A MarkIV format to vdif data converter

Des Small, July 2011

Introduction

The original EVN correlator at JIVE was (and is) a Mark 4 correlator, and it accepts data in the corresponding Mark IV data format. This format dates from the era in which magnetic tape was the only available storage medium for recording and playing back data at the speeds appropriate to VLBI, and as a result the original definiton is in terms of parallel tracks of bit-level data in which a given track records, for example, the sign or magnitude bit of each sample for a given channel.

With the advent of hard disks as the primary recording medium for VLBI the happy coincidence that maps tape tracks to channels became obscured. Data on disk is recorded in 32- or 64-bit chunks, and the individual tracks map to bits within these data structures and are not directly randomly accessible either on disk or in the CPU.

In fact, recovering the individual channels from these chunks – the so-called "corner-turning"[1] problem – is quite computationally demanding on modern CPUs, despite being conceptually trivial. Modern general-purpose CPUs are simply not designed to handle this problem well. Specialist hardware, or a suitably-programmed FPGA, could be used to accomplish this task much more efficiently, but special-purpose hardware is prohibitively expensive and general-purpose CPUs are (in comparison) very cheap. The Uniboard FPGA-based correlator being developed at JIVE to replace the existing Mark IV hardware correlator is tightly constrained in the number of logic gates available, and there is no expectation that the budget will extend to corner-turning now or in future iterations of the board.

In fact, although general-purpose CPUs are very far from optimal for the corner-turning problem they are in practice fast enough: it is anticipated that the corner-turning can be handled in real-time on the Mark5 data processing units, which (at the time of writing) are equipped with two Xeon 3.20GHz CPUs, each supporting two hyperthreads, per unit.

The vdif format

The vdif format is a new, flexible format for storing and transmitting VLBI data, first proposed in 2009. It supports a legacy mode that allows storage of Mark IV data (almost?) unmodified, but our interest here is in its use to store data in a form in which individual channels have been separated out. The format allows files to be created with one channel per file, or with all channels included as separate "threads" within a single file, as well as other possiblilities not required for this project.

The Uniboard-based correlator will use a subset of vdif as its input format, and this is therefore the targer format for the converter; the software correlator SFXC currently incorporates its own dechannelising code, but this is being extracted and merged with the code described here for use in eVLBI streaming operations and mixed mode streaming and recording. The SFXC correlator also accepts vdif input, and it was used for testing the output of the convertor. (The author is grateful to Aard Keimpema for his cheerful cooperation with the many iterations of this process.)

Development strategy

First a reference code was written in Erlang. This was not intended to be (and isn't) fast enough for

1 "A corner turn operation is defined as a copy of the object with a change in the storage order of the underlying data." http://www.ll.mit.edu/HPECchallenge/ct.html

production use, but the clarity of a functional programming language without mutable state makes it very pleasant to use for this kind of problem: the code looks almost like a mathematical specification of the problem. Also Erlang has an excellent syntax for working with data at the binary level as a result of its use for telecoms applications at Ericsson where it was developed.

Once this code was successfully tested a generic C version was developed, using the Erlang output as a gold standard for comparison. But, as the SFXC developers had noted previously, even generic C is not fast enough for real-time conversion of high-speed data; they had therefore implemented a version that produced machine-generated C code and linked it into the correlator. We did this too, except that our machine-generated code production was done more conveniently in Python.

The generic and machine-generated C versions of the program share as much code as possible, of course, including the code to process the files that specify the details of the conversion to be performed.

From Vex files to converter input

The vex file format defines the map from tracks (i.e., bits within a 32-bit or 64-bit integer) to components of channels. An example is given in Text 1 below.

```
$TRACKS;
*
def MKIV.8Ch2bit1to2;
* mode =  1    stations =Ef:Wb:Jb:On:Mc:Nt:Tr:Ur:Sh:Hh:Cm
*    format = MKIV1:2, and fan-out = 2
*    mode requires  8.00Mb/s/tr; stations using disks
     track_frame_format = Mark4;
     data_modulation = off;
     fanout_def =   : &CH01 : sign : 1:  2:  4;
     fanout_def =   : &CH01 :  mag : 1:  6:  8;
     ...
```

Text 1: Vex file specification of channels for N08C1

For the Erlang version of the program, this was manually transformed into Erlang syntax, from which it is easy to use Erlang tools to process it. The fragment above is translated into the form of Text 3 below. This code was written before the correlator control system for the next-generation EVN correlator was mature; now that it is, it would be straightforward to extract this data from the experiment database that encodes the same information as the Vex files.

```
def channels
CH01=04, 02
CH02=08, 06
CH03=12,10
CH04=16, 14
CH05=20, 18
CH06=24, 22
CH07=28, 26
CH08=32, 30
CH09=05, 03
CH10=09, 07
CH11=13, 11
CH12=17, 15
CH13=21, 19
CH14=25, 23
CH15=29, 27
CH16=33, 31
def streams
S1=CH01, CH02, CH03, CH04, CH05, CH06, CH07, CH08, CH09, CH10,
CH11, CH12, CH13, CH14, CH15, CH16
def targets
path=output/VDIF/f08l1/
```

Text 2: Complete C program configuration file for experiment F08L1

The input format for the C program is shown in Text 2 below; in addition to specifying which bits are included in a channel, it also defines output "streams"; that is to say, a set of channels which should be included as threads in the a single file of the vdif representation of the data. The C input file parser is extensible so that other options can be added. Note that the input format for the C program requires the sign and magnitude channels to be combined into a single list of bits – in experiment f08l1 sign and magnitude are each single bits for each sample. Again, this format is easy to generate from the experiment database, and it is designed so that it can also be fed into the standard input of the C program, so that the program can be spawned and configured by an Erlang process running within in the correlator control system.

Extensions and Further Work

A version of the offline code was developed for the JIVE space project to handle input in the VLBA data format. The code described here and that used in SFXC are currently being merged for use on the Mark5 data units by Harro Verkouter.