Wireless Lab - Mini Project Report

NavIC Receiver Implementation using RTL-SDR

Under the guidance of :

Dr. Swetha N Shah

Submitted by :

M UDHAY KUMAR

(P20EC022)

NavIC Receiver with RTLSDR

Abstract

Since ancient times, mankind has tried to find its bearings by using milestones and stars. A new era has begun, however, thanks to satellite communication providing precise timing and location information in real time. To take advantage of current and future demands of general public in the Indian Subcontinent region along with providing self reliant Navigation system for Indian Defence Operations, ISRO has developed the Indian Regional Navigational Satellite System (IRNSS) commonly termed as NavIC, making the induction of new receivers and these processing devices need to be developed.

Traditional GNSS/RNSS receivers use hardware based digital signal processing. This approach usually provides high performance with the advantage of low power consumption, although the flexibility of the receiver functionality is limited. By using open architectures, component replacement can occur more easily than in a closed system, significantly reducing total ownership costs in the future. Through software defined based Radio receivers, however, the whole signal processing is defined in software. By using such a system, the receiver can be reconfigured depending on the application, providing the receiver with enhanced adaptive capabilities.

This work focusses on the the implementation IRNSS NavIC system on RTL-SDR which is cheap and famous among hobbyists and DIY communities.This system works on NavIC L1 band and generates the Ephermis Data from Satellites, which contains all the key information ranging from Time of Transmission data(TOW), Satellite Location , and other crutial data of SLV, which is key for obtaining User Location or Getting accurate Timing of Atomic clocks precision.

The design approach is presented in the rest of document and all the hardware and software developed for this work is explained in detail. The system validation was successfully performed in the field with real environment constraints, proving its capabilities.

1. GNSS/RNSS Fundamentals

1.1 Introduction

In order to design a software-defined single frequency NavIC receiver, firstly the fundamentals of satellite navigation and its principles must be analysed. This section deals with the systems overview, also the characteristics of the signal and data transmitted from the satellites and received by the Radio Frequency (RF) front-end. These characteristics will impose steps to take in the course of the system development.

1.2 Satellite navigation principles

Satellite navigation systems are comprised of three functional segments: The space segment, corresponding to the operating satellites constellation; the control segment, corresponding to the ground stations involved in the monitoring of the system; and the user segment, represented by the receivers for the civilian and military use.

All satellite navigation systems determine coordinates using the same basic principles. Consider the simple model shown in below Figure, where a transmitter and a receiver with synchronized clocks are located in a one dimensional plane. Knowing that a radio wave propagates at the speed of light of around 300000 km/s, by measuring the delay from when the signal is transmitted to when it is received, it is possible to assess the receiver distance from the transmitter. However, there can be a discrepancy between the clocks of the transmitter and the receiver, leading to a large error in the calculated distance.

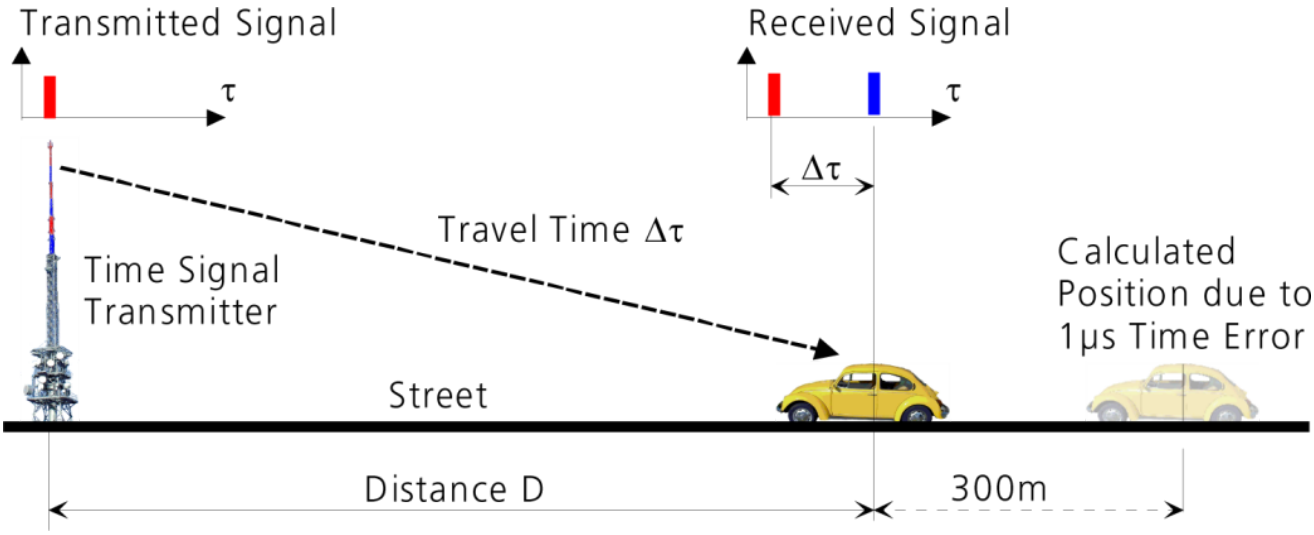


Figure 1.1 – 1D simple model navigation principle

To overcome the local clock synchronization discrepancy, a second synchronized time signal transmitter where the separation to the first transmitter is known, can be used to outfit the exact same clocks, as shown in Figure below.

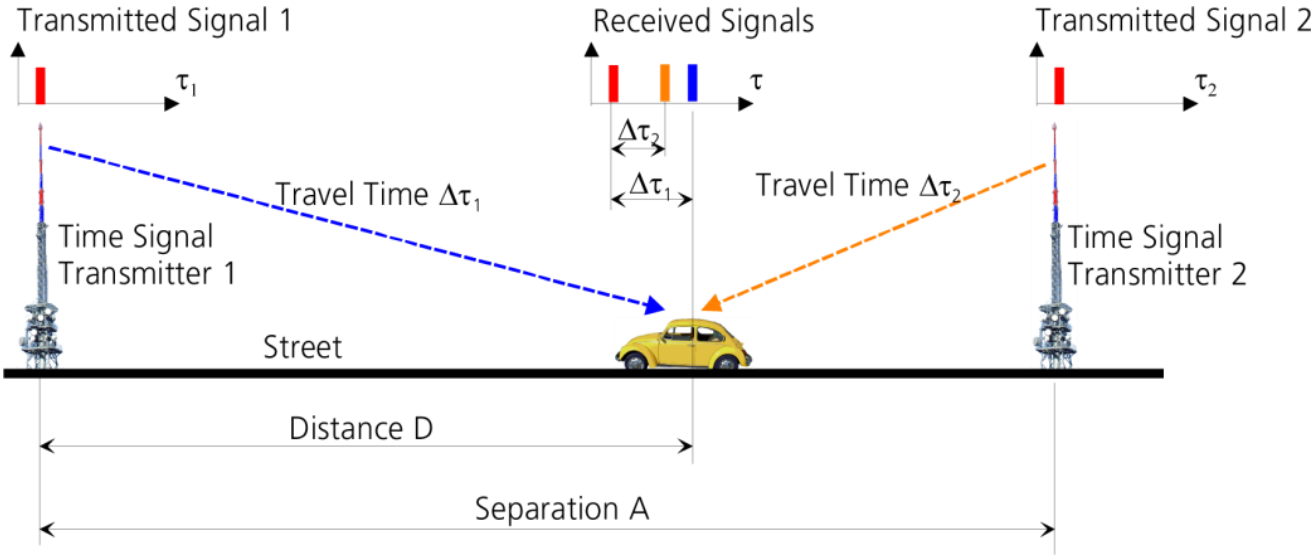


Figure 1.2 – 1D simple model with two transmitter

From this model a conclusion can be drawn: in order to exactly determine the position and time along a one dimension plane, at least two time signal transmitters are required. Now, this conclusion can be extrapolated to more dimensions. It is then possible to say that when an unsynchronized receiver clock is employed in calculating position, it is necessary that the number of time signal transmitters exceed the number of unknown dimensions by a value of one.

Satellite navigation systems use satellites as time-signal transmitters. In order for a GNSS receiver to determine its position, it must receive time signals from at least four separate satellites, represented in Figure 2.3, to calculate the signal travel times Δt_1 to Δt_4

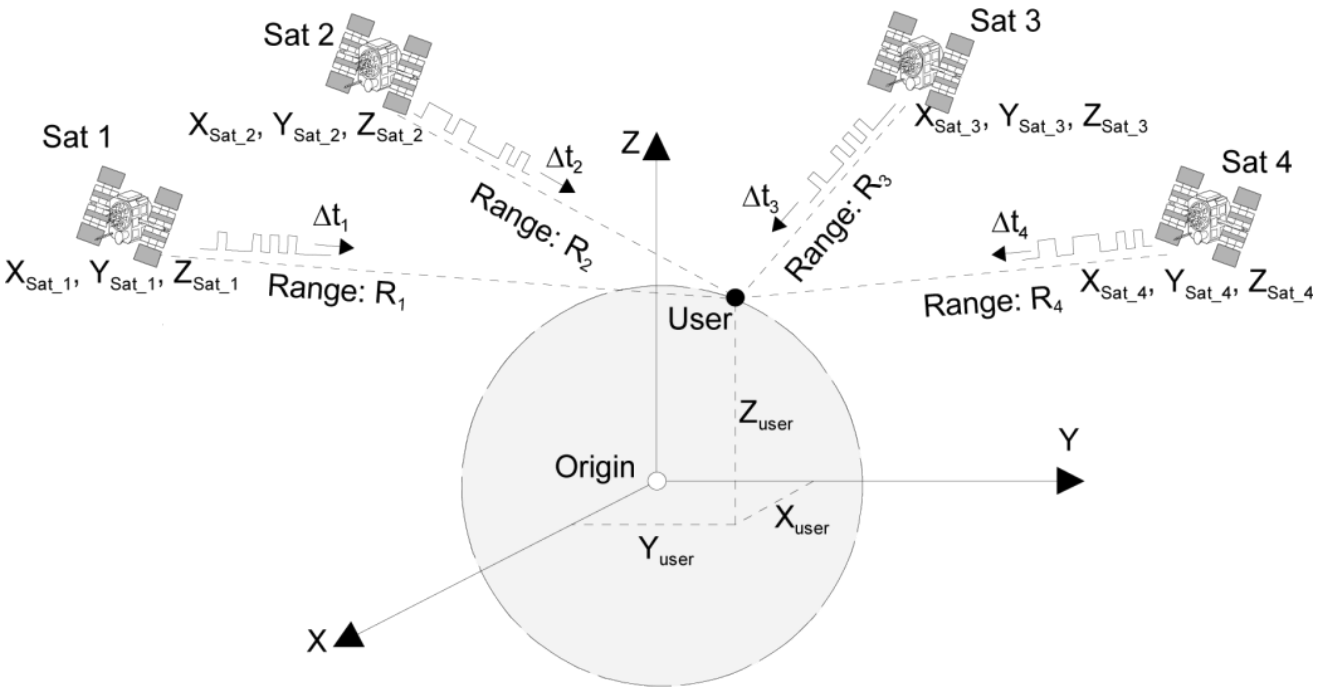


Figure 1.3 – Four satellites are required to determine a position in 3-D space.

Calculations are effected in a Cartesian, three-dimensional coordinate system with a geocentric origin. The range of the user from each of the four satellites, R_1 to R_4, can be determined with the help of the calculated signal travel times between the satellites and the receiver. As the locations X Sat , Y Sat and Z Sat of the four satellites are known, the user coordinates can be calculated.

2. NavIC/IRNSS System Overview

2.1 IRNSS FREQUENCY BANDS

The IRNSS SPS(Standard Positioning System) service is transmitted on L5 (1164.45 – 1188.45 MHz) and S (2483.5-2500 MHz) bands. The frequency in L5 band has been selected in the allocated spectrum of Radio Navigation Satellite Services as indicated in Figure below.

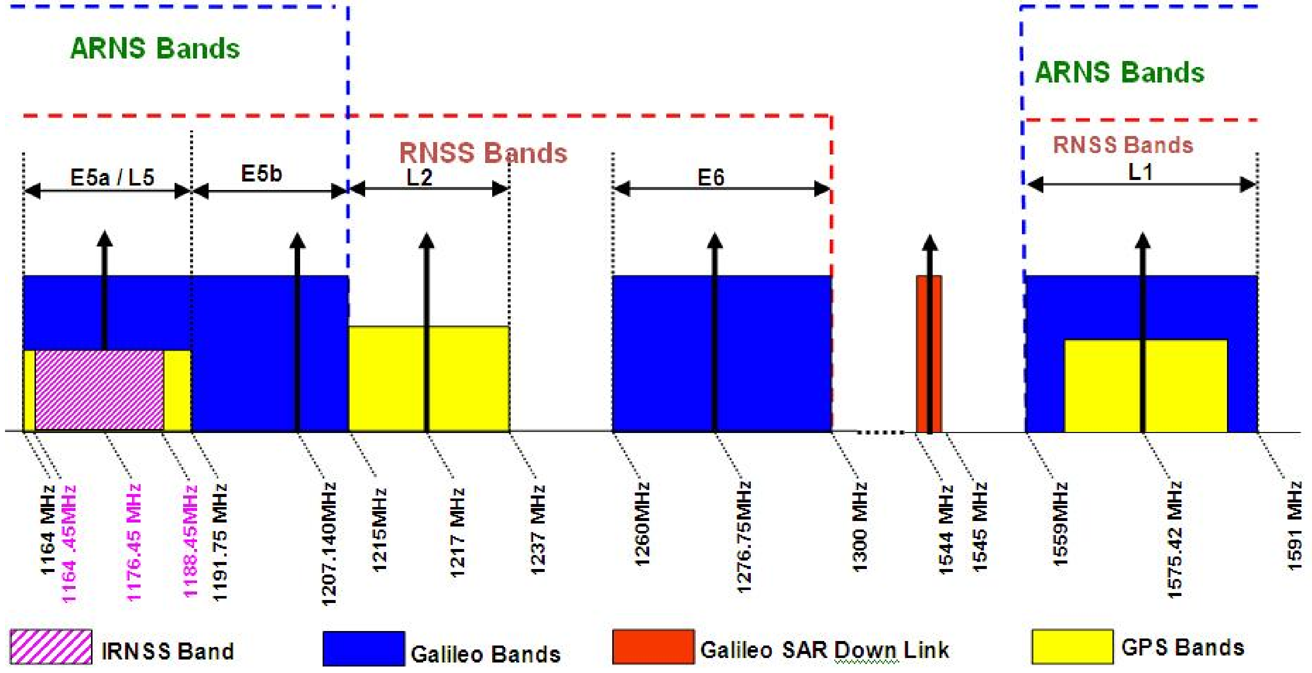


Figure 2.1 : Spectrum for Radio Navigation Satellite Services in L Band

The Below table shows the L5 and S band carrier frequecies and Bandwidth.

Signal	**Carrier Frequency**	**Bandwidth**
SPS – L5	1176.45 MHz	24 MHz (1164.45 -1188.45 MHz)
SPS – S	2492.028 MHz	16.5MHz (2483.50 – 2500.00MHz)

2.2 MODULATION SCHEME

Standard Positioning Service :

The SPS signal is BPSK(1) modulated on L5 and S bands. The navigation data at data rate of 50 sps (1/2 rate FEC encoded) is modulo 2 added to PRN code chipped at 1.023 Mcps identified for SPS service. The CDMA modulated code, modulates the L5 and S carriers at 1176.45MHz and 2492.028 MHz respectively.

The mathematical description of the SPS signal is given by:

$$s_{sps}(t) = \sum_{i=-\infty}^{\infty} c_{sps}(|i|L_{sps}).\, d_{sps}([i]_{CD\_sps}).\, rect_{Tc,sps}(t - iT_{c,sps}) \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (1)$$

Polarization Charecteristics :

All the IRNSS signals are Right Hand Circularly Polarized. The antenna axial ratio does not exceed 2.0 dB.

SPS Code Generation :

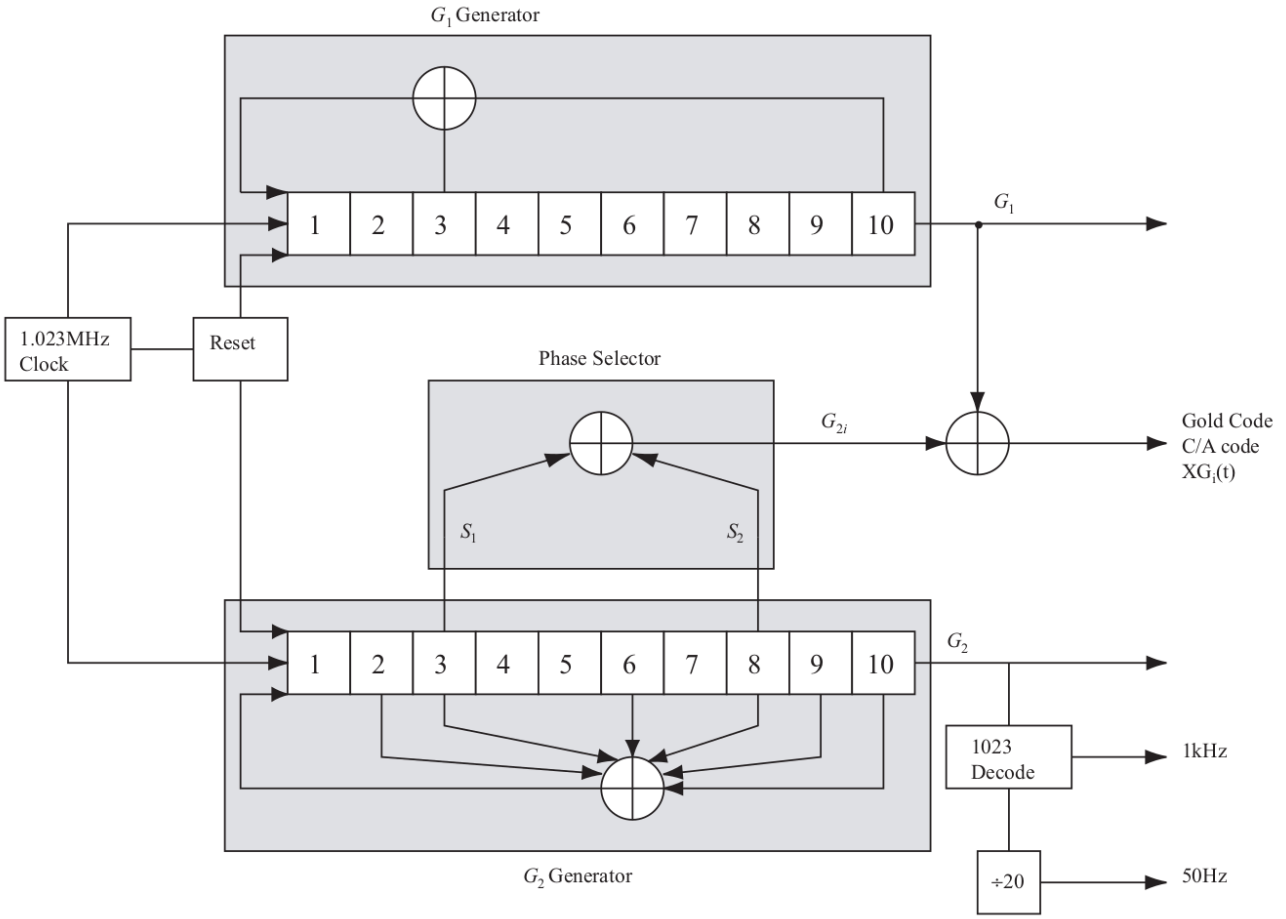
For SPS code generation, the two polynomials G1 and G2 are as defined below:

$$G1 : X10 + X3 + 1$$
$$G2 : X10 + X9 + X8 + X6 + X3 + X2 + 1$$

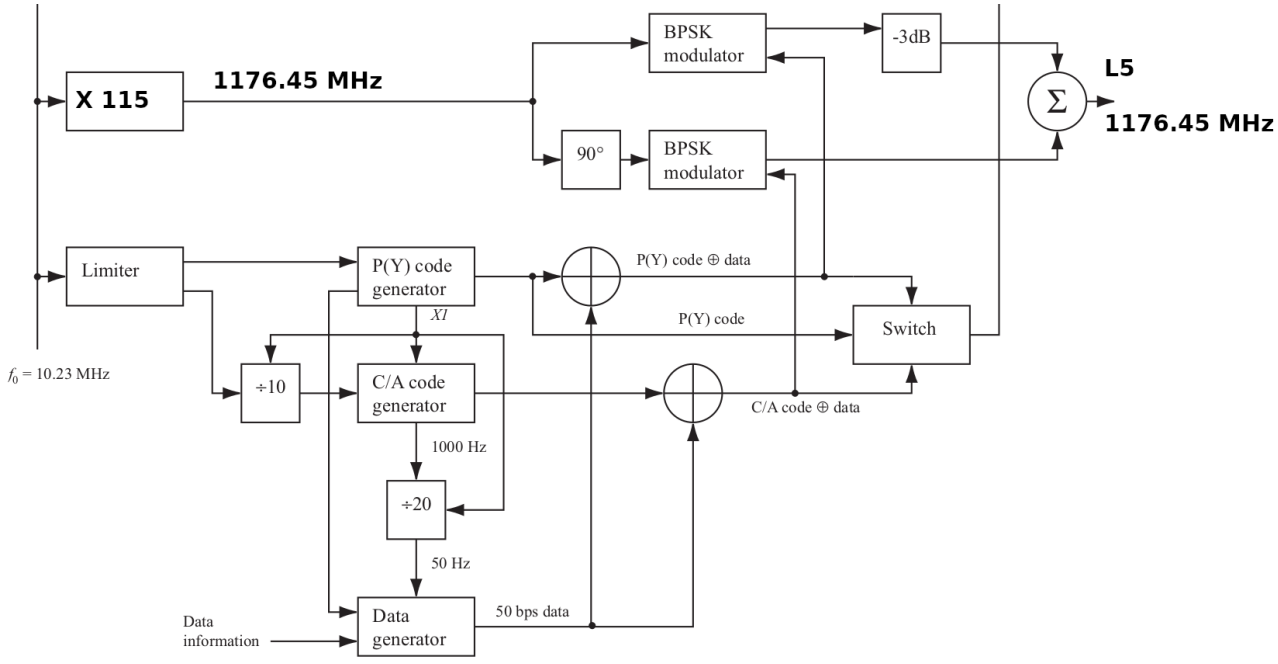
Polynomial G1 and G2 are similar to the ones used by GPS C/A signal. The G1 and G2 generators are realized by using 10 bits Maximum Length Feedback Shift Registers (MLFSR). The initial state of G2 provides the chip delay. The G1 register is initialized with all bits as 1. G1 and G2 are XOR‘ed for the generation of the final 1023 chip long PRN sequence. The time period of the PRN sequence is 1 millisecond.

Code Phase assignment for SPS signals :

PRN ID	G2 Register Initial Value
1	1110100111
2	0000100110
3	1000110100
4	0101110010
5	1110110000
6	0001101011
7	0000010100
8	0100110000
9	0010011000
10	1101100100
11	0001001100
12	1101111100
13	1011010010
14	0111101010



Navic Transmiter Block Representation :

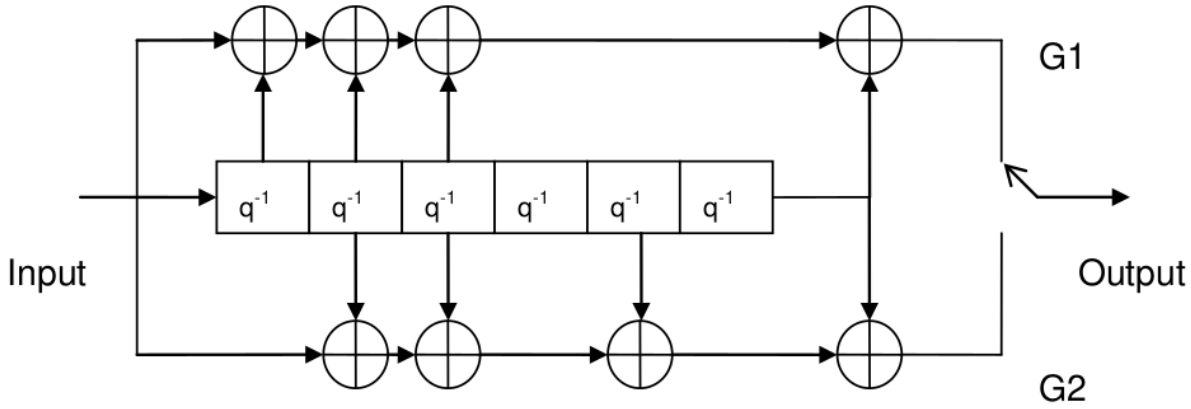


Navigation Data Structure:

IRNSS Signal In Space transmits Navigation message contents on SPS service, in L5 and S bands. The IRNSS Master frame comprises of four Sub Frames. Each Subframe is 600 symbols transmitted at 50 sps. Each subframe has 16 bit Sync word followed by 584 bits of interleaved data.

The 584 symbols of interleaved data is obtained from FEC encoding 292 subframe bits.

The Navigation data subframe of 292 bits is rate 1/2 convolution encoded and clocked at 50 symbols per second. Figure below depicts the convolution coding scheme in IRNSS :



FEC encoding parameters are :

Parameter	Value
Coding Scheme	Convolution
Coding Rate	1⁄2
Constraint Length	7
Generator Polynomial	G1 = (171)o G2 = (133)o
Encoding Sequence	G1 then G2

Each subframe of 292 bits, after encoding, results in 584 symbols.

The 584 symbols of FEC encoded navigation data is interleaved using a block interleaver with n columns and k rows. Data is written in columns and then, read in rows in the form of 73 x 8 (i.e.. n=73 rows x k=8 col)form.

The IRNSS Master Frame is of 2400 symbols long made of four sub frames. Each sub frame is 600 symbols long. Sub frames 1 and 2 transmit fixed primary navigation parameters. Sub frames 3 and 4 transmit secondary navigation parameters in the form of messages. The master frame structure is shown in Figure 10. All subframes transmit TLM, TOWC, Alert, Autonav, Subframe ID, Spare bit, Navigation data, CRC and Tail bits. Subframe 3 and 4 in addition transmit Message ID and PRN ID.

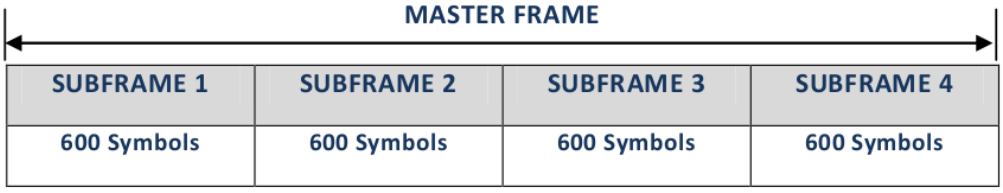


Figure 10: Master Frame Structure

Each Subframe is 292 bits long (without FEC encoding and Sync Word). The start of each subframe is with TLM word of 8 bits. Each subframe ends with 24 bit CRC followed by 6 tail bits.

In subframes 1 and 2 the Navigation data is allotted 232 bits starting from bit number 31. In subframes 3 and 4 the Navigation data is allotted 220 bits starting from bit number 37. The structure of a typical subframe 1 & 2 is shown in Figure 11. The structure of a typical subframe 3 & 4 is shown in Figure 12.

1	9	26	27	28	30	31	263	287
TLM	TOWC	ALERT	AUTONAV	SUBFRAME ID	SPARE	DATA	CRC	Tail
8 BITS	17BITS	1 BIT	1 BIT	2 BIT	1 BIT	232 BITS	24BITS	6BITS

Figure 11: Structure of Subframe 1 & 2

1	9	26	27	28	30	31	37	257	263	287
TLM	TOWC	ALERT	AUTONAV	SUBFRAME ID	SPARE	MESSAGE ID	DATA	PRN ID	CRC	Tail
8 BITS	17BITS	1 BIT	1 BIT	2 BIT	1 BIT	6 BITS	220 BITS	6	24 BITS	6 BITS

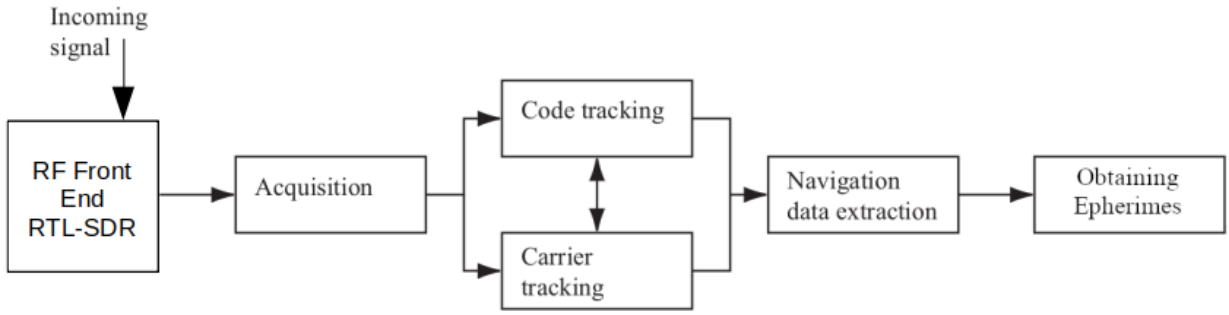
Figure 12: Structure of Subframe 3 & 4

The navigation data (NAV data) includes IRNSS satellite ephemeris, IRNSS time, satellite clock correction parameters, status messages and other secondary information etc. Navigation data modulated on top of the ranging codes can be identified as primary and secondary navigation parameters.

By understanding the signal generation structure on the NavIC system, we can clearly understand which methodology and design to be used in our Receiver Implementation using RTL-SDR. The following Methodology section describes the implementation details of the receiver.

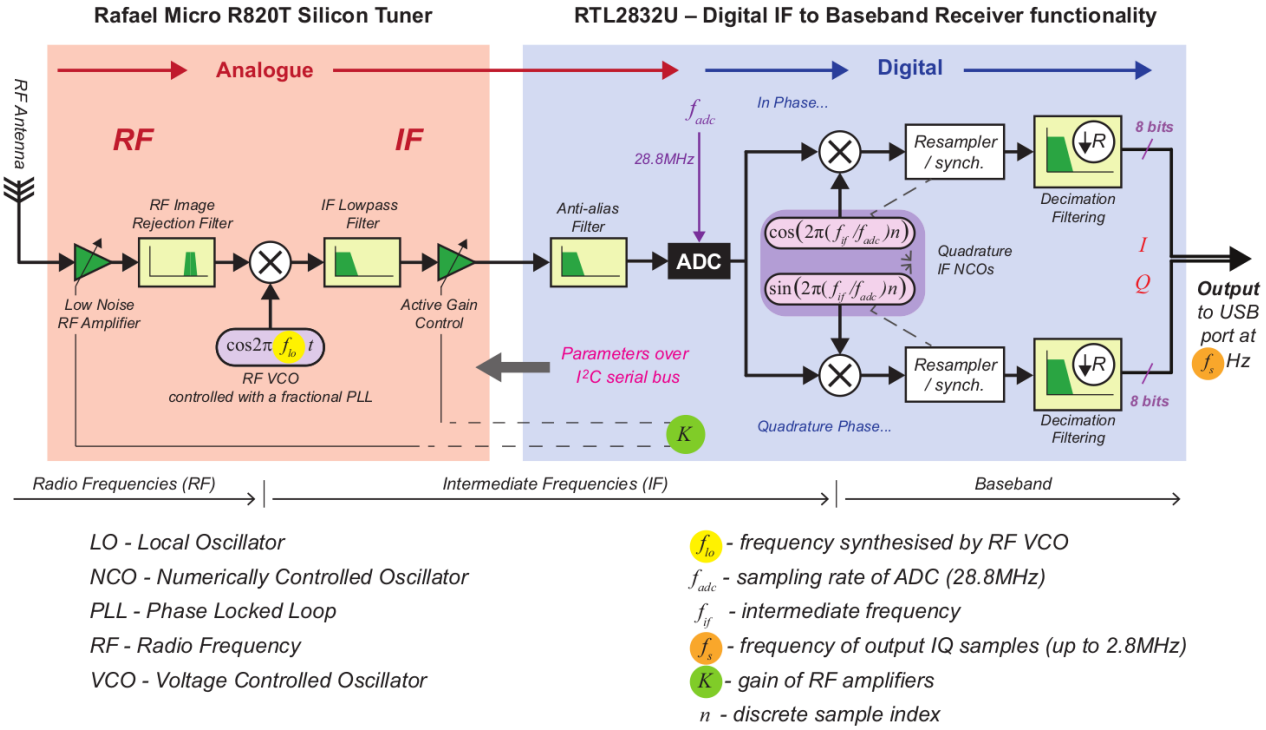
3. Methodology

3.1 The Block Level Architecture :



3.2 RF Front End:

The RTL-SDR is used as RF Front End its architecture is shown below:



LO - Local Oscillator
NCO - Numerically Controlled Oscillator
PLL - Phase Locked Loop
RF - Radio Frequency
VCO - Voltage Controlled Oscillator

f_{lo} - frequency synthesised by RF VCO
f_{adc} - sampling rate of ADC (28.8MHz)
f_{if} - intermediate frequency
f_s - frequency of output IQ samples (up to 2.8MHz)
K - gain of RF amplifiers
n - discrete sample index

The Antenna should be RHCP to get good reception of the signal. Where as if we use a Horizonal/Vertical Polarized dipole antena , which is very common and available at ease and low cost the loss will be 3dB, which is bearable when the signal strength is good as in clear sky.

3.3 Signal Acquisition :

The purpose of acquisition is to provide rough estimates of the carrier frequency and code phase from the visible satellites signals.

Doppler Effect is a consequence of the relative motion of the satellites, resulting in frequency deviations up to ±10 kHz, in worst cases. This effect is important when generating a local signal to remove the incoming carrier from the signal.

One of the standard methods of acquisition is Serial Search Acquisition. The algorithm is based on multiplication of generated PRNs and carrier signals. A PRN code is generated and multiplied by the incoming signal prior to be multiplied again by a generated carrier signal. This method performs one frequency sweep of IF in the range of ±10 kHz in steps of 500 Hz and a code phase sweep over 1023 different code phases, performing a total of about 41k combination which is exhaustive.

Instead of multiplying the input signal with a PRN code with 1023 different code phases as done in the serial search acquisition, the Parallel Code Phase Search Acquisition method makes a circular cross correlation between the input and the PRN code without shifted code phase.

Also this method parallelized the frequency space search eliminating the necessity of searching through the 41 possible frequencies. The goal of the Parallel Code Phase Search Acquisition is to parallelize the code phase.Primarily the incoming signal is multiplied with a generated cosine and sine carrier wave, obtaining an I and a Q signal. These two are combined as a complex input to the Fourier transform.

A PRN code is generated with zero code phase. Further, a Fourier transform is performed, and the result is complex conjugated. This complex conjugate is multiplied with the result of the Fourier transform, finally this result is given to an inverse Fourier transform.

As for the case of the FFT, for the output of the IFFT the absolute value also needs to be computed for all components. If a peak is present, the index of this peak symbolizes the PRN code phase of the incoming signal.

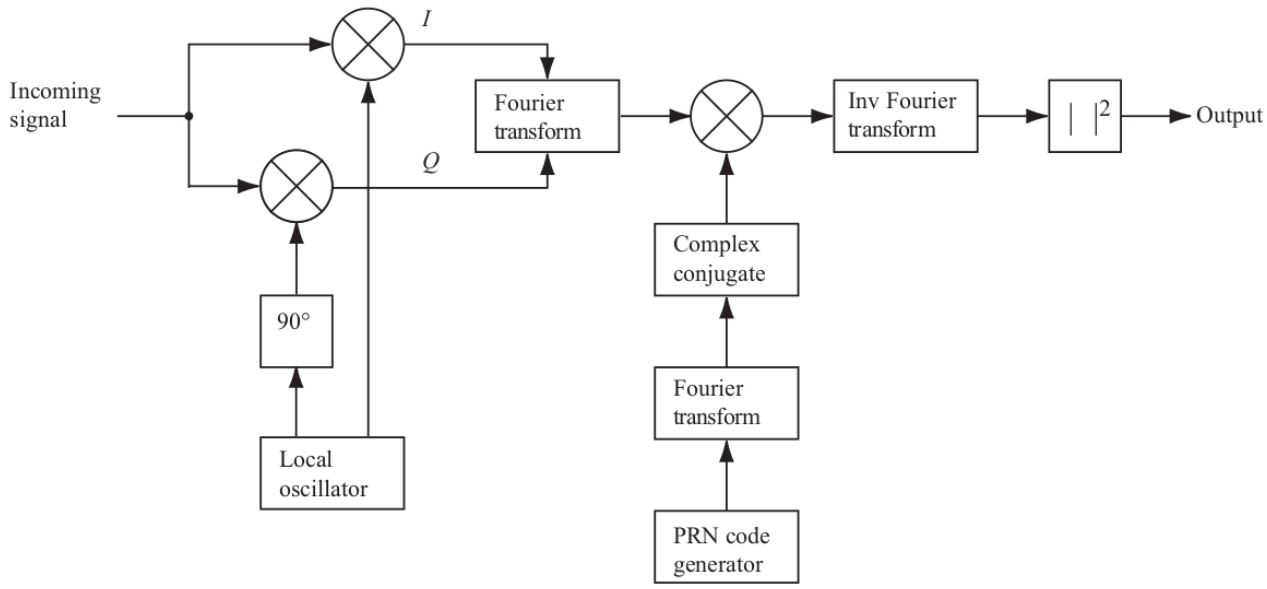


Fig: Parallel Code Phase Search algorithm

3.4 Signal Tracking :

The acquisition provides rough parameters of the frequency and code phases. The main purpose of tracking is to refine these values, keep track, and demodulate the navigation data.

The receiver must first replicate the PRN code that is transmitted by the satellite, and then it must shift the phase of the replica code until it correlates with the satellites PRN code.It also must detect the satellite in the carrier phase dimension by replicating the carrier frequency plus Doppler induced effect.

Figure 3.4 depicts a generic digital receiver channel. The IF is removed from the carrier, plus carrier Doppler, by the replica carrier signals, also plus carrier Doppler, to produce in-phase (I) and quadrature (Q) sampled data. Any misalignment in the replica carrier phase with respect to the incoming satellite signal carrier phase produces a phase angle of the prompt I and Q vector magnitude, so that the amount and direction of the phase change can be corrected by the carrier tracking loop.

When the phase-locked loop (PLL) is phase locked, I and Q signals are then correlated with early, prompt, and late replica codes. As a result, there are three replica code phases designated as early (E), prompt (P), and late (L). E and L are typically separated in phase by 1 chip and P is in the middle.

When the prompt replica code phase is aligned with the incoming satellite code phase producing maximum correlation, the early phase is aligned a fraction of a chip period early, and the late phase is aligned the same fraction of the chip period later. Any misalignment in the replica code phase with respect to the incoming satellite code phase produces a difference in the vector magnitudes of the early and late correlated outputs so that the amount and direction of the phase change can be detected and corrected by the code tracking loop.

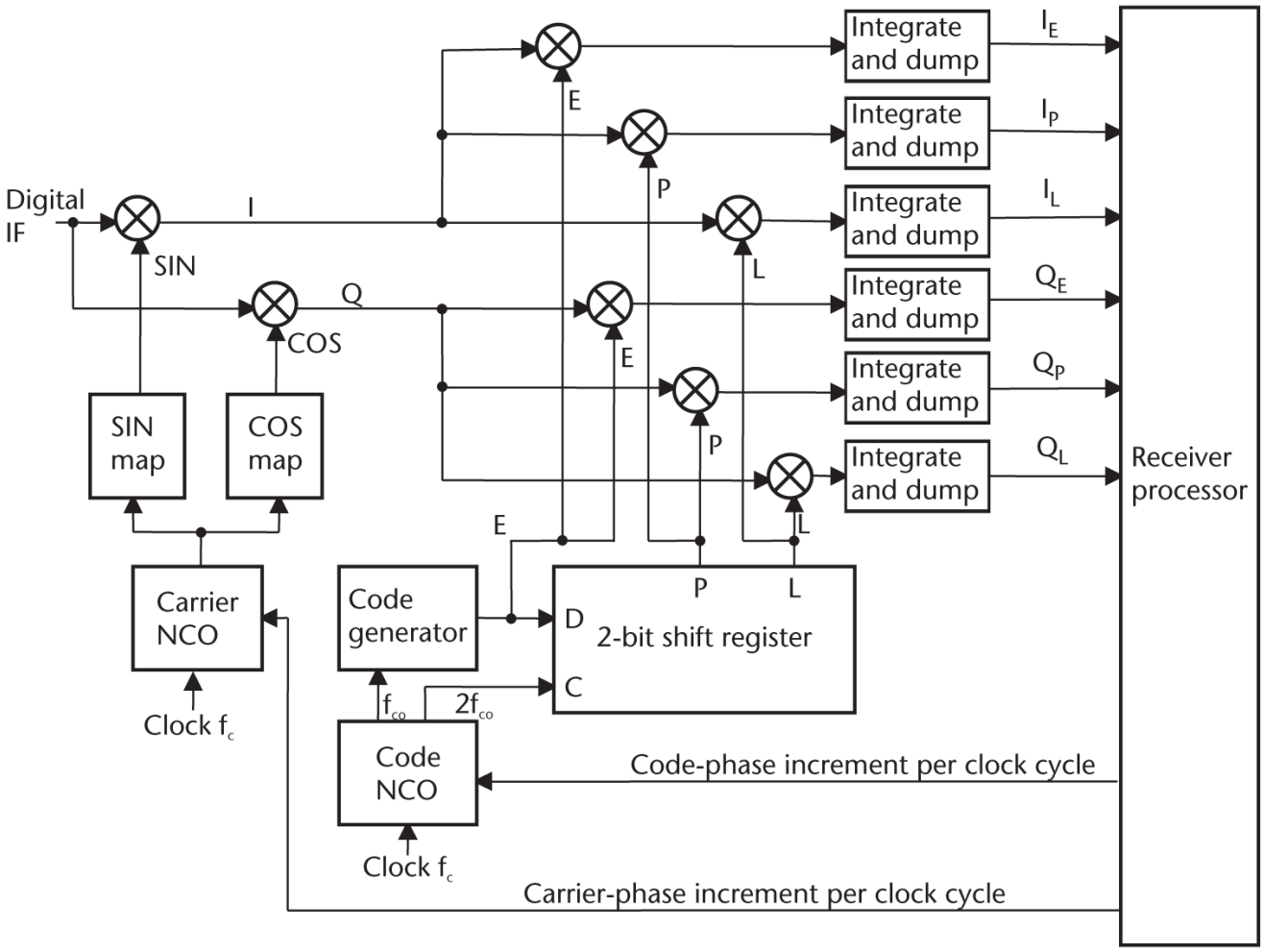


Fig 3.4 : Generic digital receiver channel

With the demodulated navigation data, the subframes can be acquired, then the ephemeris are obtained, and finally the pseudoranges as well.

3.5 Navigation Data Extraction:

It comprises of following Linear Piped sub-blocks:

1. Preamble Detection.
2. FEC decoding using Trellis coding.
3. CRC error checking.
4. Sub-Frame and Satellite ID evaluation.
5. Finally TOW, Satellite Location and other Ephemeris Extraction.

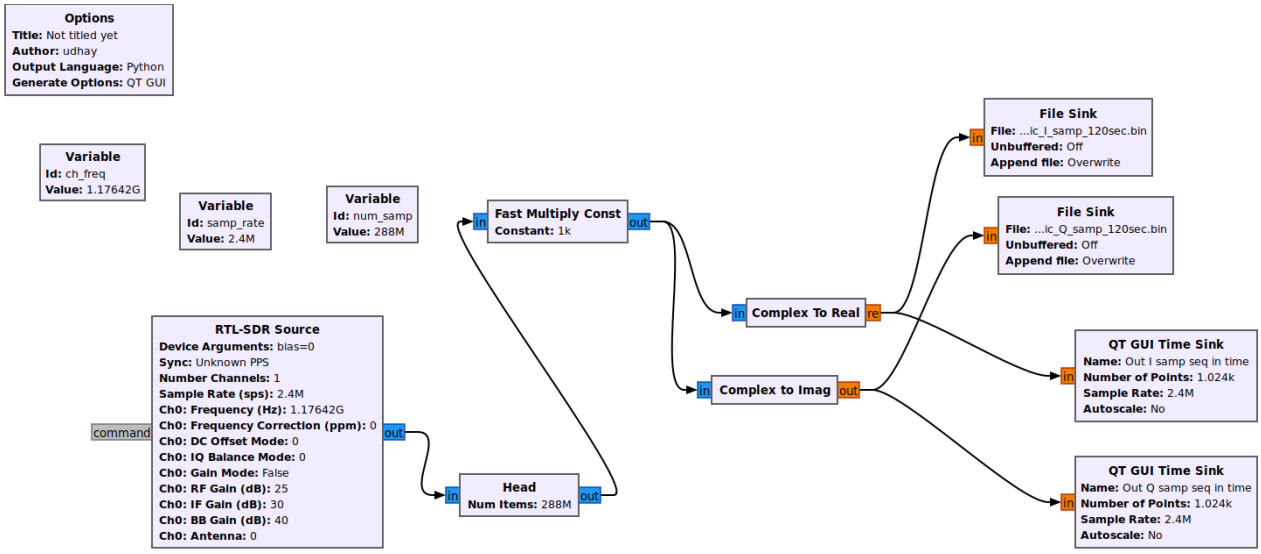
4. Implementation and Evaluation

Following Apparatus are Required:

1. RTL-SDR
2. RHCP active antenna (with 3.5V to 5V DC bias) is preferrable than Tee-Dipole Antenna
3. Computing system
4. Osmocomm RTL-SDR driver software.
5. GNU Radio with Python along with Numpy and Matplotlib Libraries.

RF Front End with RTL-SDR

The RTL-SDR is used to obtain I-Q samples at $f_c = 1176.42MHz$ through GNU Radio Block diagram to store the IQ data to ram temparty file system as shown in the Below Design:



In this implementation instead of using Active RHCP antenna,the **Passive Dipole antenna** in **Horizontal Polarization** with V-shape of 120^o between the two arms as show in below figure:

The above figure shows the Passive Antenna connected to RTL-SDR dongle grabbing IQ Samples as show on the Laptop screen using GNU Radio Companion Block design.

The Stored IQ data files are used in the next blocks for the processing at leisure time or off-the site location.

Implementation in Python

Navic Receiver Block

In [1]:
```
import numpy as np
from numpy import pi
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:
```
from navicCAcodegen import genNavicCaCode as genCaCode
from navicCAcodegen import genNavicCaTable as genCaTable
```

Initializations or Params

In [3]:
```
## Initialize block
msToProcess = int(40*1000)
numberOfChannels = 8
skipNumberOfBytes = 0

Gain = 2 # 1000

dataType = 'float32' # 'int8' # 'float32'

## sampled at 2.4MHz ie. Mega Samples per Second
samplingFreq = 2.4e6 # 2.4e6
codeFreqBasis = 1023000.0
codeLength = 1023

Ts = 1/samplingFreq

prnTot = 14 # 14 for Navic 32 for GPS
acqSatelliteList = range(1, prnTot+1)
acqSearchBand = 14e3 ##  In Hz for balancing Doppler shift in freq
acqThreshold = 1.5 # 2.5

## required for seeking in tracking module
bytes_per_sample = 4


## Code Tracking params
dllDampingRatio = 0.7
dllNoiseBandwidth = 5.0 # 2.0
dllCorrelatorSpacing = 0.5

## Carrier tracking loop parameters
pllDampingRatio = 0.5
pllNoiseBandwidth = 30.0

# Period for calculating pseudoranges and position
navSolPeriod = 500.0 ## Corresponding to 50 bits/sec
elevationMask = 10.0
useTropCorr = True
truePosition = (np.nan,np.nan,np.nan)

## Some constant values
c = 299792458.0
startOffset = 0 #0 # 68.802

## Derived constants
samplesPerCode = np.long(np.round(samplingFreq / (codeFreqBasis / codeLength)))
```

SDR front end part

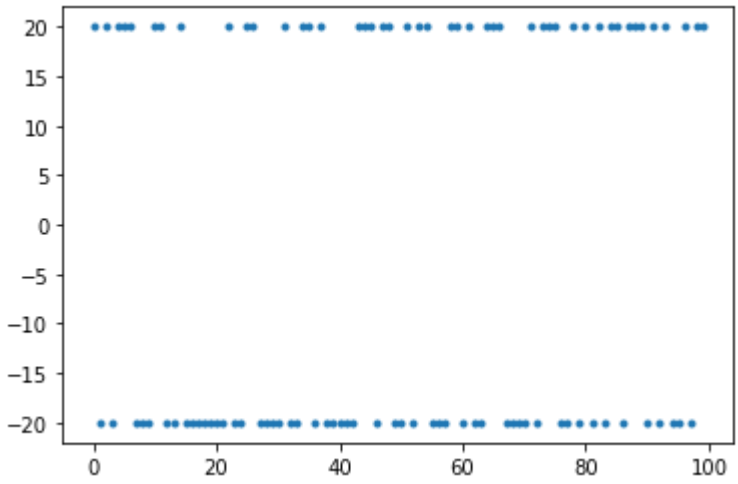
In [4]:
```
file_i_dat = '/home/udhay/data/workspace/NavIC_receiver/raichur_data_rtlsdr/NavicL1_I
file_q_dat = '/home/udhay/data/workspace/NavIC_receiver/raichur_data_rtlsdr/NavicL1_Q
```

Acquisition using Parallel Code Phase Search Acquisition

In [7]:
```
## Implementing Parallel Code Phase Search Acquisition

caCodeTable = genCaTable(samplingFreq)
## Get small time frame data
i_data = np.fromfile(file_i_dat,dataType,count=2*samplesPerCode)
q_data = np.fromfile(file_q_dat,dataType,count=2*samplesPerCode)

i_data[i_data<0] = -10
i_data[i_data>=0] = 10

q_data[q_data<0] = -10
q_data[q_data>=0] = 10

x_t = Gain * (i_data + 1j * q_data)

#x_t = (i_data - i_data.mean()) + 1j * (q_data - q_data.mean())

#x_t = x_t[:int(2**14)]
```

In [8]:
```
plt.plot(x_t[:100].real,'.')
```

Out[8]:
```
[<matplotlib.lines.Line2D at 0x7f960d2d88b0>]
```



In [9]:
```
def acquisition(PRN):
        caCode = caCodeTable[PRN,:]
        H_f = np.fft.fft(caCode).conj()

        n = np.arange(0,len(x_t))
        t_omega = 1j*2*pi*Ts*n

        results = np.zeros((len(freqBins),samplesPerCode))
        for findex in range(len(freqBins)):
            fd = freqBins[findex]
            g_t = x_t * np.exp(fd*t_omega)
            G1_f = np.fft.fft(g_t[:samplesPerCode])
            G2_f = np.fft.fft(g_t[:samplesPerCode])
            Y1_f = G1_f * H_f
            Y2_f = G2_f * H_f
            s1_t = abs(np.fft.ifft(Y1_f))**2
            s2_t = abs(np.fft.ifft(Y2_f))**2
            if s1_t.max() >= s2_t.max():
                results[findex] = s1_t
            else:
                results[findex] = s2_t
        ## For Peak metric
        frequencyBinIndex = results.max(1).argmax()
        peakSize = results.max(0).max()
        codePhase = results.max(0).argmax()
        samplesPerCodeChip = np.longlong(round(samplingFreq/codeFreqBasis))
        ## Get the peak from remaining sig power
```

```
        excludeRangeIndex1 = codePhase - samplesPerCodeChip
        excludeRangeIndex2 = codePhase + samplesPerCodeChip
        if excludeRangeIndex1 <= 0:
            codePhaseRange = np.r_[excludeRangeIndex2:samplesPerCode + excludeRangeIn

        elif excludeRangeIndex2 >= samplesPerCode - 1:
            codePhaseRange = np.r_[excludeRangeIndex2 - samplesPerCode:excludeRangeIn

        else:
            codePhaseRange = np.r_[0:excludeRangeIndex1 + 1, excludeRangeIndex2:sampl

        ## Calculate peak metric
        secondPeakSize = results[frequencyBinIndex, codePhaseRange].max()
        print(f'firstPeak = {peakSize} , secondPeak = {secondPeakSize}')
        peakMetric[PRN] = peakSize / secondPeakSize
        ## Acquisition Result for each PRN
        if peakMetric[PRN] >= acqThreshold:
            codePhase_[PRN] = codePhase
            carrFreq[PRN] = freqBins[frequencyBinIndex]
```

In [10]:

```
freqBins = np.int16(np.arange(-acqSearchBand//2,acqSearchBand//2 + 1,step=50))

# Carrier frequencies of detected signals
carrFreq = np.zeros(prnTot)

# C/A code phases of detected signals
codePhase_ = np.zeros(prnTot)

# Correlation peak ratios of the detected signals
peakMetric = np.zeros(prnTot)

## Call Acquisiton routine
for PRN in range(prnTot):
    acquisition(PRN)
```

```
firstPeak = 25398534.63359537 , secondPeak = 16690440.990901574
firstPeak = 24752560.066268332 , secondPeak = 14280857.107571963
firstPeak = 26259282.347832646 , secondPeak = 13186344.916658588
firstPeak = 23201741.983238008 , secondPeak = 16741155.278784389
firstPeak = 24572514.479839887 , secondPeak = 20945593.913220827
firstPeak = 26884446.301375117 , secondPeak = 12873805.76014831
firstPeak = 23861383.89796119 , secondPeak = 17662617.81345564
firstPeak = 27370389.024455026 , secondPeak = 16274939.289470414
firstPeak = 27135113.605787218 , secondPeak = 18111094.5447838
firstPeak = 23792003.90846893 , secondPeak = 12904751.365619864
firstPeak = 23298312.728742965 , secondPeak = 16767233.39876163
firstPeak = 24051532.9223821 , secondPeak = 14694590.072963096
firstPeak = 26986983.449044034 , secondPeak = 17305351.441131245
firstPeak = 25738216.405084886 , secondPeak = 15486614.527138112
```

In [11]:

```
## Print Acquisition result:
print("Acquisition Results are :")

for PRN in range(prnTot):
    if peakMetric[PRN] >= acqThreshold:
            print(f'For PRN : {PRN} , CodePhase = {codePhase_[PRN]} , Peak Metric = {
```

```
Acquisition Results are :
For PRN : 0 , CodePhase = 1912.0 , Peak Metric = 1.5217413756437483 and Doppler Freq =
-5000.0
For PRN : 1 , CodePhase = 1844.0 , Peak Metric = 1.7332685202167653 and Doppler Freq =
4050.0
For PRN : 2 , CodePhase = 2182.0 , Peak Metric = 1.9913996269473235 and Doppler Freq =
4650.0
For PRN : 5 , CodePhase = 1176.0 , Peak Metric = 2.0883060380324863 and Doppler Freq =
5800.0
For PRN : 7 , CodePhase = 609.0 , Peak Metric = 1.6817506067235017 and Doppler Freq =
-3050.0
For PRN : 9 , CodePhase = 1092.0 , Peak Metric = 1.8436623251691846 and Doppler Freq =
3500.0
For PRN : 11 , CodePhase = 468.0 , Peak Metric = 1.6367610666890975 and Doppler Freq =
```

```
-1700.0
For PRN : 12 , CodePhase = 1613.0 , Peak Metric = 1.5594588495268318 and Doppler Freq
= 6400.0
For PRN : 13 , CodePhase = 1984.0 , Peak Metric = 1.6619653288317007 and Doppler Freq
= 3900.0
```

Preparing for Tracking

In [12]:
```python
## Sort the PRN according to Peak Metric (approx signal strength)
PRNindexes = sorted(enumerate(peakMetric),key=lambda x: x[-1], reverse=True)

## Total satellites we consider for tracking
sat_in_View = min(numberOfChannels,sum(peakMetric>=acqThreshold))

## Store the Tracking Sat PRN nums
PRN = np.zeros(sat_in_View,dtype=np.int)
acquiredFreq = np.zeros(sat_in_View)
codePhase = np.zeros(sat_in_View,dtype=np.int)
status = ['-' for _ in range(sat_in_View)]

for i in range(sat_in_View):
    sat_prn = PRNindexes[i][0]
    ## Actual Sat Number starts from 1
    PRN[i] = sat_prn + 1
    acquiredFreq[i] = carrFreq[sat_prn]
    codePhase[i] = codePhase_[sat_prn]
    status[i] = 'T'

## Create a channel data structure for easy access
channel = np.core.records.fromarrays([PRN, acquiredFreq, codePhase, status],
                                     names='PRN,acquiredFreq,codePhase,status
```

In [13]:
```python
# calcLoopCoef.m
def calcLoopCoef(LBW, zeta, k):
    # Function finds loop coefficients. The coefficients are used then in PLL-s
    # and DLL-s.
    # [tau1, tau2] = calcLoopCoef(LBW, zeta, k)
    #   Inputs:
    #       LBW           - Loop noise bandwidth
    #       zeta          - Damping ratio
    #       k             - Loop gain
    #   Outputs:
    #       tau1, tau2    - Loop filter coefficients

    # Solve for natural frequency
    Wn = LBW * 8.0 * zeta / (4.0 * zeta ** 2 + 1)

    # solve for t1 & t2
    tau1 = k / (Wn * Wn)

    tau2 = 2.0 * zeta / Wn

    return tau1, tau2
```

In [15]:
```python
## Initialize Tracking variables
codePeriods = msToProcess

# --- DLL variables -----
# Define early-late offset (in chips)
earlyLateSpc = dllCorrelatorSpacing

# Summation interval
PDIcode = 0.001

# Calculate filter coefficient values
tau1code, tau2code = calcLoopCoef(dllNoiseBandwidth, dllDampingRatio, 1.0)

# --- PLL variables -----------------------------------------------------
```

```
# Summation interval
PDIcarr = 0.001

# Calculate filter coefficient values
tau1carr, tau2carr = calcLoopCoef(pllNoiseBandwidth, pllDampingRatio, 0.25)


# Initialize a temporary list of records
rec = []
```

In [16]:
```
## Start time is stored in file creation time itself
import pathlib,datetime
startTime = datetime.datetime.fromtimestamp(pathlib.Path(file_i_dat).stat().st_mtime)
startTime
```

Out[16]:
```
datetime.datetime(2021, 5, 2, 12, 22, 36, 287107)
```

Tracking Starts !

In [23]:
```
# Start processing channels

## open the whole if samples file for processing
ifid = open(file_i_dat,'rb')
qfid = open(file_q_dat,'rb')

for channelNr in range(sat_in_View):
    # Initialize fields for record(structured) array of tracked results
    status = '-'

    # The absolute sample in the record of the C/A code start:
    absoluteSample = np.zeros(msToProcess)

    # Freq of the C/A code:
    codeFreq_ = np.Inf * np.ones(msToProcess)

    # Frequency of the tracked carrier wave:
    carrFreq_ = np.Inf * np.ones(msToProcess)

    # Outputs from the correlators (In-phase):
    I_P_ = np.zeros(msToProcess)

    I_E_ = np.zeros(msToProcess)

    I_L_ = np.zeros(msToProcess)

    # Outputs from the correlators (Quadrature-phase):
    Q_E_ = np.zeros(msToProcess)

    Q_P_ = np.zeros(msToProcess)

    Q_L_ = np.zeros(msToProcess)

    # Loop discriminators
    dllDiscr = np.Inf * np.ones(msToProcess)

    dllDiscrFilt = np.Inf * np.ones(msToProcess)

    pllDiscr = np.Inf * np.ones(msToProcess)

    pllDiscrFilt = np.Inf * np.ones(msToProcess)

    PRN = 0

    # Only process if PRN is non zero (acquisition was successful)
    if channel[channelNr].PRN != 0:
        # Save additional information - each channel's tracked PRN
        PRN = channel[channelNr].PRN
```

```
        # continue signal processing at any point in the data record (e.g. for long
        # records). In addition skip through that data file to start at the
        # appropriate sample (corresponding to code phase). Assumes sample
        # type is schar (or 1 byte per sample)
        ifid.seek(skipNumberOfBytes * bytes_per_sample + channel[channelNr].codePhase
        qfid.seek(skipNumberOfBytes * bytes_per_sample + channel[channelNr].codePhase

        # Here PRN is the actual satellite ID instead of the 0-based index
        caCode = genCaCode(channel[channelNr].PRN - 1)

        caCode = np.r_[caCode[-1], caCode, caCode[0]]

        # define initial code frequency basis of NCO
        codeFreq = codeFreqBasis

        remCodePhase = 0.0

        carrFreq = channel[channelNr].acquiredFreq

        carrFreqBasis = channel[channelNr].acquiredFreq

        remCarrPhase = 0.0

        oldCodeNco = 0.0

        oldCodeError = 0.0

        oldCarrNco = 0.0

        oldCarrError = 0.0

        for loopCnt in range(np.long(codePeriods)):
            if loopCnt == 0:
                print(f'Tracking started for :{channelNr+1} of {sat_in_View} ; PRN is

            if loopCnt % 400 == 0:
                try:
                    print('#',end='')
                finally:
                    pass
            # Read next block of data ------------------------------------------------
            # Find the size of a "block" or code period in whole samples
            # Update the phasestep based on code freq (variable) and
            # sampling frequency (fixed)
            codePhaseStep = codeFreq / samplingFreq

            blksize = np.ceil((codeLength - remCodePhase) / codePhaseStep)
            blksize = np.long(blksize)

            # interaction
            irawSignal = Gain * np.fromfile(ifid, dataType, blksize)
            qrawSignal = Gain * np.fromfile(qfid, dataType, blksize)

            samplesRead = len(irawSignal)

            # If did not read in enough samples, then could be out of
            # data - better exit
            if samplesRead != blksize:
                print ('Not able to read the specified number of samples for tracking
                fid.close()
                trackResults = None
                break
            # Set up all the code phase tracking information -----------------------
            # Define index into early code vector
            tcode = np.linspace(remCodePhase - earlyLateSpc,
                                blksize * codePhaseStep + remCodePhase - earlyLateSpc
                                blksize, endpoint=False)
```

```
            tcode2 = np.ceil(tcode)

            earlyCode = caCode[np.int64(tcode2)]

            tcode = np.linspace(remCodePhase + earlyLateSpc,
                                blksize * codePhaseStep + remCodePhase + earlyLateSpc
                                blksize, endpoint=False)

            tcode2 = np.ceil(tcode)

            lateCode = caCode[np.int64(tcode2)]

            tcode = np.linspace(remCodePhase,
                                blksize * codePhaseStep + remCodePhase,
                                blksize, endpoint=False)

            tcode2 = np.ceil(tcode)

            promptCode = caCode[np.int64(tcode2)]

            remCodePhase = tcode[blksize - 1] + codePhaseStep - 1023.0

            # Generate the carrier frequency to mix the signal to baseband ----------
            time = np.arange(0, blksize + 1)* Ts

            trigarg = carrFreq * 2.0 * pi * time + remCarrPhase

            remCarrPhase = trigarg[blksize] % (2 * pi)

            carrCos = np.cos(trigarg[0:blksize])

            carrSin = np.sin(trigarg[0:blksize])

            # Generate the six standard accumulated values ---------------------------
            # First mix to baseband
            iBasebandSignal = irawSignal[:] * carrCos - qrawSignal[:] * carrSin
            qBasebandSignal = irawSignal[:] * carrSin + qrawSignal[:] * carrCos

            I_E = (earlyCode * iBasebandSignal).sum()

            Q_E = (earlyCode * qBasebandSignal).sum()

            I_P = (promptCode * iBasebandSignal).sum()

            Q_P = (promptCode * qBasebandSignal).sum()

            I_L = (lateCode * iBasebandSignal).sum()

            Q_L = (lateCode * qBasebandSignal).sum()

            # Find PLL error and update carrier NCO ----------------------------------
            # Implement carrier loop discriminator (phase detector)
            carrError = np.arctan(Q_P / I_P) % (2.0 * np.pi)

            carrNco = oldCarrNco + \
                      tau2carr / tau1carr * (carrError - oldCarrError) + \
                      carrError * (PDIcarr / tau1carr)

            oldCarrNco = carrNco

            oldCarrError = carrError

            carrFreq = carrFreqBasis + carrNco

            carrFreq_[loopCnt] = carrFreq

            # Find DLL error and update code NCO -------------------------------------
            codeError = (np.sqrt(I_E * I_E + Q_E * Q_E) - np.sqrt(I_L * I_L + Q_L * Q_
                    np.sqrt(I_E * I_E + Q_E * Q_E) + np.sqrt(I_L * I_L + Q_L * Q_L))
```

```
            codeNco = oldCodeNco + \
                      tau2code / tau1code * (codeError - oldCodeError) + \
                      codeError * (PDIcode / tau1code)

            oldCodeNco = codeNco

            oldCodeError = codeError

            codeFreq = codeFreqBasis - codeNco

            codeFreq_[loopCnt] = codeFreq

            # Record various measures to show in postprocessing --------------------
            # Record sample number (based on 8bit samples)
            absoluteSample[loopCnt] = ifid.tell()

            dllDiscr[loopCnt] = codeError

            dllDiscrFilt[loopCnt] = codeNco

            pllDiscr[loopCnt] = carrError

            pllDiscrFilt[loopCnt] = carrNco

            I_E_[loopCnt] = I_E

            I_P_[loopCnt] = I_P

            I_L_[loopCnt] = I_L

            Q_E_[loopCnt] = Q_E

            Q_P_[loopCnt] = Q_P

            Q_L_[loopCnt] = Q_L
        print(' Done',end='\n')
        # If we got so far, this means that the tracking was successful
        # Now we only copy status, but it can be update by a lock detector
        # if implemented
        status = channel[channelNr].status
        rec.append((status, absoluteSample, codeFreq_, carrFreq_,
                    I_P_, I_E_, I_L_, Q_E_, Q_P_, Q_L_,
                    dllDiscr, dllDiscrFilt, pllDiscr, pllDiscrFilt, PRN))
print('\nTracking Process is Completed')
ifid.close()
qfid.close()
```

```
Tracking started for :1 of 8 ; PRN is 6 in progress################################
################################################################ Done
Tracking started for :2 of 8 ; PRN is 3 in progress################################
################################################################ Done
Tracking started for :3 of 8 ; PRN is 10 in progress################################
################################################################ Done
Tracking started for :4 of 8 ; PRN is 2 in progress################################
################################################################ Done
Tracking started for :5 of 8 ; PRN is 8 in progress################################
################################################################ Done
Tracking started for :6 of 8 ; PRN is 14 in progress################################
################################################################ Done
Tracking started for :7 of 8 ; PRN is 12 in progress################################
################################################################ Done
Tracking started for :8 of 8 ; PRN is 13 in progress################################
################################################################ Done

Tracking Process is Completed
```

Store Tracking results

In [25]:
```
## Store tracking results in a Data structure
```

```
trackResults = np.rec.fromrecords(rec, dtype=[('status', 'S1'), ('absoluteSample', 'ol
                                              ('carrFreq', 'object'), ('I_P', 'obj
                                              ('I_L', 'object'),
                                              ('Q_E', 'object'), ('Q_P', 'object')
                                              ('dllDiscr', 'object'),
                                              ('dllDiscrFilt', 'object'), ('pllDis
                                              ('pllDiscrFilt', 'object'),
                                              ('PRN', 'int64')])
np.save('./trackResults.npy',trackResults)
print('Storing of Tracking Data is done for Extracting Navigation Data')
```

```
Storing of Tracking Data is done for Extracting Navigation Data
```

Post Processing Block

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:
```
numberOfChannels = 8
```

Post Processing for epherimis Data Extracting

In [3]:
```
## Load the saved Tracking result
trackResults = np.load('./trackResults.npy',allow_pickle=True)
```

In [4]:
```
trackResults.status
```

Out[4]:
```
array([b'T', b'T', b'T', b'T', b'T', b'T', b'T', b'T'], dtype='|S1')
```

In [5]:
```
def navPartyChk(ndat):

        if ndat[1] != 1:
            ndat[2:26] *= (-1)

        # --- Calculate 6 parity bits -----------------------------------------------
        # The elements of the ndat array correspond to the bits showed in the table
        # 20-XIV (ICD-200C document) in the following way:
        # The first element in the ndat is the D29* bit and the second - D30*.
        # The elements 3 - 26 are bits d1-d24 in the table.
        # The elements 27 - 32 in the ndat array are the received bits D25-D30.
        # The array "parity" contains the computed D25-D30 (parity) bits.
        parity = np.zeros(6)
        parity[0] = ndat[0] * ndat[2] * ndat[3] * ndat[4] * ndat[6] * \
                    ndat[7] * ndat[11] * ndat[12] * ndat[13] * ndat[14] * \
                    ndat[15] * ndat[18] * ndat[19] * ndat[21] * ndat[24]

        parity[1] = ndat[1] * ndat[3] * ndat[4] * ndat[5] * ndat[7] * \
                    ndat[8] * ndat[12] * ndat[13] * ndat[14] * ndat[15] * \
                    ndat[16] * ndat[19] * ndat[20] * ndat[22] * ndat[25]

        parity[2] = ndat[0] * ndat[2] * ndat[4] * ndat[5] * ndat[6] * \
                    ndat[8] * ndat[9] * ndat[13] * ndat[14] * ndat[15] * \
                    ndat[16] * ndat[17] * ndat[20] * ndat[21] * ndat[23]

        parity[3] = ndat[1] * ndat[3] * ndat[5] * ndat[6] * ndat[7] * \
                    ndat[9] * ndat[10] * ndat[14] * ndat[15] * ndat[16] * \
                    ndat[17] * ndat[18] * ndat[21] * ndat[22] * ndat[24]

        parity[4] = ndat[1] * ndat[2] * ndat[4] * ndat[6] * ndat[7] * \
                    ndat[8] * ndat[10] * ndat[11] * ndat[15] * ndat[16] * \
                    ndat[17] * ndat[18] * ndat[19] * ndat[22] * ndat[23] * \
                    ndat[25]

        parity[5] = ndat[0] * ndat[4] * ndat[6] * ndat[7] * ndat[9] * \
                    ndat[10] * ndat[11] * ndat[12] * ndat[14] * ndat[16] * \
                    ndat[20] * ndat[23] * ndat[24] * ndat[25]

        # --- Compare if the received parity is equal the calculated parity --------
        if (parity == ndat[26:]).sum() == 6:
            # Parity is OK. Function output is -1 or 1 depending if the data bits
            # must be inverted or not. The "ndat[2]" is D30* bit - the last  bit of
            # previous subframe.
            status = -1 * ndat[1]

        else:
            # Parity failure
```

```
            status = 0

        return status
```

In [6]:

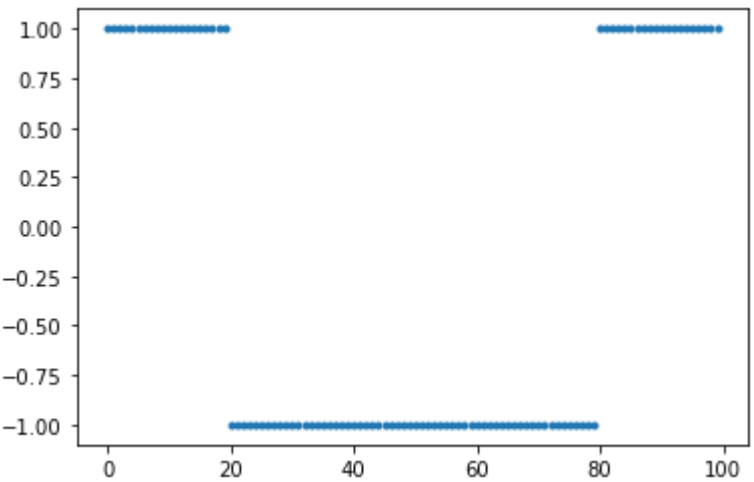
```
def findPreambles(trackResults):
    # findPreambles finds the first preamble occurrence in the bit stream of
    # each channel. The preamble is verified by check of the spacing between
    # preambles (6sec) and parity checking of the first two words in a
    # subframe. At the same time function returns list of channels, that are in
    # tracking state and with valid preambles in the nav data stream.

    # [firstSubFrame, activeChnList] = findPreambles(trackResults, settings)

    #   Inputs:
    #       trackResults    - output from the tracking function
    #       settings        - Receiver settings.

    #   Outputs:
    #       firstSubframe   - the array contains positions of the first
    #                         preamble in each channel. The position is ms count
    #                         since start of tracking. Corresponding value will
    #                         be set to 0 if no valid preambles were detected in
    #                         the channel.
    #       activeChnList   - list of channels containing valid preambles

    # Preamble search can be delayed to a later point in the tracking results
    # to avoid noise due to tracking loop transients
    searchStartOffset = 0

    # --- Initialize the firstSubFrame array -----------------------------------
    firstSubFrame = np.zeros(len(trackResults), dtype=int)

    # --- Generate the preamble pattern ----------------------------------------
    preamble_bits = np.r_[1, - 1, - 1, - 1, 1, - 1, 1, 1]

    # "Upsample" the preamble - make 20 vales per one bit. The preamble must be
    # found with precision of a sample.
    preamble_ms = np.kron(preamble_bits, np.ones(20))

    # --- Make a list of channels excluding not tracking channels --------------
    activeChnList = (trackResults.status != b'-').nonzero()[0]

    # === For all tracking channels ...
    for channelNr in range(len(activeChnList)):
        # Correlate tracking output with preamble ================================
        # Read output from tracking. It contains the navigation bits. The start
        # of record is skipped here to avoid tracking loop transients.
        bits = trackResults[channelNr].I_P[searchStartOffset:].copy()

        bits[bits > 0] = 1

        bits[bits <= 0] = - 1

        # have to zero pad the preamble so that they are the same length
        tlmXcorrResult = np.correlate(bits,
                                      np.pad(preamble_ms, (0, bits.size - preamble_ms
                                      mode='full')

        # Find all starting points off all preamble like patterns ================
        # clear('index')
        # clear('index2')
        xcorrLength = (len(tlmXcorrResult) + 1) // 2

        index = (np.abs(tlmXcorrResult[xcorrLength - 1:xcorrLength * 2]) > 153).nonze

        # Analyze detected preamble like patterns ================================
        for i in range(len(index)):
            index2 = index - index[i]
```

```
            if (index2 == 6000).any():
                # === Re-read bit vales for preamble verification ==============
                # Preamble occurrence is verified by checking the parity of
                # the first two words in the subframe. Now it is assumed that
                # bit boundaries a known. Therefore the bit values over 20ms are
                # combined to increase receiver performance for noisy signals.
                # in Total 62 bits mast be read :
                # 2 bits from previous subframe are needed for parity checking;
                # 60 bits for the first two 30bit words (TLM and HOW words).
                # The index is pointing at the start of TLM word.
                bits = trackResults[channelNr].I_P[index[i] - 40:index[i] + 20 * 60].

                bits = bits.reshape(20, -1, order='F')

                bits = bits.sum(0)

                bits[bits > 0] = 1

                bits[bits <= 0] = - 1

                if navPartyChk(bits[:32]) != 0 and navPartyChk(bits[30:62]) != 0:
                    # Parity was OK. Record the preamble start position. Skip
                    # the rest of preamble pattern checking for this channel
                    # and process next channel.
                    firstSubFrame[channelNr] = index[i]

                    break
        # Exclude channel from the active channel list if no valid preamble was
        # detected
        if firstSubFrame[channelNr] == 0:
            # Exclude channel from further processing. It does not contain any
            # valid preamble and therefore nothing more can be done for it.
            activeChnList = np.setdiff1d(activeChnList, channelNr)

            print ('Could not find valid preambles in channel %2d !' % channelNr)
    return firstSubFrame, activeChnList
```

In [7]:
```
## Obtain each channel data and the starting position of data bits
subFrameStart,activeChnList = findPreambles(trackResults)
```

In [8]:
```
channelNr = 0
bitLoc = subFrameStart[channelNr]
## Clean data Bits
bits = trackResults[channelNr].I_P[bitLoc:].copy()
bits[bits > 0] = 1
bits[bits <= 0] = - 1
plt.plot(bits[:100],'.')
```

Out[8]:
```
[<matplotlib.lines.Line2D at 0x7f4cec0763d0>]
```



In [9]:
```
field_str = 'weekNumber,accuracy,health,T_GD,IODC,t_oc,a_f2,a_f1,a_f0,'
```

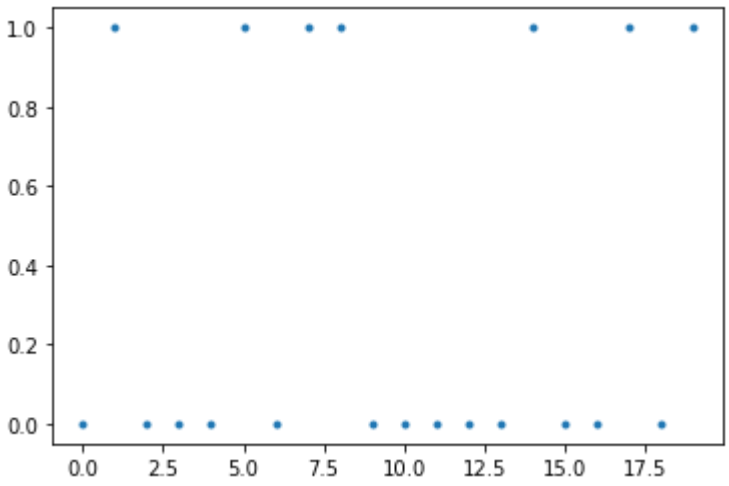
```
field_str += 'IODE_sf2,C_rs,deltan,M_0,C_uc,e,C_us,sqrtA,t_oe,'
field_str += 'C_ic,omega_0,C_is,i_0,C_rc,omega,omegaDot,IODE_sf3,iDot'
eph = np.recarray((32,), formats=['O'] * 27, names=field_str)
```

In [10]:
```
channelNr = 0
#for channelNr in activeChnList:
navBitsSamples = trackResults[channelNr].I_P[subFrameStart[channelNr] - 20:
                                                    subFrameStart[channelNr] + 1500 *
```

In [11]:
```
navBitsSamples = navBitsSamples.reshape(20, -1, order='F')
navBits = navBitsSamples.sum(0)
navBits = (navBits > 0) * 1
```

In [12]:
```
plt.plot(navBits[:20],'.')
```

Out[12]:
```
[<matplotlib.lines.Line2D at 0x7f4cdc4441c0>]
```



In [13]:
```
import ephemeris
```

In [14]:
```
# Decode ephemerides ===================================================
field_str = 'weekNumber,accuracy,health,T_GD,IODC,t_oc,a_f2,a_f1,a_f0,'
field_str += 'IODE_sf2,C_rs,deltan,M_0,C_uc,e,C_us,sqrtA,t_oe,'
field_str += 'C_ic,omega_0,C_is,i_0,C_rc,omega,omegaDot,IODE_sf3,iDot'
eph = np.recarray((32,), formats=['O'] * 27, names=field_str)
for channelNr in activeChnList:
    # === Convert tracking output to navigation bits =======================
    # --- Copy 5 sub-frames long record from tracking output ---------------
    navBitsSamples = trackResults[channelNr].I_P[subFrameStart[channelNr] - 20:
                                                 subFrameStart[channelNr] + 1500 * 20

    navBitsSamples = navBitsSamples.reshape(20, -1, order='F')

    navBits = navBitsSamples.sum(0)

    # The expression (navBits > 0) returns an array with elements set to 1
    # if the condition is met and set to 0 if it is not met.
    navBits = (navBits > 0) * 1

    # The function ephemeris expects input in binary form. In Matlab it is
    # a string array containing only "0" and "1" characters.
    navBitsBin = list(map(str, navBits))

    eph[trackResults[channelNr].PRN - 1], TOW = ephemeris.ephemeris(navBitsBin[1:], na

    if eph[trackResults[channelNr].PRN - 1].IODC is None or \
            eph[trackResults[channelNr].PRN - 1].IODE_sf2 is None or \
            eph[trackResults[channelNr].PRN - 1].IODE_sf3 is None:
        # --- Exclude channel from the list (from further processing) ------
        activeChnList = np.setdiff1d(activeChnList, channelNr)

transmitTime = TOW
```

In [15]:
```
TOW
```

Out[15]:
```
587454
```

Obtained Ephemeris Data

In [16]:
```
eph
```

Out[16]:
```
rec.array([(None, None, None, None, None, None, None, None, None, None, None, None, No
ne, None, None, None, None, None, None, None, None, None, None, None, None, None, Non
e),
           (None, None, None, None, None, None, None, None, None, None, None, None, No
ne, None, None, None, None, None, None, None, None, None, None, None, None, None, Non
e),
           (1321, 0, 0, -4.190951585769653e-09, 503, 590384, 0.0, 3.296918293926865e-1
2, 0.00010679662227630615, 85, 42.84375, 4.9302053628427535e-09, 1.6538727585988888,
2.261251211166382e-06, 0.006752743385732174, 1.2351199984550476e-05, 5153.63525772094
7, 590384, -1.2293457984924316e-07, -0.09233881716237433, 5.960464477539063e-08, 0.927
4761104056594, 121.59375, 0.5958616266113392, -8.207127574091831e-09, 85, -3.057270204
718485e-10),
           (None, None, None, None, None, None, None, None, None, None, None, None, No
ne, None, None, None, None, None, None, None, None, None, None, None, None, None, Non
e),
           (None, None, None, None, None, None, None, None, None, None, None, None, No
ne, None, None, None, None, None, None, None, None, None, None, None, None, None, Non
e),
           (1321, 0, 0, -4.6566128730773926e-09, 502, 590400, 0.0, 5.0590642786119133e
-11, 0.00047653587535023 69, 83, 70.875, 4.7191251419328674e-09, -0.30145329840845597,
3.7103891372680664e-06, 0.00639992649666965, 1.2036412954330444e-05, 5153.56384468078
6, 590400, -4.284083843231201e-08, -0.036748810826905856, 1.0617077350616455e-07, 0.93
50026688541725, 135.53125, -1.926313917701005, -8.134624554050025e-09, 83, -1.59292349
45145378e-10),
           (None, None, None, None, None, None, None, None, None, None, None, None, No
ne, None, None, None, None, None, None, None, None, None, None, None, None, None, Non
e),
           (None, None, None, None, None, None, None, None, None, None, None, None, No
ne, None, None, None, None, None, None, None, None, None, None, None, None, None, Non
e),
           (1321, 1, 0, -5.587935447692871e-09, 756, 590400, 0.0, -1.7053025658242404e
-12, -4.227878525853157e-05, 164, 36.46875, 5.031281001620245e-09, -0.667327573969190
2, 2.0917505025863647e-06, 0.016684173489920795, 6.759539246559143e-06, 5153.662576675
415, 590400, -8.381903171539307e-08, -2.1406362819791758, 2.7008354663848877e-07, 0.95
54143961588313, 247.03125, 1.1382087838559416, -8.512497436829482e-09, 164, 4.55376111
100008e-10),
           (None, None, None, None, None, None, None, None, None, None, None, None, No
ne, None, None, None, None, None, None, None, None, None, None, None, None, None, Non
e),
           (None, None, None, None, None, None, None, None, None, None, None, None, No
ne, None, None, None, None, None, None, None, None, None, None, None, None, None, Non
e),
           (None, None, None, None, None, None, None, None, None, None, None, None, No
ne, None, None, None, None, None, None, None, None, None, None, None, None, None, Non
e),
           (None, None, None, None, None, None, None, None, None, None, None, None, No
ne, None, None, None, None, None, None, None, None, None, None, None, None, None, Non
e),
           (None, None, None, None, None, None, None, None, None, None, None, None, No
ne, None, None, None, None, None, None, None, None, None, None, None, None, None, Non
e),
           (1321, 1, 0, -2.7939677238464355e-09, 250, 590400, 0.0, 5.343281372915953e-
12, 0.00042775925248861313, 169, 96.25, 4.806628786810908e-09, -0.33825178554457946,
4.82611358165741e-06, 0.00912147096823901, 2.5816261768341064e-06, 5153.549310684204,
590400, -3.5390257835388184e-08, 1.113253290050884, -1.5832483768463135e-07, 0.9617965
225714421, 329.125, 2.3960378657602988, -8.5839289836687e-09, 169, 4.064455015151444e-
10),
           (None, None, None, None, None, None, None, None, None, None, None, None, No
ne, None, None, None, None, None, None, None, None, None, None, None, None, None, Non
e),
           (None, None, None, None, None, None, None, None, None, None, None, None, No
ne, None, None, None, None, None, None, None, None, None, None, None, None, None, Non
```

```
e),
           (1321, 0, 0, -1.0244548320770264e-08, 490, 590400, 0.0, -3.410605131648481e
-12, -0.00015218276530504227, 191, -113.625, 4.726268296616789e-09, -1.86941354492065
7, -5.8300793170928955e-06, 0.005856757517904043, 3.593042492866516e-06, 5153.59786033
6304, 590400, 7.450580596923828e-08, 2.13047496418236, -3.3527612686157227e-08, 0.9615
654590495761, 309.59375, -2.8131870016047764, -8.388563703063442e-09, 191, -2.83583240
9516913e-10),
           (None, None, None, None, None, None, None, None, None, None, None, None, No
ne, None, None, None, None, None, None, None, None, None, None, None, None, None, Non
e),
           (None, None, None, None, None, None, None, None, None, None, None, None, No
ne, None, None, None, None, None, None, None, None, None, None, None, None, None, Non
e),
           (1321, 1, 0, -1.1641532182693481e-08, 487, 590400, 0.0, 1.9326762412674725e
-12, 0.00010179728269577026, 45, 97.15625, 5.083426030812873e-09, -0.4479158230233504,
5.070120096206665e-06, 0.009839744423516095, 2.421438694000244e-06, 5153.631958007812
5, 590400, 1.7508864402770996e-07, 1.0810757412696768, -5.960464477539063e-08, 0.94835
13398970446, 324.59375, -3.138082767868972, -8.706434086497957e-09, 45, 3.760870941084
7717e-10),
           (1321, 2, 0, -1.816079020500183e-08, 473, 590400, 0.0, 7.958078640513122e-1
3, 2.2018328309059143e-05, 44, -115.71875, 4.8162720456342025e-09, 2.620382956828128,
-5.939975380897522e-06, 0.004725061357021332, 3.1869858503341675e-06, 5153.68164443969
7, 590400, 8.009374141693115e-08, 2.139478035607851, -7.450580596923828e-09, 0.9590182
6119999, 317.34375, -1.5072976185502522, -8.352847929643834e-09, 44, -3.84658879729183
2e-10),
           (None, None, None, None, None, None, None, None, None, None, None, None, No
ne, None, None, None, None, None, None, None, None, None, None, None, None, None, Non
e),
           (None, None, None, None, None, None, None, None, None, None, None, None, No
ne, None, None, None, None, None, None, None, None, None, None, None, None, None, Non
e),
           (None, None, None, None, None, None, None, None, None, None, None, None, No
ne, None, None, None, None, None, None, None, None, None, None, None, None, None, Non
e),
           (1321, 0, 0, -6.05359673500061e-09, 499, 590400, 0.0, 1.0231815394945443e-1
2, 2.2032298147678375e-05, 140, -61.5625, 3.762656729755752e-09, 1.1289011782368077, -
2.8889626264572144e-06, 0.0164188725175336, 1.2412667274475098e-05, 5153.575479507446,
590400, -1.5832483768463135e-07, -3.133395532992539, 1.6391277313232422e-07, 0.9865527
654711853, 158.28125, 0.6846128449186781, -7.705320957546331e-09, 140, -6.893144269984
435e-11),
           (None, None, None, None, None, None, None, None, None, None, None, None, No
ne, None, None, None, None, None, None, None, None, None, None, None, None, None, Non
e),
           (None, None, None, None, None, None, None, None, None, None, None, None, No
ne, None, None, None, None, None, None, None, None, None, None, None, None, None, Non
e),
           (None, None, None, None, None, None, None, None, None, None, None, None, No
ne, None, None, None, None, None, None, None, None, None, None, None, None, None, Non
e),
           (None, None, None, None, None, None, None, None, None, None, None, None, No
ne, None, None, None, None, None, None, None, None, None, None, None, None, None, Non
e),
           (None, None, None, None, None, None, None, None, None, None, None, None, No
ne, None, None, None, None, None, None, None, None, None, None, None, None, None, Non
e),
           (None, None, None, None, None, None, None, None, None, None, None, None, No
ne, None, None, None, None, None, None, None, None, None, None, None, None, None, Non
e)],
          dtype=[('weekNumber', 'O'), ('accuracy', 'O'), ('health', 'O'), ('T_GD',
'O'), ('IODC', 'O'), ('t_oc', 'O'), ('a_f2', 'O'), ('a_f1', 'O'), ('a_f0', 'O'), ('IOD
E_sf2', 'O'), ('C_rs', 'O'), ('deltan', 'O'), ('M_0', 'O'), ('C_uc', 'O'), ('e', 'O'),
('C_us', 'O'), ('sqrtA', 'O'), ('t_oe', 'O'), ('C_ic', 'O'), ('omega_0', 'O'), ('C_i
s', 'O'), ('i_0', 'O'), ('C_rc', 'O'), ('omega', 'O'), ('omegaDot', 'O'), ('IODE_sf3',
'O'), ('iDot', 'O')])
```

Future Scope

The Limitations of this Implementation is a work of Future Scope By working for following Improvements:

1. The Porting of System into C++ library implementation for real time execution and multi-threading capabilities to reduce the processing overhead and time.
2. The Integration of RTL-SDR lib into the system implementation to remove the need of storing the IQ data for processing.
3. To implement the Psuedo Ranging Algorithm to compute user location.