COLLECTION (Application)
      db    5h,7h       ;   USAGE_PAGE (Keyboard)
      db    19h,0e0h    ;   USAGE_MINIMUM (Keyboard LeftControl)
      db    29h,0e7h    ;   USAGE_MAXIMUM (Keyboard Right GUI)
      db    15h,0h      ;   LOGICAL_MINIMUM (0)
      db    25h,1h      ;   LOGICAL_MAXIMUM (1)
      db    75h,1h      ;   REPORT_SIZE (1)
      db    95h,8h      ;   REPORT_COUNT (8)
      db    81h,2h      ;   INPUT (Data,Var,Abs)
      db    95h,1h      ;   REPORT_COUNT (1)
      db    75h,8h      ;   REPORT_SIZE (8)
      db    81h,3h      ;   INPUT (Cnst,Var,Abs)
;     db    95h,5h      ;   REPORT_COUNT (5)
;     db    75h,1h      ;   REPORT_SIZE (1)
;     db    5h,8h       ;   USAGE_PAGE (LEDs)
;     db    19h,1h      ;   USAGE_MINIMUM (Num Lock)
;     db    29h,5h      ;   USAGE_MAXIMUM (Kana)
;     db    91h,2h      ;   OUTPUT (Data,Var,Abs)
;     db    95h,1h      ;   REPORT_COUNT (1)
;     db    75h,3h      ;   REPORT_SIZE (3)
;     db    91h,3h      ;   OUTPUT (Cnst,Var,Abs)
      db    95h,6h      ;   REPORT_COUNT (6)
      db    75h,8h      ;   REPORT_SIZE (8)
      db    15h,0h      ;   LOGICAL_MINIMUM (0)
      db    25h,65h     ;   LOGICAL_MAXIMUM (101)
;     db    5h,7h       ;   USAGE_PAGE (Keyboard)
      db    19h,0h      ;   USAGE_MINIMUM (Reserved (no event indicated))
      db    29h,65h     ;   USAGE_MAXIMUM (Keyboard Application)
      db    81h,0h      ;   INPUT (Data,Ary,Abs)
      db    0c0h        ; END_COLLECTION
RptDscend:
```