**Notes for Future Work**

**Cody Bullock | August 8, 2025**

The main features the Rust code is lacking that keeps it from reaching the distances of the ST GUI are the *reflected power sweep* and *frequency selection*. I’ve noticed that the Rust code will tune itself to the first frequency in whatever list it’s pulling from, rather than selecting a frequency that experiences less reflections than others. The tuning results that are pulled from the GUI are useless unless the proper tuning frequency is selected. There are various functions within the C source code that would seem either capable of doing this or at least be somewhat helpful, but I haven’t been able to get any results from them. Some of those functions are:

*Set\_AntennaPower(STUHFL\_T\_ST25RU3993\_AntennaPower \*antPwr);*

*Get\_AntennaPower(STUHFL\_T\_ST25RU3993\_AntennaPower \*antPwr);*

*Set\_ChannelList(STUHFL\_T\_ST25RU3993\_ChannelList \*channelList);*

*Get\_ChannelList(STUHFL\_T\_ST25RU3993\_ChannelList \*channelList);*

*Get\_FreqReflectedPower(STUHFL\_T\_ST25RU3993\_FreqReflectedPowerInfo \*freqReflectedPower);*

*Set\_TuningCaps(STUHFL\_T\_ST25RU3993\_TuningCaps \*tuning);*

*Get\_TuningCaps(STUHFL\_T\_ST25RU3993\_TuningCaps \*tuning);*

Using these functions on Rust instances/variables changed nothing at the Rust level of the code, and seemingly nothing on the C side either. These functions should be tested more thoroughly, but I believe custom functions must be made for things like reflected power sweeps and frequency selection.

A very important source code function is the *STUHFL\_F\_Gen2\_Inventory* function found in *stuhfl\_sl\_gen2.h*. This function is the main driver behind the *inventory\_once()* and *inventory\_runner()* functions by finding all local tags and setting a lot of antenna parameters. My main interest in this function is its ability to change values on the Rust code side when a lot of other functions from the source code seem incapable of this. This function also seems responsible for selecting the operating frequency of the reader. I didn’t have the time to figure out exactly how this function worked, but I do know that it’s able to pull frequencies from the lists found in the Rust code. For some reason, the reader always tunes to the first frequency in the list it’s looking through, which isn’t very effective because it ignores the reflection coefficient. Again, this is why the *reflected power sweep* is important. The Rust code currently doesn’t have a way to tune to frequencies with low reflection coefficients, which leads to it operating at inefficient frequencies poor operation frequencies.

The most positive change I made was changing the default settings of the Tx and Rx levels to 0dB and 19db respectively. The *structs.rs* and *gen2\_structs.rs* files contain easily changeable settings that can be found in the GUI, like check boxes or items from drop-down menus. I didn’t spend a lot of time changing the default settings in these files, but I’m sure they can be fine tuned to increase the range a little further.

Main Points:

The Rust code needs some kind of ***Reflected Power Sweep*** to function similarly to the ST GUI App. The C source code has functions related to it, but it’s difficult to find which ones actually work, or how they should be implemented. If using the functions on the page above doesn’t work, then we should look into creating our own reflected power sweep function.