L1 Test Auto Tone

# Algorithm Background

L1 Transceiver config will affect the line quality, it has 5 sub-parameters as below:

"preEmphasis" : {

            "range" : [-20, 20, 1] #[begin, end, step]

        },

        "mainTap" : {

            "range" : [-20, 20, 1]

        },

        "postEmphasis" : {

            "range" : [-20, 20, 1]

        },

        "txCoarseSwing" : {

            "range" : [0, 6, 1]

        },

        "ctle" : {

            "range" : [0, 63, 1]

        }

If we want to traverse all configs, there will be max 30876608 (41\*41\*41\*7\*64) instances. Each instance will take nearly 30s to test. So that it is impossible to do that. We must design a more effective algorithm to decrease instances.

# Auto tone Algorithm

We design mainly two sub-algorithms to decrease test instances. The first algorithm used to search a basic set of transceiver values roughly. The second algorithm used to get the accurate result based on the rough results.

## Rough Tone Algorithm

Firstly, we decrease the value range and increase the steps to decrease the instance, as below definition:

"TuneRough" : {

        "DAC" : {

            "preEmphasis" : {

                "range" : [-6, 0, 2] #[begin, end, step]

            },

            "mainTap" : {

                "range" : [8, 20, 2]

            },

            "postEmphasis" : {

                "range" : [-6, 0, 2]

            },

            "txCoarseSwing" : {

                "range" : [4, 4, 2]

            },

            "ctle" : {

                "range" : [16, 16, 2]

            }

        },

     … …

    }

In the definition we can seriously decrease instance. It will be max 112(4\*7\*4) instance.

Then we adjust the values priority to optimize the search sequence. We make the more middle value as higher priority. For example:

preEmphasis is from -6 to 0, step is 2. Adjust it:

[-6, -4, -2, 0] 🡪 [-2, -4 ,0, -6]

mainTap is from 8 to 20, step is 2. Adjust it:

[8, 10, 12, 14, 16, 18, 20] 🡪 [14, 8, 16, 10, 18, 8, 20]

At last, we can get a final matrix:

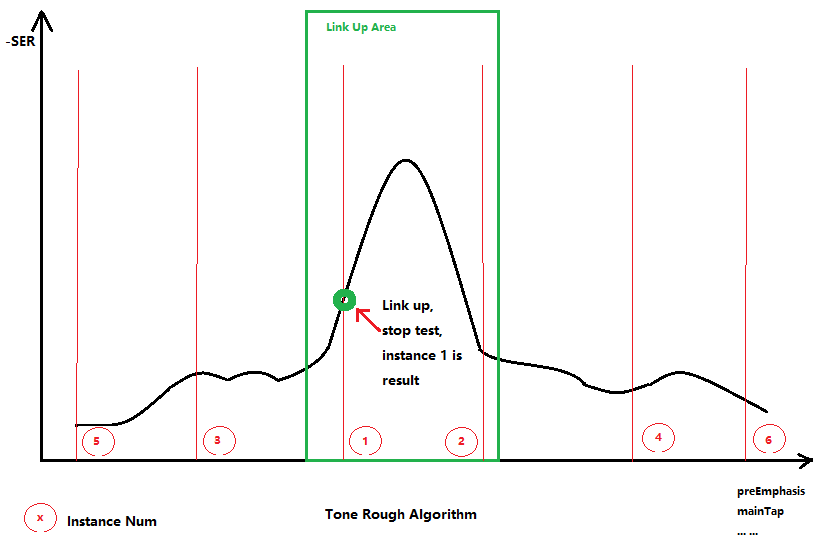
**[-2, -4 ,0, -6]**

**[14, 8, 16, 10, 18, 8, 20]**

**[-2, -4 ,0, -6]**

**[4]**

**[16]**



The instance 1 is [-2, 14, -2, 4, 16], The instance 2 is [-4, 14, -2, 4, 16] … and so on. Follow the priority to do the real testing on STC-L1 device. If we find a result is in “Link Up” area then stop the first algorithm. And the current instance values are the rough result (basic parameter values).

In the real test, we can get the basic parameter values within the first 3 instances.

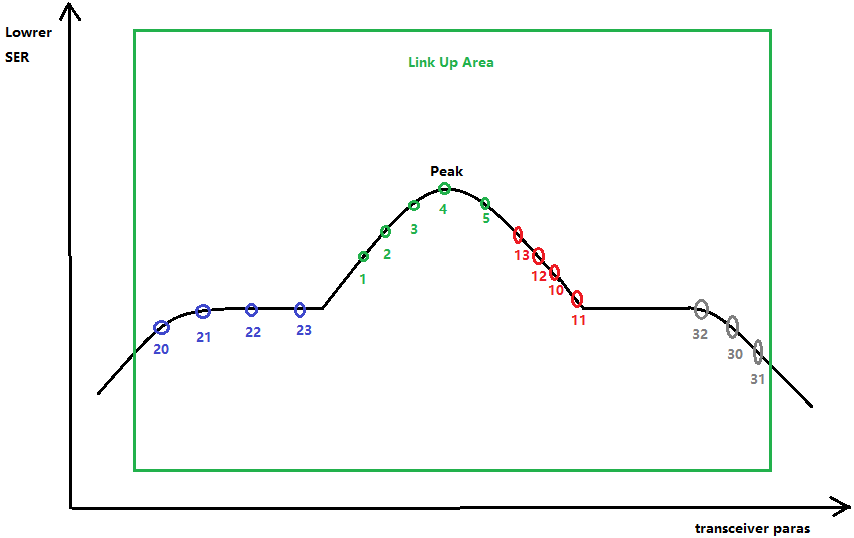
The algorithm one is to search a basic transceiver setting which can make link up. It does not need accurate, link up and 0 codeword error is enough. So, after decreasing instance and adjusting the priority, this algorithm will finish soon normally.

## Accurate Tone Algorithm

Accurate Tone Algorithm is based on the result of Rough Tone Algorithm. It will adjust each parameters step by step to approach the best parameter set.

We have a precondition is the basic parameter set locates in a quadric form area, such as the “Link Up Area”. Based on this we can find the peak point. The peak has lowest SER.

We use the full instance parameter matrix to do the accurate tone with step 1.



The algorithm procedure is as below:

From basic parameter. Set current parameter to first transceiver parameter(preEmphasis).

1. Current parameter value moves right (+= step)
2. Get the SER
3. If SER decrease, go to step1
4. If SER decrease, record better para go to step1
5. If SER increase and no better para found, step \*= -1(reverse direction), reset current parameter, go to step1
6. If SER increase and better value found, record the better para as peek.
7. Set current parameter to next transceiver parameter, if no more finish

For examples of the picture above:

* If the basic parameter from algorithm one is at point 1, the procedure will be as:

1 🡪 2(record) 🡪 3(record) 🡪 4(record) 🡪 5(discard).

4 as peed found.

* If the basic parameter from algorithm one is at point 10, the procedure will be as:

10 🡪 11(discard) 🡪 12(record) 🡪 13(record) … 🡪 5(record) 🡪 4(record) ) 🡪 3(discard).

4 as peed found.

* If the basic parameter from algorithm one is at point 20, the procedure will be as:

20 🡪 21(record) 🡪 22(record) 🡪 23(record) … 🡪 1(record) 🡪 2(record) 🡪 3(record) 🡪 4(record) 🡪 5(discard).

4 as peed found.

* If the basic parameter from algorithm one is at point 30, the procedure will be as:

20 🡪 31(discard) 🡪 32(record) 🡪 …🡪11(record) 🡪 10(record) 🡪 12(record) 🡪 13(record) 🡪 5(record) 🡪 4(record) 🡪 3(discard).

4 as peed found.

From the description above, we can get the peek point of transceiver parameters. The remained issue is SER jitter so we will use 10% of SER as redundancy. As follow:

offset = abs(current\_line\_quality – best\_line\_quality)

     if offset < int(best\_line\_quality \* 0.1):

        # Deem as jitter

        line\_quality = best\_line\_quality

# Test results

Follow these two algorithms, we wrote a python script to find the best transceiver parameter, and we got a good result which is better than AN result.

Text

Description automatically generated

In this case, the script use 11 instance to find the peak point (best transceiver parameter) within 10 min. The final SER is 60.