DESIGN OF AUTOMATIC DEPENDENT SURVEILLANCE-BROADCAST *(ADS-B)* RADAR USING SOFTWARE DEFINED RADIO

Submitted by

Baidarvi Guha Thakurta
Roll No – *91/RPE/131003*
Reg. No – *A01-2112-0501-10*

Arnav Mukhopadhyay
Roll No – *91/RPE/131016*
Reg. No – *133-1121-0041-13*

Under the Supervision of

Prof. Abhirup Das Barman

UNIVERSITY OF CALCUTTA
INSTITUTE OF RADIOPHYSICS AND ELECTRONICS
Sisir Mitra Bhavan, 92 APC Roy Road,
Kolkata – 700009, West Bengal India

TO WHOM IT MAY CONCERN

This is to certify that, ***Arnav Mukhopadhyay (Roll – 91/RPE/131016)*** and ***Baidarvi Guha Thakurta (Roll- 91/RPE/131003)***, students of ***B.Tech. Semester-6 in Radio Physics and Electronics***, have successfully completed the project work entitled, "***DESIGN OF AUTOMATIC DEPENDENT SURVEILLANCE-BROADCAST (ADS-B) RADAR USING SOFTWARE DEFINED RADIO***" in partial fulfilment of the requirements for the degree in Bachelor of Technology in Radio Physics and Electronics from University of Calcutta, under my supervision and guidance.

Date:

Place: Kolkata

(Abhirup Das Barman)
Professor

ABSTRACT

Automatic Dependent Surveillance-Broadcast (ADS-B) is a modern surveillance mechanism which forms the part of next generation radar systems [1], extending the radar coverage for detecting e.g. planes beyond the line of sight. The essential characteristic of ADS-B is that, it can be implemented easily by Software Defined Radio. This project aims to develop a complete prototype radar system using Realtek Software Defined Radio (RTL-SDR) and a custom designed digital signal processing software to extract digital data frame from the received radio frequency samples by SDR. The design is mainly targeted to reduce cost of the prototype radar, so that it can be implemented at home or small office.

In this project we design several interconnected modules na mely,J-Pole antenna, Digital Signal Processing software and software based network interfaces which can be interfaced seamlessly with standard available radar display software like Virtual Radar, ADSBScope, etc. The J-Pole antenna is fabricated to work at 1090 MHz, and selected because of its low cost, high gain and sturdiness. The digital signal processing software designed for frame decoding is capable of real-time operation. The proposed implementation consumes low processor power but provide high data-throughput even with large volume of data. The data rate used for this design is 2 MSPS, which can be processed through the designed software using very less CPU processing power by use of SIMD/SSE4.2 vector instruction sets. The benefit of which is that, the software, unlike other standard ADSB software like ADSB#, can be easily deployed on low power computers. The software network interface is designed to make the designed software interoperate with standard display software. The detection of aircraft locally can be integrated with global ADS-B systems by Internet-of-Things (IOT) approach.

Keywords- *Next Generation Radar Systems, Automatic Dependent Surveillance Radar-Broadcast, Digital Signal Processing, SIMD, SSE4.2 vector processing, Internet of things*

Contents

OBJECTIVE

The major goal of this project is to design a Physical Layer (ES0) Decoder for receiving Automatic Dependent Surveillance- Broadcast, and hence forth, establish a prototype of complete ADS-B radar on a computer. In order to attain this, the specific sub-objectives are:

1. Design of the Physical Layer (ES0 [1]) decoder
    a. Design a J-Pole antenna so that the electromagnetic line of sight is comparably high [6].
    b. Implementation of a custom signal processing software capable of handling real-time data at high speed and support asynchronous data transfer from Realtek Software Defined Radio (RTL-SDR).
    c. Maintain receiver coherence in software, using preamble based synchronization method [2, 3].
    d. Reduce the processing overhead for invalid decoded frame, by use of parity checking for CRC validation [3], using highly efficient polynomial method [7].
    e. Decoding logic for data-frame encoded with Manchester line coding [2], and packing each 8 bits into a single byte frame.
    f. Frame validation is implemented by checking each frame for data-type used and reject those which are invalid.
    g. The system designed will be power conservative and capable of being deployed on Ultra-Mobile Windows PC (UMPC), due to usage of SSE instructions [12], still maintaining real-time nature with least possible data-loss.
    h. Comparative study of designed application with a standard open-system ADSB application (ADSB# [8]).
2. Prototype of complete ADS-B Radar on computer [8]
    a. Study of various data exchange format used by standard display application such as ADSB-Scope and Virtual Radar.
    b. Implementation of a standard data exchange format for real-time streaming of decoded data frame to display application.
    c. Implementation of TCP network based communication socket for transferring data from the decoder application to display application. Hence, implement a complete prototype ADS-B radar system.

INTRODUCTION

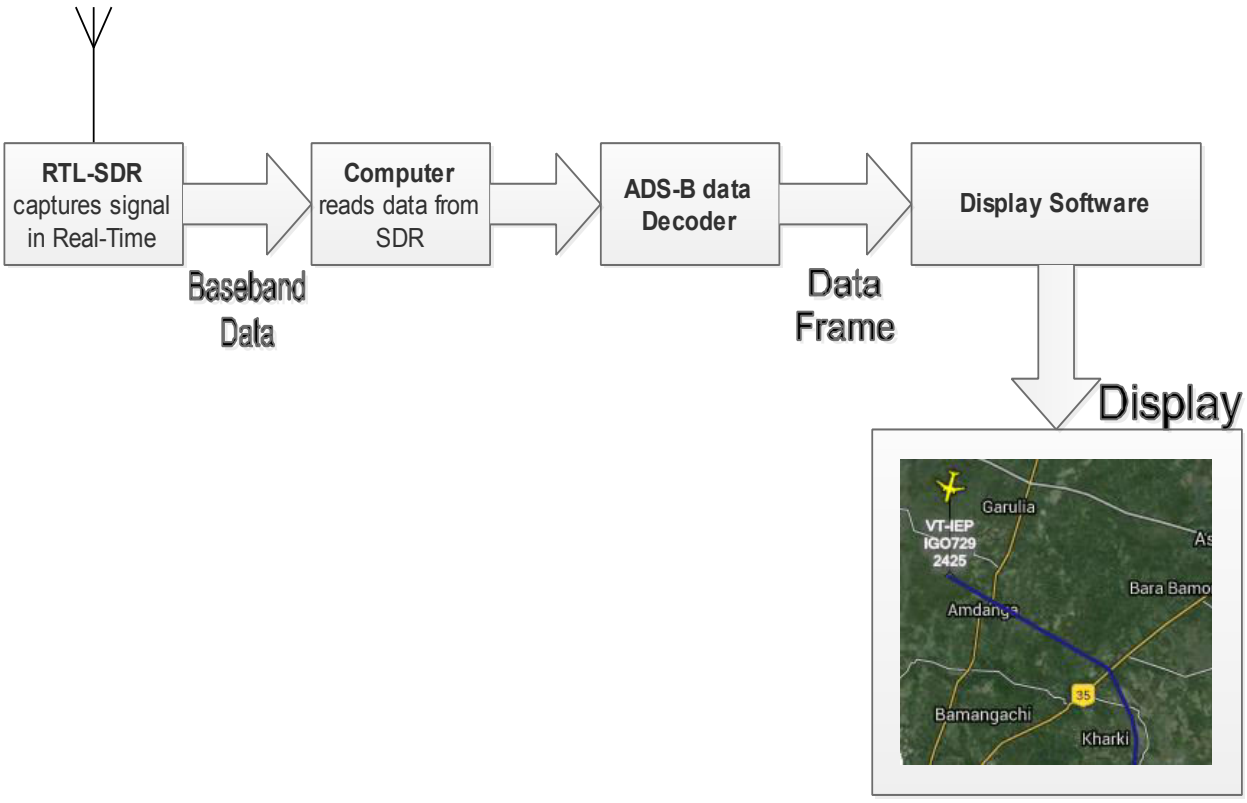


FIGURE-1: Complete flowgraph of ADS-B Radar

This thesis implements an ADS-B physical layer decoder on a computer that will use the cheap 20 $ RTL-SDR hardware to capture and decode the RF data and stream the data to a standard display software. The ADS-B is a modern way of monitoring air traffic control without using bulky radar installations and within few years, it had been accepted as an extended support to the primary radar installations [1]. Hence, this system had caught the eyes of many researchers as well as amateur hobbyist, hence in a matter of few years, many such projects have come up [8]. Some projects have used microcontroller [9], others have used a completely dedicated radio [10], and a very few projects have used software defined radios to capture real time ADS-B data packets [8]. This software aims a similar feature of implementing the system with Software Defined Radio and a custom made Digital Signal Processing software, with a motivation to learn the signal format and frame format used for data transmission, so as to implement a power conservative software that will have capability to handle bulk data using Intel SSE (vector) instructions and parallelism, resulting in high data output per clock cycle [5]. Further, a polynomial based CRC verification is implemented to drop frames that are invalid [7], this let us preserve network data bandwidth, and prevent overloading of the display software with noise [2]. Also, synchronization of receiver module with the aircraft transmitter is achieved using preamble detection mechanism [2, 3], which allows us to reduce the CPU cycles required by the frame detection algorithm. This is done not only to ensure that coherence is achieved even with propagation effects, that occurs in Amplitude Shift Keying used by ADS-B transmitter; but also makes CPU consumption

linearly dependent on the number of ADS-B frames received, as the time complexity of the frame detection algorithm is linear. But still, the usage of several internal optimizations and compiler auto-vectorization, which substitutes SSE instructions wherever possible, leads to a maximum CPU usage of 12 %. Under average load condition, the CPU usage will hardly cross 3 %.

This software, unlike other commercially or freely available software, allows users to use any compatible RTL-SDR hardware and is still easy to setup and use. The software is written for Microsoft Windows operating system, and can be executed not only on Desktop and Laptop computers, but also on low power Windows based ultra-mobile PC (UMPC) because of its extensive uses of SSE instructions [12]. Hence, the cost to implement a complete system is not more than Rs. 8000. And it should also be noted that, there is no legal hassle against system implementation, as the complete system is designed using open system [11] specifications that are available for designers and researchers to consult.

DESIGN

RTL-SDR AND DATA CAPTURE

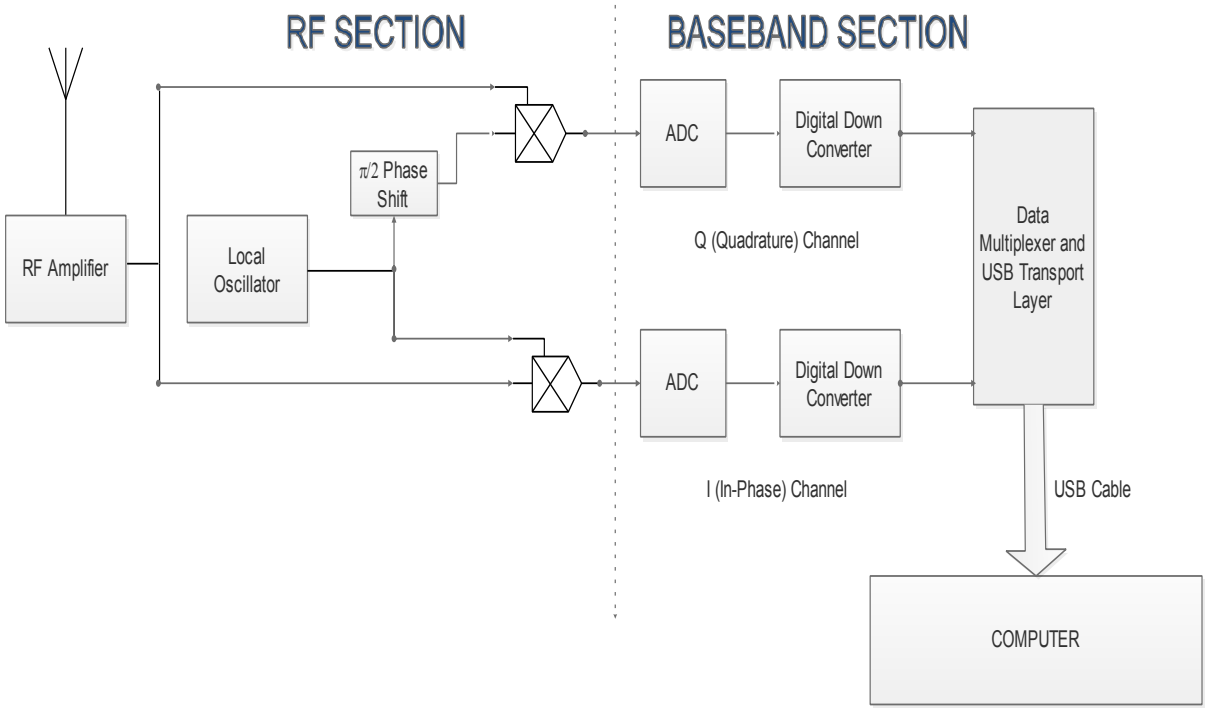


FIGURE 2 - RTL-SDR Block diagram

Introduction

Realtek Software Defined Radio or, RTL-SDR in short, is the data source for the system. It receives the radio frequency signal from the antenna and amplifies these weak signal by the internal Radio Frequency amplifier and pass it to a Quadrature Demodulator [13]. The quadrature demodulator is composed of a Multiplier (with an internal Low-Pass filter) [13]. Each of the multiplier gets the same local oscillator frequency, which is equal to 1090 MHz for ADS-B reception. Although the oscillator frequency is the same, they have a phase difference of 90°. This phase difference results in the output analog signal, which are always is phase quadrature with each other, and hence, named analog I and Q, representing In-Phase $(0° \; phase)$ and Quadrature-Phase $(90° \; phase)$, respectively. These are then digitized using an analog-to-digital converter, which is capable of sampling at 3.2 MHz and the digital signal represented the baseband spectrum of the received radio frequency signal, where the local oscillator had been tuned to [13].

It should be noted that, the ADC can sample at sample rate of 3.2 MSPS but this is too much traffic to be carried by the USB 2.0 data channel. Hence, a digital down converter resamples the signal to a much lower sample rate. The sample rate of the digital down converter is set by the application program running on the computer. In the case of ADS-B system, the sample rate is selected to be 2 MSPS, which means 2 million Quadrature Sample, each of which is 8 byte in length (*I sample is 8 byte and Q sample is 8 byte*) [13].

The digital down converter limits the available data bandwidth by passing digitized spectrum through a 16-tap FIR filter, which limits the spectrum length to Nyquist bandwidth corresponding to desired sample rate, before passing it to a down conversion circuitry. The output of the down-converter is at the desired sample rate, and the USB channel is capable of carrying the data blocks. Hence, I and Q digital samples are multiplexed to form a 16-bit data, and then accumulated within USB data frame and transmitted to the computer for further processing.

Synchronous and Asynchronous Data Transfer Modes

The USB data transfer is done serially through a bidirectional bus. The bus is shared among many other USB devices, which are connected in a master-slave connection. The data transfer is done in a bulk mode, where a large volume of data is exchanged between the end-point entities. This requires a specific size buffer on both entities involved in communication. RTL-SDR is one of such device which has the capability to transfer data to the computer in bulk.

Also, there are two data transfer modes in RTL-SDR –

- Synchronous mode
- Asynchronous mode

Synchronous Mode

In synchronous data transfer mode, the RTL-SDR does the conversion and pushes the converted data onto an internal FIFO buffer. When the computer needs to read some data, it will poll the RTL-SDR and request for the data. The RTL-SDR then send the content of the FIFO at that moment to the computer. But if requested size of data to be read is larger than the FIFO size, multiple read operations are performed over the USB and hence the function will block till all the data is read. In case of any error that occurs during this operation, the function will exit immediately with partial data been read or even no data being read.

Hence, with this mode, it is the duty of the application running on the computer to periodically read the data from the RTL-SDR. But due to the inherent delay involved in the user level read operation, there may occur cases where a part of the sample data may be lost. Hence, this mode cannot completely provide real-time operation.

Also, the data loss during the read operation will also result in failure to detect proper ADS-B frames, as preamble or frame parity may be damaged during data read operation. Hence, this mode is not used in the design of ADS-B decoder.

Asynchronous Mode

This mode of read operation is real time in nature, and does not result in data loss. This is because, in asynchronous data mode, it is not the duty of the application program to periodically read the sampled data. Here, the device driver itself reads the sampled data asynchronously. A specified memory size is allocated by the device driver and the RTL-SDR is requested to fill the specified driver memory repeatedly. Hence, this data reading is much faster than the synchronous mode and also no sampled data is lost during read, because the read is actually performed in the kernel level rather than at user level. Therefore, for the data capture of the ADS-B system the asynchronous read mode is best suited.

Setting up RTL-SDR.

The RTL-SDR connects to USB 2.0 port of the computer. When it is connected for the first time, Microsoft Windows operating system will load the default driver, which will not work with this application. Hence, after the system successfully installs the RTL-SDR, run *zadig* and install the WinUSB driver [14]. This is the required step. After, the WinUSB driver is setup, restart the system. Once the system restarts, the RTL-SDR can be successfully used as Software defined radio, and with the presented application too.

J-POLE ANTENNA

Introduction

Antenna is a crucial part of the whole system, which is responsible for intercepting the electromagnetic signal from space. Hence, for proper reception and increasing range of reception, a selection of proper antenna is required. The very simplest antenna that can be used for reception of ADS-B signal is a small piece of bare wire, but the reception range is limited to aircrafts in the vicinity. Therefore to increase the range a more powerful antenna is required. The antennas considered for practical usage in various projects are Half-wave dipole antenna, Franklin collinear dipole antenna, Coaxial Collinear antenna [15]. Some advanced projects have also considered the use of J-Pole antenna [16]. In this project, a modified version of the simple J-Pole antenna [16] is used, so that the coverage is high. The antenna designed for this project is custom made, and in this section, the design specifications are discussed and also by using this antenna, aircrafts from over 35000 feet and 100 km away can effectively be tracked. But there is also a problem of gain overload of the RTL-SDR, which occurs when an aircraft in vicinity, transmits a signal which is powerful enough to mask the data frames arriving from far away, resulting in the aircraft at further distance vanishing from the display. This problem cannot be removed as this is a problem with any radio receivers. But the problem can eliminated by using distributed ADS-B setup forming a sensor network environment, which is beyond the scope of this paper. Hence, the discussion in this section is limited to the design and application of the custom made J-Pole antenna for receiving ADS-B signal, whereby the receiver is more sensitive and able to track aircrafts at distances far away from the antenna.

J-Pole Antenna

J-Pole antenna is special antenna which is DC short and can be easily matched to a 50 Ω feed line [17]. It has a gain of 2.5 – 3 dBi [18] and finds its application in broadband antenna. Comparing its size it is portable and small, such that is can be carried in a bag-pack. Although there are benefits with this antenna, but it must be carefully noted that the antenna is not tied to the ground, as any ground connection becomes a part of the antenna itself and causes tuning problem. The gain of the antenna is not uniform, it is maximum on one side and minimum on the other [17].

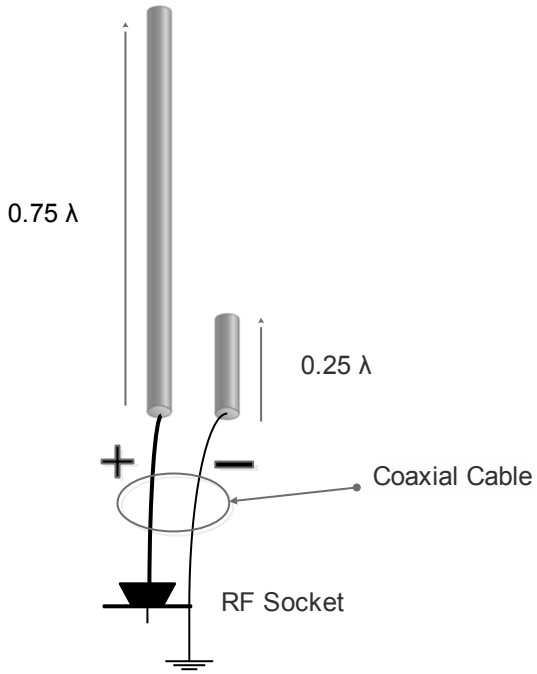
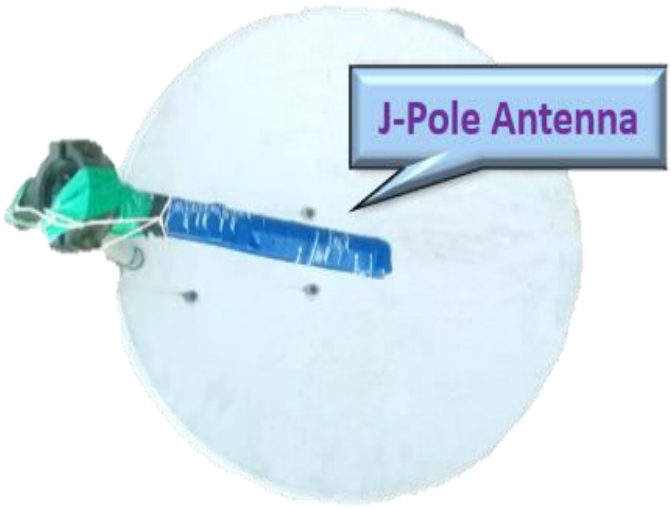


FIGURE 3 - J-Pole Antenna



Consider the above J-Pole antenna design. The name of the antenna derives from the physical structure of the antenna. The antenna has two sections – a $0.75\ \lambda$ section which is connected to the positive end of the socket and uses internal conductor for the connection, when coaxial cable is used. The second section is$0.25\ \lambda$, which is connected to the ground or earth section of the RF socket. The length of each section is calculated as shown here under –

ADS-B frequency: $f = 1090\ MHz$

Wavelength: $\lambda = \frac{c}{f} = 27.5\ cm$

Short Section: $l_{-} = 0.25\ \lambda = 6.9\ cm$

Long Section: $l_{+} = 0.75\ \lambda = 20.6\ cm$

Coverage

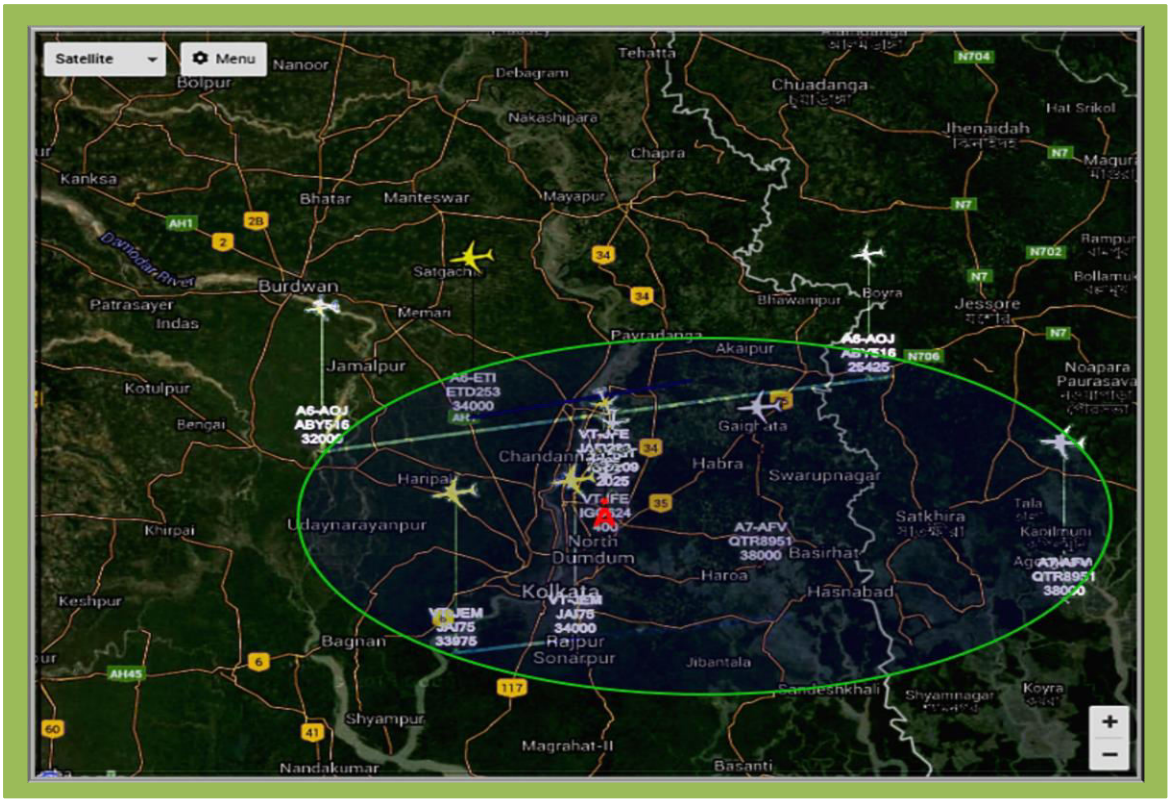


FIGURE 4 - Overlay image showing the coverage of J-Pole antenna

The figure above can be used to determine the coverage range of the antenna designed. From overlaying several images of the aircrafts onto a single image, and drawing an ellipse covering the position of the farthest aircraft, we see that the antenna covers a fairly large area. Considering the location of antenna at *Barasat (Latitude: 22.34^o, Longitude: 88.15^o)*, it can be seen that the coverage on the right-side is more that the coverage on the left-side. But the overall coverage is very high, considering the nearby obstructions of the locality. Hence, this antenna is usable in small surveillance system, and is also cheap and easy to design.

ADS-B TECHNICAL SPECIFICATIONS

Introduction

Automatic Dependent Surveillance Broadcast (*ADS-B*) is the modern Radar system that extends the capability of existing Radar. It forms the part of NextGen surveillance system developed by USA. It is also operational in India, after 2015. It is projected to be implemented by all countries by the year 2020.

The main advantage of the new system is the use of GPS satellite navigation to pin point aircraft data in real time, which increases positional awareness not only to the ground controller but also the nearby aircrafts. The system is operated independently by a computer and transmits signals periodically on a single frequency of 1090 MHz, which is time shared between multiple aircrafts. All Civilian Air traffic are mandated to broadcast these periodically without outside intervention. But on request by Ground Controller, the information may be re-broadcast again.

The transmission system is Asynchronous yet periodic, because although the information are broadcasted periodically, but since the channel is time-shared, the aircraft checks for presence of traffic on the channel to avoid collision of data. The receiver can maintain coherence with the ADS-B data traffic, by synchronizing and training their clock with the Preamble bits, which is a sequence of bits encoded with Return to Zero (RZ) line coding. Once preamble is detected, and clock synchronized, the rest of the received data can be processed for frame data.

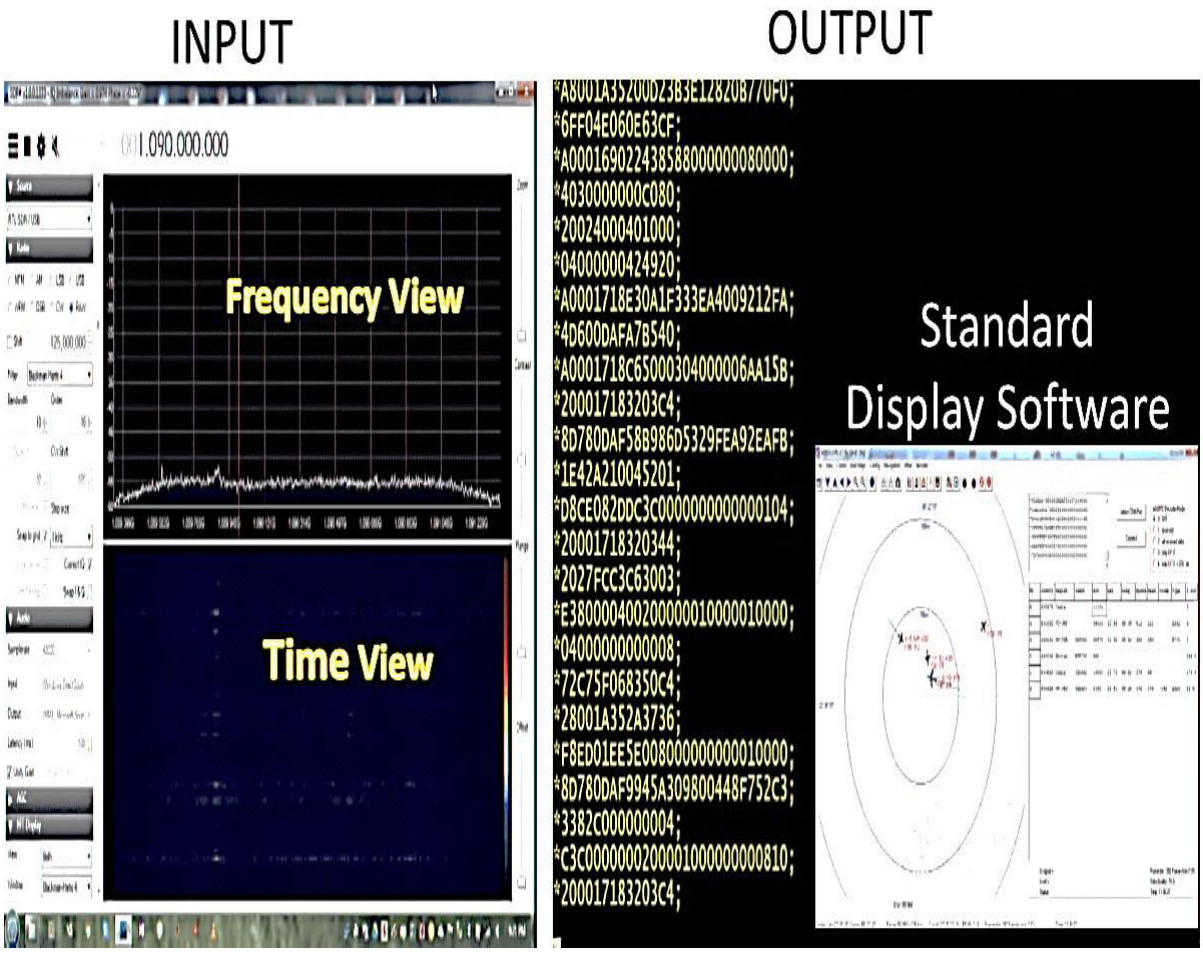
The modulation used by ADS-B is Amplitude Shift Keying that is transmitting the carrier when the data bit is high and stop carrier when data bit is low. Hence, the preamble detection will require the threshold detection technique to establish amplitude level for 1 and 0, so that the sequence of preamble training bits are properly decoded. The Amplitude Shift Keying was selected due to its simplicity, as well as its ability to perform better than other form of digital modulation in situations where fading is high and multi-path effect results in

The data frame bits follow the preamble bits. The data frames are coded with Manchester line coding and on transmission looks similar to Phase-Shift Keying. Hence, to detect Phase-Shift Keying, requires receiver coherence in order to lock to the transmitter carrier. Any error in carrier phase will cause the data frame to become invalid and rejected. Hence in order to circumvent this, differential phase detection or slope method of detection is used, so that phase coherence error is eliminated. In this method of detection, only burst noise or packet interference may cause bit error, otherwise, multipath has limited effect on the detection process.

Technical Specifications

Transmitted Frequency:	1090 MHz
Modulation:	Amplitude Shift (On/Off) Keying
Modulation Type:	Digital
Transmitter Power:	Maximum 430 W (burst)
Data Transmission Mode:	Asynchronous
Receiver synchronized using:	Preamble bits
Data Start Indication:	Preamble bits
Preamble Line Coding:	Return to Zero (RZ)
Preamble Bits:	[1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0]
Data Frame Line Coding:	Manchester Line Coding
Data Frame Types:	Short Frame, Long Frame
Frame type detected using:	Data Format Byte (1st byte of received frame)
Short Frame Length:	56 bits
Long Frame Length:	112 bits
Frame Parity Checking:	CRC bits appended to data frame
CRC checking Polynomial:	0xfffa0480

ADS-B DATA PROCESSING

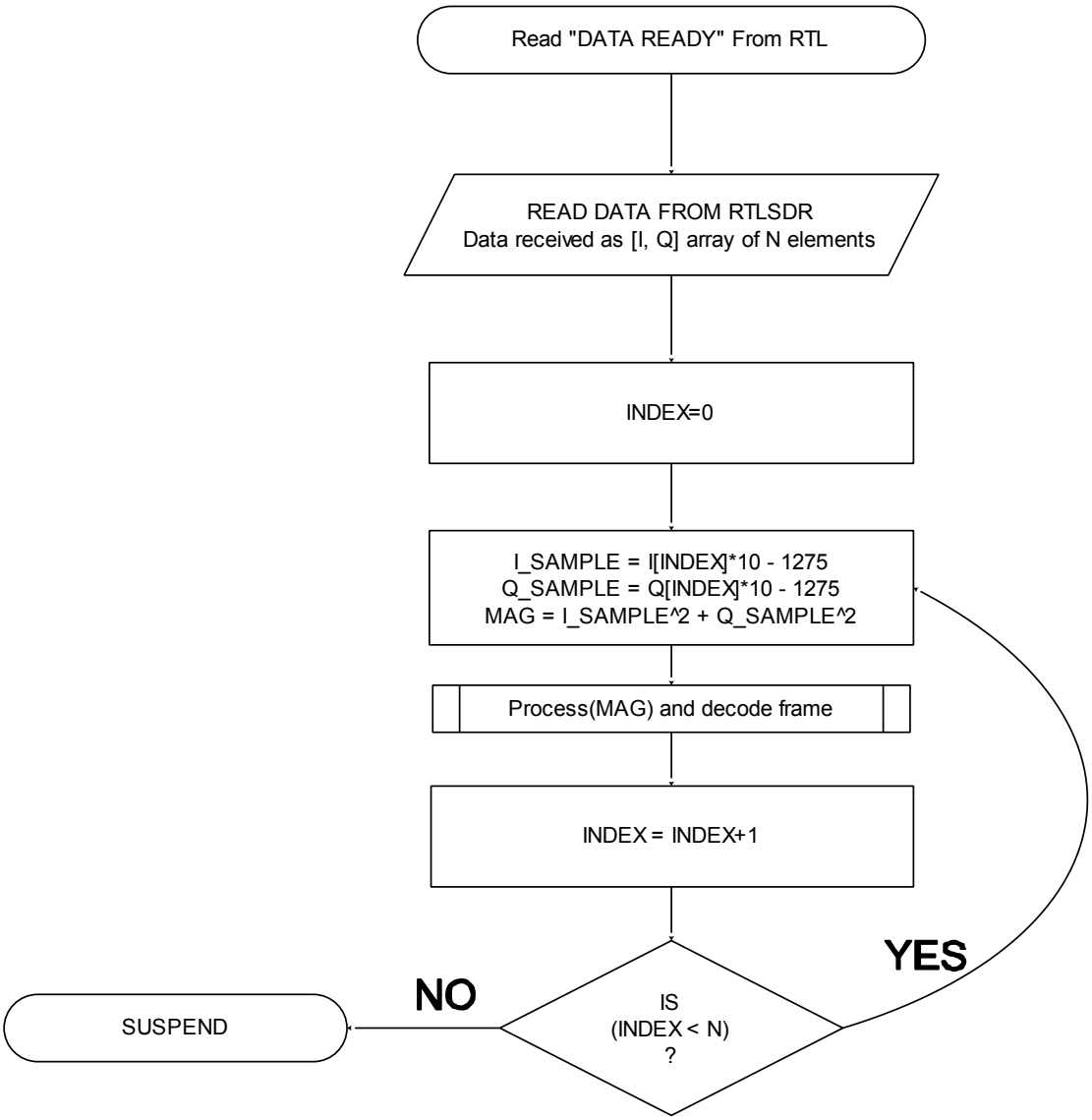


Introduction

The aim of this section is to look deeper into the actual signal processing that is used to recover the actual data from the input radio frequency signal, as shown in the figure above. The input signal is the digital equivalent of the amplitude component received at 1090 MHz. The quadrature demodulator sends I, Q samples to computer, but since ADS-B uses amplitude shift keying, the received signals needs to be converted to magnitude, before it can be further processed. Generally, this magnitude conversion is done using the relation $\sqrt{(I^2 + Q^2)}$, which requires floating point operation. But since floating point operation requires more clock cycles, the calculations are modified to find magnitude using integral operations, which are further accelerated using SSE4.2 operations, so that the data throughput is further increased.

The magnitude are then scanned for the presence of preamble bits. Unless a preamble is detected in the received signal, the magnitude signal is discarded. The preamble detection logic utilizes known preamble method, so as to terminate the detection immediately when the received signal does not matches the preamble pattern, which saves a lot of time. Once preamble is detected, the rest of the received magnitude is used for detection of the data frame, called *extended squitter*. Then packed bytes are CRC validated to check for data integrity. Then if the frame is valid, the packed bytes are send to the display software through TCP/IP software network.

Magnitude Conversion



The IQ signals sent to computer from RTLSDR is 8 bit unsigned data, with a data in range of 0 to 255. Hence, it is converted to the range of -1275 to +1275, by using the formula:

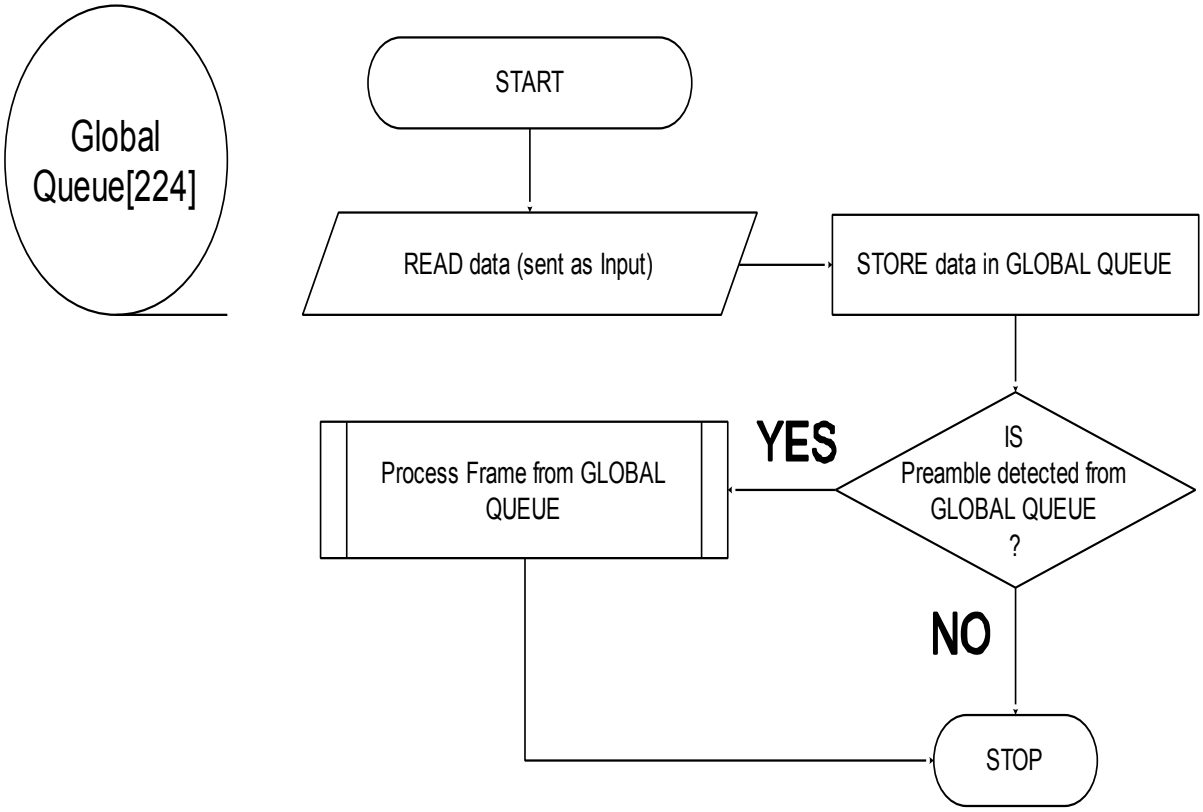
$$I_1 = I \times 10 - 1275$$

$$Q_2 = Q \times 10 - 1275$$

From the new values, which are themselves integer, now the magnitude or more specifically, square of magnitude is obtained using the relation, $mag = {I_1}^2 + {Q_1}^2$, which is again an integer. Now the magnitude mag is always positive in the range of 0 to 3251250.

Using the above flowchart, each and all of the IQ data are converted to integral magnitude which is passed down for further processing.

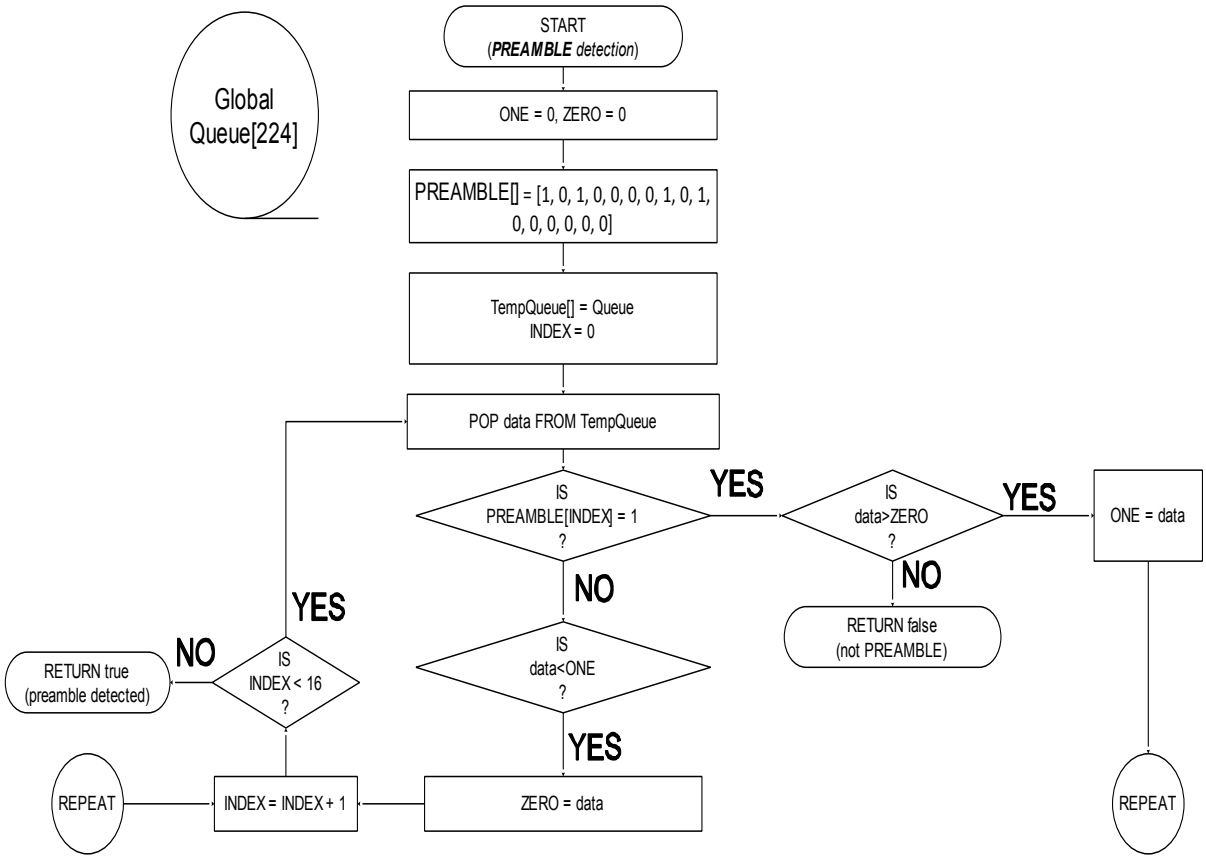
Magnitude processing



Each of the magnitude data is first pushed into a global queue, which allows for detecting preamble in each and every received sample. The global queue used in this case, is a circular queue capable of storing 224 magnitudes. Then the queue is traversed for a valid preamble, and only when a preamble is detected then the queue is passed on for reading the data frame.

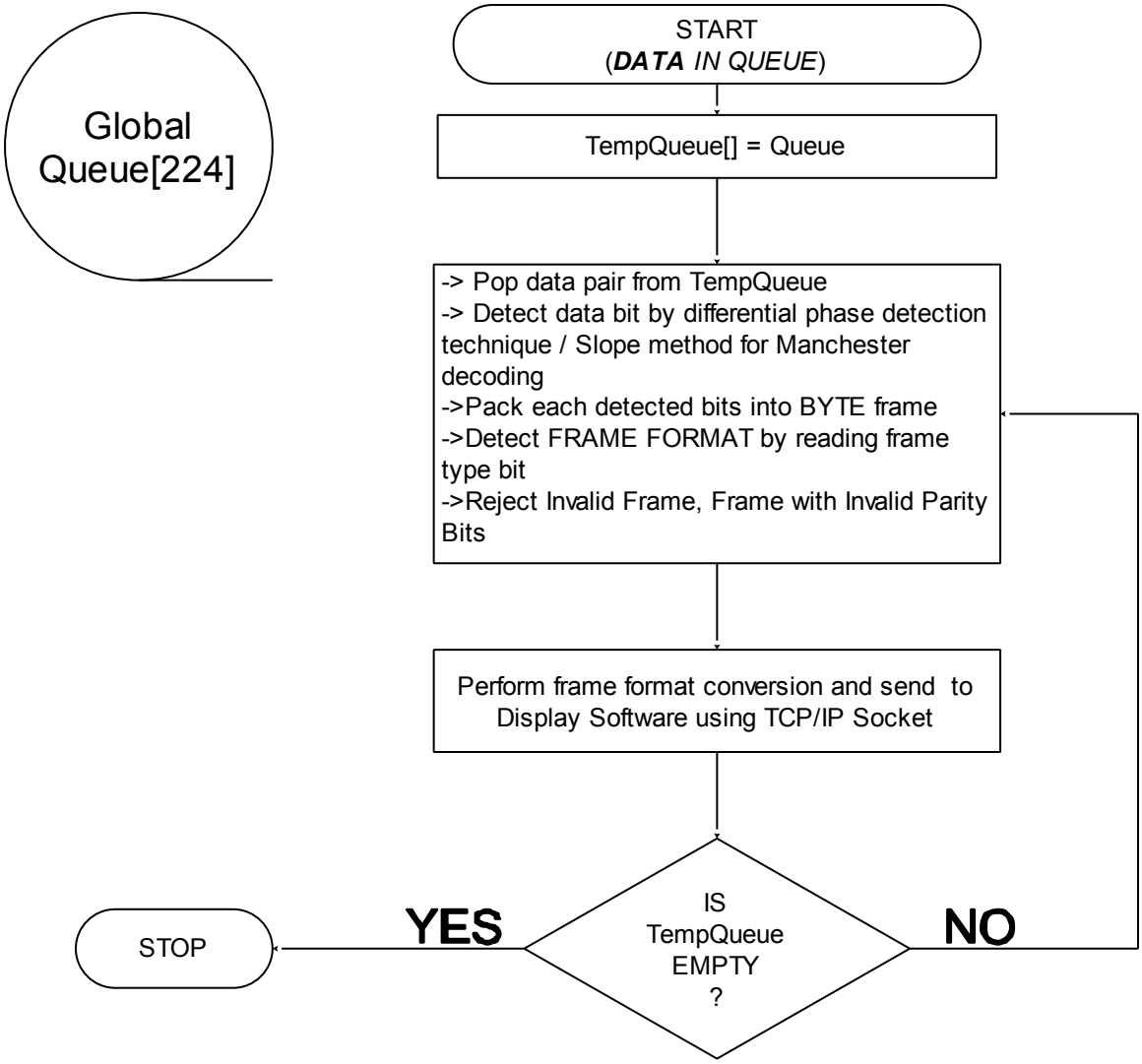
This algorithm is very fast as it utilizes only integer calculations, also the queue is aligned to 128 byte boundary, which allows for SSE optimization, running the calculations faster.

Preamble Detection



The preamble detection uses known preamble detection method, instead of threshold method, so as to make it capable of detecting very weak signals which will not pass when using threshold method. The preamble is a 16 bits of data, represented as 1 or 0. The algorithm declares two variables *one* and *zero*, all initialized to same value 0. Then global queue is copied to a local (*temporary*) queue. Data is popped from the queue and compared against the selected preamble bit. If the preamble bit is 1 and the magnitude is more than magnitude of zero, then this magnitude data is saved as magnitude of *one*, otherwise, the preamble bit is not 1, which means that it is not a proper preamble. Same is done for zero. This process returns *true* only when a complete preamble is properly detected.

ADS-B Frame Detection and Packing



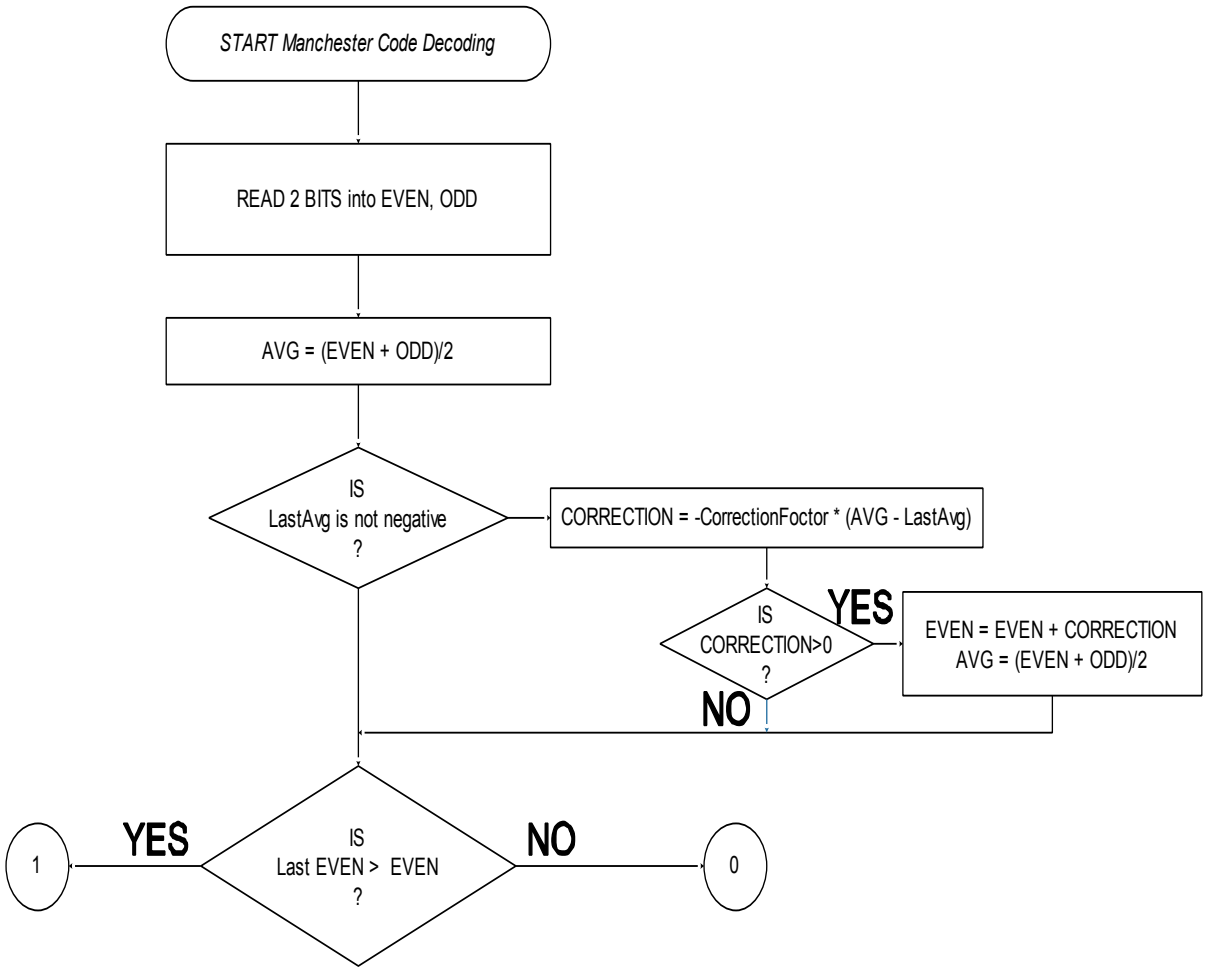
Once a proper preamble is detected, the next 52 or 112 bits will represent data frame called *extended squitter*, which are coded using Manchester line coding. The detection of Manchester line coding is discussed in the next section. Once sufficient number of bits are detected, they are packed into bytes. Then the first data byte called *frame[0]* contains the data format. ADS-B data format are classified into two types:

- Short frame (52 bits)
- Long frame (112 bits)

If data format bit pattern value greater than or equal to "1000 0XXX", then the remaining data is Long frame, otherwise it is a Short frame.

When all frames are detected, then last 3 bytes (MSB) are checked, so that they are all non-zero. If these bytes are zero, the whole frame is dropped, as there is no data present. Also, in accordance with ADS-B standards the valid data formats bits (=frame[0] bits 3-8) values are 0, 4, 5, 11, 17, 20, 21. Hence, before streaming the data to the display software, the frames which are of the mentioned data format will be kept and the rest will be dropped, decreasing the undesirable traffic and bandwidth wastage. Further, CRC check on the data frame is done to drop all frames that fails the check.

Manchester Decoding



In Manchester code representation, a bit 1 is represented by (1, 0) bit pair, whereas 0 is represented by a (0, 1) bit pair. These are detected using slope detection method. If the slope is rising then it must be 1 whereas a decreasing slope is 0. But there lies a problem caused by fading where slope may not be adequately high enough for comparison. Hence, a correction is applied to even bit every time the previous average bit is positive. This lets even bit to be moved to a value much higher than the noise floor. Now comparing successive even bits, the value being 1 or 0 can be determined.

CRC Checking

CRC checking is done to ensure that the decoded digital data is valid. Frame parity error may occur due to interference, noise, bad weather condition, partial data loss, etc. But CRC check cannot be used to recover the damaged bits. The only solution, in case of CRC failure is to drop the frame entirely.

CRC checking can be done using table method and polynomial method. In table method, too many memory seek will cause too many idle time and decrease data processing speed. But polynomial method decreases this by using only SHIFT and XOR operation which is very fast and does not require any memory seek to be performed. Hence, the polynomial method is chosen for the design.

The ADS-B polynomial used is0xfffa0480, which XORed with frame bits and then SHIFT, to obtain the CRC values, which are XORed ultimately to the end bits and obtain 0 only whence the frame data is valid.

There are 2 frames – short frame and long frame, which are checked using following logic:

```
uint32_t getShortFrame_Parity(uint8_t aFrame[])
{
    ssealign uint32_t f0 = aFrame[0];
    ssealign uint32_t f1 = aFrame[1];
    ssealign uint32_t f2 = aFrame[2];
    ssealign uint32_t f3 = aFrame[3];
    ssealign uint32_t f4 = aFrame[4];
    ssealign uint32_t f5 = aFrame[5];
    ssealign uint32_t f6 = aFrame[6];

    ssealign uint32_t data = (f0<<24) | (f1<<16) | (f2<<8) | f3;

    for(ssealign int i=0; i<32;i++)
    {
        if((data & 0x80000000) !=0)
        {
            data ^= ADSBConstants::Polynomial;
        }
        data <<= 1;
    }


    ssealign uint32_t result0= (data>>24) & 0xff;
    ssealign uint32_t result1= (data>>16) & 0xff;
    ssealign uint32_t result2= (data>>8) & 0xff;

    result0 ^= f4;
    result1 ^= f5;
    result2 ^= f6;

    result0 &= 0xff;
    result1 &= 0xff;
    result2 &= 0xff;

    result0 <<= 16;
    result1 <<= 8;

    return result0 | result1 | result2;
```

```
}

uint32_t getLongFrame_Parity(uint8_t aFrame[])
{
    ssealign uint32_t f0 = aFrame[0];
    ssealign uint32_t f1 = aFrame[1];
    ssealign uint32_t f2 = aFrame[2];
    ssealign uint32_t f3 = aFrame[3];
    ssealign uint32_t f4 = aFrame[4];
    ssealign uint32_t f5 = aFrame[5];
    ssealign uint32_t f6 = aFrame[6];
    ssealign uint32_t f7 = aFrame[7];
    ssealign uint32_t f8 = aFrame[8];
    ssealign uint32_t f9 = aFrame[9];
    ssealign uint32_t f10 = aFrame[10];
    ssealign uint32_t f11 = aFrame[11];
    ssealign uint32_t f12 = aFrame[12];
    ssealign uint32_t f13 = aFrame[13];

    ssealign uint32_t data = (f0<<24) | (f1<<16) | (f2<<8) | f3;
    ssealign uint32_t data1= (f4<<24) | (f5<<16) | (f6<<8) | f7;
    ssealign uint32_t data2= (f8<<24) | (f9<<16) | (f10<<8);

    for (ssealign int i = 0; i < 88; i++)
    {
        if ((data & 0x80000000) != 0)
        {
            data ^= ADSBConstants::Polynomial;
        }
        data <<= 1;


        if ((data1 & 0x80000000) != 0)
        {
            data |= 1;
        }
        data1 <<= 1;

        if ((data2 & 0x80000000) != 0)
        {
            data1 |= 1;
        }
        data2 <<= 1;
    }


    ssealign uint32_t result0= (data>>24) & 0xff;
    ssealign uint32_t result1= (data>>16) & 0xff;
    ssealign uint32_t result2= (data>>8) & 0xff;

    result0 ^= f11;
    result1 ^= f12;
    result2 ^= f13;

    result0 &= 0xff;
    result1 &= 0xff;
    result2 &= 0xff;

    result0 <<= 16;
    result1 <<= 8;
```

```
        return  result0 | result1 | result2;

    }


    uint32_t getICAOAddres(uint8_t aFrame[])
    {
        ssealign uint32_t f0 = aFrame[0];
        ssealign uint32_t f1 = aFrame[1];
        ssealign uint32_t f2 = aFrame[2];
        ssealign uint32_t f3 = aFrame[3];

        ssealign uint32_t df = (f0>>3) & 0x1f;
        ssealign uint32_t icao=0;

        switch(df)
        {
        case 11:
        case 17:
        case 18:
            icao =  (f1<<16) | (f2<<8) | f3;
            break;
        default:
            icao = df>=16 ? getLongFrame_Parity(aFrame) :
getShortFrame_Parity(aFrame);
            if(icao == 0)
            {
                icao = (f1<<16) | (f2<<8) | f3;
            }

            break;
        }

        return icao;
    }
```

It can be seen from the above code, that more frame bytes are used for long frame than the short frame.

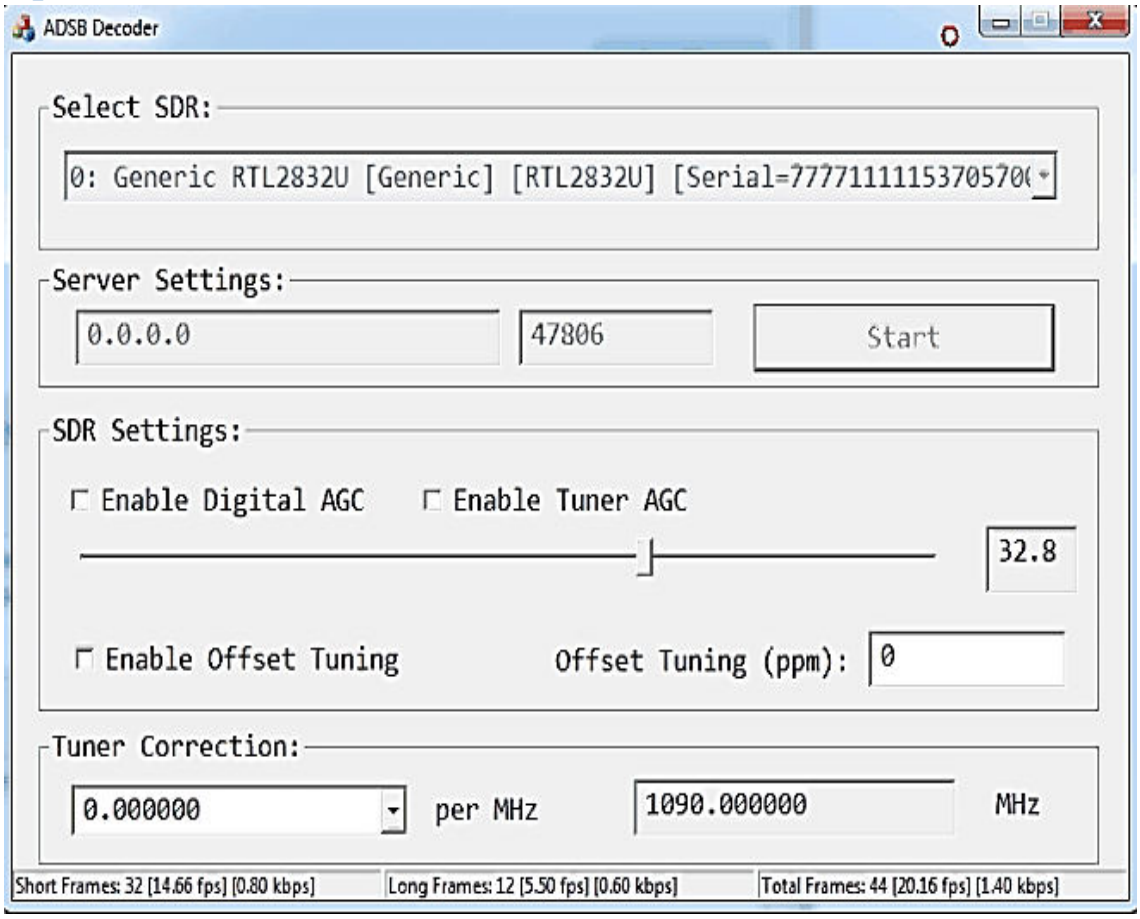
Standard Data Exchange Format

The software designed will decode the ADS-B digital data from RF signal, but in order to extract the positional information and display them on a user friendly system, a standard data exchange format is required. The standard data exchange format popularly used are byte binary data format and BEAST data format. The byte binary data format is binary stream format that are supported only by few software. They are also very hard to debug and understand. The BEAST data format is a text based format, which contain one frame in one line. The frame starts with * and ends with semi-colon. In between these identifiers the data frame bytes are placed in hexadecimal representation. This format is a text format and hence, can be easily debugged using telnet or any text display software. This data exchange is supported by all standard radar display software and hence, adopted for easy interoperation of the designed software.

Software Network Interface

The software network interface uses Windows Socket application programming interface to facilitate for detaching the decoder application from display application. This ensures that both application can reside on different computers connected through LAN or the Internet. It also provides enough freedom to add the capability to upload data stream to IOT taking advantage of bigger collaboration and global coverage, which the primary radar can never provide.

Graphical User Interface



The software is designed using Microsoft Visual C++ 2012 and depends on QSP, a custom digital processing driver which can deal with huge volume of real-time data, The User Interface is shown above, and the application is named ADSB_GUI. It can be deployed on any Microsoft Windows 7 or above system, without any added power requirement. It is very easy to understand by its WYSIWYG view. The software also supports a large number of SDR dongles which are compatible with RTL-SDR driver and also multiple manufacturers of such dongles.

DISPLAY SOFTWARE

Introduction

The display software forms the front end of the designed application. The designed application streams the received data in a standard frame format called the beast format, to the display software using TCP/IP network stream. The display software receives the frame and decodes the necessary information to be displayed on the screen. There are many software available for the purpose, but for this project only two software were used and tested – ADSBScope and Virtual Radar.

ADSBScope

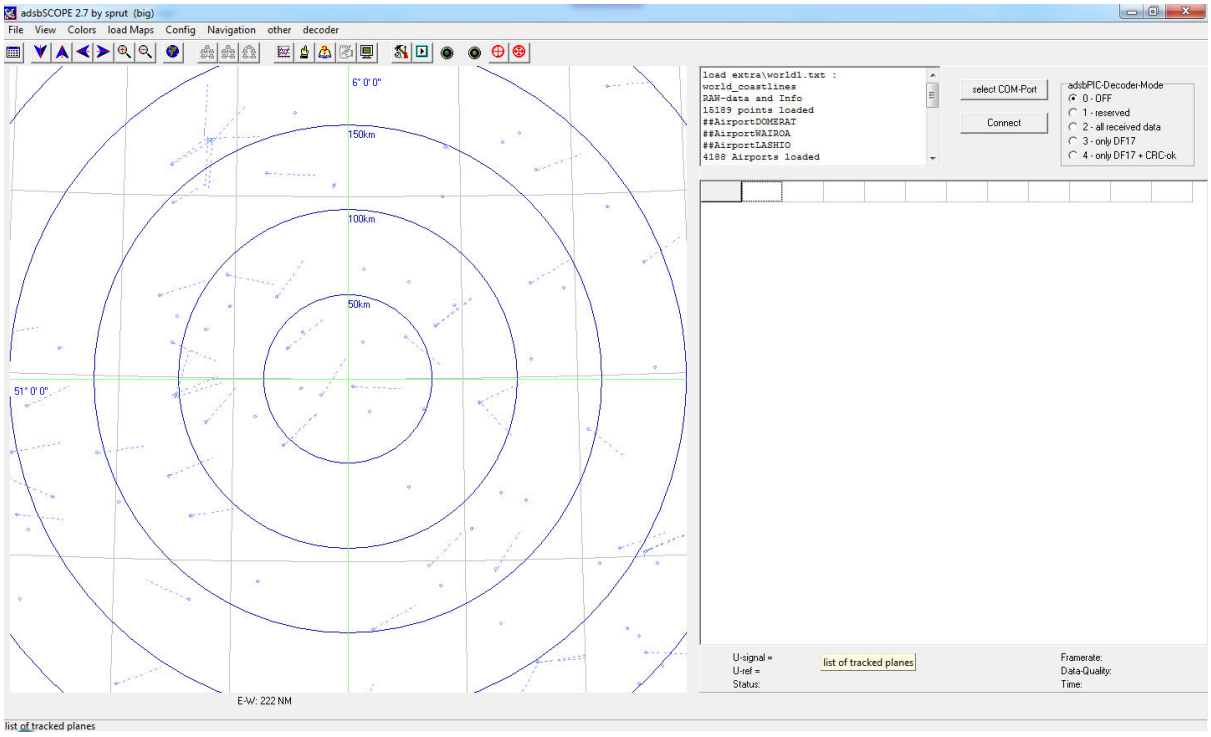


Figure 1: ADSBScope User Interface

ADSBScope is a freeware software that can be downloaded and run without any installation process and is very easy to setup and use. The software requires very less CPU power and can be run on any PC. The benefit of this software is the small memory consumption which enables it to run on UMPC. It also feature a complete RADAR display and can download map from the Internet and display it on the display as background. Thus, the software does not require any additional display interfaces.

Virtual Radar

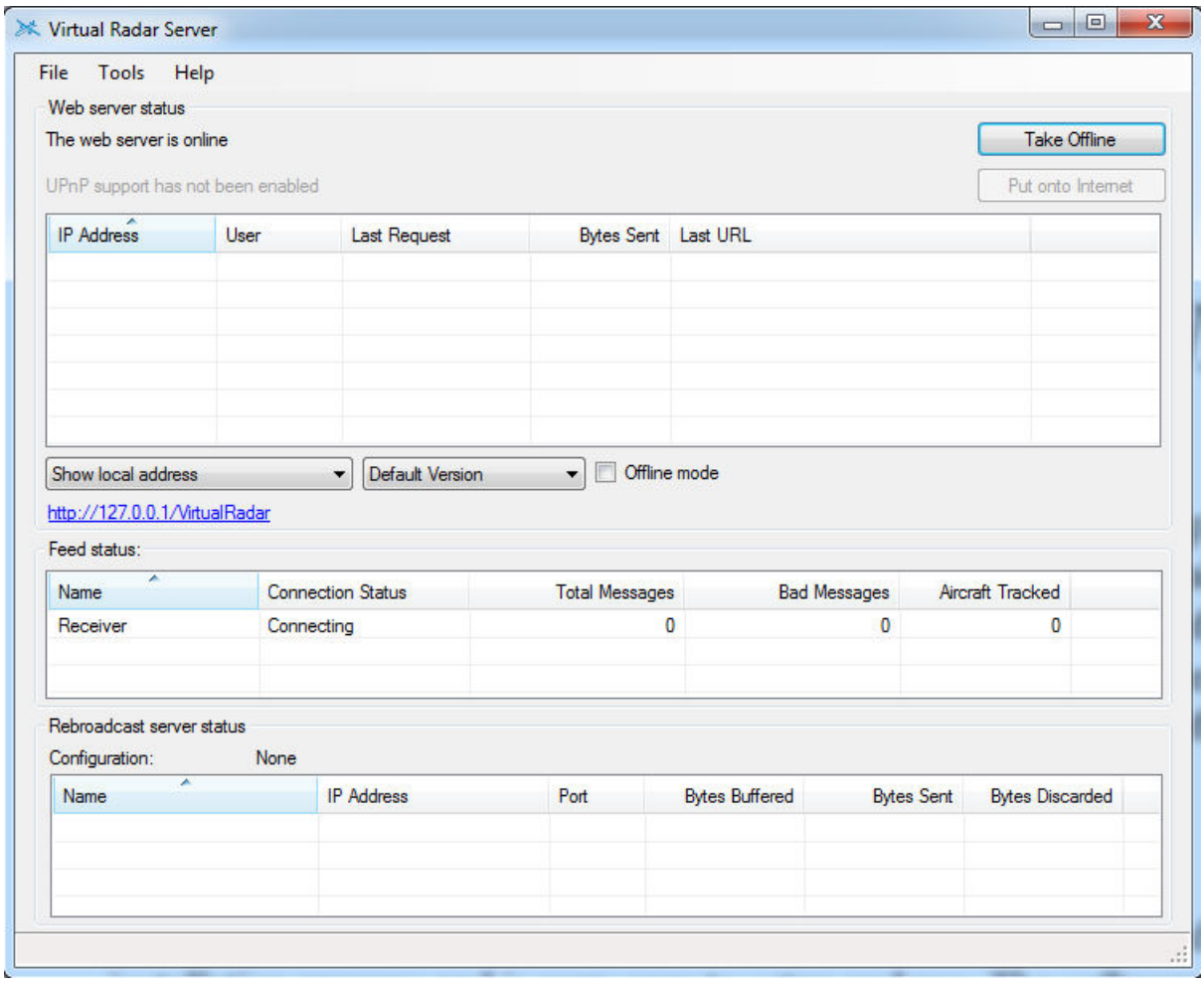


Figure 2 : Virtual Radar Graphical User Interface

Virtual Radar is a more sophisticated application that uses Microsoft .NET framework for processing. Current version is not supported to run on Window 8.1 UMPC and hence, the test using this software is limited to desktop PC only. It is also conservative on CPU power but requires Web browser to display flight information. An active Internet connection is also necessary, as the display plots the positional information of the aircraft on Google Maps, which is streamed in at real time.

RESULTS

ADSB# vs ADSB_GUI

ASDB_GUI

The software designed as a part of this project is named ADSB_GUI. It is a Microsoft Windows based application that can decode the ADSB data frames from the received radio frequency signals. The software can be deployed very easily as it does not require additional components to be installed.

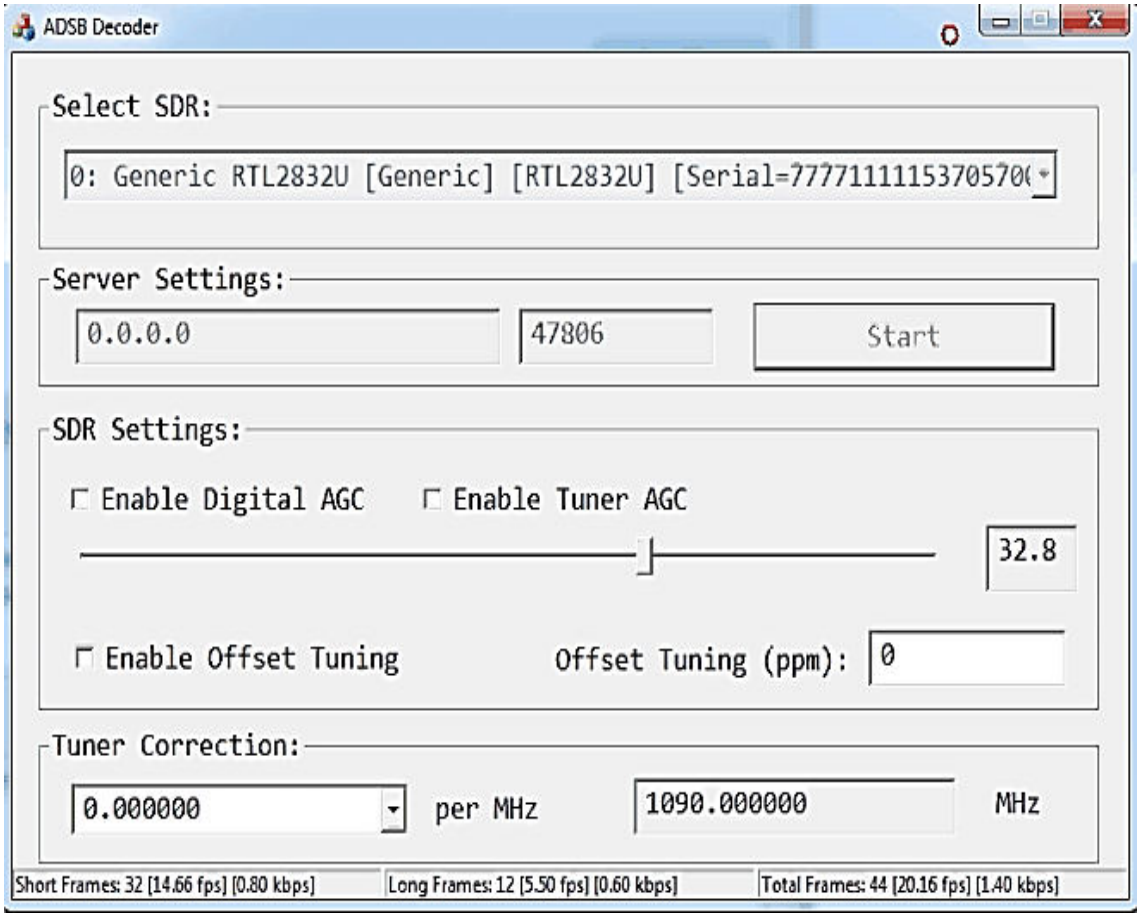


Figure 3 : ADSB_GUI Graphical User Interface

The software interface allows to select one of the connected RTLSDR, connected to the USB port of the computer, and also allows for settings of Server listener IP address and Port value. Once the Start button is pressed, the application blocks until a client display software connect to the application. Thereafter, settings like Gain, Tuner Correction, AGC controls and offset tuning controls are enable to fine tune the RTLSDR for proper reception of the radio frequency signal.

Additionally, in the status bar the packet frame rate as well as the total bytes transferred is displayed.

ADSB#

A competing and existing software is ADSB#, developed by Youssef, an engineer who developed SDR#. The software is popular among researcher, aviation enthusiast as well as radio operator as a popular ADS-B decoder software, which existed prior to the development of our software, but has controls similar to that seen in the previous user interface. The software, unlike our software, require Microsoft .NET Framework and cannot be deployed on low power ultra-mobile computer (UMPC), and the current versions does not work with RTL-SDR but requires AirSpy SDR to work.

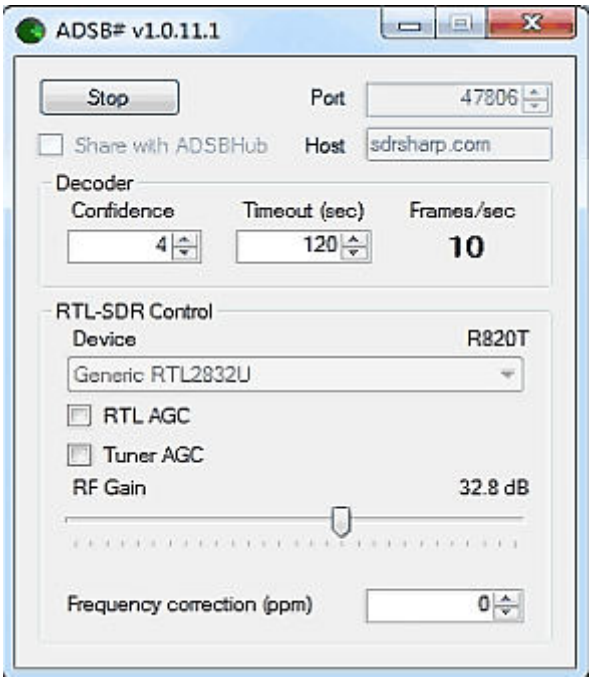


Figure 4: ADSB# Graphical User Interface

Comparison of ADSB_GUI and ADSB#

CPU/Processor Consumption

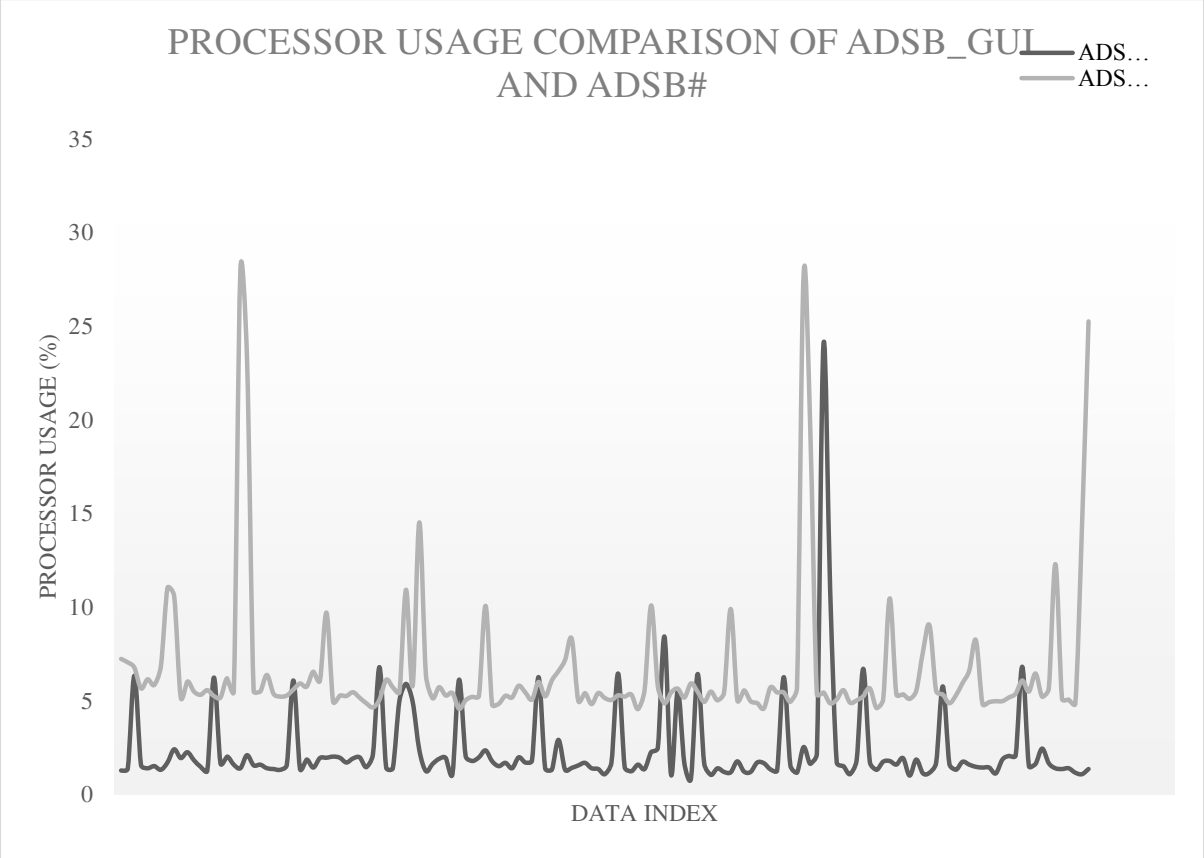


Figure 5: Processor usage

The above graph visually proves that the designed application ADSB_GUI consumes less CPU at any time compared to the ADSB# software. This advantage is brought about by the usage of SIMD/SSE4.2 instructions and auto-vectorization, wherever possible. This again increases battery life and requires less power for operation, and enables easy deployment on the low power computers.

Comparison of Requirement Specifications

The common requirement for both the software is Microsoft Windows operating system, preferably Windows 7 or above. But ADSB_GUI does not rely on any additional installation as it is designed using Microsoft Visual C++ 2012 and the executable contains x86 based machine code. Whereas the ADSB# is a Microsoft IL binary requiring Microsoft .NET Framework 3.0, which limits its operation on Windows 8.1 based UMPC. The Windows 8.1 UMPC does not allow installation of Microsoft .NET 2.0 and 3.0 framework. Here again ADSB_GUI takes an advantage.

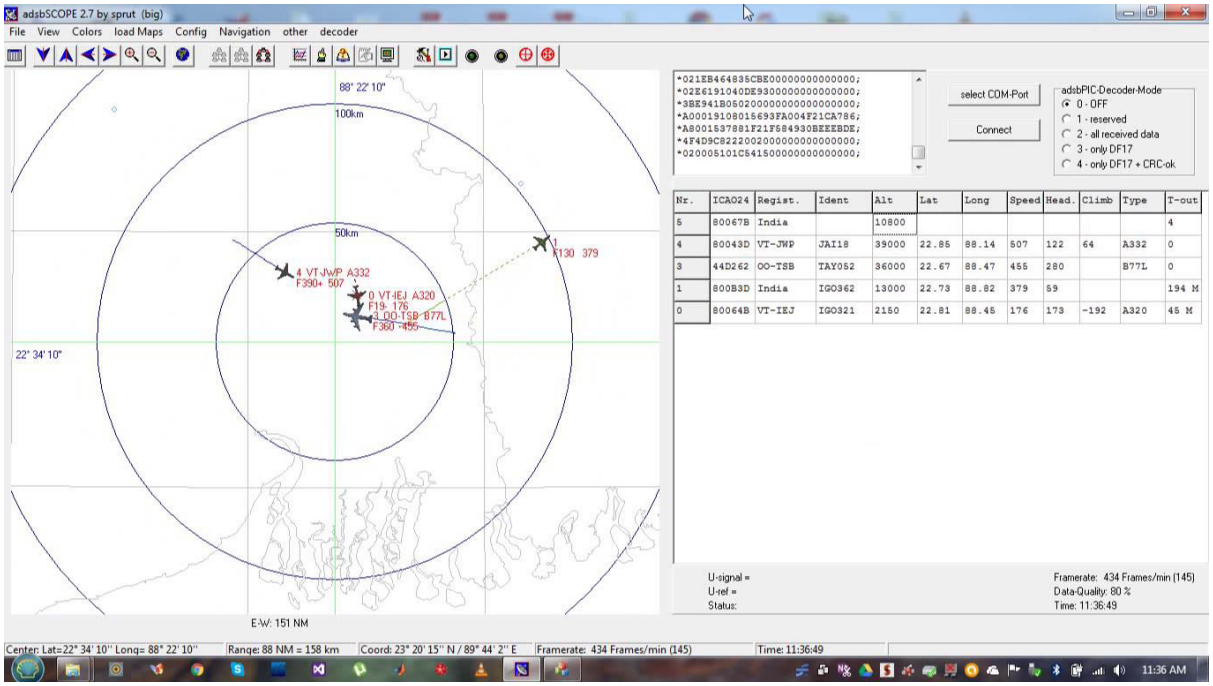
Development Approach

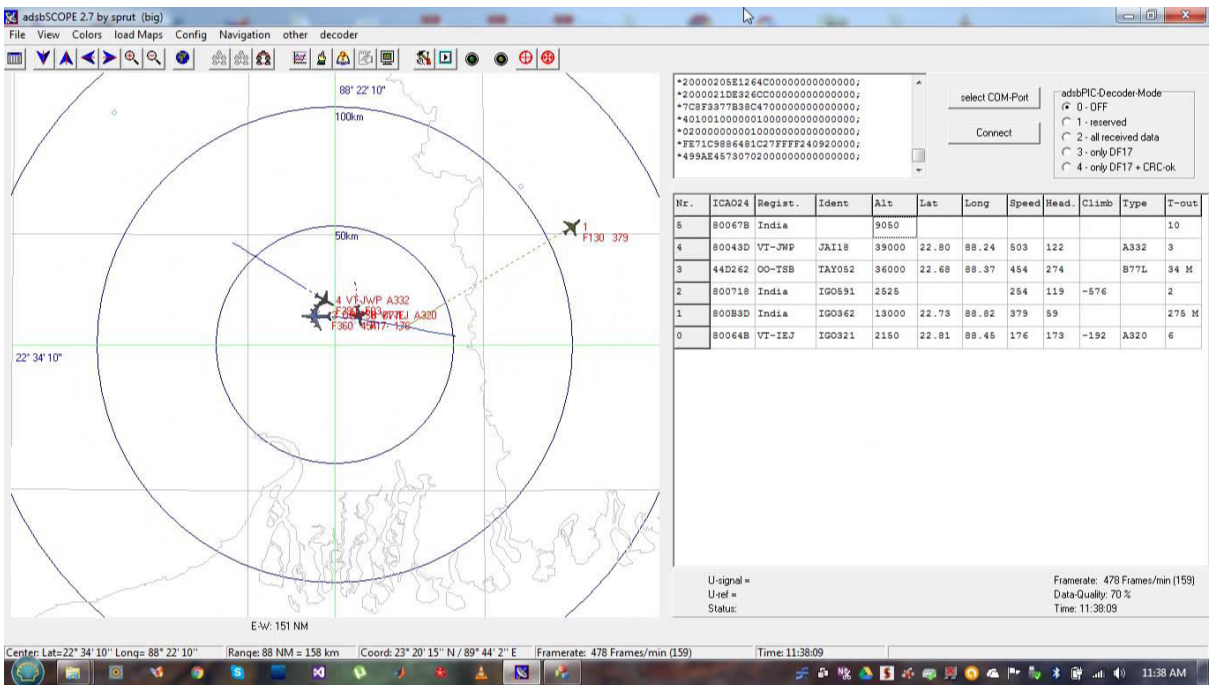
The development approach used in ADSB# uses an internal Dictionary to store ICAO codes, which are assigned a timeout interval after which it is discarded. This approach increases CPU power consumption due to comparison with Dictionary storage.

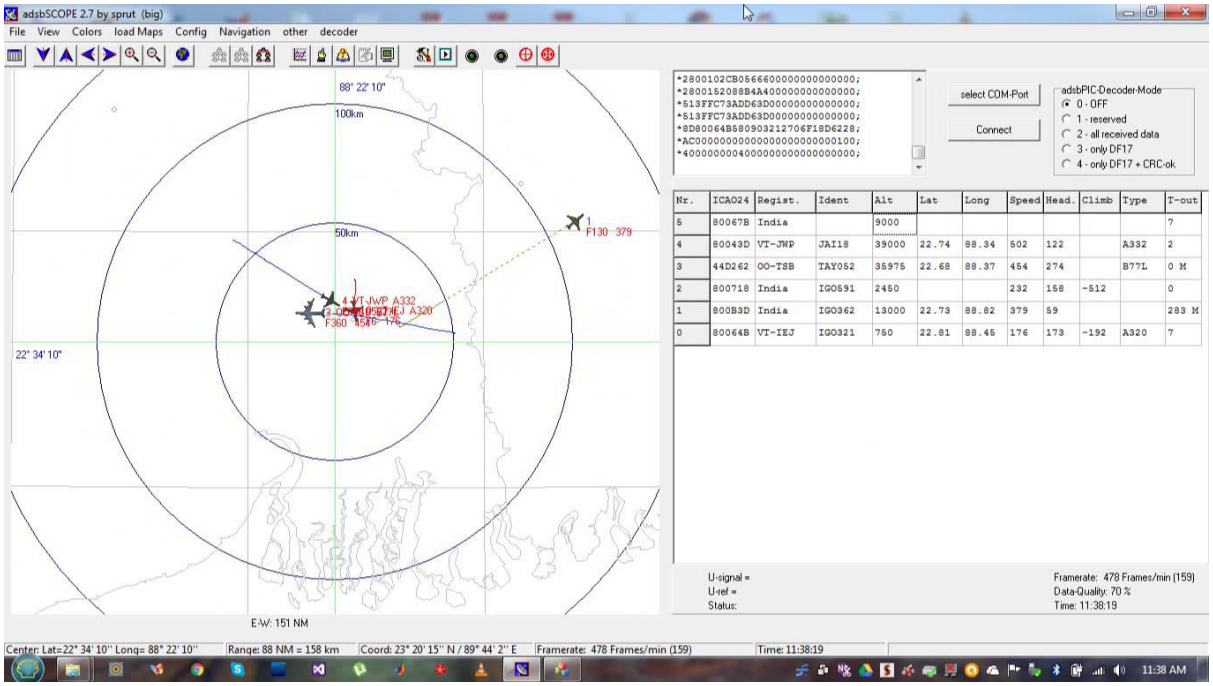
But ADSB_GUI uses a completely different approach. The code internally uses an adaptive level to establish a noise floor, which is used in to correct noise and discard any frame that is corrupted by burst noise. Even the preamble is detected by known preamble method, whose algorithm is efficient enough to detect even weak signals that can be distinguished by the ADC in RTLSDR. This enables more packet detection even in bad weather and indoor reception condition.

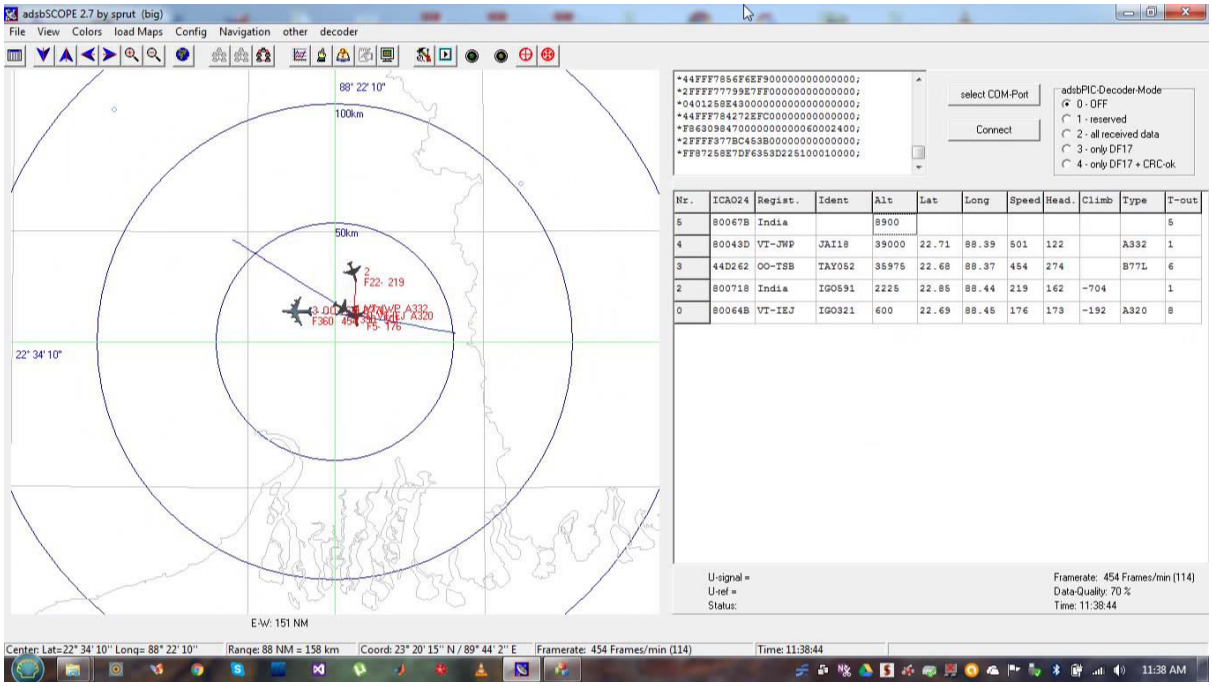
Observations using ADSBScope with ADSB_GUI

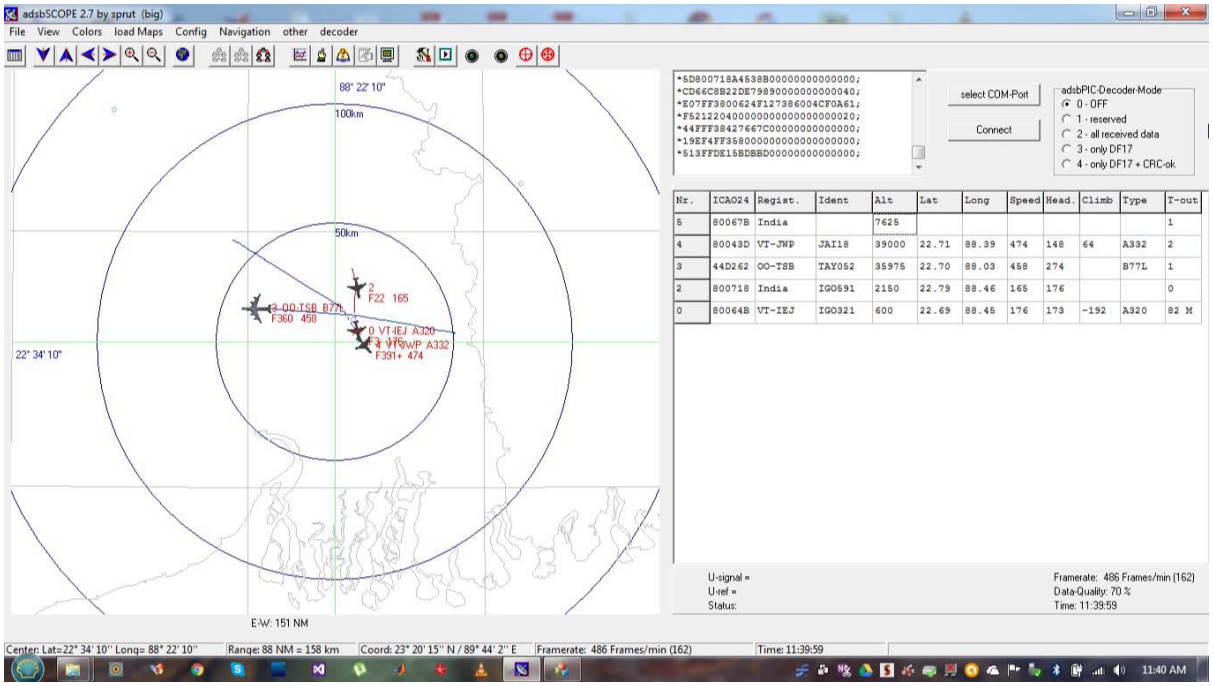
Observations on Windows 7 Desktop/Laptop Computer

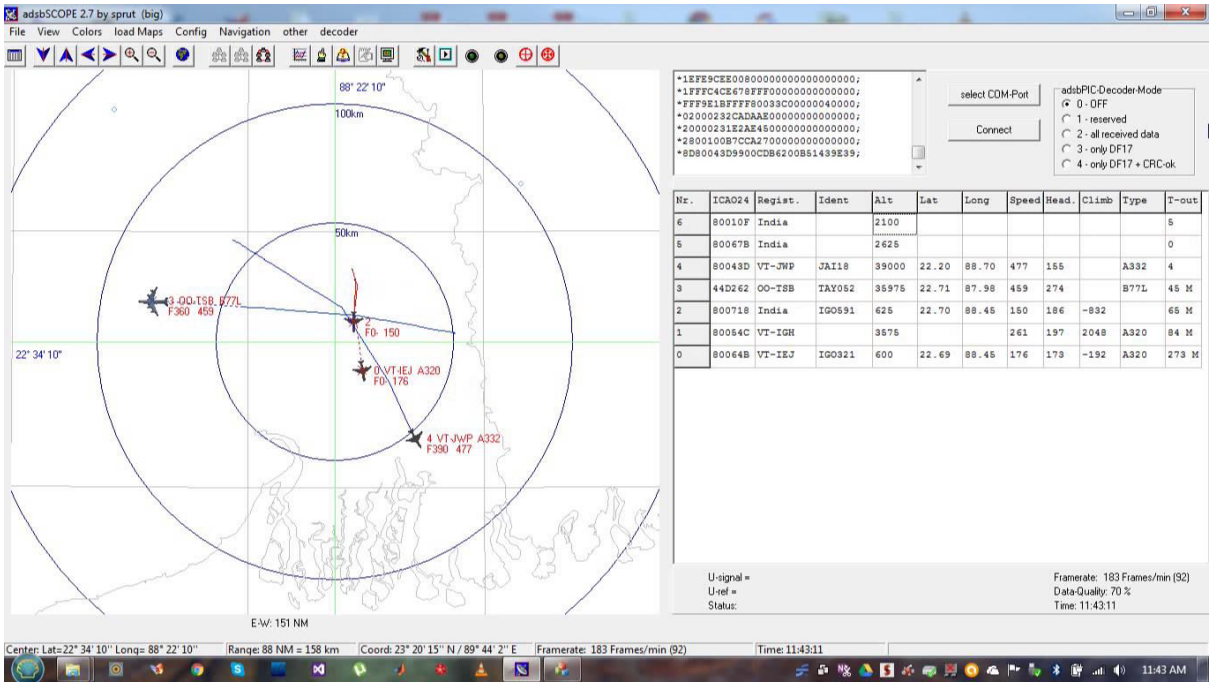


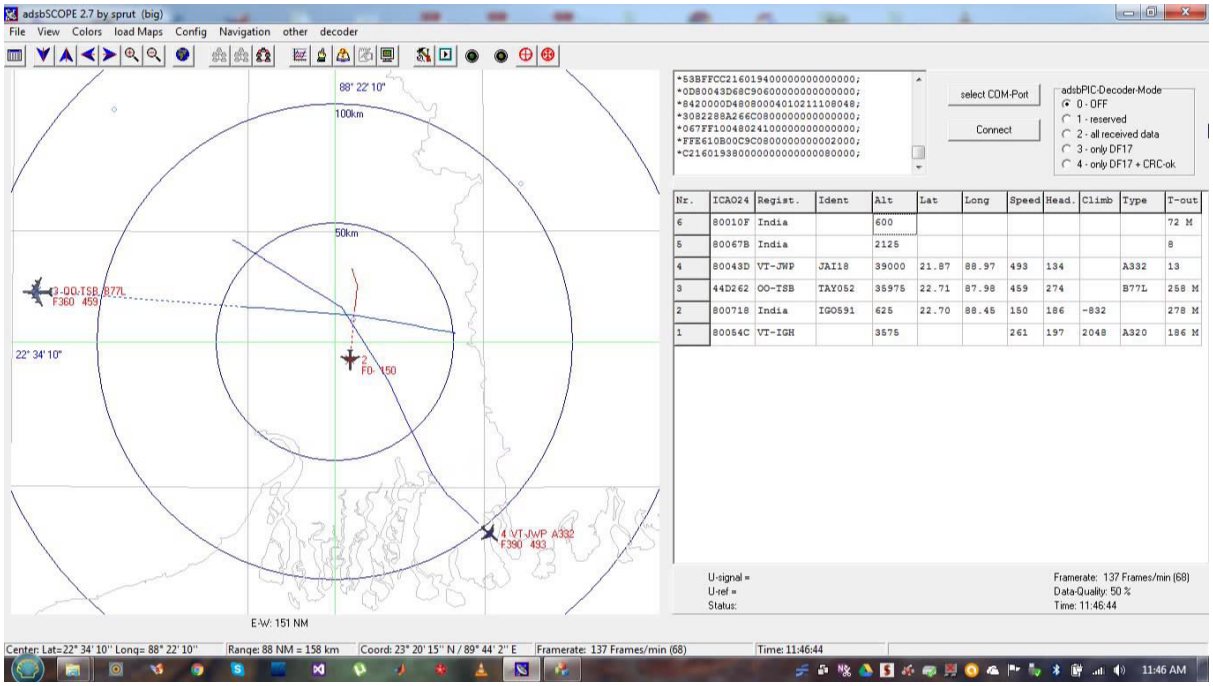


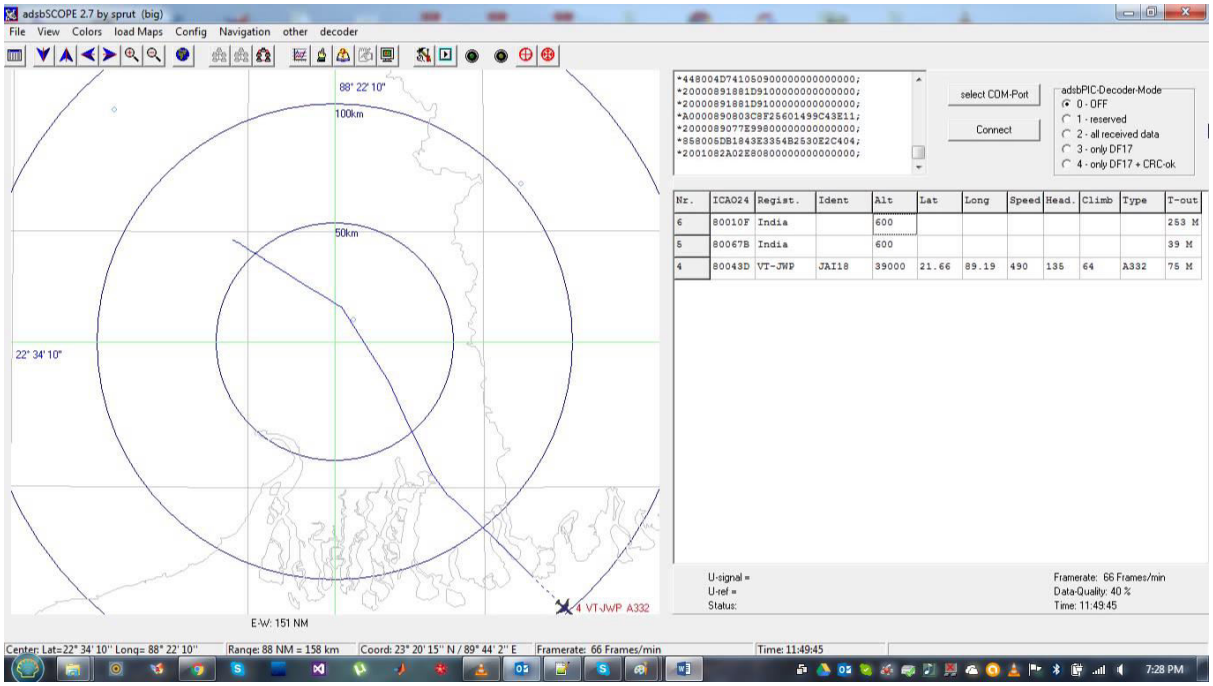

adsbSCOPE 2.7 by sprut (big)
File View Colors load Maps Config Navigation other decoder
88° 22' 10"
100km
50km
22° 34' 10"
E-W: 151 NM
*2800102CB056660000000000000;
*280015208B8A400000000000000;
*513FFC73ADD63D0000000000000;
*513FFC73ADD63D0000000000000;
*8D80064B5809032127O6F18D6228;
*AC0000000000000000000000100;
*4000000004000000000000000;
select COM-Port
Connect
adsbPIC-Decoder-Mode
0 - OFF
1 - reserved
2 - all received data
3 - only DF17
4 - only DF17 + CRC-ok
Nr. ICAO24 Regist. Ident Alt Lat Long Speed Head. Climb Type T-out
5 80067B India 9000 7
4 80043D VT-JWP JAI18 39000 22.74 88.34 502 122 A332 2
3 44D262 OO-TSB TAY052 35975 22.68 88.37 454 274 B77L 0 M
2 800718 India IGO591 2450 232 158 -512 0
1 800B3D India IGO362 13000 22.73 88.82 379 59 283 M
0 80064B VT-IEJ IGO321 750 22.81 88.45 176 173 -192 A320 7
U-signal =
U-ref =
Status:
Framerate: 478 Frames/min (159)
Data-Quality: 70 %
Time: 11:38:19
Center: Lat=22° 34' 10'' Long= 88° 22' 10''
Range: 88 NM = 158 km
Coord: 23° 20' 15'' N / 89° 44' 2'' E
Framerate: 478 Frames/min (159)
Time: 11:38:19
11:38 AM


adsbSCOPE 2.7 by sprut (big)
File View Colors load Maps Config Navigation other decoder
88° 22' 10"
100km
50km
22° 34' 10"
E-W: 151 NM
*44FFF7856F6EF900000000000000;
*2FFFF77799E7FF000000000000000;
*0401258E4300000000000000000;
*44FFF78427 2EFC000000000000;
*F863098470000000000060002400;
*2FFFF777BC453B000000000000000;
*FF87258E7DF6353D225100010000;
select COM-Port
Connect
adsbPIC-Decoder-Mode
0 - OFF
1 - reserved
2 - all received data
3 - only DF17
4 - only DF17 + CRC-ok
Nr. ICAO24 Regist. Ident Alt Lat Long Speed Head. Climb Type T-out
5 80067B India 8900 5
4 80043D VT-JWP JAI18 39000 22.71 88.39 501 122 A332 1
3 44D262 OO-TSB TAY052 35975 22.68 88.37 454 274 B77L 6
2 800718 India IGO591 2225 22.85 88.44 219 162 -704 1
0 80064B VT-IEJ IGO321 600 22.69 88.45 176 173 -192 A320 8
U-signal =
U-ref =
Status:
Framerate: 454 Frames/min (114)
Data-Quality: 70 %
Time: 11:38:44
Center: Lat=22° 34' 10'' Long= 88° 22' 10''
Range: 88 NM = 158 km
Coord: 23° 20' 15'' N / 89° 44' 2'' E
Framerate: 454 Frames/min (114)
Time: 11:38:44
11:38 AM

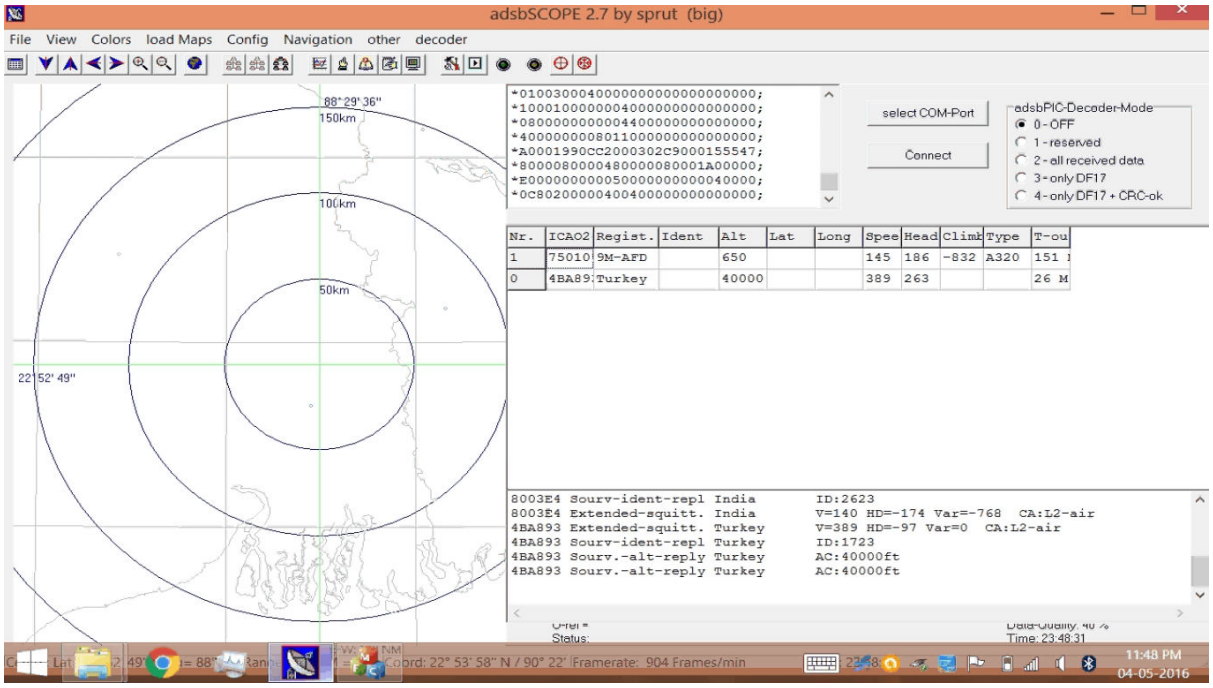

adsbSCOPE 2.7 by sprut (big)
File View Colors load Maps Config Navigation other decoder
88° 22' 10"
100km
50km
22° 34' 10"
2 F22 165
3 OO-TSB B77L F360 458
0 VT-IEJ A320
4 VT-JWP A332 F391+ 474
E-W: 151 NM
*5D800718A4638B00000000000000;
*CD66C8B22DE79890000000000040;
*E07FF3800624F127386004CF0A61;
*F5212204000000000000000000020;
*44FFF38427667C00000000000000;
*19EF4FF358000000000000000000;
*513FFDE15BDBBD00000000000000;
select COM-Port
Connect
adsbPIC-Decoder-Mode
0 - OFF
1 - reserved
2 - all received data
3 - only DF17
4 - only DF17 + CRC-ok
Nr. ICAO24 Regist. Ident Alt Lat Long Speed Head. Climb Type T-out
5 80067B India 7625 1
4 80043D VT-JWP JAI18 39000 22.71 88.39 474 148 64 A332 2
3 44D262 OO-TSB TAY052 35975 22.70 88.03 458 274 B77L 1
2 800718 India IGO591 2150 22.79 88.46 165 176 0
0 80064B VT-IEJ IGO321 600 22.69 88.45 176 173 -192 A320 82 M
U-signal =
U-ref =
Status:
Framerate: 486 Frames/min (162)
Data-Quality: 70 %
Time: 11:39:59
Center: Lat=22° 34' 10'' Long= 88° 22' 10''
Range: 88 NM = 158 km
Coord: 23° 20' 15'' N / 89° 44' 2'' E
Framerate: 486 Frames/min (162)
Time: 11:39:59
11:40 AM

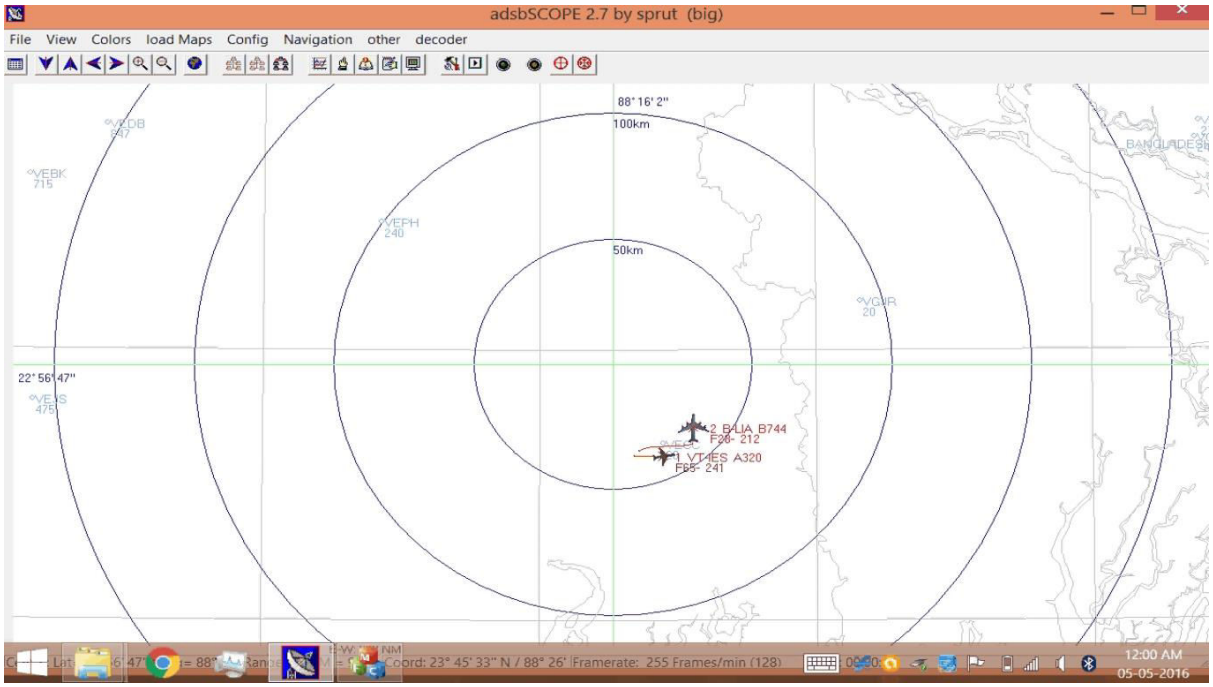

adsbSCOPE 2.7 by sprut (big)
File View Colors load Maps Config Navigation other decoder
88° 22' 10"
100km
50km
22° 34' 10"
3 OO-TSB B77L F360 459
2 F0 150
0 VT-IEJ A320 F0 176
4 VT-JWP A332 F390 477
E-W: 151 NM
*1EFE9CEE008000000000000000;
*1FFFC4CE678FFF00000000000000;
*FFF9E1BFFFF80033C00000040000;
*02000232CADAAE00000000000000;
*20000231E2AE4500000000000000;
*2800100B7CCA2700000000000000;
*8D80043D9900CDB6200B51439E39;
select COM-Port
Connect
adsbPIC-Decoder-Mode
0 - OFF
1 - reserved
2 - all received data
3 - only DF17
4 - only DF17 + CRC-ok
Nr. ICAO24 Regist. Ident Alt Lat Long Speed Head. Climb Type T-out
6 80010F India 2100 5
5 80067B India 2625 0
4 80043D VT-JWP JAI18 39000 22.20 88.70 477 155 A332 4
3 44D262 OO-TSB TAY052 35975 22.71 87.98 459 274 B77L 45 M
2 800718 India IGO591 625 22.70 88.45 150 186 -832 65 M
1 80054C VT-IGH 3575 261 197 2048 A320 84 M
0 80064B VT-IEJ IGO321 600 22.69 88.45 176 173 -192 A320 273 M
U-signal =
U-ref =
Status:
Framerate: 183 Frames/min (92)
Data-Quality: 70 %
Time: 11:43:11
Center: Lat=22° 34' 10'' Long= 88° 22' 10''
Range: 88 NM = 158 km
Coord: 23° 20' 15'' N / 89° 44' 2'' E
Framerate: 183 Frames/min (92)
Time: 11:43:11
11:43 AM


adsbSCOPE 2.7 by sprut (big)
File View Colors load Maps Config Navigation other decoder
88° 22' 10"
100km
50km
22° 34' 10"
OO-TSB B77L
F360 459
2
FO- 150
4 VT-JWP A332
F390 493
E-W: 151 NM
select COM-Port
Connect
adsbPIC-Decoder-Mode
0 - OFF
1 - reserved
2 - all received data
3 - only DF17
4 - only DF17 + CRC-ok
Nr. ICAO24 Regist. Ident Alt Lat Long Speed Head. Climb Type T-out
6 80010F India 600 72 M
5 80067B India 2125 8
4 80043D VT-JWP JAI18 39000 21.87 88.97 493 134 A332 13
3 44D262 OO-TSB TAY052 35975 22.71 87.98 459 274 B77L 258 M
2 800718 India IGO591 625 22.70 88.45 150 186 -832 278 M
1 80054C VT-IGH 3575 261 197 2048 A320 186 M
U-signal =
U-ref =
Status:
Framerate: 137 Frames/min (68)
Data-Quality: 50 %
Time: 11:46:44
Center: Lat=22° 34' 10'' Long= 88° 22' 10''
Range: 88 NM = 158 km
Coord: 23° 20' 15'' N / 89° 44' 2'' E
Framerate: 137 Frames/min (68)
Time: 11:46:44
11:46 AM


adsbSCOPE 2.7 by sprut (big)
File View Colors load Maps Config Navigation other decoder
88° 22' 10"
100km
50km
22° 34' 10"
4 VT-JWP A332
E-W: 151 NM
select COM-Port
Connect
adsbPIC-Decoder-Mode
0 - OFF
1 - reserved
2 - all received data
3 - only DF17
4 - only DF17 + CRC-ok
Nr. ICAO24 Regist. Ident Alt Lat Long Speed Head. Climb Type T-out
6 80010F India 600 253 M
5 80067B India 600 39 M
4 80043D VT-JWP JAI18 39000 21.66 89.19 490 135 64 A332 75 M
U-signal =
U-ref =
Status:
Framerate: 66 Frames/min
Data-Quality: 40 %
Time: 11:49:45
Center: Lat=22° 34' 10'' Long= 88° 22' 10''
Range: 88 NM = 158 km
Coord: 23° 20' 15'' N / 89° 44' 2'' E
Framerate: 66 Frames/min
Time: 11:49:45
7:28 PM

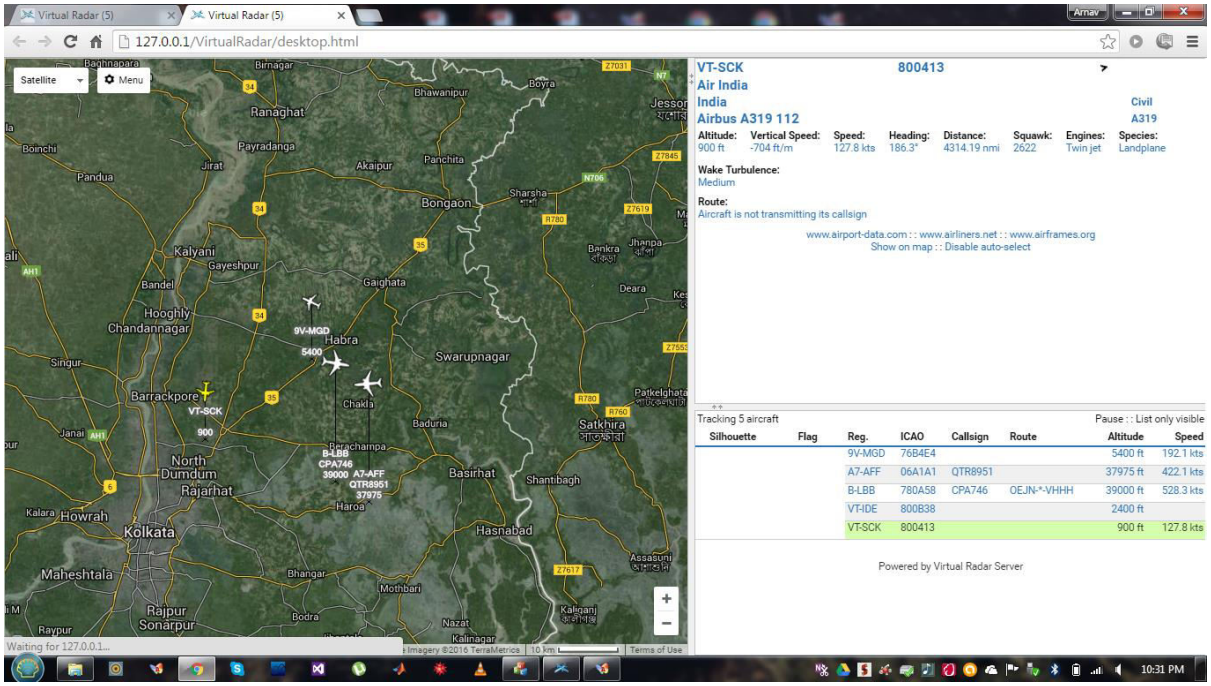
Observations on Windows 8.1 iBall Slide 701i Ultra-Mobile PC (UMPC)

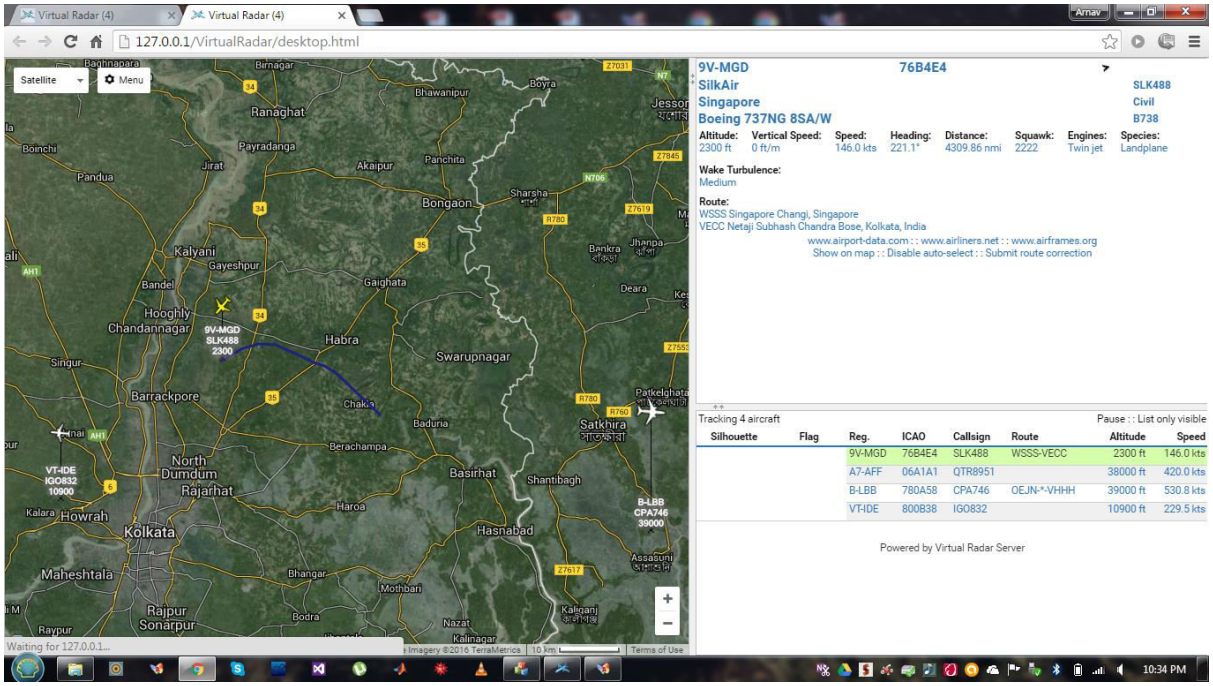


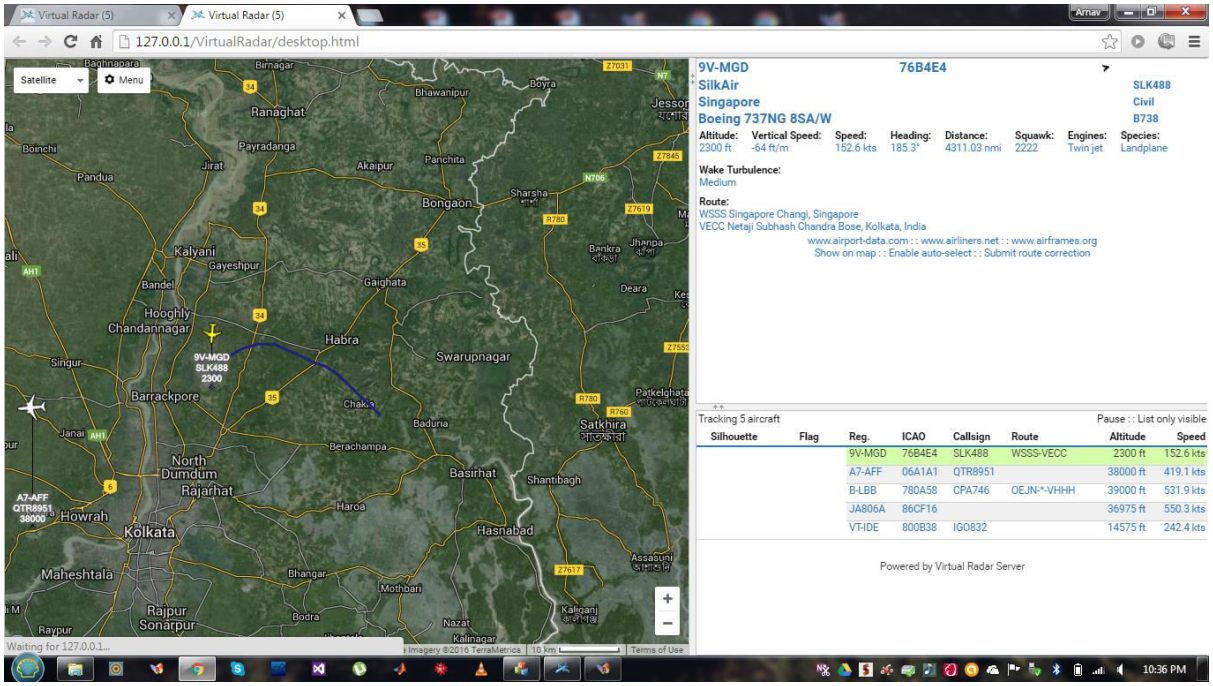


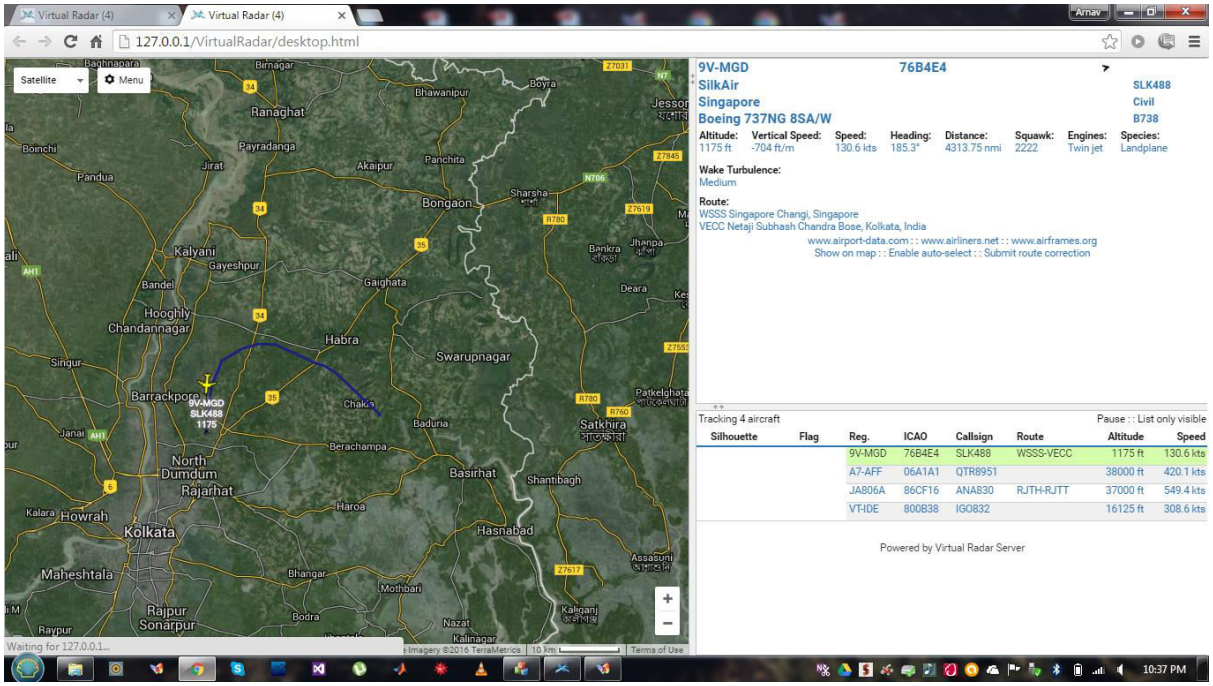
Observations using Virtual Radar with ADSB_GUI

Observations on Windows 7 Desktop/Laptop Computer









The Virtual Radar software does not run on Windows 8.1 iBall 701i Tablet. This is because, the operating system does not allow user to install Microsoft .NET Framework 2.0 and 3.0 framework, which is required by Virtual Radar software.

CONCLUSIONS

- We have designed ADS-B software in Software Defined Radio for the detection of aircraft parameters which can be interfaced with commercial display software.
- Our designed algorithm consumes less CPU power.
- Our software can detect ADS-B signals in indoor and in bad weather condition.
- The software can be uploaded in IOT and can be displayed globally.

References

[1] "Next Generation Air Transportation System", *Wikipedia*, 2016. [Online]. Available: https://en.wikipedia.org/wiki/Next_Generation_Air_Transportation_System. [Accessed: 21- Apr- 2016].

[2] A. Abdulaziz, A. S.Yaro, A. Adam, M. Kabir, H. Salau, A. Abdulaziz, A. S.Yaro, A. Adam, M. Kabir and H. Salau, "Optimum Receiver for Decoding Automatic Dependent Surveillance Broadcast (ADS-B) Signals", *American Journal of Signal Processing*, vol. 5, no. 2, pp. 23-31, 2015.

[3] M. Huang, R. Narayanan, Y. Zhang and A. Feinberg, "Tracking of Noncooperative Airborne Targets Using ADS-B Signal and Radar Sensing", *International Journal of Aerospace Engineering*, vol. 2013, pp. 1-12, 2013.

[4] "Synchronous vs. Asynchronous", *Engr.iupui.edu*, 2016. [Online]. Available: http://www.engr.iupui.edu/~skoskie/ECE362/lecture_notes/LNB25_html/text12.html. [Accessed: 21- Apr- 2016].

[5] H. Jeong, S. Kim, W. Lee and S. Myung, "Performance of SSE and AVX Instruction Sets", *Arxiv.org*, 2012. [Online]. Available: http://arxiv.org/abs/1211.0820v1. [Accessed: 21- Apr- 2016].

[6] "ADSB J-pole Antenna", *Lll.lu*, 2016. [Online]. Available: http://www.lll.lu/~edward/edward/adsb/antenna/ADSBantenna.html. [Accessed: 21- Apr- 2016].

[7] "Cyclic redundancy check", *Wikipedia*, 2016. [Online]. Available: https://en.wikipedia.org/wiki/Cyclic_redundancy_check. [Accessed: 21- Apr- 2016].

[8] J. Kopcak, "ADS-B Decoding with RTL-SDR, ADSBSharp, and Virtual Radar Server | Jeffrey Kopcak, MBA", *Jeffreykopcak.com*, 2014. [Online]. Available: https://www.jeffreykopcak.com/2014/11/30/ads-b-decoding-with-adsbsharp-and-virtual-radar-server/. [Accessed: 21- Apr- 2016].

[9] "Simple ADS-B decoder", *Rxcontrol.free.fr*, 2016. [Online]. Available: http://rxcontrol.free.fr/PicADSB/. [Accessed: 21- Apr- 2016].

[10] G. Koellner, "MODE-S BEAST - Your Airspace Observer", *Modesbeast.com*, 2016. [Online]. Available: http://www.modesbeast.com/. [Accessed: 21- Apr- 2016].

[11] Director General of Civil Aviation, "AIR TRAFFIC MANAGEMENT CIRCULAR NO. 15 of 2014 Automatic Dependent Surveillance – Broadcast (ADS-B)", New Delhi, 2016.

[12] G. Mitra, B. Johnston, A. Rendell, E. McCreath and J. Zhou, "Use of SIMD Vector Operations to Accelerate Application Code Performance on Low-Powered ARM and Intel Platforms", *2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum*, pp. 1107-1116, 2013.

[13] "RTL2832 Software Defined Radio", *Ab9il.net*, 2016. [Online]. Available: https://www.ab9il.net/software-defined-radio/rtl2832-sdr.html. [Accessed: 25- Apr- 2016].

[14] "zadig Archives - rtl-sdr.com", *rtl-sdr.com*, 2015. [Online]. Available: http://www.rtl-sdr.com/tag/zadig/. [Accessed: 25- Apr- 2016].

[15] "ADS-B DIY Antenna", *Plane Finder Forum*, 2016. [Online]. Available: http://forum.planefinder.net/threads/ads-b-diy-antenna.23/. [Accessed: 26- Apr- 2016].
[16] "ADSB J-pole Antenna", *Lll.lu*, 2016. [Online]. Available: http://www.lll.lu/~edward/edward/adsb/antenna/ADSBantenna.html. [Accessed: 26- Apr- 2016].
[17] "J-PoleSection", *Aa1zb.net*, 2016. [Online]. Available: http://aa1zb.net/Antennas/J-Poles/J-PoleSection.html. [Accessed: 28- Apr- 2016].
[18] "J-pole antenna", *Wikipedia*, 2016. [Online]. Available: https://en.wikipedia.org/wiki/J-pole_antenna. [Accessed: 28- Apr- 2016].