```
////////////////////////////////////////////////////////////////////////////////
/////////
//
// ACQUISTION and Fine resolution frequency search for L1 band signal
// 92行目set_offset  138行目for文を変更する
// 変数は適当に付加してください
// ビルドにはfftw3.hが必要です（以下サイトで利用済みです）
// https://www.denshi.e.kaiyodai.ac.jp/kubo/sdr.html
////////////////////////////////////////////////////////////////////////////////
/////////

#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <math.h>
#include "global_extern.h"
#include "fftw3.h"

using namespace std;

const double cs = 299792458.0;//speed of light
const double pi = 3.1415926535898;//circular constant
const int SIZE = 409600 * 2 + 10;
const int SIZE_F = 120;

void makeCaTable();
void generateCAcode(int);
void makeL1CTable(int);
void generateL1Ccode(int);
void makeE1Table();
void generateE1code(int, char, int*);
void makeB1CTable(int);
void generateB1Ccode(int);
void makeB1ITable();
void generateB1Icode(int);

double F1 = 1575420000.0;
double data1, mean, sum = 0, out_1[SIZE][2] = { 0 };
double value, in6[4194306] = { 0 }, fftMax = 0.0, uniqFftPts = 0;
signed char signal[524288 * 2][16] = { 0 }, signal_Q[524288 * 2][16] = { 0 };

void acquisition(signed char data[]) //符号なし
{
        int i, j, k, prn, count;
        int samplesPerCode, numberOfFrqBins;
        int frequencyBinIndex, codePhase, stock_frequencyBinIndex = 0;
        int flag_absRes1 = 0, flag_absRes2 = 0;
        int samplesPerCodeChip, excludeRangeIndex1, excludeRangeIndex2;
        int codePhaseRange[SIZE] = { 0 };
        int codeValueIndex[SIZE * 10] = { 0 }, longCaCode[SIZE * 10] = { 0 };
        int fftNumPts = 0, fftMaxIndex = 0;
        int n_fine = 0;
        int type_flag = 1;//1:NCH 0:CH
        int N = settings.N;
        int summation, setsign[64] = { 0 };

        double ts;
        float frqBins[SIZE_F] = { 0 };
```

```
        double phasePoints[SIZE] = { 0 };
        float sinCarr[SIZE] = { 0 }, cosCarr[SIZE] = { 0 };
        float I1[SIZE] = { 0 }, Q1[SIZE] = { 0 }, I2[SIZE] = { 0 }, Q2[SIZE] = {
0 };
        float absRes1[SIZE][16] = { 0 }, correlation[SIZE][SIZE_F] = { 0 };
        float stock_absRes[SIZE] = { 0 };
        float peakSize = 0, store_peak = 0, secondPeakSize = 0;
        float signalIODC[524288 * 2] = { 0 }, xCarrier[524288 * 2] = { 0 };
        float dp_estimate[PRN_max] = { 0 };

        float absRes[41000][60][16] = { 0 };//メモリの関係で別に設定
        float absRes_L1C[SIZE][SIZE_F] = { 0 };//メモリの関係で別に設定

        int Integration = 10;//L1C/A(SBAS,L1S),B1I
        settings.acqSearchBand = 28;

        if (settings.frequency == 1 || settings.frequency == 3 ||
settings.frequency == 7) {//L1C E1 B1C
                Integration = 1;//コード1つ分
                settings.acqSearchBand = 56;
        }
        if (settings.frequency == 6) {//B1I
                F1 = 1561098000.0;
        }

printf("¥tprn¥tpeakSize/secondPeakSize¥tfrequencyBinIndex¥tcodePhase¥n");
        // Initialization //
        SATn = 0;
        for (i = 0; i <= 63; i++) {
                AcqResults_carrFreq[i] = 0;//周波数
                AcqResults_codePhase[i] = 0;//コード位相
                AcqResults_peakMetric[i] = 0;//ピークの高さ
        }
        //Find number of samples per spreading code
        data1 = (settings.samplingFreq / (settings.codeFreqBasis /
settings.codeLength));
        samplesPerCode = int(0.5 + data1);

        ///*
        ////////////////////////////////////////////////////////////
                //get IQ_rawdata
                //set offset 3->30-40ms 4->40-50ms
        int set_offset = 3;//いろいろ変更//SBAS 3,ca

        //read I signal
        for (j = Integration * set_offset + 1; j <= Integration * (set_offset +
1); j++) {
                for (i = 0; i < samplesPerCode; i++)
                        signal[i][j - Integration * set_offset] = data[2 * i +
(j - 1) * samplesPerCode * 2];
        }
        for (i = Integration * set_offset * samplesPerCode; i < Integration *
(set_offset + 1) * samplesPerCode; i++)
                sum = sum + data[2 * i];
        mean = sum / (Integration * samplesPerCode);
        for (i = 0 + Integration * set_offset * samplesPerCode; i < Integration
* (set_offset + 1) * samplesPerCode; i++)
                signalIODC[i - Integration * set_offset * samplesPerCode] =
```

```
data[2 * i] - mean;

        //read Q signal
        for (j = Integration * set_offset + 1; j <= Integration * (set_offset +
1); j++) {
                for (i = 0; i < samplesPerCode; i++)
                        signal_Q[i][j - Integration * set_offset] = data[2 * i +
1 + (j - 1) * samplesPerCode * 2];
        }

//for(i=Integration*set_offset*samplesPerCode;i<Integration*(set_offset+1)*sampl
esPerCode;i++)
        //      sum = sum+data[2*i+1];
        //mean = sum/(Integration*samplesPerCode);

//for(i=0+Integration*set_offset*samplesPerCode;i<Integration*(set_offset+1)*sam
plesPerCode;i++)
        //      signalIODC[i-Integration*set_offset*samplesPerCode] =
data[2*i+1]-mean;
////////////////////////////////////////////////////////////////
//*/
/*
////////////////////////////////////////////////////////////////
        //get rawdata
        //set offset 3->30-40ms 4->40-50ms
        int set_offset=0;
        for(j=Integration*set_offset+1;j<=Integration*(set_offset+1);j++){
                for(i=0;i<samplesPerCode;i++)
                        signal[i][j-Integration*set_offset] =
data[i+(j-1)*samplesPerCode];
        }

for(i=Integration*set_offset*samplesPerCode;i<Integration*(set_offset+1)*samples
PerCode;i++)
                sum = sum+data[i];
        mean = sum/(Integration*samplesPerCode);

for(i=0+Integration*set_offset*samplesPerCode;i<Integration*(set_offset+1)*sampl
esPerCode;i++)
                signalIODC[i-Integration*set_offset*samplesPerCode] =
data[i]-mean;
////////////////////////////////////////////////////////////////
*/
// Correlate signals //
// Using DLL (based on FFTW)
////////////////////////////////////////////////////////////////////////////////
///////
////////////////////////////////////////////////////////////////////////////////
///////

        for (prn = 1; prn <= 39; prn++)//1-36(L1C/A,E1) 1-10(L1C) 1-4(L1S)
1-39(L1 SBAS) 19-45(B1C) 1-63(B1I)
        {
                if (settings.frequency == 0) {
                        //switch(prn){//L1S用
                        //case  1:  break;
                        //case  2:  break;
                        //case  3:  break;
```

```
        //case  4:  break;
        //default:continue; break;
        //}
}
if (settings.frequency == 1) {//for L1C
                makeL1CTable(prn);
}
if (settings.frequency == 6) {//for B1I
        switch (prn) {//SDR contest B1I 24機
        case   1:  break;
        case   2:  break;
        case   3:  break;
        case   4:  break;
        case   6:  break;
        case   8:  break;
        case   9:  break;
        case  13:  break;
        case  14:  break;
        case  16:  break;
        case  23:  break;
        case  24:  break;
        case  25:  break;
        case  27:  break;
        case  28:  break;
        case  32:  break;
        case  33:  break;
        case  38:  break;
        case  39:  break;
        case  41:  break;
        case  58:  break;
        case  59:  break;
        case  60:  break;
        case  62:  break;
        default:continue; break;
        }
}
if (settings.frequency == 7) {//for B1C
        makeB1CTable(prn);
        switch (prn) {//SDR contest B1C 12機
        case  23:  break;
        case  24:  break;
        case  25:  break;
        case  27:  break;
        case  28:  break;
        case  30:  break;
        case  32:  break;
        case  33:  break;
        case  38:  break;
        case  39:  break;
        case  41:  break;
        case  42:  break;
        default:continue; break;
        }
}

Time_itg[prn] = Integration;
Acqflag[prn] = type_flag;
```

```
                //Number of the frequency bins for the given acquisition band
(500Hz steps)
                numberOfFrqBins = int(settings.acqSearchBand) * 2 + 1;

                ////////////////////////////////////
                //calculate ts again
                        //Find sampling period
                ts = 1.0 / settings.samplingFreq;

                //Find phase points of the local carrier wave
                for (i = 0; i < samplesPerCode; i++) {
                        phasePoints[i] = i * 2.0 * pi * ts;
                }
                ////////////////////////////////////////////
                for (count = 1; count <= Integration; count++) {

                        fftw_complex* out;
                        double in[SIZE];
                        fftw_plan p;

                        double convCodeIQ1[SIZE][2] = { 0 },
convCodeIQ2[SIZE][2] = { 0 };
                        out = (fftw_complex*)fftw_malloc(sizeof(fftw_complex) *
N);

                        if (settings.frequency == 0) {//L1-C/A
                                for (i = 0; i < N; i++) {
                                        in[i] = double(caCodesTable[prn -
1][i]);
                                }
                        }
                        if (settings.frequency == 1||settings.frequency==7)
{//L1C B1C
                                for (i = 0; i < N; i++) {
                                        in[i] = double(cCodesTable[prn - 1][i]);
                                }
                        }
                        if (settings.frequency == 3) {//E1
                                for (i = 0; i < N; i++) {
                                        in[i] = double(E1BCodesTable[prn -
1][i]);
                                }
                        }
                        if (settings.frequency == 6) {//E1
                                for (i = 0; i < N; i++) {
                                        in[i] = double(B1ICodesTable[prn -
1][i]);
                                }
                        }
                        p = fftw_plan_dft_r2c_1d(N, in, out, FFTW_ESTIMATE);
                        fftw_execute(p); // repeat as needed
                        fftw_destroy_plan(p);

                        for (j = 0; j < N; j++) {
                                out_1[j][0] = out[j][0];
                                out_1[j][1] = out[j][1];
                        }
                        for (i = N / 2 + 1; i < N; i++) {//remaining half
```

```
                        out_1[i][0] = out_1[N - i][0];
                        out_1[i][1] = out_1[N - i][1];
                }
                for (i = 0; i < N / 2 + 1; i++) {//sign inversion
                        out_1[i][1] = -1.0 * out_1[i][1];
                }
                fftw_free(out);

                store_peak = 0;// important because we have to calculate
several SVs
                //--- Make the correlation for whole frequency band (for
all freq. bins)

////////////////////////////////////////////////////////////////////////////
////////

////////////////////////////////////////////////////////////////////////////
////////
                for (i = 1; i <= numberOfFrqBins; i++) {
                        //--- Generate carrier wave frequency grid
(???kHz step) -----------
                                frqBins[i - 1] = settings.IF -
(settings.acqSearchBand / 2) * 1000 / (settings.acqSearchBand / 14) +
                                        (i - 1) * 500 /
(settings.acqSearchBand / 14);

                        //--- Generate local sine and cosine
------------------------------
                        for (j = 0; j < N; j++) {
                                sinCarr[j] = sin(frqBins[i - 1] *
phasePoints[j]);
                                cosCarr[j] = cos(frqBins[i - 1] *
phasePoints[j]);
                        }

                        //--- "Remove carrier" from the signal
----------------------------
//                      for(j=0;j<N;j++){
//                              I1[j] = sinCarr[j] * signal[j][count];
//                              Q1[j] = cosCarr[j] * signal[j][count];
//                      }

//                      for(j=0;j<N;j++){
//                              I1[j] = sinCarr[j] * signal[j][count];
//                              Q1[j] = sinCarr[j] * signal_Q[j][count];
//                      }

                        //--- "Remove carrier" from the signal
----------------------------
                        for (j = 0; j < N; j++) {
                                I1[j] = sinCarr[j] * signal[j][count] -
cosCarr[j] * signal_Q[j][count];
                                Q1[j] = cosCarr[j] * signal[j][count] +
sinCarr[j] * signal_Q[j][count];
                        }

                        //--- Convert the baseband signal to frequency
```

```
domain --------------
                                //for I1 Q1: IQfreqDom1 = fft(I1 + j*Q1)
                                fftw_complex* in2, * out2;
                                in2 =
(fftw_complex*)fftw_malloc(sizeof(fftw_complex) * N);
                                out2 =
(fftw_complex*)fftw_malloc(sizeof(fftw_complex) * N);
                                for (j = 0; j < N; j++) {
                                        in2[j][0] = I1[j];
                                        in2[j][1] = Q1[j];
                                }
                                p = fftw_plan_dft_1d(N, in2, out2, FFTW_FORWARD,
FFTW_ESTIMATE);
                                fftw_execute(p);
                                fftw_destroy_plan(p);
                                fftw_free(in2);

                                //--- Multiplication in the frequency domain
(correlation in time domain)
                                for (j = 0; j < N; j++) {
                                        convCodeIQ1[j][0] = out2[j][0] *
out_1[j][0] - out2[j][1] * (1.0 * out_1[j][1]);
                                        convCodeIQ1[j][1] = out2[j][1] *
out_1[j][0] + out2[j][0] * (1.0 * out_1[j][1]);
                                }
                                fftw_free(out2);//fftw_free(out3);

                                //--- Perform inverse DFT and store correlation
results ------------
                                //for convCodeIQ1: acqRes1 =
abs(ifft(convCodeIQ1)) .^ 2
                                fftw_complex* in4, * out4;
                                in4 =
(fftw_complex*)fftw_malloc(sizeof(fftw_complex) * N);
                                out4 =
(fftw_complex*)fftw_malloc(sizeof(fftw_complex) * N);
                                for (j = 0; j < N; j++) {
                                        in4[j][0] = convCodeIQ1[j][0];
                                        in4[j][1] = convCodeIQ1[j][1];
                                }
                                p = fftw_plan_dft_1d(N, in4, out4,
FFTW_BACKWARD, FFTW_ESTIMATE);
                                fftw_execute(p);
                                fftw_destroy_plan(p);
                                fftw_free(in4);

                                for (j = 0; j < N; j++) {//divided by N
                                        out4[j][0] = out4[j][0] / N;
                                        out4[j][1] = out4[j][1] / N;
                                }

                                //Non-Coherent
                                if (settings.frequency == 0 ||
settings.frequency == 6) {
                                        for (j = 0; j < N; j++) {
                                                absRes1[j][count] =
pow(sqrt(out4[j][0] * out4[j][0] + out4[j][1] * out4[j][1]), 2.0);
                                                absRes[j][i][count] =
```

```
absRes1[j][count];
                                        }
                                }
                                if (settings.frequency == 1 ||
settings.frequency == 3 ||settings.frequency==7) {
                                        for (j = 0; j < N; j++) {
                                                absRes1[j][count] =
pow(sqrt(out4[j][0] * out4[j][0] + out4[j][1] * out4[j][1]), 2.0);
                                                absRes_L1C[j][i] =
absRes1[j][count];
                                        }
                                }
                                fftw_free(out4);//fftw_free(out5);

                        }//numberOfFrqBins : i
                }//Integration : count
                cout << "....";
                cout << "  ";

                //Non-Coherent
                if (settings.frequency == 0 || settings.frequency == 6) {
                        for (count = 1; count <= Integration; count++) {
                                for (i = 1; i <= numberOfFrqBins; i++) {
                                        for (j = 0; j < N; j++) {
                                                correlation[j][i] =
correlation[j][i] + absRes[j][i][count];
                                        }
                                }
                        }
                }
                if (settings.frequency == 1 || settings.frequency ==
3||settings.frequency==7) {
                        for (i = 1; i <= numberOfFrqBins; i++) {
                                for (j = 0; j < N; j++) {
                                        correlation[j][i] = correlation[j][i] +
absRes_L1C[j][i];
                                }
                        }
                }

////////////////////////////////////////////////////////////////////////////////
///////

////////////////////////////////////////////////////////////////////////////////
///////
                //Look for correlation peak
                for (i = 1; i <= numberOfFrqBins; i++) {

                        for (j = 0; j < N; j++) {
                                if (correlation[j][i] >= store_peak) {
                                        peakSize = correlation[j][i];
                                        store_peak = peakSize;
                                        frequencyBinIndex = i;
                                        codePhase = j + 1;
                                        flag_absRes1 = j;
                                }
                        }
                        stock_frequencyBinIndex = frequencyBinIndex;
```

```
                }//numberOfFrqBins

                //--- Find 1 chip wide C/A code phase exclude range around the
peak ----
                samplesPerCodeChip = int(0.5 + (settings.samplingFreq /
settings.codeFreqBasis));
                excludeRangeIndex1 = codePhase - samplesPerCodeChip;
                excludeRangeIndex2 = codePhase + samplesPerCodeChip;

                //--- Correct C/A code phase exclude range if the range includes
array boundaries
                int num = 0;
                if (excludeRangeIndex1 < 1) {
                        for (k = excludeRangeIndex2; k <= samplesPerCode +
excludeRangeIndex1; k++)
                                codePhaseRange[num++] = k;
                }
                else if (excludeRangeIndex2 > samplesPerCode) {
                        for (k = excludeRangeIndex2 - samplesPerCode; k <=
excludeRangeIndex1; k++)
                                codePhaseRange[num++] = k;
                }
                else {
                        for (k = 1; k <= excludeRangeIndex1; k++)
                                codePhaseRange[num++] = k;
                        for (k = excludeRangeIndex2; k <= samplesPerCode; k++)
                                codePhaseRange[num++] = k;
                }

                //--- Find the second highest correlation peak in the same freq.
bin ---
                store_peak = 0;
                for (j = 0; j < N - samplesPerCodeChip - samplesPerCodeChip + 1;
j++) {
                        if (correlation[codePhaseRange[j + 1] -
1][frequencyBinIndex] >= store_peak) {
                                secondPeakSize = correlation[codePhaseRange[j +
1] - 1][frequencyBinIndex];
                                store_peak = secondPeakSize;
                        }
                }
                printf("¥t%d,¥t%f,¥t¥t%d,¥t¥t¥t%d¥n",
                        prn, peakSize / secondPeakSize, frequencyBinIndex,
codePhase);

                if (frequencyBinIndex >= settings.acqSearchBand + 1)
                        setsign[prn] = -1;
                else
                        setsign[prn] = +1;

                for (j = 0; j < N; j++) {
                        Acquisition1[prn][j] =
correlation[j][frequencyBinIndex];
                }

////////////////////////////////////////////////////////////////////////////
////////////////
```

```
////////////////////////////////////////////////////////////////////////////////
////////////////
                                //If the result is above threshold, then there
is a signal ...
                if (peakSize / secondPeakSize > settings.acqThreshold) {

                                // Fine resolution frequency search //
                                fftw_plan pp;//re definition
                                if (settings.frequency == 0) {
                                        summation = 10;//10ms分のコードでFFT
                                        //--- Generate 10msec long C/A codes
sequence for given PRN --------
                                        for (k = 1; k <= summation *
samplesPerCode; k++) {
                                                value = floor((ts * k) / (1.0 /
settings.codeFreqBasis));
                                                codeValueIndex[k - 1] =
int(value);
                                        }
                                        generateCAcode(prn);
                                        for (k = 0; k < summation *
samplesPerCode; k++) {
                                                value = codeValueIndex[k] % 1023
+ 1;
                                                longCaCode[k] =
CAcode[int(value) - 1];
                                        }
                                }
                                if (settings.frequency == 1) {
                                        summation = 1;//もともと10ms分
                                        //--- Generate 10msec long L1C codes
sequence for given PRN --------
                                        for (k = 1; k <= summation *
samplesPerCode; k++) {
                                                value = floor((ts * k) / (1.0 /
settings.codeFreqBasis));
                                                codeValueIndex[k - 1] =
int(value);
                                        }
                                        generateL1Ccode(prn);
                                        for (k = 0; k < summation *
samplesPerCode; k++) {
                                                value = codeValueIndex[k]%1023 +
1;
                                                //longCaCode[k] =
cCodesTable[prn - 1][k];
                                                longCaCode[k] = Ccode[int(value)
- 1];
                                        }
                                }
                                if (settings.frequency == 3) {
                                        summation = 1;
                                        //--- Generate 4msec long E1B codes
sequence for given PRN --------
                                        for (k = 1; k <= summation *
samplesPerCode; k++) {
                                                value = floor((ts * k) / (1.0 /
```

```
settings.codeFreqBasis));
                                        codeValueIndex[k - 1] =
int(value);
                                }
                                for (k = 0; k < summation *
samplesPerCode; k++) {
                                        //value = codeValueIndex[k]%4092
+ 1;
                                        longCaCode[k] =
E1BCodesTable[prn - 1][k];
                                }
                        }
                        if (settings.frequency == 6) {
                                summation = 10;//10ms分のコードでFFT
                                //--- Generate 10msec long B1C codes
sequence for given PRN --------
                                for (k = 1; k <= summation *
samplesPerCode; k++) {
                                        value = floor((ts * k) / (1.0 /
settings.codeFreqBasis));
                                        codeValueIndex[k - 1] =
int(value);
                                }
                                for (k = 0; k < summation *
samplesPerCode; k++) {
                                        //value = codeValueIndex[k] %
2046 + 1;
                                        longCaCode[k] =
B1ICodesTable[prn - 1][k];
                                }
                        }
                        if (settings.frequency == 7) {
                                summation = 1;//もともと10ms分
                    //--- Generate 10msec long B1C codes sequence for given
PRN --------
                                for (k = 1; k <= summation *
samplesPerCode; k++) {
                                        value = floor((ts * k) / (1.0 /
settings.codeFreqBasis));
                                        codeValueIndex[k - 1] =
int(value);
                                }
                                for (k = 0; k < summation *
samplesPerCode; k++) {
                                        longCaCode[k] = cCodesTable[prn
- 1][k];
                                }
                        }
                        //Remove C/A code modulation from the original
signal (Using detected C/A code phase)
                        j = 0;
                        for (k = codePhase; k <= codePhase + summation *
samplesPerCode - 1; k++) {
                                xCarrier[j++] = signalIODC[k - 1] *
longCaCode[j];
                        }

                        //--- Find the next highest power of two and
```

```
increase by 8x --------
                                int length = summation * samplesPerCode;
                                fftNumPts = 0;
                                value = length;
                                while (value >= 1.0) {
                                        value = value / 2.0;
                                        fftNumPts++;
                                }
                                fftNumPts = int(8 * pow(2.0, fftNumPts));

                                // Compute the magnitude of the FFT, find
maximum and the associated carrier frequency
                                fftw_complex* out6;
                                n_fine = fftNumPts;
                                out6 =
(fftw_complex*)fftw_malloc(sizeof(fftw_complex) * n_fine);
                                for (i = 0; i < samplesPerCode * summation; i++)
{
                                        in6[i] = xCarrier[i];
                                }
                                for (i = samplesPerCode * summation; i < n_fine;
i++) {
                                        in6[i] = 0;
                                }
                                pp = fftw_plan_dft_r2c_1d(n_fine, in6, out6,
FFTW_ESTIMATE);
                                fftw_execute(pp); // repeat as needed
                                fftw_destroy_plan(pp);

                                for (i = 0; i < n_fine; i++) {
                                        out6[i][0] = sqrt(out6[i][0] *
out6[i][0] + out6[i][1] * out6[i][1]);
                                }

                                uniqFftPts = ceil((double(fftNumPts) + 1) / 2);
                                //                      uniqFftPts =
ceil((double(fftNumPts) + 1));
                                fftMax = 0.0;
                                for (i = 4; i<int(uniqFftPts) - 5 - 1; i++) {
                                        if (out6[i][0] >= fftMax) {
                                                fftMax = out6[i][0];
                                                int F = 7000 /
settings.acqSearchBand;//追加
                                                fftMaxIndex = i % F;//サーチ幅
[Hz]で割ってみる
                                                //fftMaxIndex = i;
                                        }
                                }
                                //fftFreqBins = (0 : uniqFftPts-1) *
settings.samplingFreq/fftNumPts;
                                for (i = 0; i < uniqFftPts; i++) {
                                        in6[i] = double(i) *
settings.samplingFreq / double(fftNumPts);
                                }

                                //--- Save properties of the detected satellite
signal -------------
                                AcqResults_carrFreq[prn - 1] =
```

```
in6[fftMaxIndex]*setsign[prn];
                                //AcqResults_carrFreq[prn-1]  =
frqBins[frequencyBinIndex-1];
                                AcqResults_codePhase[prn - 1] = codePhase;
                                AcqResults_peakMetric[prn - 1] = peakSize /
secondPeakSize;
                                SVn[SATn] = prn;
                                SATn++;

                                fftw_free(out6);//fftw_free(out6);

                }//peak/second_peak is OK
////////////////////////////////////////////////////////////////////////////
/////////////////
////////////////////////////////////////////////////////////////////////////
/////////////////

                for (j = 0; j < N; j++) {
                        for (i = 1; i <= numberOfFrqBins; i++) {
                                correlation[j][i] = 0;
                        }
                }
                peakSize = 0;
                secondPeakSize = 0;
                store_peak = 0;
        }

        cout.precision(10);
        cout << "Acquistion Finish!!!" << endl;
        cout << "¥tDoppler Ferquency" << endl;
        for (i = 0; i < SATn; i++) {
                cout << SVn[i] << " |¥t" << AcqResults_carrFreq[SVn[i] - 1] -
settings.IF << endl;
        }

        //acquisition_1
        for (i = 0; i < SATn; i++) {
                Doppler[i] = AcqResults_carrFreq[SVn[i] - 1] - settings.IF;
                fprintf(fp_mysoft[0], "%d,%f,%f,%f,%d,%f¥n", SVn[i], Doppler[i],
                        AcqResults_codePhase[SVn[i] - 1],
AcqResults_peakMetric[SVn[i] - 1], Acqflag[SVn[i]], Time_itg[SVn[i]]);
        }

        //acquisition1_2
        fprintf(fp_mysoft[1], " ");
        for (i = 0; i < SATn; i++) {
                fprintf(fp_mysoft[1], ",%d", SVn[i]);
        }
        fprintf(fp_mysoft[1], "¥n");
        for (j = 0; j < N; j++) {
                fprintf(fp_mysoft[1], "%d", j);
                for (k = 0; k < SATn; k++) {
                        fprintf(fp_mysoft[1], ",%f", Acquisition1[SVn[k]][j]);
                }
                fprintf(fp_mysoft[1], "¥n");
        }
        fprintf(fp_mysoft[1], "¥n");
```

```
        fclose(fp_mysoft[0]);
        fclose(fp_mysoft[1]);

        //      exit(0);
}
```