PS/2 BLE Keyboard

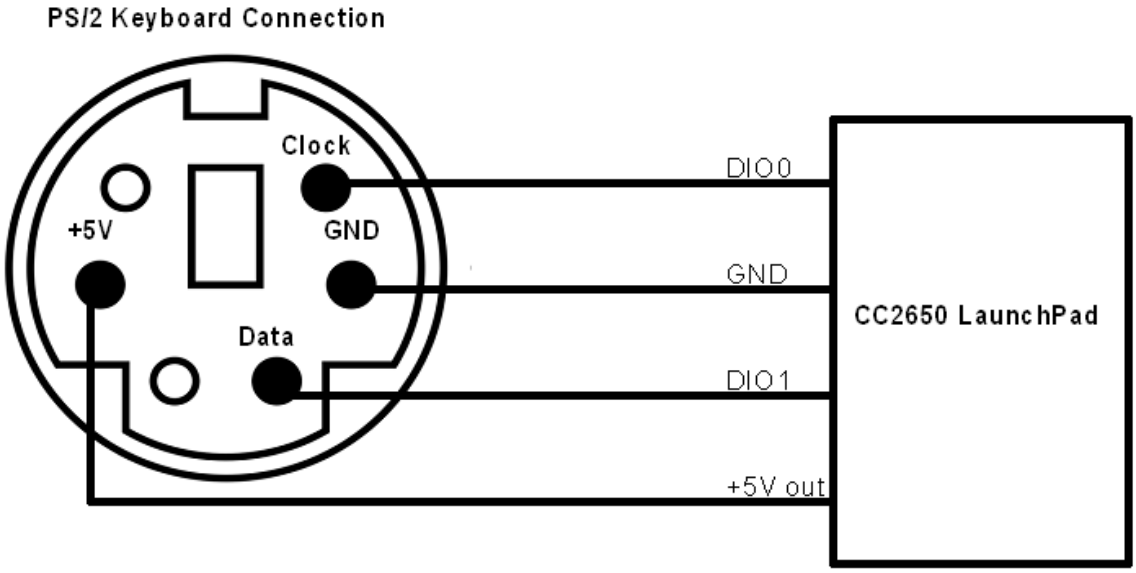
Advanced Computer Systems (Fall 2016), Tel Aviv University
Nerya Meshulam (neryam@mail.tau.ac.il)

Overview

The main goal of the project is to create a Bluetooth adapter for wired keyboards.

I wanted to learn more about real time signal processing and the BLE capabilities of the CC2650, as well as adding some capabilities to the OS, and I think this project is a great way to achieve it.

Basic Design



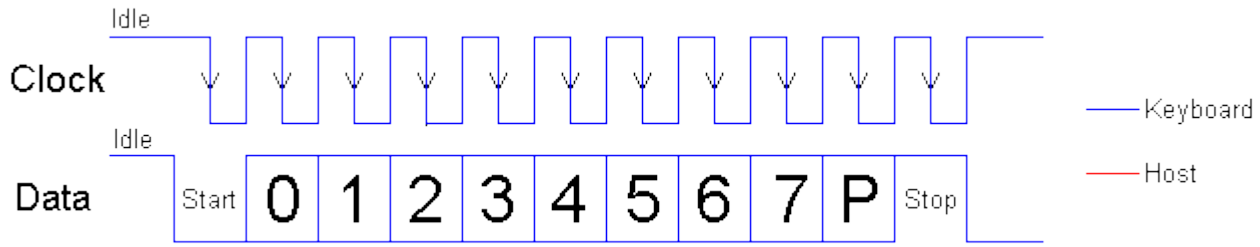
PS/2 Protocol

PS/2 is a synchronous serial communication, typically used for user input devices like keyboard and mouse.
The interface consists of 4 lines:

1. +5V
2. GND
3. Clock: used by the device to signal the timing of data being transmitted. The host relies on this signal both to send and receive data, and can inhibit the device by pulling this line low
4. Data: used for bi-directional data transmission. Depending on the direction of communication, reading the state of this line on either the rising or falling edge of the clock signal provides one bit of data being transmitted

Device to Host communication

Each keystroke is represented by one or more bytes transmitted from the device to the host. The device can send data to host when both Data and Clock lines are high. In order to initiate communication, the device will take the Data line low, start generating clock pulses on the Clock line and then send the data on Data line:

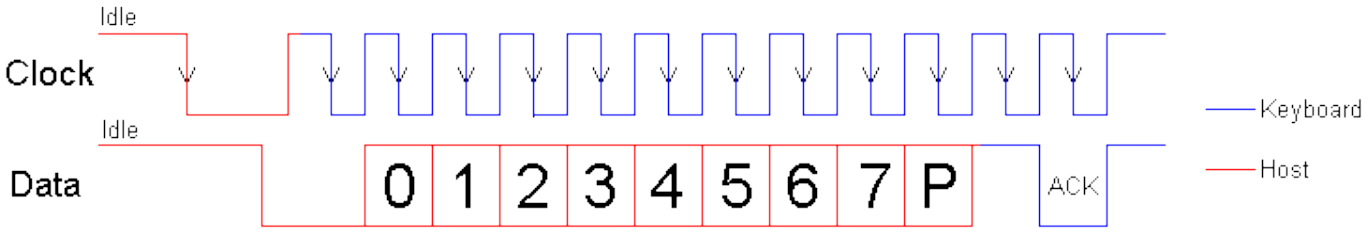


Device to Host timing diagram (Reference: [Beyond Logic: Interfacing the AT Keyboard])

Host to Device communication

Sometimes the host need to send data to the device. For example, when the host need to turn on/off the LEDs on the keyboard, or request the device to resend last transmitted data.

To initiate Host to Device communication, the host will take the Clock line Low for at least 100us. Then the Data line will be taken Low and the Clock line will be released by the host. Now, the device must start generating clock pulses, so the host can start transmitting the data:



Host to Device timing diagram (Reference: [Beyond Logic: Interfacing the AT Keyboard])

Scan codes

Each keystroke generates several scan codes transmitted to host. Any individual key has its own unique scan code(s), each scan code represented by a single byte.

The keyboard distinguishes two states for any key, the 'key pressed' state and 'key released' state. For each keystroke the keyboard will send the scan code for the pressed key, and after the key released it will send an '0xF0' (Break code) following by the scan code of the key.

Some keys are represented by two scan codes, the first scan code is '0xE0' (Extended code) and the second is a regular scan code. For example, 'L ALT' and 'R ALT' keys are represented by the same scan code ('0x11'), but 'R ALT' represented with '0xE0' prior to '0x11'.

Full scan code table can be found here: http://www.computer-engineering.org/ps2keyboard/scancodes2.html

HID Over GATT

HID over GATT profile is an adaptation of the USB HID specification to operate over a Bluetooth low energy wireless link.

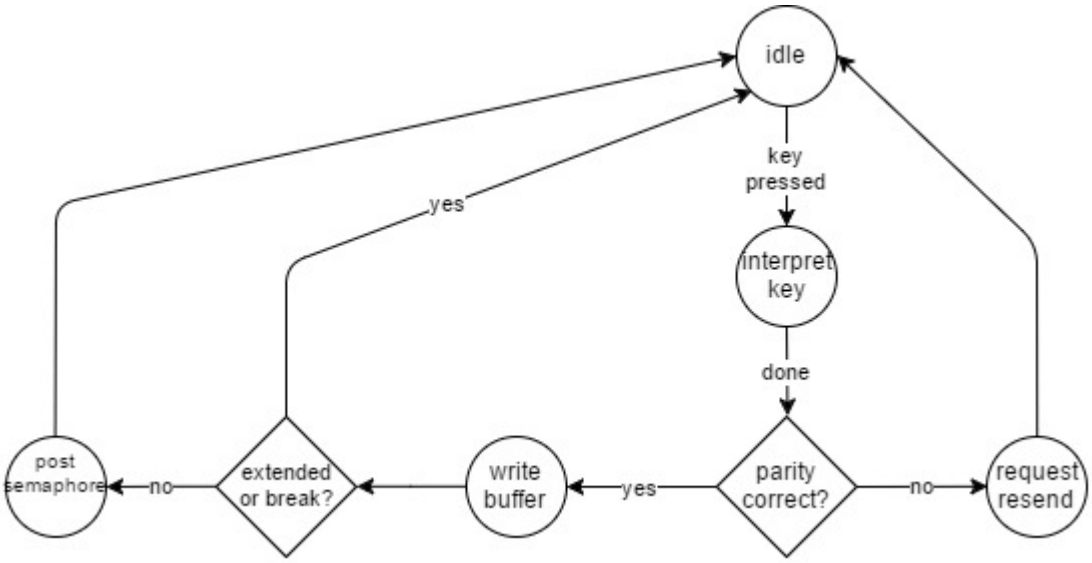
Currently pressed keys are reported by 8 bytes as follows: [modifier, 0x00, Key1, Key2, Key3, Key4, Key5, Key6].
The modifier byte is a bitmap, which means that each bit corresponds to one key (ALT, CTRL, SHIFT, GUI - left and right for each of those).
The keys' byte is a scan code similar to the PS/2 scan code mentioned above, but with different values and without Extended codes or Break codes (to represent 'key released' state we need to send another report without the released key).

Detailed information and scan codes table can be found in: http://www.usb.org/developers/hidpage/.

The BLE part of the project is based on 'hid_emu_kbd' example from BLE SDK 2.2.0.

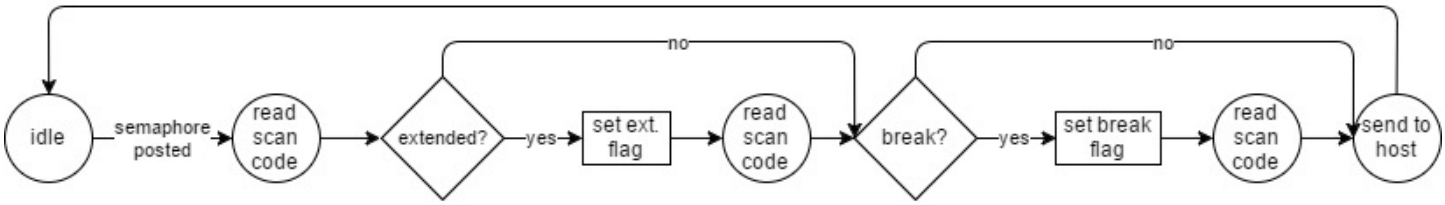
Program Flow

The program is interrupt based. Each time the clock changes from high to low it triggers the callback function. The function reads the Data line and saves the value. On the 11th run its saves the scan code on a buffer, and posting a semaphore so the idle task that pending on this semaphore could process it.



Simplified flow of the callback function

The idle task read a scan code from the buffer. If the scan code is '0xE0' (Extended code) or '0xF0' (Break code) it sets the relevant flag and read another scan code. Now the idle task translates the PS/2 scan codes to one USB scan code, and sends it as a message to the HID service with the relevant flags.

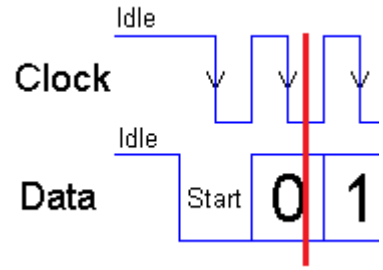


Simplified flow of the idle task

The HID service updates the report mentioned above according to the message, and sends it to the host.

Problems Encountered

- Timing issue: at the beginning of the project I've sampled the Data line immediately after the fall of the Clock line and as a result 20% of the scan codes received from the keyboard contained error (one or more inverted bits).
  After some searching I've changed the sample timing to be delayed for 0.25 of the clock time (~17us), because that's the time the Data signal is steady for sampling. This step reduced the errored scan codes to about 2%.



In red: new sampling time