Analyzing Keyboard Firmware Part 1 Sweet, clicky fun

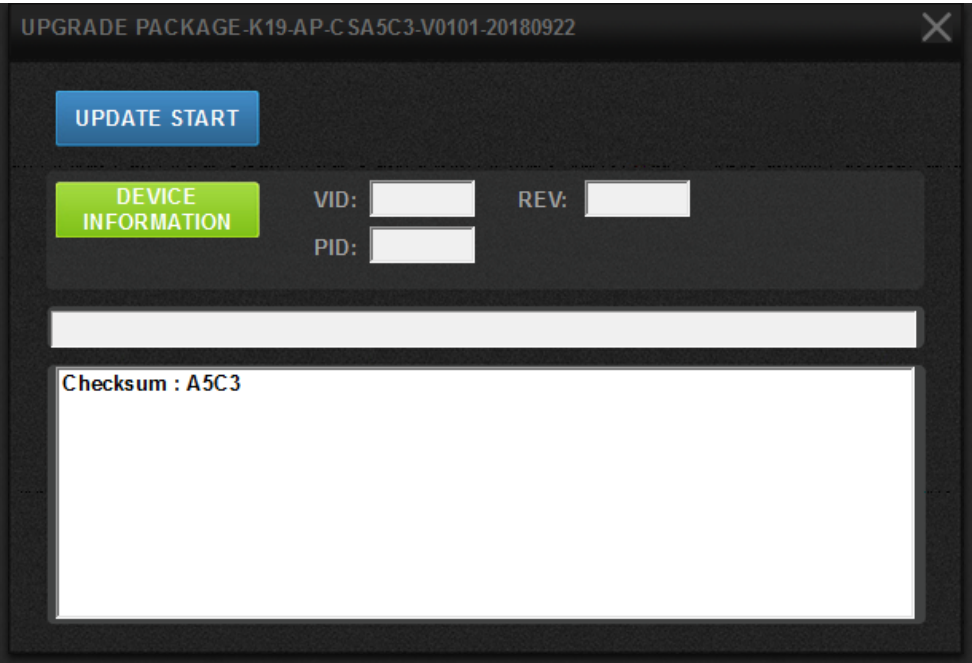
Recently I bought a cheap SPC Gear GK530 mechanical keyboard to test out the Kailh Brown keyswitches. Overall it has a nice feel and you can change the epic RGB-lights to just be a dimmed constant color so I would recommend it for typing.

Unfortunately the macro/rebind feature is not very much to my liking. Right now you have to switch to *Gaming mode* (`FN+Win`) and then you can use use your modified keymap (with macros). In this mode you cannot use the Windows key (because it is locked) and it is generally quite awkward.

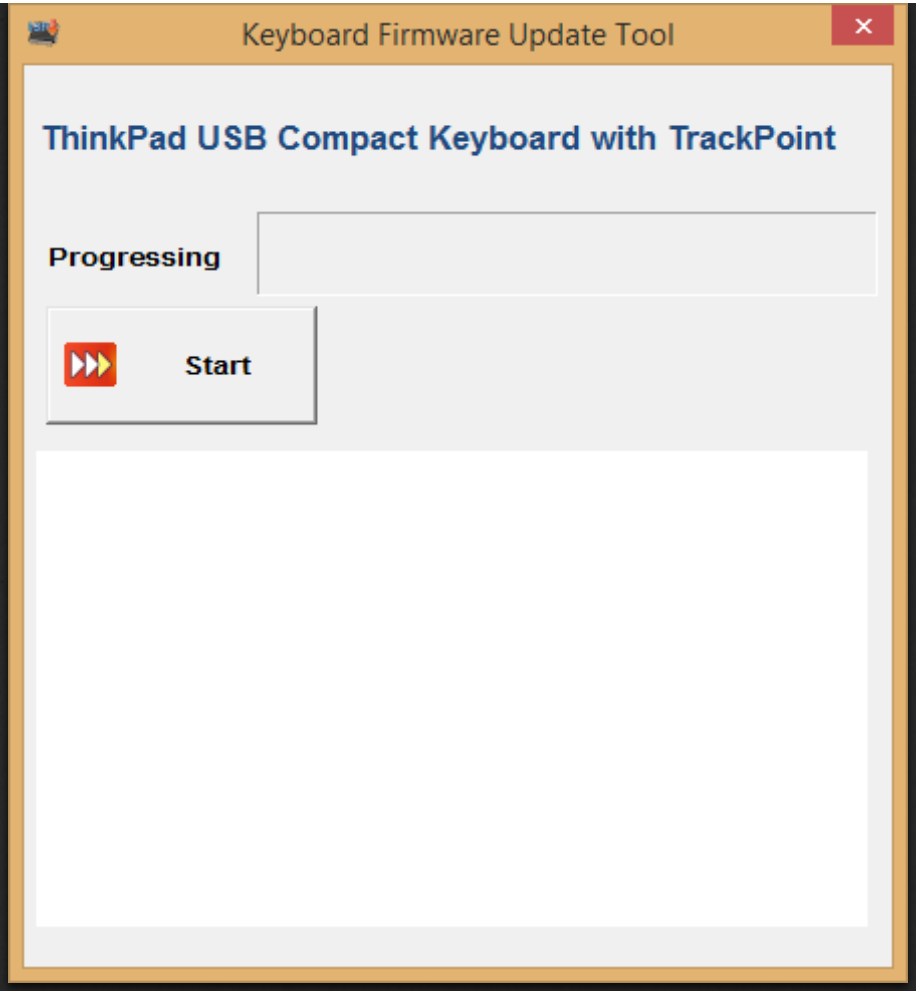
My preferred mode would be that holding `Caps Lock` enables my custom layer, which unfortunately does not seem to be supported. At this point I thought it would be a fun project to try and hack the firmware to support this behavior.

Obtaining the firmware

Luckily for us SPC Gear provides two firmware update utilities on their Software page. The utility is called `1 - GK530 keyboard upgrade.exe` and when I started it I was reminded of a similar utility I took at look at in the past.



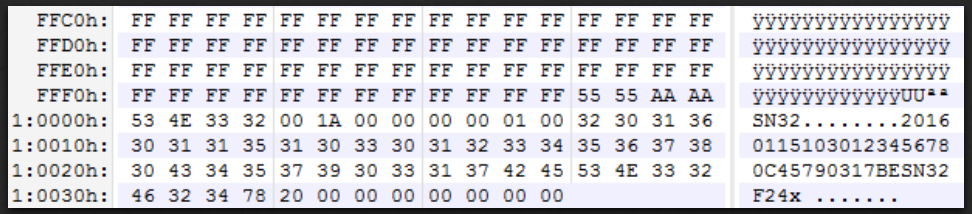
The similar utility was a firmware upgrade tool for my current keyboard:



When looking at a firmware upgrade tool in the past I documented it on a Github issue. From a quick glance it appears that nothing has changed, except that the firmware is no longer 'encypted' now and the UI looks a bit more fancy.

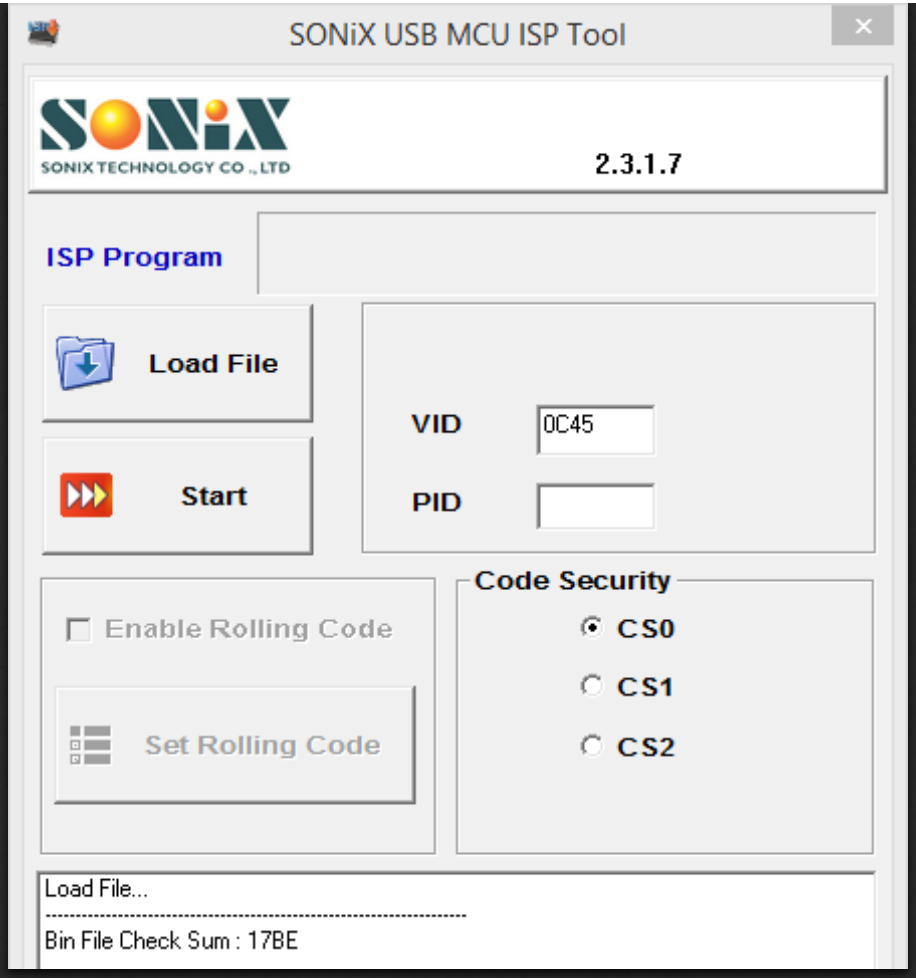
To obtain the image:

- Extract the `"BINARY"` resource (170). You can use CFF Explorer for this task.
- The extracted blob is an executable with no functionality, which has an 'overlay' appended to it (data after the end of the last section).
- The overlay is the firmware image (65k) with some additional metadata at the end.



```
  FFC0h:  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
  FFD0h:  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
  FFE0h:  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
  FFF0h:  FF FF FF FF FF FF FF FF FF FF FF FF 55 55 AA AA  ÿÿÿÿÿÿÿÿÿÿÿÿUUªª
1:0000h:  53 4E 33 32 00 1A 00 00 00 00 01 00 32 30 31 36  SN32........2016
1:0010h:  30 31 31 35 31 30 33 30 31 32 33 34 35 36 37 38  0115103012345678
1:0020h:  30 43 34 35 37 39 30 33 31 37 42 45 53 4E 33 32  0C45790317BESN32
1:0030h:  46 32 34 78 20 00 00 00 00 00 00 00             F24x .......
```

The interesting string there is at the very end, which says `SN32F24x` (all SONiX chips seem to start with SN). With a bit of Google-fu and creativity you can land on the product page for the SN32F248. Over there you can also find a bunch of links to utilities and manuals, one of which is called `SONiX_USB_MCU_ISP_Tool_V2.3.1.7.7z`. Opening the extracted firmware yields a good result:



Additionally there is also a data sheet available to the public (`SN32F248_V2.0_EN.pdf`), which should help progressing further. Initial information I gathered is that the chip is based on an ARM Cortex-M0.

Next steps

My next idea of progression is to try and set up the same tooling as someone developing hardware with this chip would. Get a C compiler and actually build a ROM of our own, to have a point of reference when starting reverse engineering. I do not have any experience looking at ARM code and not much experience with embedded software, so anything to aid my understanding in that area will be good.

Hopefully till next time,

Duncan

Thanks to F for the proofread!

← Previous | Archive | Next →

Published
28 September 2019

Category
reversing

Tags

reversing 6

keyboard 2

arm 2