Introduction to Keyboard Programming

Jacob Alexander, Massdrop.com

This is an advanced introductory article around keyboard programming and dives into complex topics that may be intimidating for a beginner. However if you're interested in learning about what's going on under your keys, then there are certain things you need to understand first.

> The first thing to embrace is that computers are fundamentally incapable of feeling a keypress .When you press a key down, there is a specific signal or "code" assigned to that button that is read by the micro-controller in your keyboard.

This micro-controller is constantly monitoring all of your keys, checking to see if you happen to press one down. This scanning occurs many times per second, which is how your keyboard is able to keep up with you even if you are a very fast typist. This is where the concept of N-Key Rollover comes from, which is essentially a feature that allows you to press "N" or any number of keys, and have all of them be sent in the proper order to your keyboard's controller.

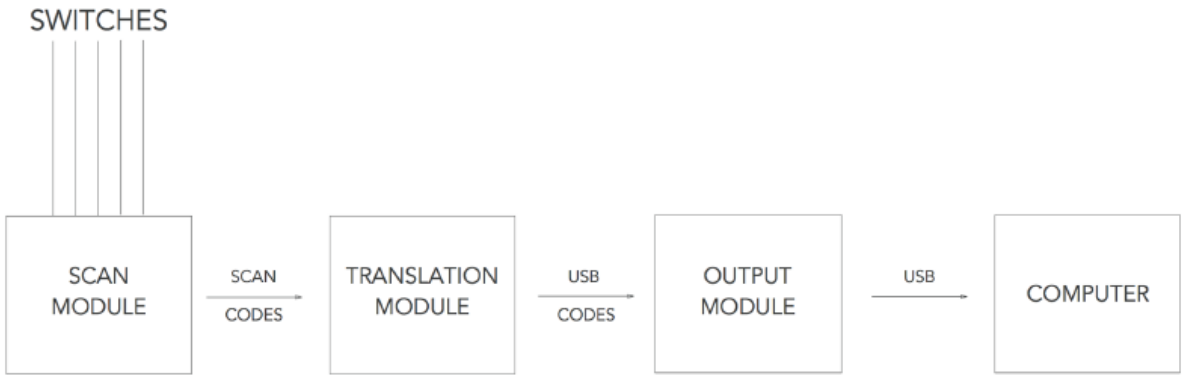
Although we've touched upon how the micro-controller in your keyboard understands when you've pressed a key, it is still quite different from your computer receiving what you've typed. You may have noticed that almost all keyboards these days are connected to computers via USB cable, and it's about time you learned what USB is. USB is a standardized connection that forces electronics to abide by certain universal methods for communicating. Keyboards are included in this list of electronics, and in order to use a USB cable, they must play by the proper rules. Fortunately for you, some very clever engineers chose the microcontroller in your keyboard specifically for its ability to receive the signals from your keypresses and convert them into something that USB will allow to pass through the cable and into your computer!

This entire process is possible because of specific firmware that was developed to sit on the keyboard's micro-controller and rapidly perform the process of translating your keypresses from physical contact into electrical signals and then outputting them into your computer.

The next few sections will take you on a deeper dive into how this process works, and will introduce you to the specific terms that are used to describe each part so that you can conduct further research. First, let's start with modules.

Modules
I generally like to think about Keyboard firmware in 3 different parts or modules.

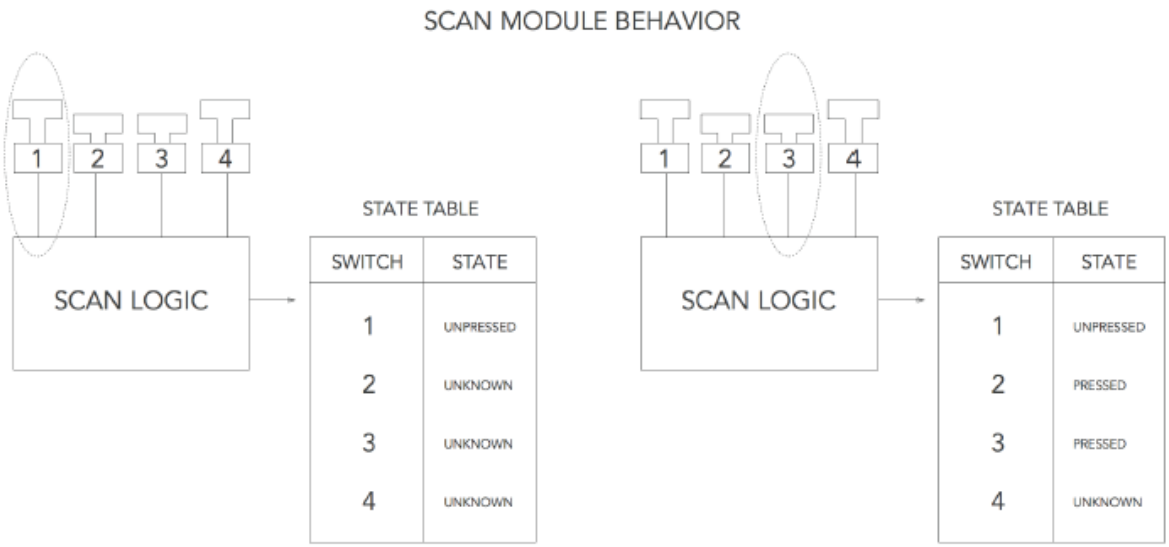
> Scan Module - Deals with collecting the state of all of the keyboard switches.
>
> Translation Module - Handles the layout mapping from Scan Codes (hardware location of each switch) to Output Codes (e.g. USB Codes), which the host computer understands.
>
> Output Module - Takes care of sending signals to the host computer (e.g. USB, Bluetooth, and PS/2).



A keyboard microcontroller consists of three modules. A Scan Module, Translation Module and Output Module.

Scan Module
The Scan module is responsible for collecting the state of every single keyboard switch. Each key is internally assigned a unique number called a "Scan code". To simplify programming and reduce costs, most microcontrollers only have a single processing core (i.e. single threaded). Unfortunately, this means that it can only do one thing at a time, or in our case, the microcontroller can only read a single switch at a time. However, what about the case where you want to press two keys at the same time?

To get around the deficiency in the microcontroller, some mildly clever software engineering is required. Rather than sending the state of each key after it was pressed to the computer, the microcontroller is programmed to cache the state of each key until all the keys are pressed. The state is easy to keep track of because each switch has a unique Scan code. This is also where scan-rate or poll-rate comes from. For a 1,000 Hz polling keyboard, the microcontroller must scan the state of every single key 1,000 times per second. After all of the states of every key have been scanned, it is then sent to the next module.



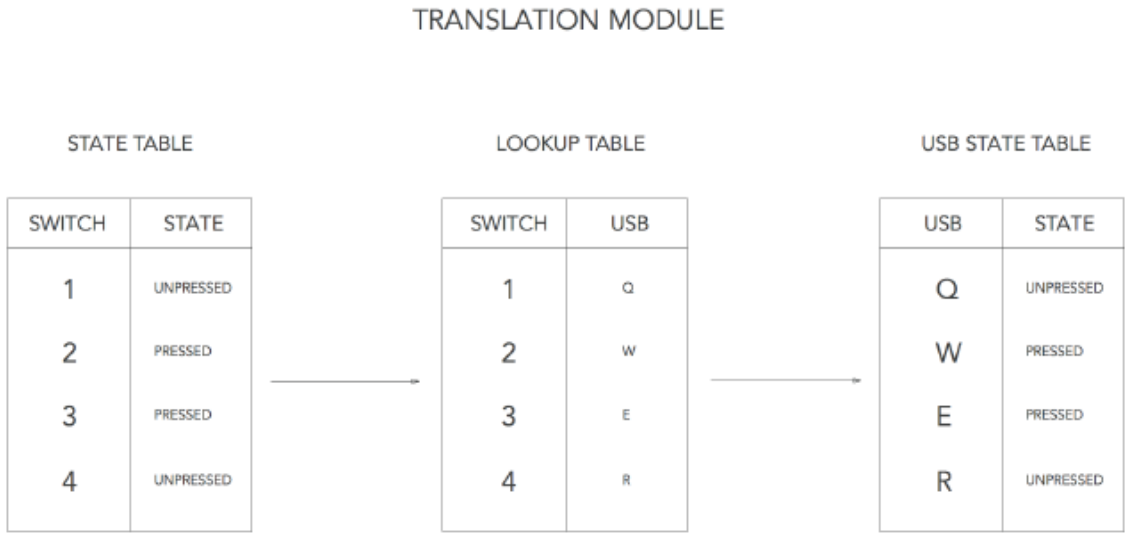
At time = 1 switch 1 is scanned and the state table is updated. The states of switches 2, 3 and 4 are unknown. At time = 3 switch 3 is scanned and the state table is updated. The states of switches 1, 2 and 3 are known while switch 4 is unknown.

Some advanced topics that I've purposefully not covered and may include in a future article include: debouncing, anti-ghosting, NKRO and capacitive sensing.

Translation Module

Now that we have a unique Scan code for each key switch the state of the keyboard is known. However, we are still missing two critical pieces of keyboard firmware programming: layout and host computer communication (e.g. USB). The Translation module is responsible for converting each Scan code to a number that your computer understands. For most modern keyboards this is a USB Code. USB Codes are defined in the USB HID Keyboard spec. The conversion is usually done using a lookup table. This table is also where the keyboard layout is defined.

The USB HID Keyboard defines that every USB keyboard should define a US ANSI-like layout and that it is up to the OS to map the keys for each language. However, as long as the OS is set to a US ANSI layout, the keyboard firmware can define the layout to be anything, as the OS will not do any additional key mappings.



The known switch states are translated using the lookup table and stored in the USB state table. Some advanced topics that belong in this section (but really need their own article) include: macros, layers and KLL (Keyboard Layout Language).

Output Module

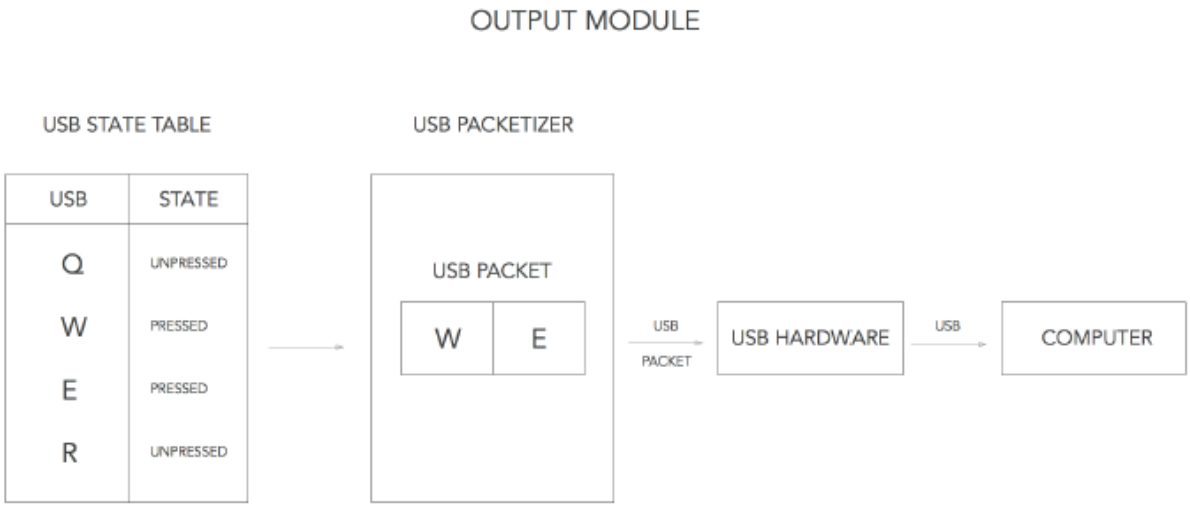
Now that the unique state of each key is known and converted to identifiers that the host computer understands, they have to be sent to the host computer. Basically, all of the identifiers (e.g. USB Codes) that are currently pressed are sent to the host computer in a single message. This identifies to the OS that all of the keys are currently pressed. Any keys that were not sent are not being pressed. As a bit of history, this logic does not always apply to older keyboards and this can pose problems when making a USB converter (like Soarer's or Hasu converters). To be a bit more specific, I'll talk about USB.

Most modern keyboards have microcontrollers with dedicated hardware for sending USB signals; however, some configuration is still needed. Two important concepts about USB need to be covered first: Endpoints and Descriptors.

> Endpoints - An endpoint is a basically a mailbox. It's a place for mail/data to go before it is sent and after it is received. USB devices can be configured to have multiple endpoints allowing things like a keyboard that is also a USB memory stick.
>
> Descriptors - A descriptor is like a map. It tells you where the endpoints are, what they are called, what each endpoint does (so the OS can properly assign a driver) and other sorts of device information like who manufactured the device.

The USB HID spec defines an additional level of descriptors for devices that implement the HID spec. For keyboards this is where things like USB NKRO are defined. These descriptors indicate to the OS HID driver what the message is going to look like (for example, 6 KRO + modifiers vs. NKRO). Once the descriptors and endpoints are set up, it's

as simple as sending a message formatted as per the HID descriptor with the USB codes corresponding to the keys being pressed. To indicate that a key is no longer being pressed, send a message that does not include that USB code.



All keys that are being pressed are sent to the USB Packetizer (makes a USB message), in this case W and E. Then the USB message is sent to the host computer.

Topics that I'll save for additional articles include: composite descriptors, USB NKRO, Bluetooth, PS/2 (and other old keyboard protocols) and debug output.

Conclusion

The goal of this article on keyboard programming is to help you understand what's going on every time you press a key. This understanding is what will provide you with insight into the differentiation between a quality keyboard and an average one. Ultimately, in order to truly use your tools to the best of their ability, you have to be able to understand what they do, and this article is your first step in that process.