Proposal: VDIFParse

Mars Buttfield-Addison

May 4, 2022

1 Purpose

Currently, the need to ingest and manipulate telescope data in the modern standard VDIF (or variant CODIF) formats is met by DiFX with its `mark5access` and `vdifio` libraries. However, the need for `mark5access` to also support several legacy formats has prevented it from using practices best suited to the VDIF format, at a significant cost to performance. The limited functionality of the `vdifio` library could not be incorporated into `mark5access` without suffering the inherent slowdown to multithreaded VDIF processing.

This presents an opportunity to create a new library that combines the functionality of the two previous libraries, designed especially for performance with the VDIF format. Coupled with a modern, easy-to-use API, such a library could also fill a gap that would allow quick creation of small VDIF processing programs. Herein proposes such a solution, called `VDIFParse`.

2 Assumed Use Cases

- Someone wants to organise/filter data into files that will be faster to decode or analyse later.
- Someone wants to know the properties of the data they have.
- Someone wants to decode data for use in a later software spectrometer or analysis pipeline.

3 Required Functionality

3.1 File Management

To make it easier for later analysis, some file management utilities should be included that will split or "clean up" files. This will not perform any analysis or decoding of the file contents beyond that required to move frames around.

- **Split file**: take in a file path (e.g., `/example.vdif`) and output new files for each thread or thread group (e.g., `/example_thread0.vdif`). This can be used to turn compound data streams into multiple simple data streams, or save later work in loading all threads from a file just to decode one.
- **Clean file**: take in a file path and output a new file that has been skew corrected–meaning frames have been sorted in order of seconds from epoch and then thread id–and inserted empty invalid frames for any time gaps.
- **Split and clean file**: perform split file and also clean each new file as it is being written to.

3.2 Data Summary

To report on the contents of a data stream, some utilities should be included that will do a quick parse of a file, extracting attributes from only frame headers.

- **Summarize file**

3.3 Data Dispatch

3.4 Data Decode

4 Technologies

Library is to be written in native C, in conformance with *at least* GNU90 ad C99. Dependencies will be kept to the C Standard Library. The library should target both OSX and Debian-based Linux.

5 API Design

Usage will centre around a struct type `DataStream`, which can be initialised in one of two modes: `FileMode` or `StreamMode`.

In `FileMode`, the expected interaction is that a user will initialise an `InputStream` using the `open_file()` function and passing a filepath to a valid VDIF or CODIF file.

```c
#include <stdio.h>
#include "vdifparse.h"

int main() {
    int res; // holds result status code for each operation

    // open new stream
    DataStream ds;
    open_file("file.vdif", &ds);

    // set format designator (saves summarising the file in advance)
    set_format_designator(&ds, "VDIF-1024-16-2-4");

    // select specific threads to include in output
    int selected_threads[2] = { 0, 1 };
    set_output_threads(&ds, selected_threads);

    // decode some samples
    float** output;
    DecodeMonitor statistics;
    int num_samples = 20000;
    decode_samples(&ds, num_samples, &output, &statistics);

    // free the memory allocated for the stream
    close(&ds);
```

```
    // (do something with output here)
}
```

In `StreamMode`, the expected interaction is that a user will initialise an (initially empty) `InputStream` using the `open_stream()` function and then push raw data into it by monitoring a specific port or piping from another process.

```
#include "vdifparse.h"

int main() {
    // open new stream (most values remain unset)
    DataStream ds = open_stream();


    close(in);
}
```

6 Style and Conventions

- Type names are in PascalCase (e.g., `DataStream`).
- Function and variable names are in snake_case (e.g., `sample_rate`).

7 Collected Prompts for Feedback

1. Nowadays, the most popular architectures (x86-64, IA-32, etc.) all use little-endianness. Can you think of a device or prospective used that would still require support for big-endianness?
2. The VDIF and CODIF specs allow for anywhere from 1-bit to 32-bit samples, but existing software only shows support for 1, 2, 4 or 8-bit decode. Should support be added for the full range of sample options?