Overhaul the gr-fec Module

Hamza Mohammed Hasan

Zewail City of Science and technology in Egypt, BSe

hamza.mohammed.hasan@gmail.com

https://github.com/StudHamza

bluemare on **matrix**

Google Summer Of Code 2024

Abstract

This document proposes a Google Summer of Code project to possibly overhaul the current gr-fec module by migrating the variable (en|de)coders implementations into more versatile and efficient implementations that are within the AFF3CT Library. This will not only add new complex FEC (en|de)coders, but will also improve the efficiency of the existing ones. Three of the main key objectives in this proposal are: 1) Introduce modern FEC codes to the gr-fec module. 2) Improve the performance of existing (en|de)coder variable algorithms, (complexity and memory wise). 3) Abstract implementation details of standard/complex (en|de)coder variables.
keywords:

1 Introduction

In digital communication, Forward-Error-Correction (FEC) is technique used to detect and possibly fix errors in data transmission over noisy channels. Gnu-radio has a gr-fec package that allows for FEC protocol implementations and performance evaluation. The API design for the FEC module allows for flexible addition of new FEC coders to the package. Some coders implementations currently available in gr-fec are: repetition, convolutional,TPC , polar, and low density parity check (LDPC) codes.
Over the years, many new FEC and possibly more performant methods have been implemented. The idea is to directly use software Error Control Codes (ECCs) in the communications; in this context, there are various available Open-source software like , AFF3CT (A Fast Forward Error Correction Toolbox) that takes care of modern code implementations; which are not in the gr-fec package, like Turbo, Reed-Solomon, and BHC codes[1]. Not only that but AFF3CT library have been optimized using SMID instructions. Luckily the AFF3CT can be used as a simulation tool as well as a library under the MIT license[2]. The main key objective in this proposal is to introduce the AFF3CT library to the gr-fec module.

2 Proposed Project

2.1 Literature Review

Why do we need better FEC module you might ask. Gnu radio is mostly used in research field, however recently it has gained more popularity with real time systems doing packetized data like Generic Monitoring systems[3] or real time UVA video transmission [4]. The upcoming wireless 5G standard raises a new range of applications for the software ECCs: the Cloud Radio Access Networks (C-RAN) in a Software-defined radio (SDR) context[5]. The gr-fec module lacks many modern implementation which will not only halt its effectiveness, but it also might not make it compatible with newer technology. Upgrading or possibly overhauling the gr-fec module will improve gnu-radio overall security, compatibility and features.

What are FEC Codes? FEC are techniques to detect and possibly fix errors of data, which are transmitted over noisy channel. Given Shannon's theorem we know its possible to send information bits nearly error-free up to a computable maximum rate through the channel. However as we approach the Shannon Limit, our FEC implementation becomes fairly complex. Some codes like the LDPC have approached this limit and are currently used in communication channels.

2.2 Proposal

The FEC API operates on two levels, the coder deployment and the coder variable. The coder deployments deals with input/output data, data (de)puncturing, communication with the coder variables and interaction with the scheduler. While the coder variables are mostly responsible for the "work" or "processing" of the data.
My proposed idea is to add a pointer to an (en|de)coder AFF3CT object to every code variable. We will initialize the encoder object during construction of the variable.

```
std::unique_ptr<module::Encoder_repetition_sys<>>(new module::Encoder_repetition_sys<>(K, N ));
```

Listing 1: Pointer to AFF3CT Repetition Encoder Object

The encoder object takes *int K* number of input bits and *int N* number of output bits. The encoder has a encode method, _encode(), which is responsible to do the work. The encode method takes in 2 stream, a constant input stream/buffer to read from and a output stream/buffer to write on. The third argument may be ignored.

```
template <typename B>
void Encoder_repetition_sys<B>
::_encode(const B *U_K, B *X_N, const size_t frame_id)
```

Listing 2: _encode method

The idea of this proposal is to add a AFF3CT (en|de)coder class pointer in each coder variable, and change the generic_work function, to implement the class methods. This will not change the over all design of the FEC API since the only thing that will change is the generic_work() method.
In this proposal I plan on implementing new coders to the gr-fec package, specifically the following:

- Turbo
- Bose–Chaudhuri–Hocquenghem codes (BCH codes)
- Reed–Solomon codes (RS)
- Raptor code (RA)

I will also move the implementation logic of the coders to the AFF3CT library to handle as explained above. And lastly, I plan on performing some bench marking analyses of the new and old code implementations using the BER generator in the gr-fec package.
Advantages of using AFF3CT implementations:

- Encapsulation of complex coder implementations
- New and up-to-date implementations of FEC coders
- The code SIMDization rest upon the MIPP wrapper [2]
- AFF3CT takes advantage of the 8-bit and 16-bit built-in fixed-point and saturated arithmetic of your CPU.
- Potential involvement of parallelism
- Potential involvement of performance measurements utilities

2.2.1 Parallelism in AFF3CT

This section discuses the potential involvement of AFF3CT multi-threaded architectures, that using pragma directives of the well-known OpenMP language, in gr-fec module.

2.3 New Flow

2.4 Deliveries

- Implementing the AFF3CT coders to existing code variables:
  - Polar, LDPC ,Conv, Rep
- Implementing new coders to the gr-fec package:
  - Turbo Codes, BCH codes, RS and RA
    - Many issues like #6016 BHC and #6769 TCP codes will be closed
- Bench marking of old and new implementations
  - Simulate both new and old blocks on gr-fec and AFF3CT performance measures.
- Add QA python tests for new blocks
- Add documentation to c++ API of gnu-radio
  - Fix poorly documented pages, like this
  - Add .grc examples for new codes
  - Tutorial on FEC API

2.5 Testing And Documentation

For every new block implemented a QA test would be written. I will also be responsible to update the gnu-radio wiki, and c++ API reference. Each block will be tested locally before pushing to origin/main. All bench-marking data will also be available for further analysis if needed.

3 Schedule

During the project period I will always be available through email, GitHub, irc-channel and matrix. The project is 350 hours, so I will contribute a minimum of 30 hours a week. During the project period i will be full time enrolled in summer school, but i will manage to put 3-4 hours a day to the project.

3.1 Milestones

- Phase 1: Complete Re-implementation of old blocks to AFF3CT-based algorithms.
- Phase 2: Complete Implementation of new codes to the gr-fec module.
- Final submission: Full documentation of old and new codes along with bench-marking statistics and future goals for the package.

3.2 Timeline

TABLE 1 Timeline

April 2 - May 1	**Coordination Period:** • Planning to bond with the community, active through IRC chat, Matrix and mailing list. • Building OOT Repetition code form AFF3CT Library, for practice. • Study feature request with label "FEC" • Refine details of the project

TABLE 1 Timeline

May 1 - May 26	**Start of work Period:** • Bench Marking existing FEC coders, store benchmark information for further analyses. • Read: Iterative Error Correction • Set-up git repository
May 27 - June 4	**Coding begins:** Write AFF3CT Repetition (en|de)coders. Test the block and update documentation. (started with this as a warm-up)
June 4 - June 20	**Delivery 1:** Move LDPC, TCP and polar blocks to AFF3CT-based implementations. (Add documentation + tests)
June 20 - July 1	**Wrapping and results:** Test, Simulate and bench-mark new blocks. Write a blog #1 about the difference.
July 2 - July 11	**New Blocks start:** Add BCH block. Test, Simulate and write QA tests for it. Add documentation for new block.
July 12	**Midterm Evaluations**
July 13 - August 1	**Delivery 2:** Add + test RS, Turbo , and RA blocks.
August 1 - August 6	**Deliveries 3 and 4:** Wrap up new and old blocks. Refine Source code and documents (c++ API)
August 7 - August 16	**Bench-marking #2:** use both packages (gr-fec (BER curve block) and utilise-AFF3CT (BER/FEC tools)) to simulate and bench-mark new performance. Compare and write blog post 2 about new updates of the gr-fec document. Document simulation comparison results.
August 17 - 25	Wrap-up functions,blocks, examples and documentations. Blog 3, write about how the AFF3CT parallelism and performance evaluation utilise could be beneficial to gr-fec module. Submit final work.

4 Acknowledgement And Licence

The entire code during the coding period will be transparent, i.e., available on GitHub. The code submitted will be GPLv3 licensed. The library used is licensed under MIT. I acknowledged and understand the rules of conduct page, including the three strikes rule.

5 About Me

My name is Hamza, currently pursuing my bachelor's degree in Communication Engineering in Egypt. I am comfortably familiar with multiple programming languages including C,python and Matlab. One of my interests is programming of digital signal processing algorithms. I also really enjoy the course discrete mathematics.
Never contributed to open-source projects before, but excited to do so. However I worked on various related areas like IoT-narrow band SDR and a POSIX OS scheduler. One of my worth mention projects is the re-implementation of MPEG-layer 2 compression, which takes advantage of the human auditory system and the psycho-acoustic model to perform a lossy compression. I am also familiar with Linux operating system.

I only recently heard about gnu-radio, and i spent most of my time in the C++ API reference and the tutorial page. I also payed around with the source code and attempted to solve "good first issues". This project aligns with my personal and academic interests and I plan on being a contributor to the gnu-radio project, even after my GsoC period. I am even thinking of using gnu-radio for my graduation project.

6 Conclusion

In conclusion, the gr-fec package is a really useful package for many fields. Whether this proposed idea of implementing AFF3CT library is accepted or not, there should be some other way to update the package for future usage.
Cyberspectrum is the best spectrum

References

[1] A. Cassagne, O. Hartmann, M. Léonardon, K. He, C. Leroux, R. Tajan, O. Aumage, D. Barthou, T. Tonnellier, V. Pignoly, B. Le Gal, and C. Jégo, "AFF3CT: A fast forward error correction toolbox!," *SoftwareX*, vol. 10, p. 100345, July 2019.

[2] "Aff3ct library examples." Accessed: 2024-03-25.

[3] R. J. Steinhagen, M. Kretz, A. Krimm, D. Kozel, J. Morman, I. Čukić, and F. Osterfeld, "Gnu radio 4.0 for real-time signal-processing and feedback applications at fair," in *Proceedings of the 14th International Particle Accelerator Conference, IPAC*, vol. 23, 2023.

[4] A. Demir and H. Çevikalp, "Real time compressed uav video transmission by using hackrf one,"

[5] Wikipedia, "Error correction code."