Optimize FOM Reduction

COM Commit Request Number 4p9_2
(Resubmit of Request 4p8_6)

Adam Gregory, Samtec

Background

- This commit request (4p9_2) is identical to the previous 4p8_6 request. That request was deferred so that people would have an opportunity to test the code changes and verify that no changes in output were observed.

Purpose

- Reduce the size of the optimize_fom function.
- Maintain the same functionality.
- The optimize_fom function is 1400 lines and has become difficult to update since it is massive, and deciphering the logic paths requires a large cognitive load. Now that COM is in a repository where many people can collaborate, the likelihood of code conflicts and errors in optimize_fom is almost certain.
- This function is a constant source of updates, so it is important to contain the expansion before active collaboration ramps up.
    - In the last 4 years, it grew from 700 lines to 1400 lines.

Summary

- This proposal reduces the number of lines to from 1400 to 400 by sending logical blocks into new subfunctions.
  - Also reduces the number of variables in the workspace from around 180 to 60.
- At this point, the 23 new subfunction names have the prefix “OptFom_” to easily identify them
  - The majority of these subfunctions would have no use outside of optimize_fom, but it is possible that some could rebranded for general purpose.
- Over 30 test cases were written to validate this update. Each case gives identical output to the version of COM without the changes.
  - Identical means that every field in the output Result struct returns true for “isequal”
  - The test cases were developed with the strategy of hitting all possible conditions within optimize_fom. There is almost 100% code coverage within optimize_fom and the new subfunctions.

Optimize FOM Flowchart

- The basic goal is to make optimize_fom look something like the pseudo-code shown here.
- This is an oversimplification, but it expresses the overall arc of the code.
- It turns optimize_fom into a facilitator and pushes the actual technical work into manageable subfunctions.
- Three new structures are introduced to assist with organization and sending data between functions
  - BEST: container to hold the best EQ data
  - THIS: container to hold the EQ data for the current loop
  - SETTINGS: container to hold settings are independent of the EQ loop

```
% SETTINGS:  hold settings that are not EQ loop dependent
% THIS:  hold data for current EQ settings
% BEST:  hold data for best EQ settings
% Result:  final output from optimize_fom
SETTINGS = Calculate_Settings();
for ctle_index = 1:num_ctle
    Compute_CTLE();
    for txffe_index = 1:num_txffe
        Compute_TXFFE();
        Find_Sample_Point();
        for itick = 1:num_sample_points
            Compute_RxFFE();
            Compute_DFE();
            Calc_Noise();
            Calc_FOM();
            %At this point, THIS holds all current loop settings
            if THIS.FOM > BEST.FOM
                BEST = THIS;
            end
        end
    end
end
Result = BEST;
```

List of subfunctions created (Pre-Loop)

1. OptFom_Initialize_Loop_Struct: Initialize the container that holds all the settings for the current loop
2. OptFom_Build_TXFFE: dynamically construct the TxFFE sweep settings and tap indices
3. OptFom_Calc_Hr.m: return Hr (combined effect of all filter gains)
4. OptFom_FD_or_TD_Fields.m: determine which fields to use in chdata for FD vs. TD Mode
5. OPTFom_Calculate_Settings: Build a container that holds miscellaneous settings that are independent of the EQ Loops

List of subfunctions created (Main functions In-Loop)

6. OptFom_Compute_CTLE.m: return total CTLE gain and modify chdata impulse response to include the CTLE
7. OptFom_Compute_TXFFE.m: calculate pulse response with TXFFE applied
8. OptFom_Find_Sample_Point.m: return cursor_i and peak_i (sample point and peak point)
9. OptFom_Compute_DFE.m: calculate all DFE taps, floating taps, and tail taps
10. OptFom_Compute_RxFFE: culcate RxFFE taps and apply to pulse response. Also returns PSD_results, MMSE_Results, and FOM when MMSE is enabled.
11. OptFom_Calc_Noise: Calculate all the noise parameters that are needed (h_J, sigma_TX, ISI_N, sigma_N, total_noise_rms)
12. OptFom_Calc_FOM.m: calculate FOM for a particular loop (not used when RxFFE with MMSE is enabled)
13. OptFom_Update_Best_Setttings: update BEST settings when current FOM > Best FOM

List of subfunctions created (Auxiliary functions In-Loop)

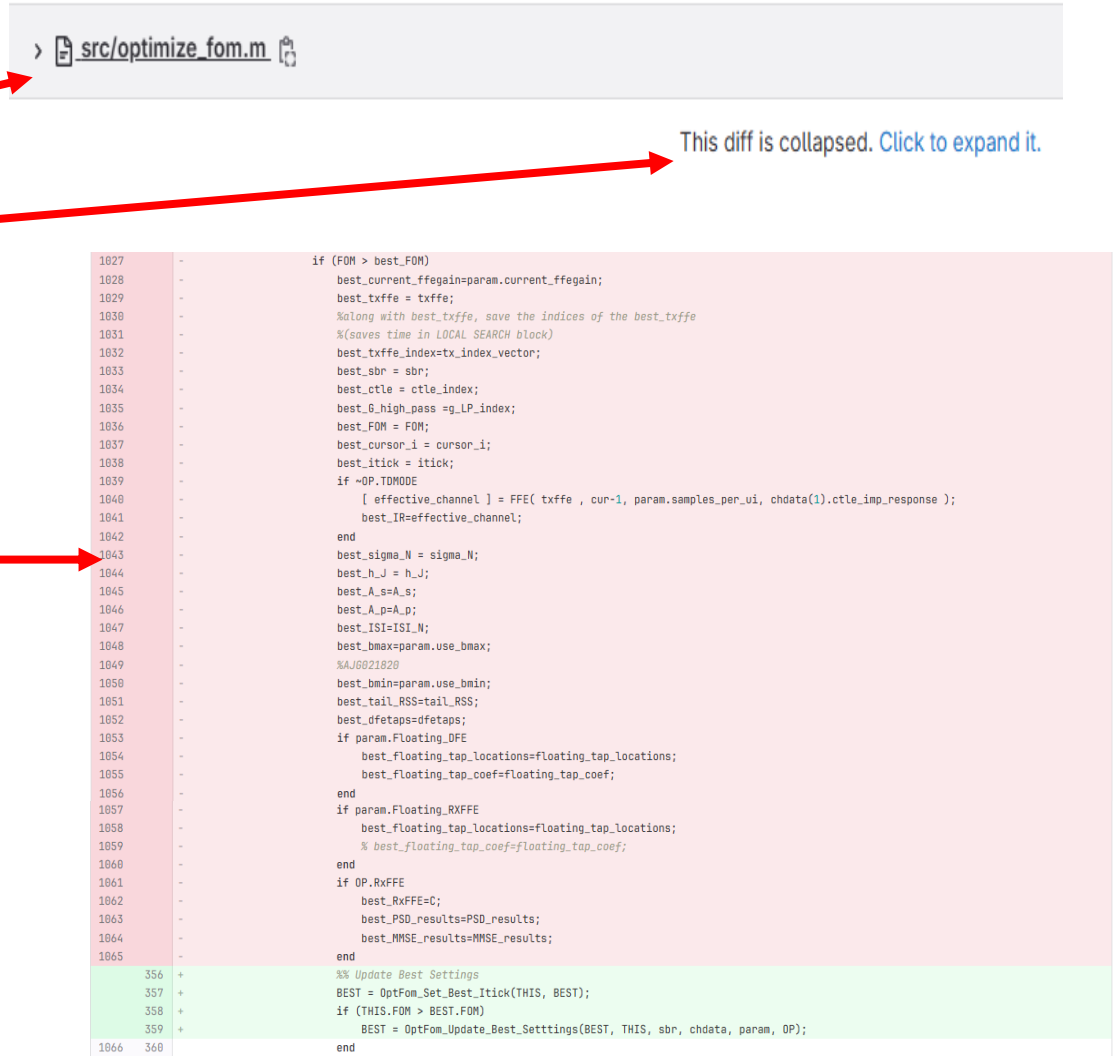
14. OptFom_Local_Search.m: run the LOCAL SEARCH routine (determine if this EQ loop should be skipped)
15. OptFom_Setup_Sampler_Sweep.m: logic to setup the sampler point sweep (sample_adjustment in the config spreadsheet)
16. OptFom_Itick_BoxSearch: handle box search itick sweep (not currently enabled but is in optimize_fom as placeholder)
17. OptFom_Itick_LocalSearch: handle Local Seach for Itick sweep (only when TS Search Mode = Middle)
18. OptFom_Set_Best_Itick: set the best itick settings for the current sweep (used in Itick Local Search)
19. OptFom_Calc_Noise_XC.m: calculate Noise_XC (obsolete)

List of subfunctions created (Post-Loop)

20. OptFom_Update_Best_Settings_EQ_Failed: update BEST settings after EQ optimization for the special case where no valid FOM was found
21. OptFom_Update_BEST_Post_Optimize: update BEST settings for fields that are only updated after EQ optimization loop
22. OptFom_Plot_Best_Results.m: debug plot at the very end of optimize_fom
23. OptFom_Create_Output: create final output structure

Change comparison

- This URL shows the diffs in optimize_fom:
  - Compare main vs. Reduce Optimize FOM
  - Scroll to the bottom to see “src/optimize_fom.m”
  - It will say “This diff is collapsed. Click to expand it”
  - Click that to see all the diffs
- In general, the diffs look as shown on the image to the right
  - A bunch of lines pushed into subfunctions
- Due to the nature of this refactoring, it is difficult to visually observe the changes.



Changes to config

- Changes to config
  - None
- Changes to output
  - None
- Download beta test code
  - Beta Test: Optimize FOM Reduction

Running this Update using the GIT repository

- You can also run this update by pulling the git repository from the 802-COM website
- Run this command from git bash:
  - git checkout Optimize_FOM_Reduce
- This will switch your working area to the branch which contains all the reduction to optimize_fom
  - You can tell it has updated because you will have all the OptFom_* functions in src\ folder
- Run this command to switch back to main branch:
  - git checkout main
- You can run tests on your own to validate the update. Something like this:
  - While in main branch:
    - Ref_Result = com_ieee8023_(.....)
  - While in Optimize_FOM_Reduce branch:
    - New_Result = com_ieee8023_(.....)
  - Then run the compare function:
    - Management.compare_results(Ref_Result,New_Result);
    - Note: The compare_results function only takes scalar structs. So if your COM result output has length > 1:
      - Management.compare_results(Ref_Result{1},New_Result{1});
  - It will say “Results are equal” or it will give a list of which fields within the result struct are not equal and the numerical difference (if applicable)

Screenshot of running the compare test on many cases

```
>> for j=1:length(new_result_sweep)
    fprintf('Case %d\n',j);
    Management.compare_results(ref_result_sweep{j},new_result_sweep{j});
end
Case 1
Results are equal
Case 2
Results are equal
Case 3
Results are equal
Case 4
Results are equal
Case 5
Results are equal
Case 6
Results are equal
```