SPAI - Lab Assignment 2

DSP Implementation: Real Time Adaptive Notch Filter

Hannes Rosseel
KU Leuven, Faculty of Engineering Technology
Campus Group T

December, 2023

The previous SPAI lab sessions prepared you for the implementation of a real-time adaptive notch filter (ANF), capable of removing feedback oscillations in closed-loop systems. This document will provide you with the assignment and deliverables details. At the end of this text, you can find the evaluation criteria for this assignment.

1 Assignment

The goal of this assignment is to build a real-time implementation both in fixed-point C and Assembler code on the TMS320C5515 eZDSP USB Stick, which implements the least mean squares (LMS) based 2nd order biquadratic ANF. During the theoretical lectures by prof. Toon van Waterschoot, you already gained the insights leading to the design of the ANF algorithm seen in Algorithm 1.

```
Algorithm 1 : $2^{nd}$ order ANF-LMS algorithm
Require: step size $\mu$, initial pole radius $\rho(-1)$, final pole radius $\rho(\infty)$, exponential decay time constant $\lambda$, input data $\{y(m)\}_{m=0}^{N-1}$, initial conditions $s(-1), s(-2), a(-1)$
Ensure: $2^{nd}$ order ANF parameter $\{a(m)\}_{m=0}^{N-1}$
1: for $m = 0, \ldots, N-1$ do
2:     $\rho(m) = \lambda\rho(m-1) + (1-\lambda)\rho(\infty)$
3:     $s(m) = y(m) + \rho(m)a(m-1)s(m-1) - \rho^2(m)s(m-2)$
4:     $e(m) = s(m) - a(m-1)s(m-1) + s(m-2)$
5:     $a(m) = a(m-1) + 2\mu e(m)s(m-1)$
6: end for
```

Algorithm 1: Summary of the second order ANF-LMS algorithm [1].

1.1 Start Files

You will not have to start from scratch. Several files are available on Toledo to help you get started:

- Python files:
  - anf.py: The ANF-LMS algorithm implementation in Python
  - file_parser.py: Used to read and write Python variables to .pcm files
- Code Composer Studio files:
  - mainoffline_anf.c: main code file in C which runs the algorithm offline
  - mainonline_anf.c: main code file which runs the algorithm in real-time
  - anf.c: fixed-point ANF C implementation
  - anf.h: ANF function header file
  - anf.asm: fixed-point ANF Assembler implementation

1.2 Tasks

1. Analyze the provided Python code and understand the ANF-LMS algorithm. Identify the correct Q-factor for each variable in the algorithm. Motivate this choice in the report.
2. Implement the 2nd order ANF summarized in **Algorithm 1** in fixed-point C code. Start with a simplified version by using a fixed pole radius ρ. Extend your implementation with an iteratively adaptive ρ when the simplified version is tested.

   Note: after each coefficient update, make sure that $|a(t)| < 2$. If this is not the case, the second constraint of the ANF is violated (= zeros should always lie on the unit circle).
3. Test the fixed-point C code in an offline manner with a self-chosen test signal by writing the signal to an input.pcm file and inputting the file in your Code Composer Studio (CCS) data folder. After your code ran successfully, analyze the created output.pcm file in Python.
4. Implement the 2nd order ANF algorithm in Assembler code, like in task 2.
5. Test the Assembler code with a self-chosen test signal by writing the signal to an input.pcm file and inputting the file in your Code Composer Studio (CCS) data folder. After your code ran successfully, analyze the created output.pcm file in Python.

6. Test both implementations in real-time using the code provided in *mainonline_anf.c*. Make sure that your code is fast enough to keep up with real-time demand. Document your testing process in the report.

7. Discuss the computational cost and results of both implementations.

1.3 Extras

Create a new *main_cascade_anf.c, anf_cascade.h, and anf_cascade.asm* file. *Implement a 4^{rd} order ANF filter by cascading two 2^{nd} order ANF filters in Assembler. The 4^{rd} order ANF filter should be able to simultaneously attenuate two narrowband noise sources in one signal.*

2 Deliverables

The deadline for this assignment can be found on Toledo. Please submit your code files, a written report, and possible test files **in a single .zip file** named **{name}_{lastname}_assignment2.zip**. Every student needs to submit the assignment individually. Clearly state the names of all group members in the report.

2.1 Code files

Your code needs to be structured and **contain sufficient self-explaining text comments**. Your code is evaluated on its readability and efficient memory use (see rubric on Toledo). You will need to submit a minimum of three code files: the main C file containing the program structure, the fixed-point C ANF function, and the Assembler ANF function.

2.2 Written report

The written report needs to summarize the implementation steps of the ANF algorithm. You will evaluate your implementation (what works, what doesn't work, what could be improved) and discuss the performed functional tests and validations of both C and Assembler implementations. You will also need to include a computational cost analysis for both C and ASM implementations, and motivate your choice of fixed-point Q-factor for each variable in the ANF-LMS algorithm.

The report will be graded on the correct use of grammar, spelling, and punctuation. The report ought to be easy to read, and its contents presented in a logical order. Add personal creativity to your report by including images, tables, additional testing, and so on.

3 Grading

The final evaluation score for the lab will be composed out of: 20% for the evaluation of Lab Assignment 1 and 80% Lab Assignment 2 which is the evaluation of the submitted files (the assessment is based on the rubric which you can find on Toledo).

Important note

The C5515 development board needs to be returned on the date of the written exam. **Bring your DSP development board to the written exam!**

References

[1] Toon Van Waterschoot. *Digital Signal Processing-2, adaptive notch filters for acoustic feedback control.* 2014. URL: `https://homes.esat.kuleuven.be/~tvanwate/courses/dsp2/1415/DSP2_coursenotes_04_adaptievefiltering.pdf` (visited on 12/02/2020).