Modify GNSS-SDRLIB to Process Firehose IQ Data Files

Purpose

The purpose of this paper is to describe the modifications to GNSS-SDRLIB that are necessary to process IQ data files produced by the Firehose RF front end (software defined receiver). I'll start out showing the results of what you should see in GNSS-SDRLIB once the modifications are made, followed by how to set up the Firehose files and how to modify GNSS-SDRLIB.

The Firehose RF Front End

The Firehose is a GNSS RF front end and digitizer. It provides three 50 MHz RF channels (and two auxiliary baseband channels), supporting signals from GPS, GLONASS, Galileo, Beidou and SBAS. Signals are transmitted on a gigabit Ethernet link using two bits for I and two bits for Q; this format provides high-quality wideband signals to any downstream software receiver or recorder.

Running GNSS-SDRLIB

When the modifications discussed in this memo are applied to GNSS-SDRLIB, you should see results similar to what is shown in the figures below. I'm using a Linux PC running Ubuntu 20.04 LTS OS. Also, this paper assumes the user already has some familiarity with running GNSS-SDRLIB with a bladeRF (or rtlsdr) RF front end.

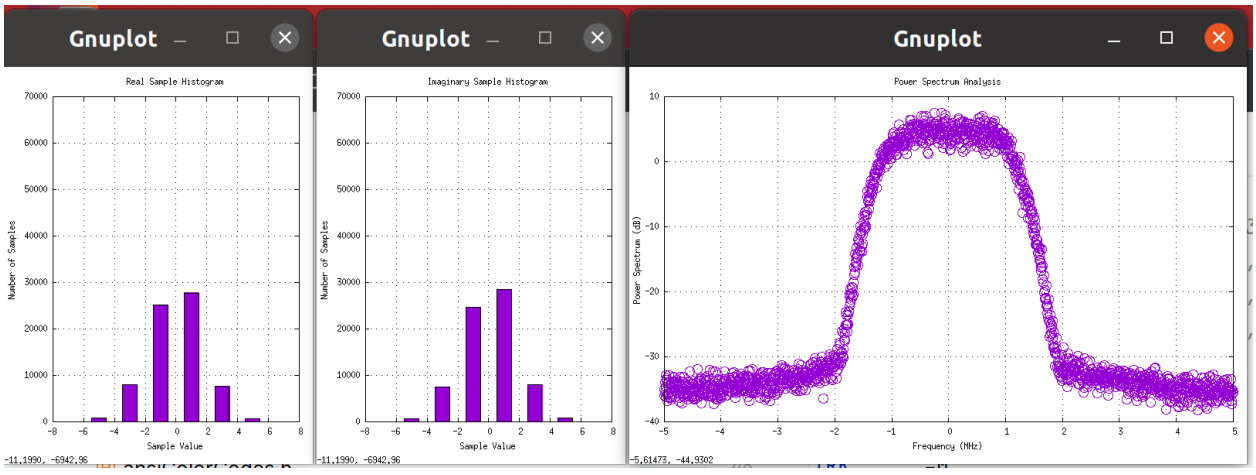

Figure 1. Histogram and Spectrum Plot for Firehose

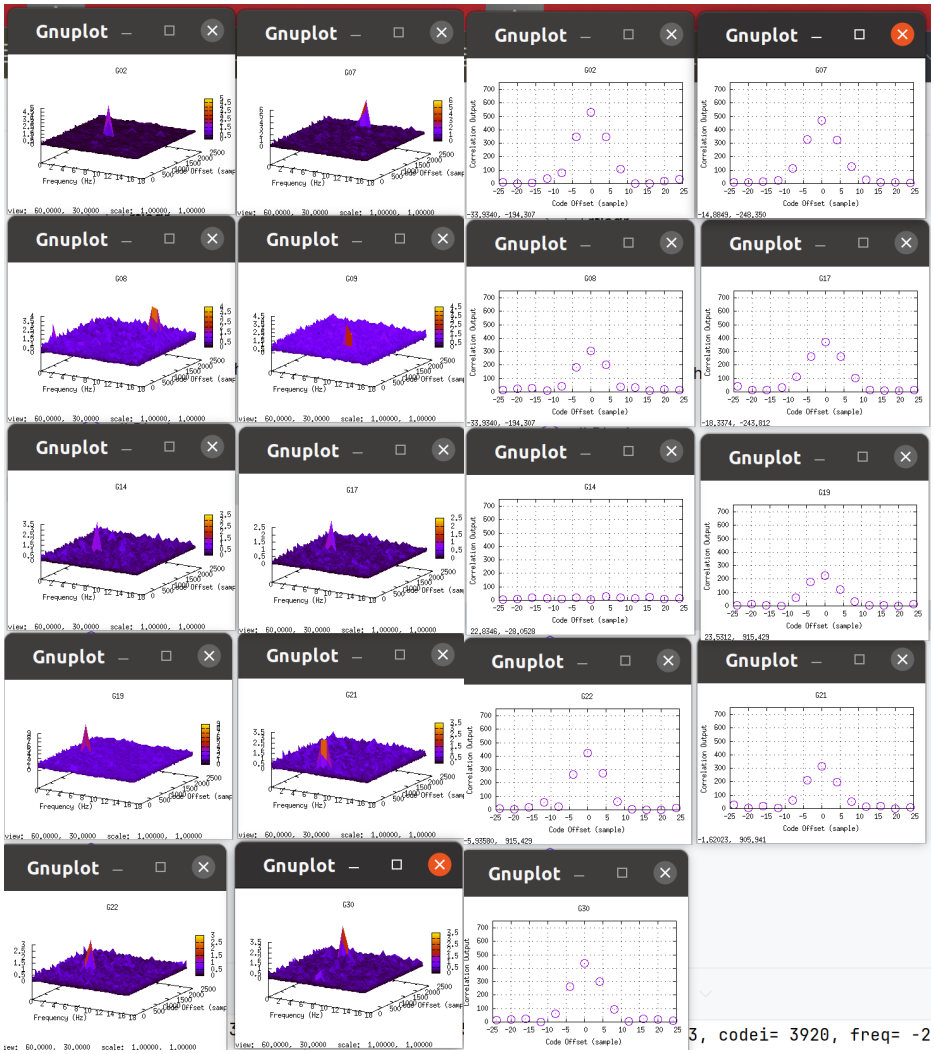

Figure 2. Acquisition and Tracking Plots

```
G05, C/N0=34.9, peak=1.1, codei= 3250, freq=  3200.0
G26, C/N0=35.2, peak=1.4, codei= 8664, freq=  3200.0
G18, C/N0=35.0, peak=1.2, codei= 1178, freq=  1600.0
G07 ID=1 tow:320646.0 week=2305 cnt=38665
G30 ID=1 tow:320646.0 week=2305 cnt=28321
G17 ID=1 tow:320646.0 week=2305 cnt=34168
G22 ID=1 tow:320646.0 week=2305 cnt=31925
G21 ID=1 tow:320646.0 week=2305 cnt=32373
G02 ID=1 tow:320646.0 week=2305 cnt=40915
G08 ID=1 tow:320646.0 week=2305 cnt=38227
G19 ID=1 tow:320646.0 week=2305 cnt=33279
G27, C/N0=35.2, peak=1.2, codei= 7618, freq=  5000.0
G09, C/N0=35.9, peak=1.4, codei= 7978, freq=  2000.0
G28, C/N0=35.3, peak=1.2, codei= 5702, freq= -5600.0
```

Figure 3. Navigation Data Decoded

Modify GNSS-SDRLIB to Process Firehose IQ Data Files

Steps for Preparing Firehose Files and For Modifying GNSS-SDRLIB

Step 1. Save the Firehose data in pcap format, then parse out L1C data to a binary file. These steps are accomplished using the Python scripts provided by the Firehose developer, Peter Monta, and found on his github repository.

To create a pcap file with ~90 seconds of data, use the following (replace en10 with applicable ethernet port) command in a Linux terminal:

```
sudo tcpdump -nn -i en10 -c 6930000 -B 100000 -w captureData.cap ether proto 0x88b5
```

To parse the pcap data into a L1C binary file, use the following Linux command:

```
./packet2wav_3ch 1 <captureData.pcap >firehose_dataL1.dat
```

Step 2. Run the Python script (provided in Appendix A) to modify the L1C binary data file parsed in Step1 so that it can be processed in GNSS-SDRLIB. This script performs three primary functions:

- Removes the carrier offset from the Firehose L1 signal
- Adjusts the range of the data to +/- 127 (int8)
- Down-samples the 70 Msps Firehose data to a more manageable 10 Msps. Since we are saving 2 bytes for each IQ sample-pair, you should get a 900 MB file is saving 45 seconds worth of 10 Msps data, as an example.

Step 3. Make several modifications to the GNSS-SDRLIB code.

1. Modify the fbladerf_pushtomem function (found in bladerf.c) so that it can read the 10 Msps Firehose file we created in Step 2. See Appendix B for the modifications.
2. In the startsdr function in sdrmain.c, disable the call to chk_initvalue. If used unmodified, it will fail the read of our modified Firehose file.
3. Change the scaling for the track plots if desired. To do this, I changed scale (in the initplotstruct function in sdrinint.c) from 5 to 30.

Step 4. Modify the bladerf ini file (bladefile.ini) to set the sampling frequency to 10 Msps, and point the FILE1 parameter to the location of the file that was created in Step 2.

Step 5. Compile and run GNSS-SDRLIB.

Appendix A. Python Script for Step 2

~/Desktop/ConvertFirehose/resampleFirehose_10M_noCO_Float32_V2.py (functions)

```python

import numpy as np
import gnsstools.nco as nco
import gnsstools.io as io
import scipy.signal
#import scipy.fftpack as fft

# Inputs
fileName = 'firehose_90s_dataL1.dat'
fileNameNew = 'firehose_45s_10M_L1.dat'
fileType = 2 # IQ
dataType = 'i1' # can also be int8
dataSize = 1 # bytes per I or Q reading
skipNumberOfBytes = 0
samplingFreq = 69984000
IF = -9334875
codeFreqBasis = 1023000
codeLength = 1023
samplesPerCode = int(np.round (samplingFreq / (codeFreqBasis / codeLength)))
numMs = 1000
resampleFs = 10e6
fs = samplingFreq
coffset = IF
fsr = resampleFs/fs
resamplesPerCode = 10000
longData = np.array([])
longData = longData.astype('float32')
numSec = 45 # Number of sec to process

# Open read and write files
print("Open read and write files")
fp = open(fileName, "rb")
fid2 = open(fileNameNew, 'ab')

for i in range(numSec):
    print("Iteration: ", i)

    # Read in int8 IQ samples
    print('Read initial samples from raw Firehose int8 IQ file')
    n = int(samplingFreq*0.001*numMs)
    x = io.get_samples_complex(fp, n)

    # Remove carrier offset
    print('Remove carrier offset')
    nco.mix(x,-coffset/fs,0)

    # resample to 10 MHz
    print('Resample to 10Msps')
    h = scipy.signal.firwin(161,1.5e6/(fs/2),window='hann')
    x = scipy.signal.filtfilt(h,[1],x)
    xr = np.interp((1/fsr)*np.arange(numMs*resamplesPerCode),np.arange(len(x)),np.real(x))
    xi = np.interp((1/fsr)*np.arange(numMs*resamplesPerCode),np.arange(len(x)),np.imag(x))
    x = xr+(1j)*xi

```

```
    # Scale data to +/- 127
    absXr = np.absolute(xr)
    maxXr = absXr.max()
    scaleFactor = 125/maxXr
    xr = xr * scaleFactor

    absXi = np.absolute(xi)
    maxXi = absXr.max()
    scaleFactor = 125/maxXi
    xi = xi * scaleFactor

    # print('Scaling float32 data to +/- 1.0')
    # dataFloatNorm = data / 3.0

    # Reformat x from complex128 to an IQ float32 array
    #print('Reformat to float32')
    #xr = xr.astype('float32')
    #xi = xi.astype('float32')
    print('Reformat to int8')
    xr = xr.astype('int8')
    xi = xi.astype('int8')
    xStacked = np.stack((xr, xi), axis=1)
    numIQvalues = int(fileType*numMs*resampleFs/1000)
    #xFloat32 = np.reshape(xStacked, numIQvalues)
    xInt8 = np.reshape(xStacked, numIQvalues)



    # Write to new bin file
    #print('Append float32 array to new file')
    #xFloat32.tofile(fid2)
    print('Append int8 array to new file')
    xInt8.tofile(fid2)

    # Append to data array
    #longData = np.append(longData, xFloat32)

# Close open files
print("Close opened files")
fp.close()
fid2.close()

# Read new bin file to make sure everything worked
print('Test the new file')
try:
    with open(fileNameNew, 'rb') as fid3:
        #dataNew = np.fromfile(fid3, 'f4')
        dataNew = np.fromfile(fid3, 'i1')

except IOError as e:
    print('Unable to read file')
fid3.close()

print('Program complete')
```

Appendix B. GNSS-SDRLIB Modifications

```
/* push data to memory buffer ------------------------------------------------
* push data to memory buffer from BladeRF binary IF file
* args   : none
* return : none
*-----------------------------------------------------------------------------*/
extern void fbladerf_pushtomembuf(void)
{
    size_t nread;
    //uint16_t buff[BLADERF_DATABUFF_SIZE*2];
    int8_t buff[BLADERF_DATABUFF_SIZE*2]; // 10Msps firehose
    int i,ind;

    mlock(hbuffmtx);

    //nread=fread(buff,sizeof(uint16_t),2*BLADERF_DATABUFF_SIZE,sdrini.fp1); //bladerf
    nread=fread(buff,sizeof(int8_t),2*BLADERF_DATABUFF_SIZE,sdrini.fp1); //firehose

    /* buffer index */
    ind=(sdrstat.buffcnt%MEMBUFFLEN)*2*BLADERF_DATABUFF_SIZE;

    for (i=0;i<nread;i++) {
        //sdrstat.buff[ind+i]=(uint8_t)((buff[i]>>4)+127.5); // bladerf
        sdrstat.buff[ind+i]=(uint8_t)(buff[i]+127); // firehose
    }

    unmlock(hbuffmtx);

    if (nread<2*BLADERF_DATABUFF_SIZE) {
        sdrstat.stopflag=ON;
        SDRPRINTF( format: "end of file!\n");
    }

    mlock(hreadmtx);
    sdrstat.buffcnt++;
    unmlock(hreadmtx);
}
```