GPU Based Receiver

Design Document

Dylan Lewis
dbl1716@gmail.com

October 23, 2021

Contents

1 Introduction **1**
1.1 Receiver Interface 1

2 Project Architecture **2**
2.1 Functional Model 2
2.2 Workflow 2

3 Design **4**
3.1 Polyphase Filter 4

4 Results **5**

5 Conclusions **5**

1 Introduction

This project is intended to create a QPSK receiver that runs at a speed greater than 50_{Mbaud}. The modem will be implemented using Graphics Proccessing Units (GPUs) to meet the desired throughput. Nvidia's Compute Unified Device Architecture or (CUDA) will be used to interface with the GPUs and use them as general compute devices.

1.1 Receiver Interface

The receiver will perform demodulation, decoding, and deframing of the digitzed input signal. It is expected that the input signal contains 2 samples per symbol, to minimize the input data rate to the modem. Input data is expected in binary format with I and Q interleaved, with a type of "float" where a float is 4 bytes.

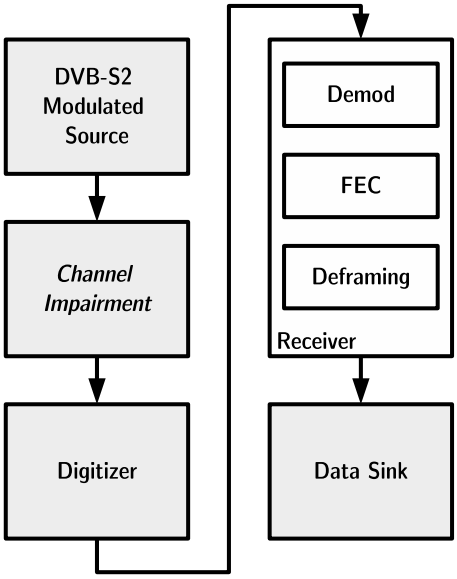

Figure 1: Top level data flow into and out of receiver

2 Project Architecture

2.1 Functional Model

The receiver model is implemented in the Julia language and was tested using Julia v1.6.3 The model is intended to be functionally equivalent to the CUDA receiver but not "bit true". The model is used to rapidly prototype the receiver and to provide a golden reference to compare against the CUDA receiver.

2.2 Workflow

The workflow utilizes some CI/CD principles to reduce project management overheads. A local Ubuntu 20.04 machine, **cuda-dev**, is used as both the build machine and as a worker. The **cuda-dev** machine is configured with the following.

- AMD Ryzen 5 3600 6-Core Processor 16GB RAM
- Nvidia GeForce GT 710
- Nvidia CUDA Linux Drivers v460.91.03
- Docker Engine v20.10.9
- Nvidia Container Toolkit (nvidia-docker2 v2.6.0-1)
- GitHub Actions Runner

The Github Actions runner is used to facilitate automatic builds and unit testing. The build environment runs as a Docker container and is based on an Nvidia provided CUDA container (v11.4.2-devel-ubuntu20.04). The build container includes the Nvidia toolkit v11.4.2 and the Julia v1.6.3 interpreter. The build container compiles the CUDA accelerated receiver and creates a docker image. The **cuda-dev** machine then runs the newly created image and executes unit tests on the compiled CUDA receiver and compares against the Julia model. Containers are used to simplify transition to a cloud environment after development and to easily replicate the build environment. See **Figure 2** for block diagram of the workflow.

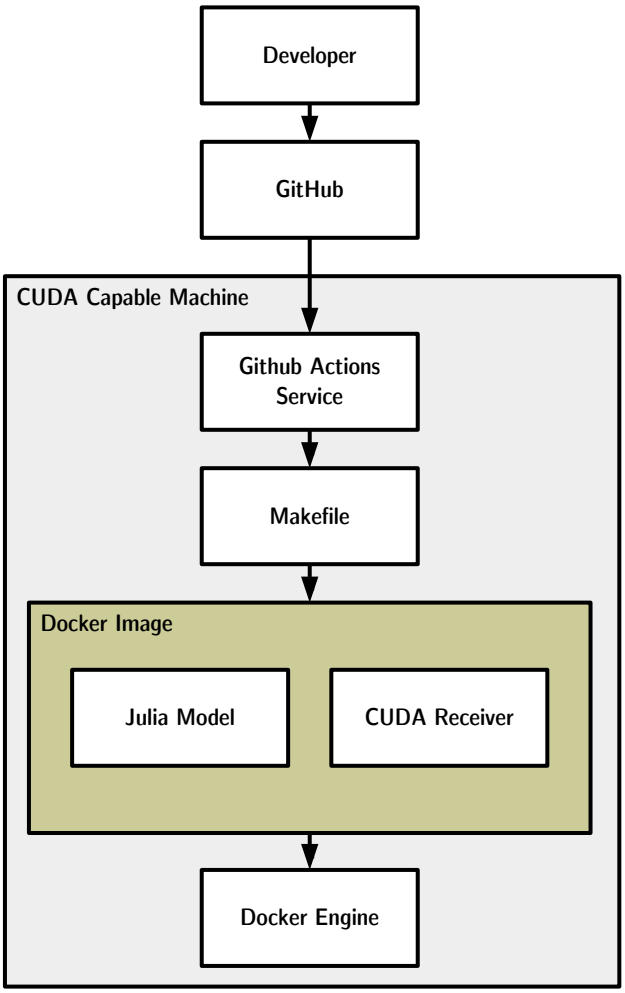

Figure 2: CI/CD

Unit testing will primarily be conducted using test vectors. The Julia model and the CUDA receiver will always run in parallel and receive the same input during unit testing.The Julia model represents the "golden vector" to which the CUDA receiver is compared.

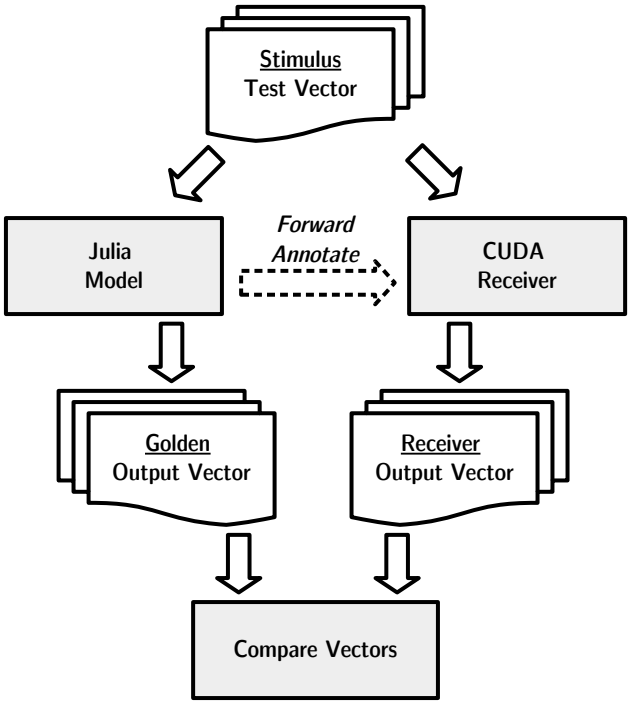

Figure 3: Modeling Iteration

3 Design

3.1 Polyphase Filter

Polyphase filtering in an efficient structure for performing filtering and sample rate conversions. Polyphase filtering is used in the demodulator portion of the receiver to provide matched filtering, symbol timing recovery and decimation of the input stream down to 1 sample per symbol. Because the input is only 2 samples per symbol we must perform interpolation on the input to recover the symbol timing. This is computationally expensive and must be done in the GPU. The GPU implementation of the polyphase filter structure will be unit tested against a CPU based polyphase filter and a CPU based interpolate then filter structure.

Polyphase Decomposition For a given system with interpolation factor P and impulse response $h[n]$ we can use polyphase decomposition to break the filter down into M subfilters which can be moved to before the interpolator using the second noble identity. To decompose a filter, first express the filter in terms of subfilters where the taps are seperated by the interpolation rate.

$$H(z) = \sum_{k=0}^{P-1} z^{-k} G_k(z^P) \tag{1}$$

$$G_k(z) = \sum_{n=-\infty}^{\infty} h[nP + k] z^{-n} \tag{2}$$

Now it is simple to move the subfilters before the interpolation using the second noble identity.

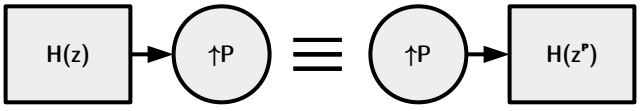

Figure 4: Second Noble Identity

4 Results

5 Conclusions