1.09 GHz Mode-S Receiver Design and VHF Radar Antenna Characterization

By

Dabin Zhang

Senior Thesis in Electrical Engineering

University of Illinois at Urbana-Champaign

Advisor: Professor Steven J. Franke

May 2013

Abstract

The design and implementation of a 1.09 GHz tuned-radio-frequency (TRF) receiver which can capture mode-S secondary surveillance radar reply frames sent by aircraft transponders is discussed in this thesis. The RF frontend of the receiver employs an RF-IC and SAW filter and detects the envelope of the input signal. The detected envelope signal is sampled and demodulated by FPGA firmware to reconstruct mode-S frames which can be retrieved via a Universal Serial Bus (USB) interface. Mode-S frame recording and decoding programs were written in C++ to extract aircraft information such as latitude, longitude, altitude, and aircraft identification number. Some decoded data has been combined with radar backscatter data collected by a VHF remote sensing radar to characterize the radiation pattern of the radar antenna arrays. The analyses confirmed that one antenna array performs according to theoretical expectation and uncovered a significant problem with another antenna array due to a coaxial cable connectivity issue.

Subject Keywords: tuned radio frequency receiver, mode-S, ADS-B, antenna pattern measurement

Contents

1. Introduction

Project Overview

This project consists of two parts. The first part is 1.09 GHz mode-S receiver design, where the receiver is designed to capture and demodulate mode-S downlink frames sent by aircraft transponders at 1.09 GHz. The second part of this project is VHF radar antenna characterization. The data captured by the mode-S receiver was decoded and position reports from particular aircraft have been combined with radar data to characterize the radiation pattern of the radar antenna array.

Transponder

A typical interrogation and reply cycle between ground radar station and aircraft is shown in Figure 1. The secondary radar interrogator transmits an interrogation frame (uplink) to the aircraft at 1.03 GHz and the aircraft transmits a reply frame (downlink) at 1.09 GHz after it receives the interrogation frame.

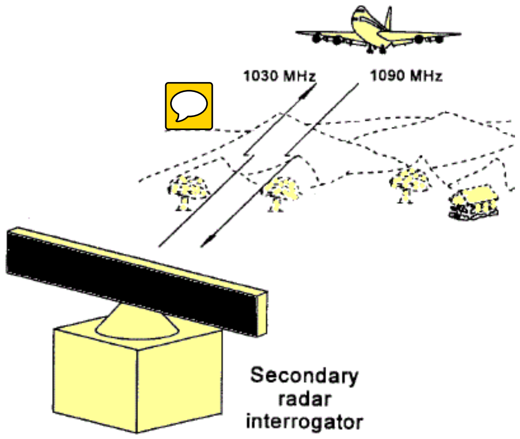


Figure 1 Typical interrogation cycle. Adapted from [1]

Figure 2 is a typical transponder front panel. The LED screen shows current altitude is FL065 which is $65\times100=6500$ feet and squawk code is 1200 which is for visual flight rules (VFR) flight. Both altitude and squawk will display on air traffic controllers' (ATC) radar screen. The squawk code can be entered through keypad and mode can be selected through the knob.

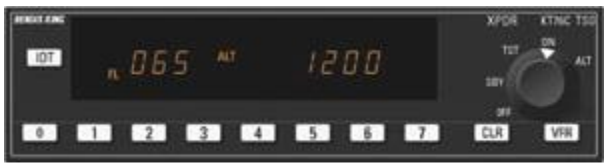


Figure 2 Front panel of KT-76C transponder. Adapted from [2]

Transponders for civil aircraft have three different modes which are mode-A, mode-C and mode-S and most commercial transponders are capable of all those modes. Mode-A contains identification information which is squawk code and mode-C contains altitude code. Mode-S, however, can carry multiple types of data with larger package size.

Mode-A and mode-C reply frames have the same format (Figure 3 [3]) which is amplitude shift keying (ASK) with 13-bit length. All mode-A/C downlink frames have a start pulse (F_1) and end pulse (F_2) which are on the front and end of data block. The SPI pulse at the end indicates the status of transponder.

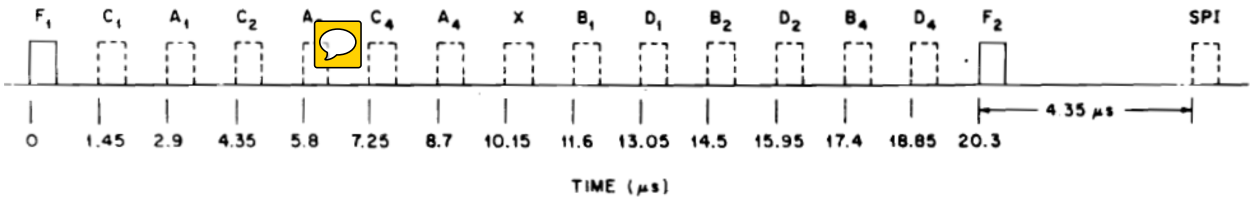


Figure 3 Mode A/C reply frame

Mode-S

Mode-S stands for “mode select”, because only a particular aircraft will respond to the radar interrogation since all aircraft are assigned a unique address code (ICAO address). This technique provides improved surveillance capability and using a discrete address permits information exchange between ground radar station and aircraft which forms a data link [4]. As can be seen in Figure 4, aircraft not only transmit data to ground air traffic controller (ATC) facility and other aircraft, but they also broadcast some data.

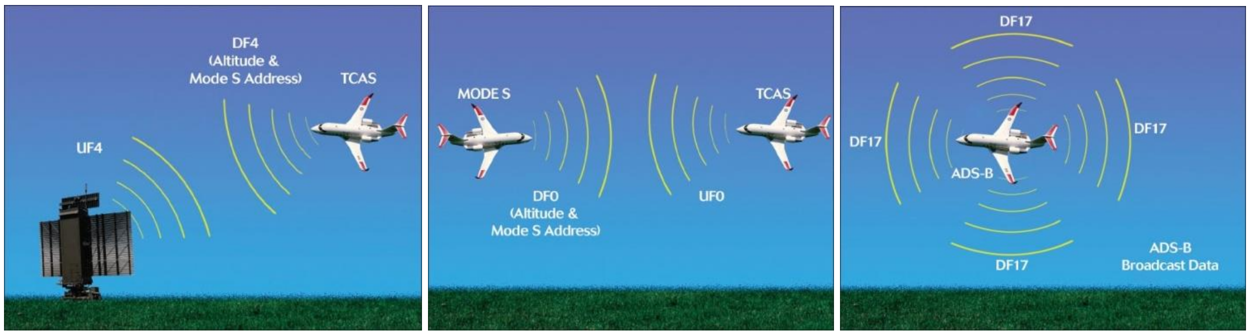


Figure 4 Mode-S data link. Adapted from [5]

Since the receiver discussed in this thesis is designed for the mode-S signal sent by aircraft transponders, only the downlink data format is discussed. As shown in Figure 3 [3], each mode-S downlink frame contains a preamble and a data block. Pulse-position modulation (PPM) is used to send these frames on 1.09 GHz carrier.

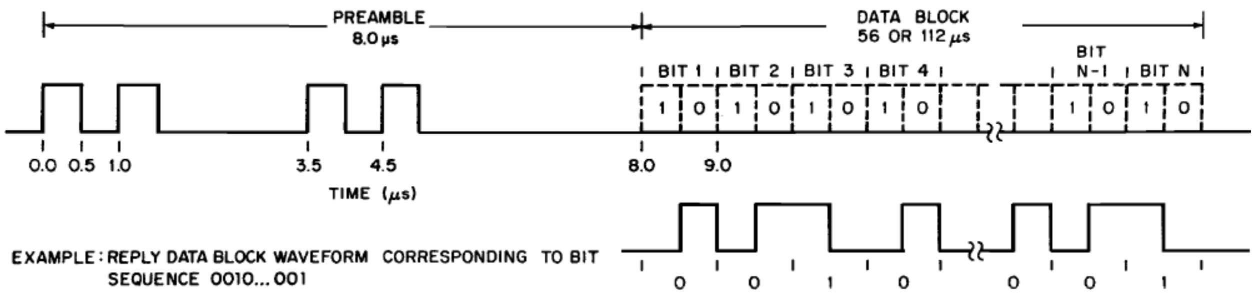


Figure 5 Mode S reply waveform

As defined by the International Civil Aviation Organization (ICAO), the first five bits of a mode-S frame indicates its type (DF0 to DF31) [4]. Moreover, the first bit of the data block represents the length of the frame, bit1=0 (DF0 to DF15) stands for 56-bit data block and bit1=1 (DF16 to DF31) stands for 112-bit data block. Since the data rate of the mode-S downlink is 1Mbps, the total length of each frame is 64μs or 120μs.

ADS-B

ADS-B stands for automatic dependent surveillance-broadcast which is a new generation of surveillance technique that allows aircraft to broadcast their own information such as GPS coordinate, altitude, flight number and other information via mode-S frames. As defined in mode-S standards, ADS-B uses DF11 and DF17 type of mode-S frames to broadcast information. Such a broadcast is called a “squitter”. As shown in Figure 6 [6], DF11 frames do not contain any aircraft information other than 24-bit ICAO address and DF17 frames contain an extra 56-bit ADS-B data block.

DF11 format Total 56-bit	01011 5-bit	Capability 3-bit	ICAO address 24-bit	Parity 24-bit	
DF17 format Total 112-bit	10001 5-bit	Capability 3-bit	ICAO address 24-bit	ADS-B data 56-bit	Parity 24-bit

Figure 6 ADS-B frame structure

The first five bits of 56-bit ADS-B data block in DF17 frames contain a subtype code (0 to 31) which indicates the type of information encoded in the 56-bit data block. Table 1 [6] shows some typical subtypes and their usage.

Table 1 ADS-B frame type

Subtype	Content
0	Barometric altitude or no altitude information, no position information
1 ~ 4	Aircraft identification and category message only
5 ~ 8	Surface position message only, type depends on horizontal containment radius limit
9 ~ 18	Airborne position message with barometric altitude
19	Airborne velocity message with difference between barometric altitude and GNSS Height
20 ~ 22	Airborne position message with GNSS Height (HAE)
23 ~ 31	Other

Figure 7 [6] shows the structure of subtype 9 to 18 56-bit ADS-B data block in DF17 frames, altitude and coordinate information are encoded.

Subtype 5-bit	Status 3-bit	Altitude 12-bit	Time flag 1-bit	CPR flag 1-bit	Latitude 17-bit	Longitude 17-bit

Figure 7 ADS-B airborne position frame’s data block

VHF radar

The VHF radar is a 2-channel VHF remote sensing radar interferometer, whose hardware was designed by Professor Steven J. Franke and software was designed by Professor Erhan Kudeki. Both channels are receiving on 49.8 MHz, but only channel 1 is transmitting at the same frequency with 15 kW (PEP) transmitting power and 140 µs pulse width for each 4 ms cycle. The average output power is 525 W and duty cycle is 3.5%. The 140 µs pulse is binary phase-shift coded with a 28 baud code. The baud length is 5 µs which sets the range resolution of the radar to 750 m. The receiving system samples the signal every 2.5 µs, corresponding to range-sampling-interval of 375 m.

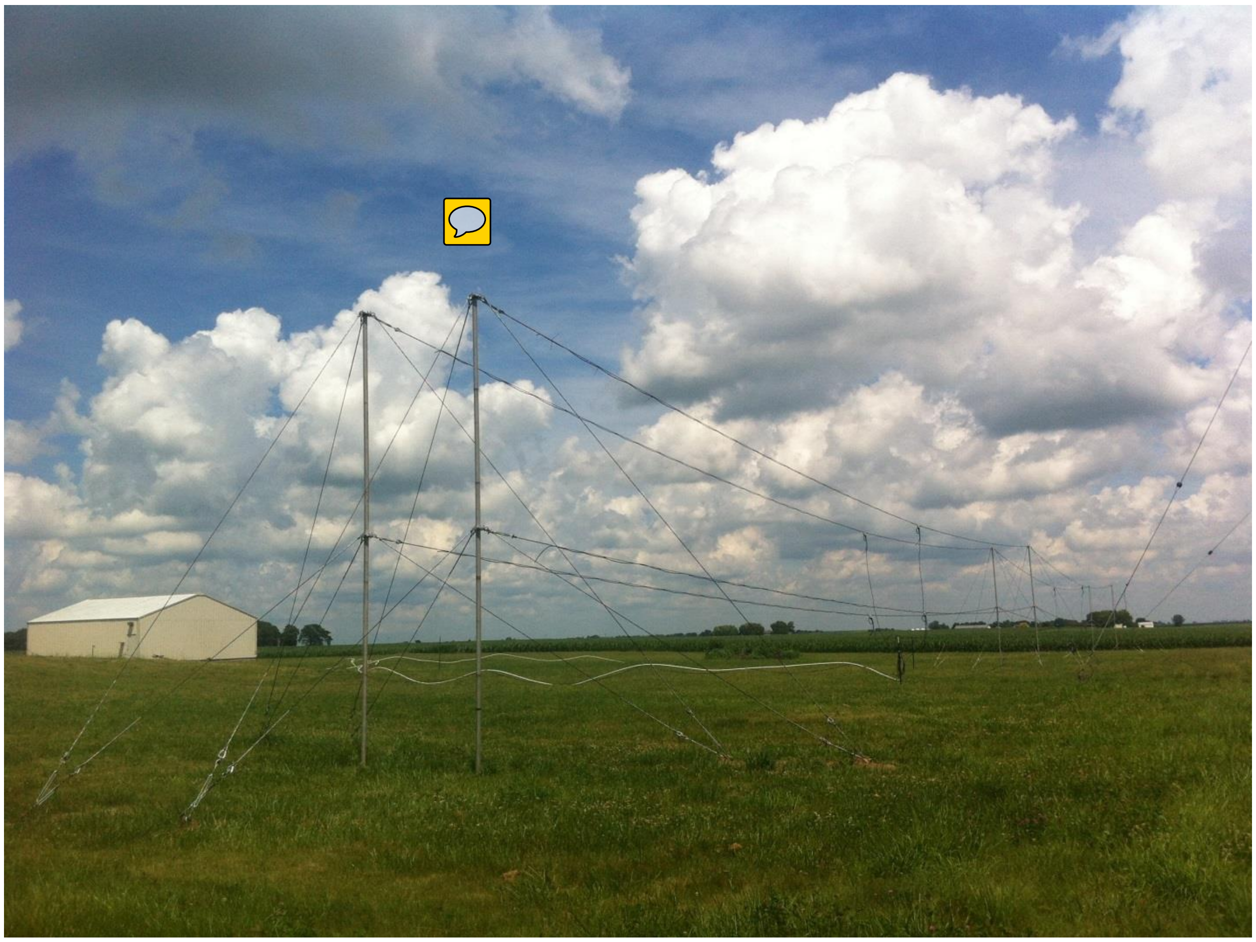


Figure 8 Photo of VHF radar antenna array

Each antenna array of the VHF radar is formed by four coaxial collinear antennas fed with different phase shift to create a narrow main beam directed toward the north. The main beam peaks at an elevation angle of approximately 17°. There are two antenna arrays built for the radar and they are lined up in the east-west direction (Figure 8). Since this antenna array is too large to perform an anechoic chamber measurement and commercial aircrafts can be spotted by the radar, the antenna pattern is characterized by aircraft echoes.

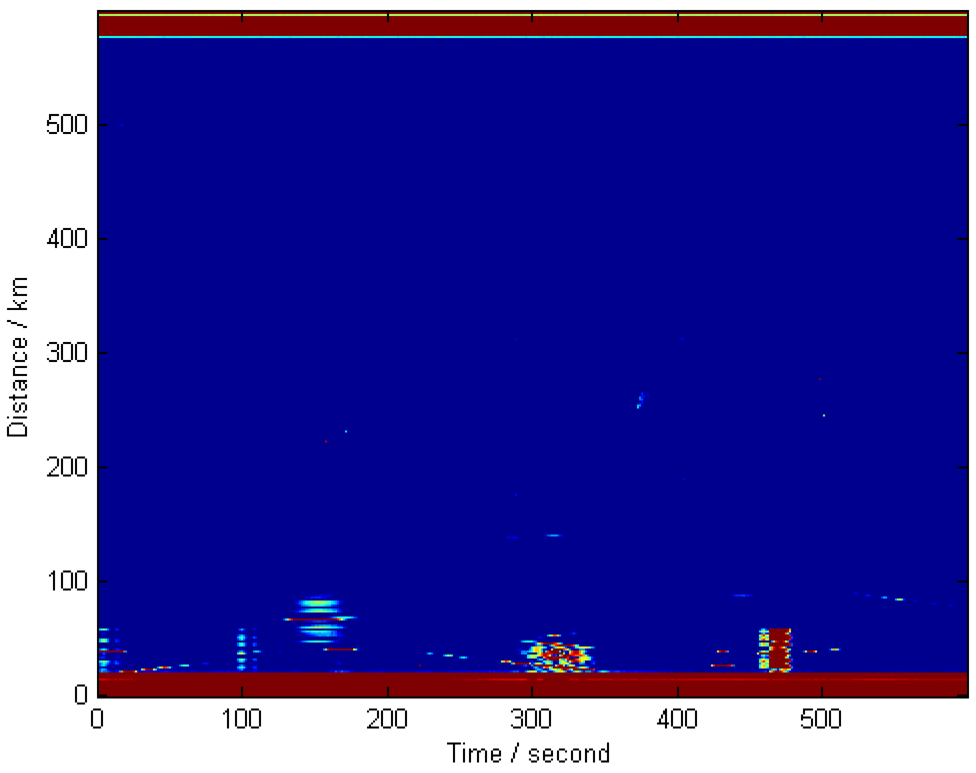


Figure 9 A typical radar image

This radar generates one MATLAB format output file containing the processed data for each ten minute period. Each output file contains two 600-second 2-dimensional radar images for each channel. As shown in Figure 9, x-axis of radar image stands for time, y-axis stands for distance between the radar antenna and aircraft and the color of each pixel on the image represents the corresponding reflection power. Most of the patterns in Figure 9 with distance smaller than 100 km are the aircraft echoes.

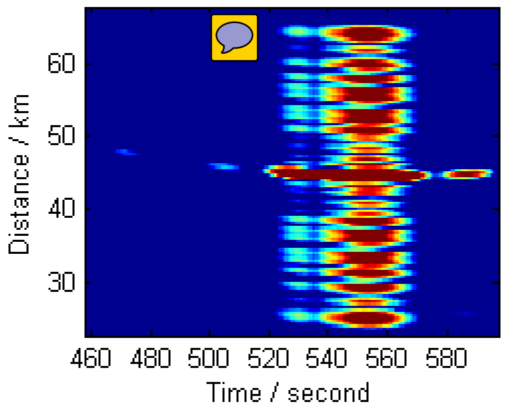


Figure 10 A typical aircraft echo

Figure 10 is a zoomed-in view of one of the aircraft echoes in another radar file. The distance between aircraft and radar antenna is around 47 km. The vertical pattern which shows up between 520 s and 570 s are range sidelobes from the 28 baud binary phase code that modulates each radar pulse. Moreover, the main trace of the aircraft is not continuous because the aircraft is moving between antenna beams (main beam, first side lobe, second side lobe, etc.) and the radar can only detect reflection power when the aircraft is in one of those beams.

2. Receiver Design

Product Overview

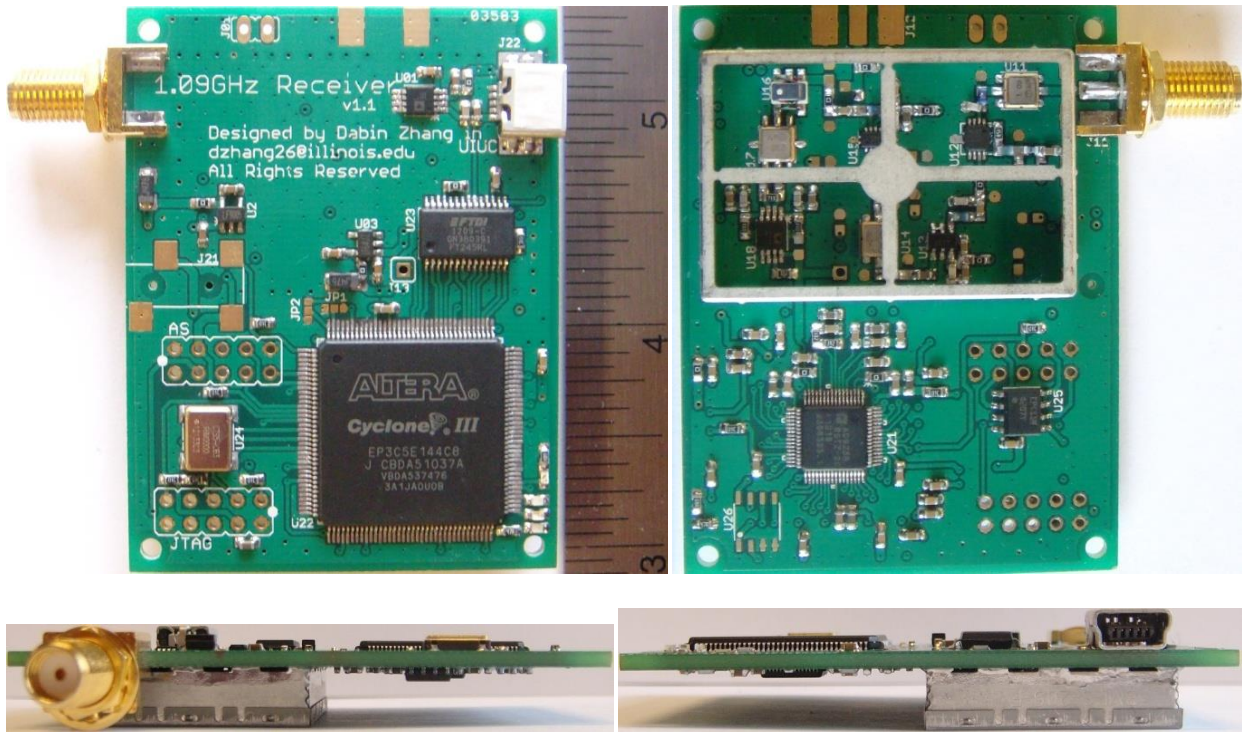


Figure 11 Top, bottom, and side view of receiver

The physical dimension of the receiver is 2.39”×1.83”×0.47” and it is implemented on a 4-layer PCB with through hole vias. The total development time of the receiver is two months with one person and $1000 funding. Average cost of each prototype receiver is $176 and the price will decrease dramatically if the quantity increases (Table 2, all prices in USD) since larger quantities will have lower component costs and automated PCB assembly techniques can be used.

Table 2 Quantity versus cost

Quantity	ICs	Other Critical part	Miscellaneous part	PCB	Assembly cost	Total cost
1	75.71	24.02	7.28	6.44	63	176.45
10	71.1	23.28	5.47	6.44	63	169.29
100	56.49	14.52	5.65	0.87	0.98	78.5
1000	44.11	10.61	5.88	0.41	0.78	61.78
10000	44.11	10.25	4.8	0.37	0.71	60.23

There are some similar products available commercially including “mode-S beast”, “AVR receiver”, “SBS receiver” and Table 3 shows the comparison in performance and price (USD).

Table 3 Comparison of some existing mode-S receivers

Model	Price	Structure	Development Team	Comment
Mode-S Beast 1 Channel	380	ADC FPGA	Multiple engineers	High price
Mode-S Beast 2 Channel	519	ADC FPGA	Multiple engineers	High price
Transponder-Mouse	510	ADC FPGA	Multiple engineers	High price
microADSB	95	Microcontroller	Small commercial company	Low performance
Kinetic SBS-3	611	ADC FPGA	Commercial company	High price

As can be seen in Table 4, those products either have a high price with high performance or low price by using simple and low performance structure. Since both of them have a relative low performance/price ratio, the goal of this design is to create a receiver which has moderate price and high performance.

RF Design

Overview

The RF part can be divided into two parts. The function of the first part is amplification and channel selection, which is accomplished with a combination of surface acoustic wave (SAW) band pass filters (BPF) and RF amplifiers. The function of the second part is envelope detection which extracts the envelope of the input RF signal, similar to an amplitude modulation (AM) demodulator. There are two kinds of SAW BPF available at 1.09 GHz with bandwidths of 30MHz and 12MHz. These are narrow enough for channel selection, so there is no need to implement an intermediate frequency (IF) stage for further channel selection. Thus this is a tuned radio frequency (TRF) receiver and only works at 1.09 GHz.

Since the distance between the receiver's antenna and the aircraft may vary from less than one kilometer to several hundred kilometers, the RF part requires both high sensitivity and high dynamic range. Moreover, the envelope signal of RF frontend is generated by a logarithmic detector which measures the overall input signal power, so the noise added by the amplification stages has to be minimized. Also, the overall power consumption has to be low because this receiver is designed to be powered from the USB bus which can provide maximum power P=5V×500mA=2.5 W.

Component Selection

To select the transponder's signal at 1.09 GHz, a proper filter has to be selected. There are four different filters available at this frequency and two of them, TA0232 and TA1090EC, were selected. TA1090EC is selected because it has wider bandwidth and lower insertion loss. TA0232 is selected because it has narrower bandwidth.

Two low noise amplifiers (LNA), ADL5523 and MAX2640, were selected to provide low noise amplification for the RF front end. ADL5523 has relative higher gain and lower noise than the MAX2640, but it has higher power consumption. To provide sufficient gain for the envelope detector part, a general purpose gain block AD8354 was selected because it does not require complex external circuitry, such as matching a network, and it has high gain.

There are multiple methods for demodulation but envelope detection method is the most convenient structure for this receiver. To ensure the performance and efficiency, a logarithmic detector is used instead of a Schottky diode and capacitor which used in AM demodulation. AD8313 is selected because it has moderate response time and power requirement.

To connect the amplification part with TA0232 SAW BPF and envelope detector, a balanced to unbalanced converter (BALUN) is needed to convert unbalanced output signal into balanced. TC1-1-13M+ is selected for this purpose since it has low insertion loss and impedance transfer ratio is 1:1.

Matching Network

All the LNAs in this design must be properly matched to achieve highest performance at the design frequency. All the matching networks were implemented as L-networks.

ADL5523

The ADL5523 requires an input matching network located at its input port and an output matching network located at the power port (pin8). Output (power) port matching is the first step to match the amplifier and input port matching can be determined when optimum S_{22} is achieved. Since the output port matching is hard to characterize with common L-network matching model and example matching networks for some common frequencies were given by the datasheet, the matching network was first built for 0.9 GHz and then move to 1.09 GHz by adjusting components values. The key element for output matching is L_6 (Figure 12).

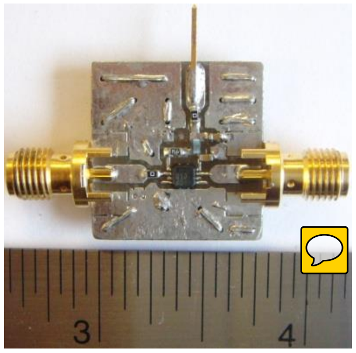


Figure 12 Prototype for ADL5523

For prototype circuit (Figure 12), finished matching network gives gain G=15.24 dB and noise figure NF=1.2 dB with power consumption P=105 mW.

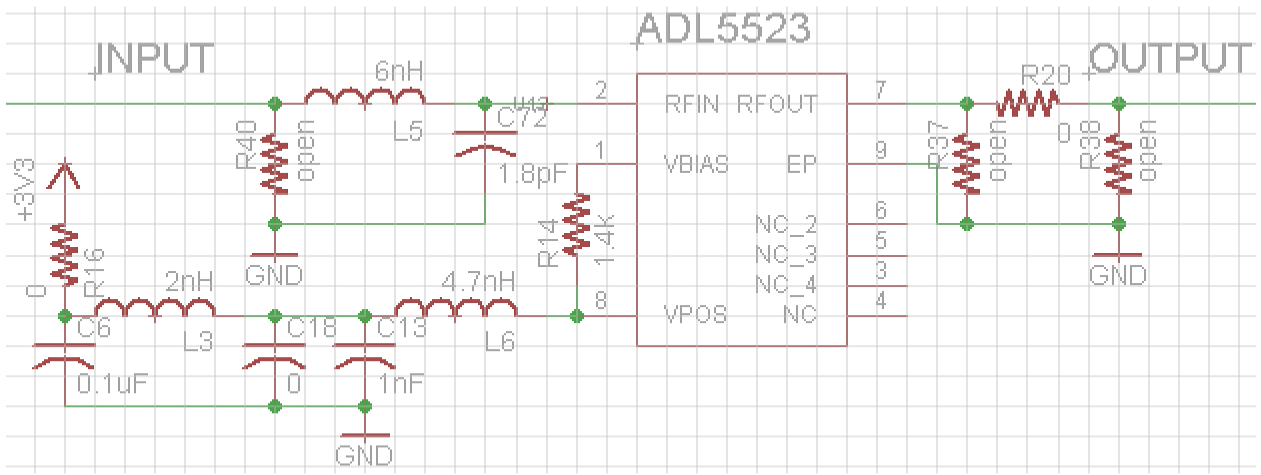


Figure 13 Circuit schematic for ADL5523

The finished circuit schematic for ADL5523 is shown in Figure 13, $R_{40}, R_{37}, R_{20}, R_{38}$ are reserved for further adjustment and not used for this design.

MAX2640

The MAX2640 requires matching networks on its input and output ports. Since the matching network design of this amplifier can be modeled as a common matching network, the S-parameter of the amplifier at 1.09 GHz has been measured and the components' value of L-network was calculated. Moreover, a recommended L-network design at 0.9 GHz was given in the datasheet and this example has been used as the starting point of matching network design.

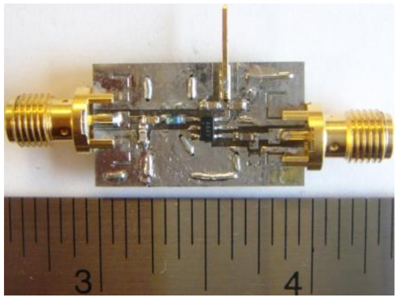


Figure 14 Prototype for MAX2640

By testing with different component, finished matching network gives gain G=14.05 dB and noise figure NF=2.17 dB on prototype board (Figure 14).

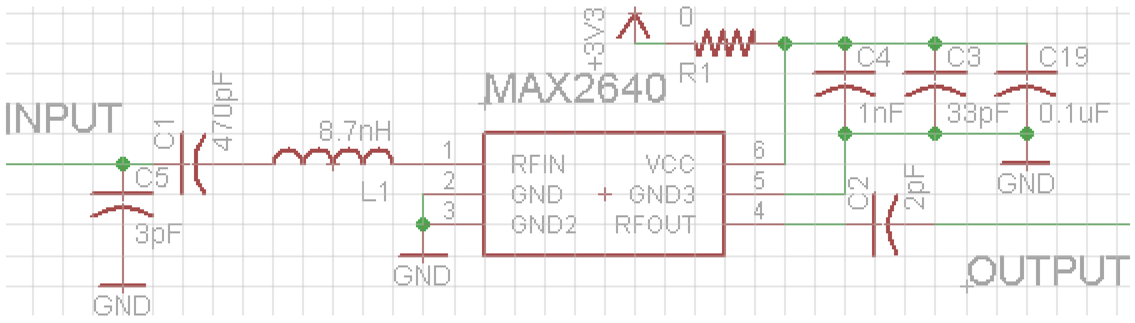


Figure 15 Circuit schematic for MAX2640

AD8354

Since this amplifier is a general purpose gain block and internally matched to 50 Ω, no extra matching circuit was implemented.

The finished prototype circuit (Figure 16) provides gain G=18.9 dB and noise figure NF=4.2 dB.

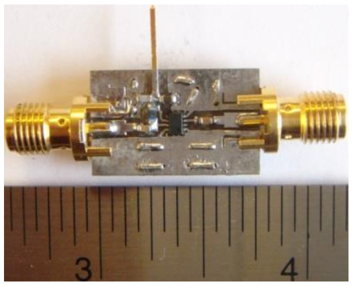


Figure 16 Prototype for AD8354

Final circuit schematic for AD8354 is shown in Figure 17. A 100 nH pull-up inductor L_2 was inserted between pin 6 (V_{CC}) and pin 7 (output) to increase gain.

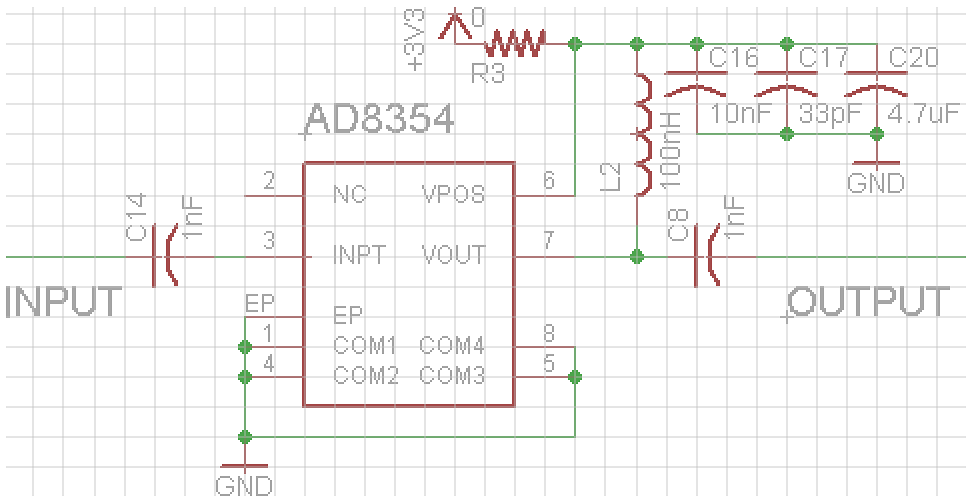


Figure 17 Circuit schematic for AD8354

Envelope Detector

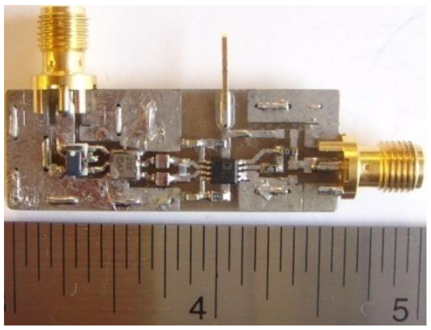


Figure 18 Prototype for envelope detector

The core of envelope detector is AD8313, a 0.1-2.5GHz 70dB logarithmic detector which output is the logarithmic of PEP input power (envelope) and this detector can detect input power from around -65 dBm to 5 dBm. In this application, the output of logarithmic amplifier is sampled by an ADC for further processing. The alternative solution of envelope detection is to implement an intermediate frequency (IF) stage and use ADC to sample the IF or base band (BF) signal. Table 4 lists the comparison of those two configurations.

Table 4 Comparison between envelope detector and IF stage

	Envelope Detector Implementation	Intermediate Frequency Implementation
Circuit Complexity	Single Chip (AD8313)	At least has mixer, LO, IF BPF
Power Consumption	3.3V×7mA=23 mW	>500 mW
Sensitivity	12 MHz BPF reduce SNR by 3	Mixer bring in extra noise
Cost	Low	High

Figure 18 shows a prototype version of the envelope detector, input signal has been converted to balanced and feed into TA0232 BPF before reach AD8313.

Final circuit schematic for AD8313 is shown in Figure 19. To match 50 Ω source to the input impedance of AD8313, a matching network were designed (R_8, C_9 and C_{10}) and R_8 can be replaced by an inductor to reduce ohmic loss.

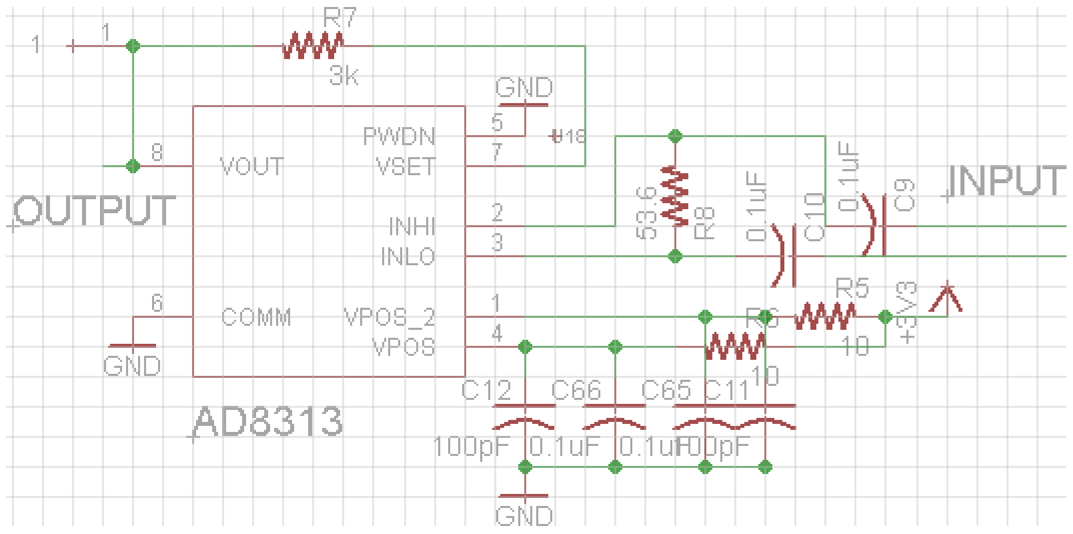


Figure 19 Circuit schematic for AD8313

Component Arrangement

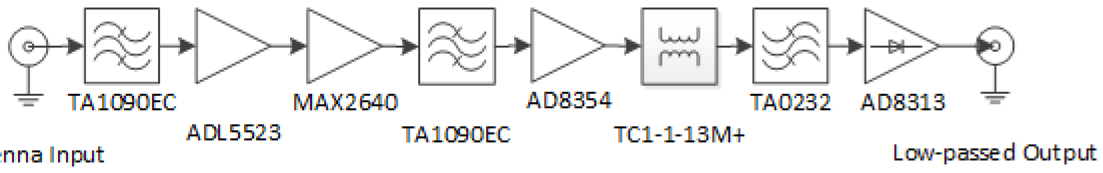


Figure 20 Block diagram for RF frontend

Since there may be strong interference sources in the receiver's working environment, a pre-selector is required to reject those unwanted components picked up by the antenna. However, insert a pre-selector between antenna and amplifier will not only bring in extra loss but can also increase the system noise dramatically. Thus TA1090EC was used for pre-selector because it has moderate insertion loss and 3 dB bandwidth.

To ensure the low noise nature of the RF front end, LNAs must be placed immediately after the pre-selector. Since ADL5523 has the lowest noise and relatively high gain (lowest noise measure), it was used as the first amplifier. MAX2640 was used as the second amplifier because it has higher noise figure than ADL5523 but lower power consumption. Moreover, MAX2640 can keep its output power under the maximum rating of later stage when an excessive strong signal is fed into its input since the 1dB compression point is only -22 dBm.

An extra filter was placed after MAX2640 to attenuate harmonics generated by intermodulation caused by strong signals due to the MAX2640's relatively low input intercept level.

Since the total gain of the first two amplifiers is only 29dB, a third amplifier AD8354 is used to provide extra gain.

Table 5 Comparison of different amplifier and filter combinations

#	Circuit Configuration (same as signal flow)				Noise Figure dB	Gain dB	Power mW
1	ADL5523	ADL5523	MAX2640		1.2887	43.40	217.14
2	ADL5523	ADL5523	AD8354		1.2532	47.52	296.10
3	ADL5523	MAX2640	AD8354		1.2732	48.00	207.27
4	MAX2640	MAX2640	AD8354		1.8974	47.78	111.86
5	MAX2640	AD8354	AD8354		2.0173	51.95	187.53
6	ADL5523	TA1090EC	MAX2640	AD8354	1.4557	45.29	203.98

Table 5 Table 5lists different configuration for the amplification part with corresponding measured performance. The third configuration has the highest performance; therefore this combination was used for final design.

The last stage of RF frontend is envelope detector AD8313. To enhance selectivity, TA0232 was placed before AD8313 to limit the equivalent noise bandwidth to 12 MHz and TC1-1-13M+ is placed between AD8354 and TA0232 to transfer the unbalanced transmission line into balanced.

Figure 21 is a photo of RF prototype board including the amplification part and envelope detector. The RF circuit is placed in a 1.5”×1” RF shield to prevent interference. The SMA socket on the left is antenna input, socket on the top is envelope signal video output and SMA connector on the bottom is an extra port reserved for RF testing.

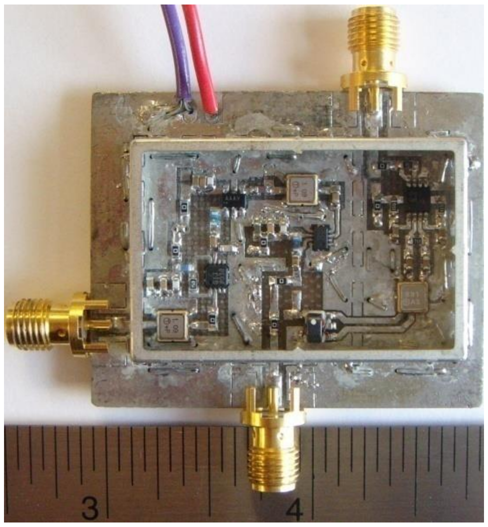


Figure 21 Prototype for RF frontend

Final PCB Design

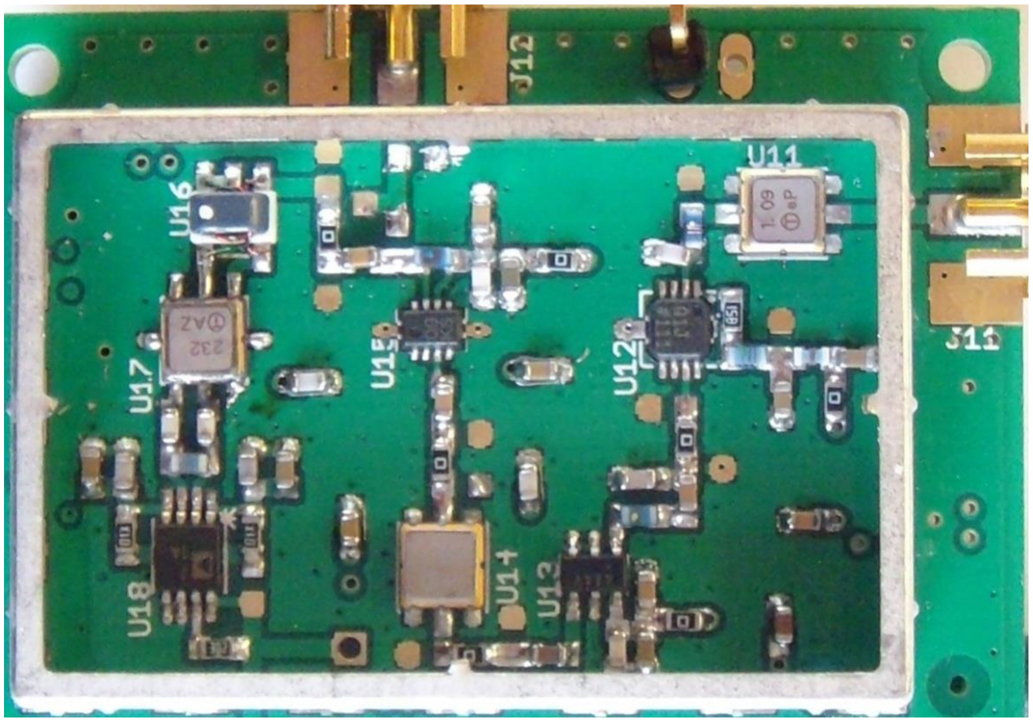


Figure 22 Final layout of RF frontend

As shown Figure 22, antenna output signal comes into the RF frontend through a SMA connector (J_{11}) via coplanar waveguide and envelope signal is fed into ADC. The envelope signal is also connected to the video test point located in the middle of U_{18} (AD8313) and U_{14} (TA1090). To have maximum isolation between different stages, the signal is guided in an “S” shaped structure.

Testing

Unless noted, 3.3 V DC power for RF frontend is provided by Agilent E3648A power supply and testing frequency is 1.09 GHz.

Amplifier-Filter Network Measurement

The amplifier network can be treated as a two port network and it can be characterized through S-parameters, noise, and nonlinearity measurements. Since the finished PCB has a reserved RF test port between the last amplifier and BALUN, the test port is used as the output of amplification part.

S-parameters and gain compression were measured using an Agilent E8357APNA with SOLT calibration standard (Agilent 85052D 3.5mm Economy Calibration Kit).

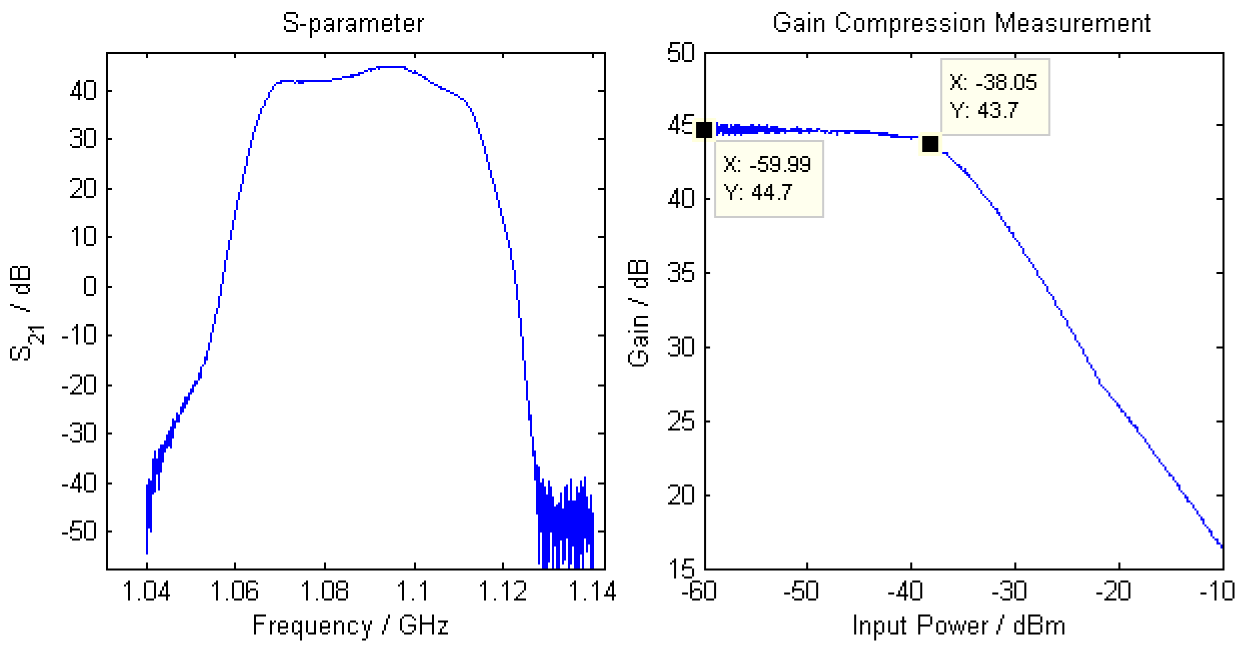


Figure 23 RF frontend S-parameter and gain compression measurement

At 1.09 GHz, measured $S_{21} = 44.07\ dB$, 1dB gain compression point $P_{1dB} = -38.05\ dBm$. 3 dB bandwidth of the amplification stage is $BW_{3dB} = 22.44\ MHz$, 30 dB bandwidth is $BW_{30dB} = 22.44\ MHz$ and $Shape\ Factor = 2.657$

Noise is measured through Noise Figure function in Agilent N9020A MXA Signal Analyzer with Agilent N4001A series noise source. As shown in Figure 24, measured noise figure NF=2.0 dB, gain G=44.4 dB at 1.09 GHz.

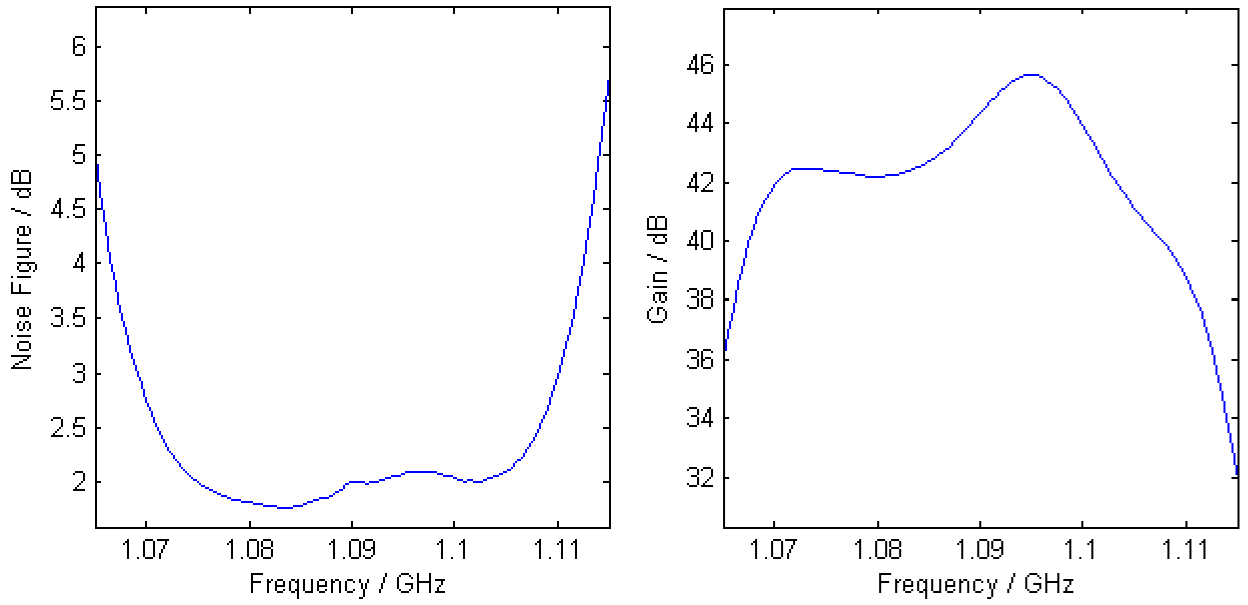
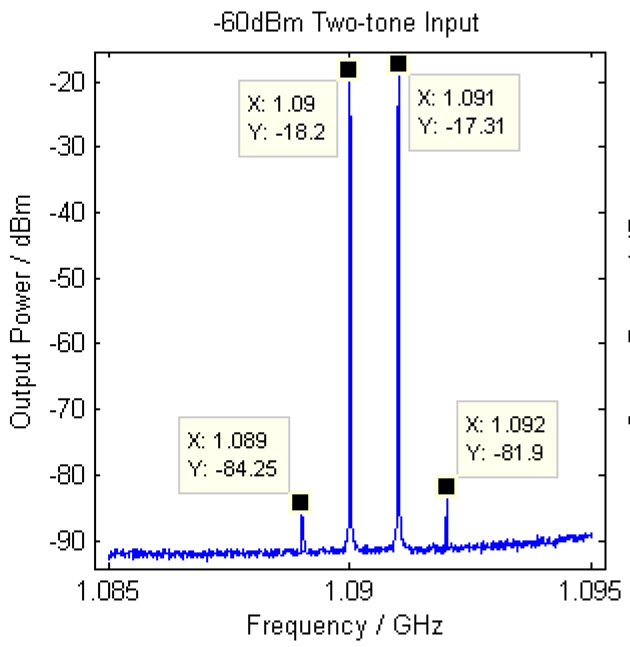


Figure 24 Noise measurement of RF frontend

Power consumption was measured by the built-in ampere meter of the power supply when a single tone is fed into the input port of RF frontend. When input signal is lower than -40 dBm, total power consumption of RF frontend is P = 3.3V × 75mA = 247.5 mW. Maximum power consumption is lower than 257.4 mW with -10 dBm input signal.

Two-tone third-order input intercept (IIP3) was measured by the following setup: two different input tones were generated by two HP E4432B ESG-D series signal generators and combined by Mini-Circuit ZESC-2-11+ splitter before fed into the receiver. The output spectrum was measured through spectrum analyzer mode in Agilent N9020A MXA Signal Analyzer at receiver's RF testing port. Measurement shows the overall two-tone third-order input intercept (IIP3) is -27.4 dBm



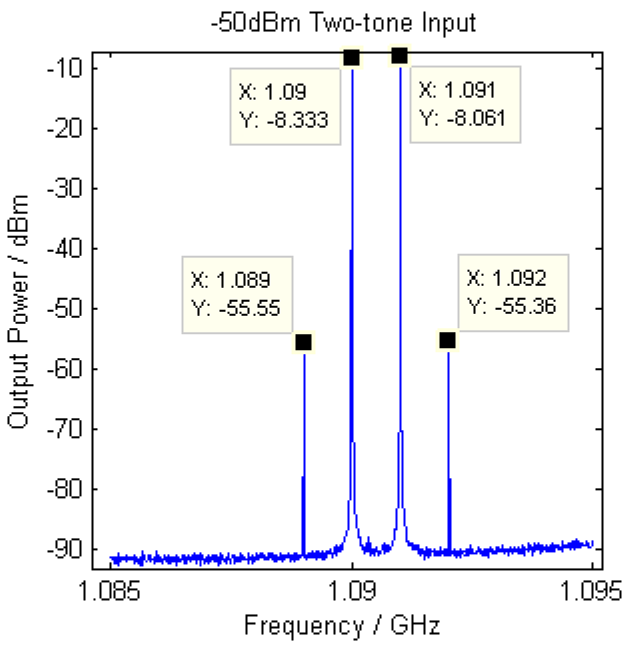


Figure 25 Two-tone third-order measurement

Envelope Detector Measurement

Since the envelope detector's output is the envelope (power) of the input signal, it can be characterized by input power and output voltage. As shown in Figure 22, output voltage of envelope detector was obtained from the video testing point by Agilent 34410A multimeter and RF testing port was used as input port which fed by HP E4432B signal generator.

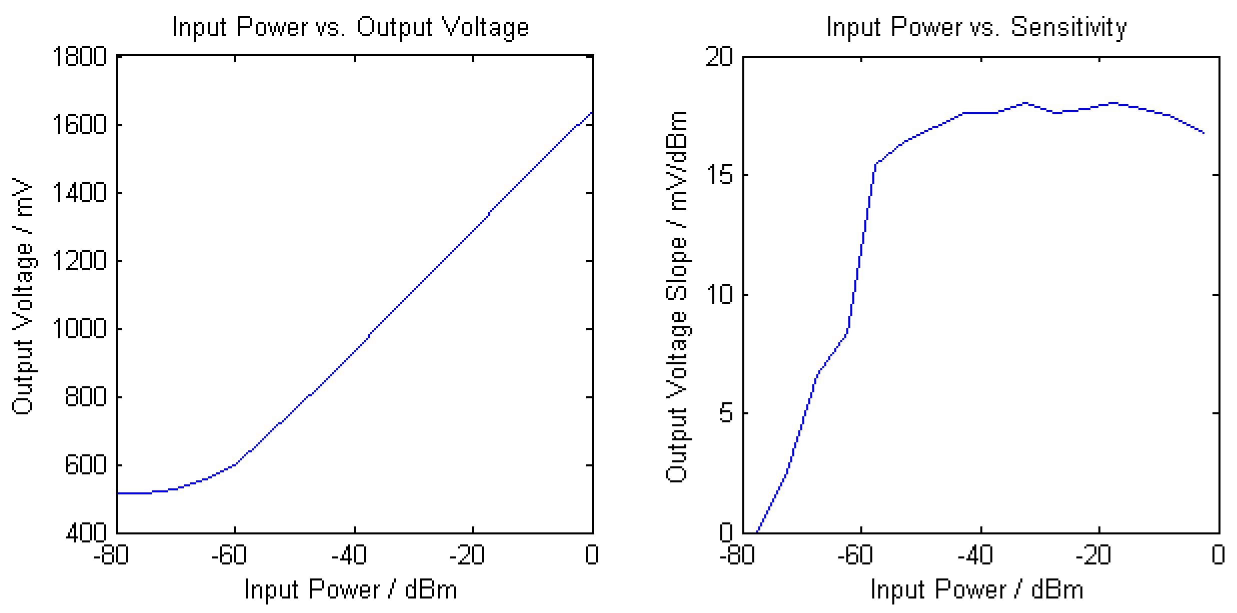


Figure 26 Envelope detector measurement

The measured input RF power versus output voltage curve and its slope (sensitivity) were plotted in Figure 26 shows the envelope detector can detect input signal stronger than -60 dBm.

RF-Video Measurement

This measurement is actually a combination of the previous two measurements which characterize the relationship between input signal power and output voltage of envelope detector. However, the result is not simply the sum of those two previous measurements because neither amplifier nor envelope detector is an ideal 50 Ω device.

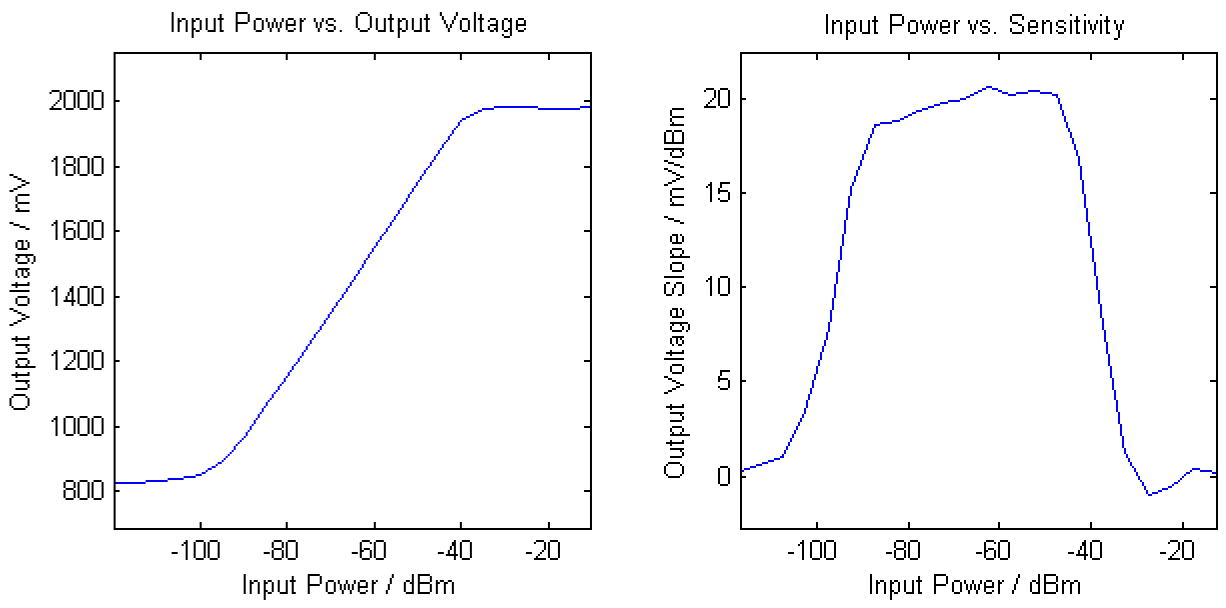


Figure 27 RF frontend RF-video measurement

The measured input RF power versus output voltage curve and its slope (sensitivity) is plotted in Figure 27 shows the RF frontend can detect input signal stronger than -95 dBm.

Digital Design

Overview

The digital part of the receiver handles sampling, mode-S frame recovering and transmitting data through USB interface. As described above, the envelope signal is generated by the RF front end and sampled by the ADC. The data rate of mode-S signal is 1 Mbps and its Nyquist sampling rate 2 MHz. The envelope signal has been oversampled because the ADC sampling rate is 16 MHz.

System Level Design

To convert the analog envelope signal into a sampled data sequence, an analog to digital convertor (ADC) is needed. Field programmable gate array (FPGA) is used for processing the sampled data flow and reconstructed mode-S frames are sent to the computer through USB interface as an ACSII string.

Component Selection

The ADC has the following requirements: sampling rate no less than 2 MSPS; low power consumption; high accuracy. The ADC should also have a user friendly package for soldering and can be powered by single power supply. Using these requirements for selection, the AD9238-20, a dual 12-Bit 20MSPS 3V ADC, was selected.

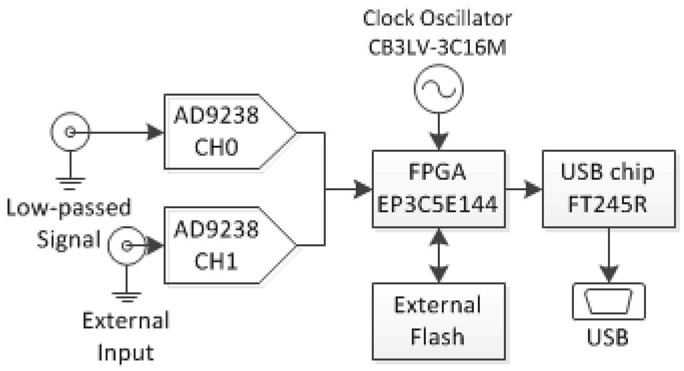


Figure 28 Block diagram for digital backend

The FPGA performs mode-S frame reconstruction and it is the main controller of digital backend. Since there are several families of FPGAs with different performance, the FPGA used in the receiver has to fit all design requirements properly to minimize cost, design difficulty and maximize performance. To estimate resource requirement of FPGA, a sample firmware was implemented on ALTERA development and education 2 (DE2) board. With considering all the above requirements, one of ALTERA CYCLONE III series' FPGA, EP3C5E144C8N, was selected.

In order to send the recovered frames to computer, the receiver should have some data interface such as universal serial bus (USB), parallel port or serial port. USB is the most commonly used interface for today's computer but it has a relative complex control protocol. Serial port is easier to use but most computers no longer support this data interface. Thus, an FIFO to USB virtual serial port interface, FT245R, was selected. This interface chip connects to computer via USB port but the computer treats this device as a serial port.

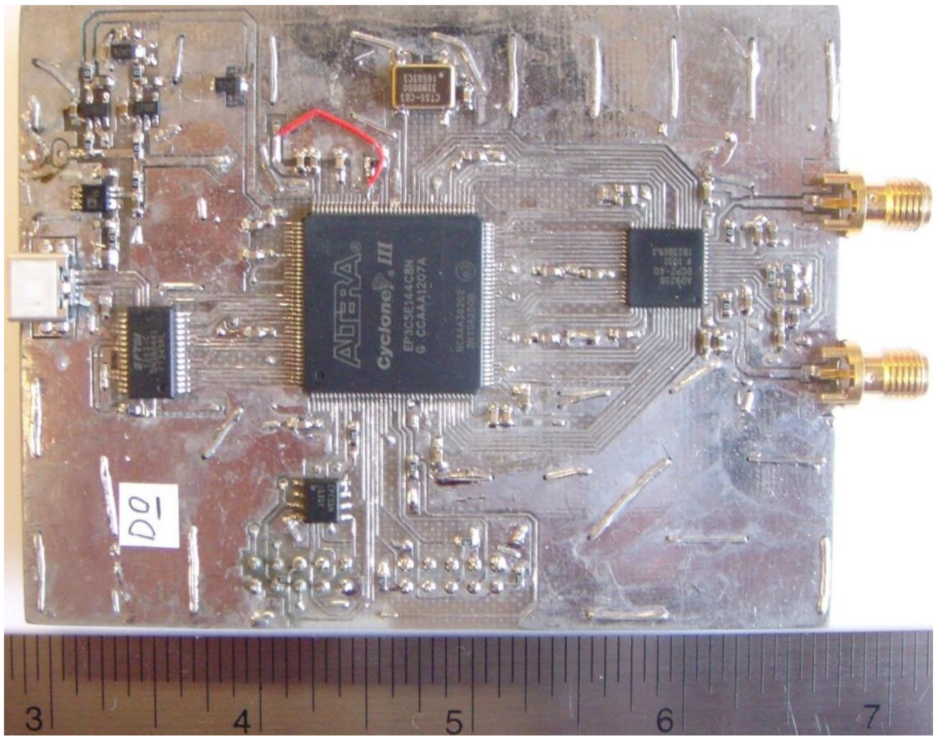


Figure 29 Prototype for digital backend

PCB Design

Since the physical dimension of FPGA is 22 × 22mm, it occupies the dominant area of the digital part. The routing between ADC and FPGA needs to be considered carefully because ADC has 64 pins and 36 of them are data interfaces which need to be connected with FPGA.

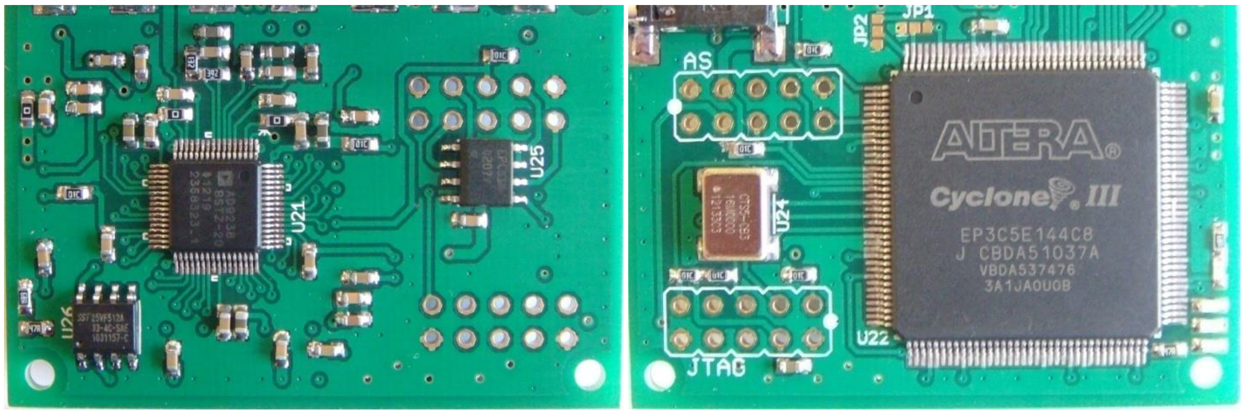


Figure 30 Final PCB layout for digital backend

To save routing area, as shown in Figure 30, the FPGA and ADC are centered but located on opposite sides of the PCB. Most of the input/output (IO) routings are under the FPGA and ADC. FPGA's external memory and configuration sockets are located on its backside while LEDs are located on the edge of PCB.

The USB interface has been placed on the backside of the RF frontend to save space because it has relative low routing complexity and its internal clock oscillator needs to be powered by the 5 V USB bus.

Firmware Design

Overview

FPGA firmware determines the functionality of the receiver and it was designed to recover mode-S frames in this design. The envelope data of input signal has been sampled by ADC and fed into FPGA, recovered frame is sent to computer via USB interface as string.

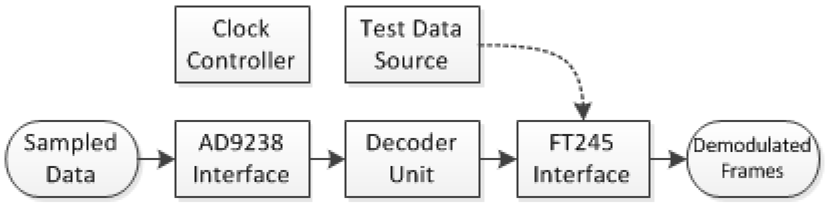


Figure 31 Block diagram for FPGA firmware

As shown in Figure 31, this firmware consists three parts: ADC interface, demodulator and FT245 (FIFO) interface. The ADC interface configures ADC and receives ADC's sampled data. Demodulator locates mode-S preamble, perform synchronization and determine the value of all the bits in the mode-S data block. Once the value of all the bits has been determined, the recovered frame will sent to the FIFO interface, convert into ACSII code string and sent to computer via USB interface.

Demodulator

The demodulator transfers the input raw data to frames, it contains arithmetic unit, demodulating cores and data buffers.

Arithmetic Unit

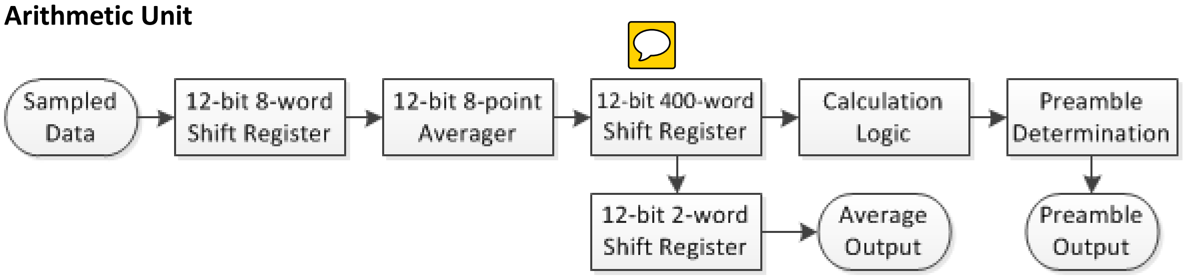


Figure 32 Block diagram for arithmetic unit

The arithmetic unit performs all the mathematical calculation for the demodulator such as averaging, subtracting and comparing. As shown in Figure 32, the input data flow has been first buffered by an 8-word shift register for an 8-point average calculation. Since the envelope signal of RF frontend has been oversampled to 16 MHz, this average calculation acts as a digital LPF to eliminate high frequency interference.

The averaged data is then fed into another longer shift register which provides 4 µs delay. As shown in Figure 5, mode-S frames have a preamble that lasts 8 µs and it can be detected by searching for peaks in the delayed data from the shift register. By calculating signal to noise ratio (SNR) of certain point of data stored in the shift register, this unit can determine whether a preamble is present or not and this judgment result is fed into the demodulating core to enable the demodulating process.

Demodulation Core

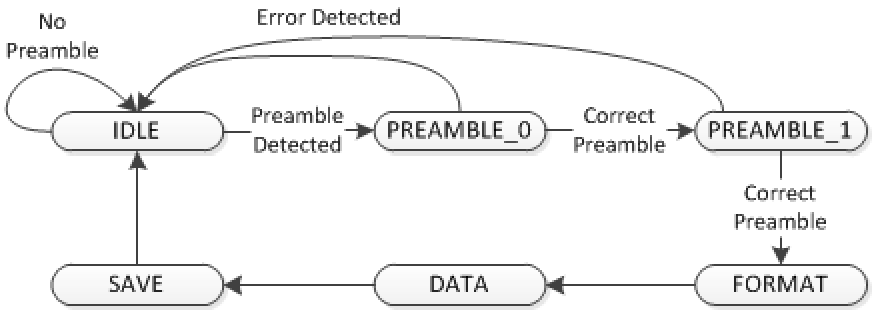


Figure 33 State diagram for demodulation core

The demodulating core will leave IDLE state when it is triggered by the arithmetic unit when a possible preamble is present. However, each preamble contains two groups of pulses which have a 2 µs distance thus first two states (PREAMBLE_0 and PREAMBLE_1) of the demodulating core were designed to verify both groups of pulses in the preamble are present and correct. If the preamble passed the check, the demodulating core will move to FORMAT state and get the first bit of data block to determine the length of the frame otherwise the demodulation core will go back to IDLE state and wait for another possible preamble. Once the length of data block has been determined, the state machine will move to DATA state which enables a build-in counter to count 56 or 112 1 µs cycles to estimate the value of all the symbols in the entire frame. For each cycle, the demodulate core will compare the average value of the first 8 sample points and second 8 sample points of the current bit to determine its actual value and store the result into a shift register. When the count is complete, the demodulation core will go to SAVE state and transfer the entire frame from shift register into data buffer (FIFO).

Data Buffer

Since the demodulator has 4 demodulate cores and all of them may generate output data at the same time, each demodulation core should have a private FIFO to ensure all the output data can be stored properly. Moreover, output data of all the demodulation cores have to merge into FIFO interface which needs another buffer and control logic (Figure 34).

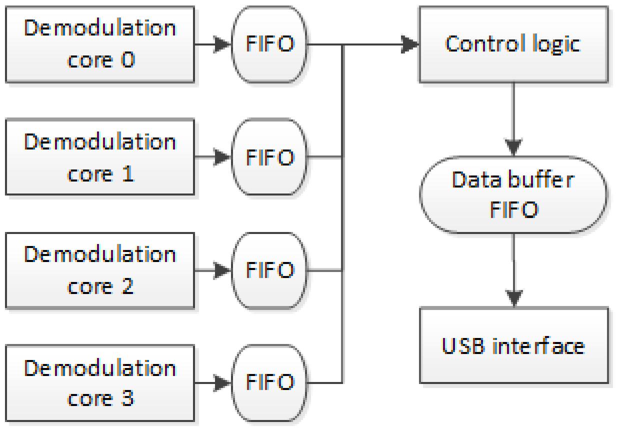
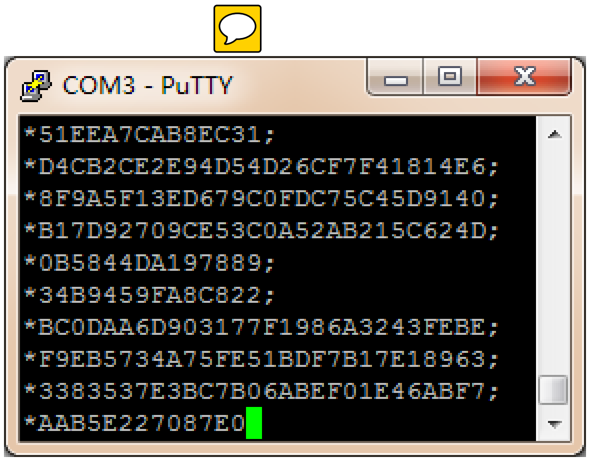


Figure 34 Block diagram for data buffer

The control logic is implemented as a state machine. This state machine checks and reads each demodulate cores' FIFO by sequence and transfer those data into a master FIFO.

USB (FT245) Interface

USB interface read the data from the previous stage and sends those data to the FT245R's built-in FIFO by sequence. This interface also inserts some formatting characters to make each frame recognizable for client software. Since the USB interface chip processes data at a relative low frequency but each cycle can process 4-bit, the actual highest data rate of the USB interface is less than 7 Mbps.



```
*51EEA7CAB8EC31;
*D4CB2CE2E94D54D26CF7F41814E6;
*8F9A5F13ED679C0FDC75C45D9140;
*B17D92709CE53C0A52AB215C624D;
*0B5844DA197889;
*34B9459FA8C822;
*BC0DAA6D903177F1986A3243FEBE;
*F9EB5734A75FE51BDF7B17E18963;
*3383537E3BC7B06ABEF01E46ABF7;
*AAB5E227087E0
```

Figure 35 Receiver data read by computer

To ensure the computer software can recognize the demodulated frames from the receiver, the ASCII string sent to computer is in AVR format. This format was designed by Edward John Cardew who designed a simple mode-S based on a TV tuner and microcontroller with firmware compiled in Atmel AVR Studio [7].

As shown in Figure 35, the formatting characters of AVR format include "*", ";", <CR> and <LF> and the mode-S data block is the located between "*" and ";". Data sent by this format is actually ASCII code and each ASCII code represents 4 bits of data. Since the raw data in the receiver is hexadecimal format, a look up table (LUT) was implemented to translate the hexadecimal number into corresponding ACSII character.

Other

There are four LEDs on the finished receiver to display information of the receiver and those LEDs can be configured for diagnostic or normal display purpose. The external flash was designed to store extra configuration for the receiver because FPGA will lose all the information for current state when powered off.

Testing

Testing FPGA firmware was designed for testing USB interface and ADC. FPGA was tested during USB or ADC test because both testing firmware requires the FPGA to be working properly.

To test the USB interface, a fake string was generated from the FPGA by a test firmware and this string can be received by computer. The test was success since the computer received all the fake frames correctly from the FPGA.

To test the ADC, a voltage meter firmware was implemented which sent the sampled data (voltage) to computer directly. By applying different voltage on the input port of ADC, the voltage was read from computer and indicated that the ADC was working properly.

System Design and Construction

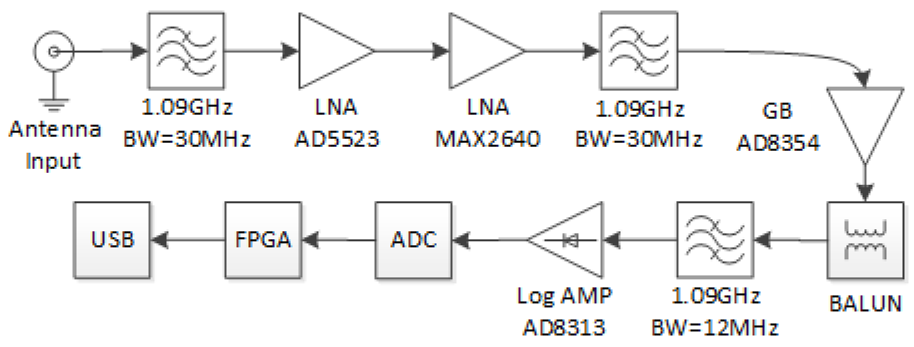


Figure 36 Block diagram for the receiver

Power Solution

As discussed above, the receiver is powered by 5 V USB bus and can draw no more than 500 mA current. The USB interface chip FT245R is powered directly by USB because its internal oscillator requires 5 V supply voltage. Since the RF frontend has best performance when supply voltage is 3.3 V and ADC works only at 3.3 V, 3.3 V was used as the main power for the receiver and it is generated through a 3.3 V voltage regulator ADP1715ARMZ-3.3-R7 which is a 500mA low-dropout CMOS linear regulator. The other supply voltages for FPGA are generated from 3.3V bus to reduce voltage drop on voltage regulators. Internal power $V_{INT} = 1.2\ V$ is generated by ADP151AUJZ-1.2-R7 and analog power $V_A = 2.5\ V$ is generated by ADP151AUJZ-2.5-R7 which is an ultra-low noise 200mA CMOS linear regulator. Figure 37 shows the power distribution network.

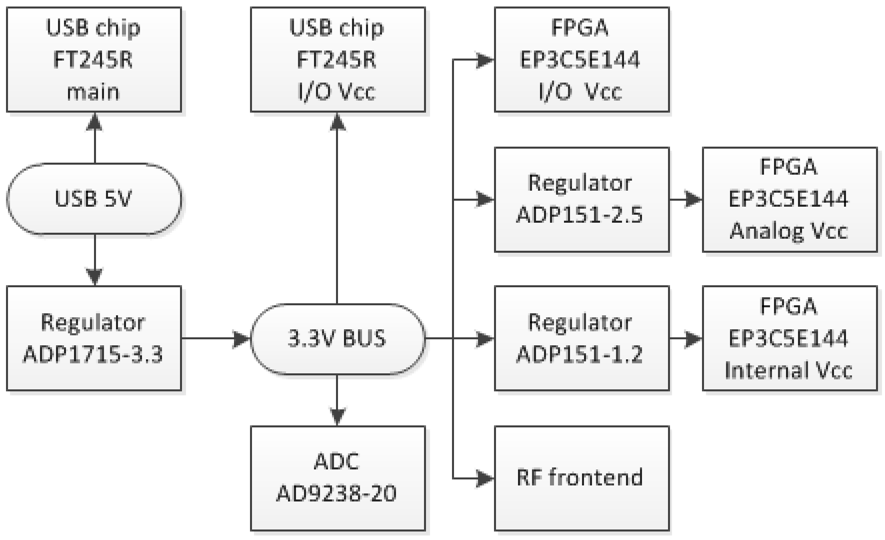


Figure 37 Power distribution network

Final PCB Design

To balance the design requirement and cost, the PCB used for this receiver is a 4-layer FR-4 substrate board with 35 µm copper thickness and through-hole via. Figure 11 illustrates the final PCB design with components installed. Since through-hole via holes influence all the layers of the PCB, via holes should be placed carefully to avoid influence the layout on the opposite side of the board. To ensure isolation for the RF frontend, via holes were placed closely and backside of RF frontend was only used by a few low complexity components.

Top and Inner Layer

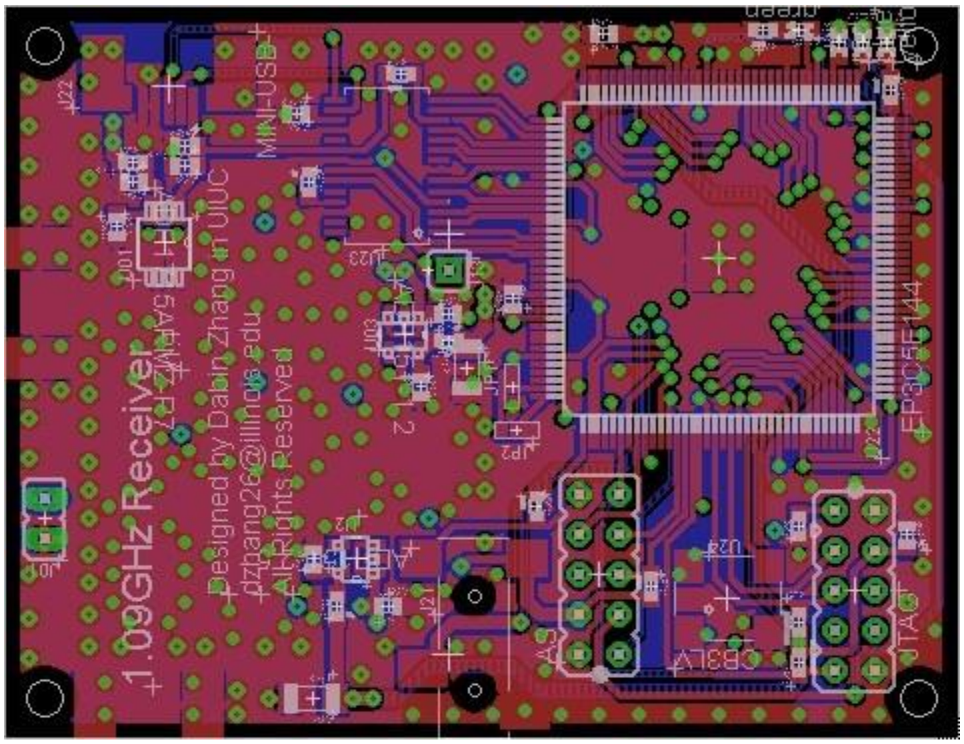


Figure 38 Layer 1 and 8 of final PCB layout

Bottom Layer

An inner layer is not shown together with bottom layer since the hidden layer was dedicated for ground.

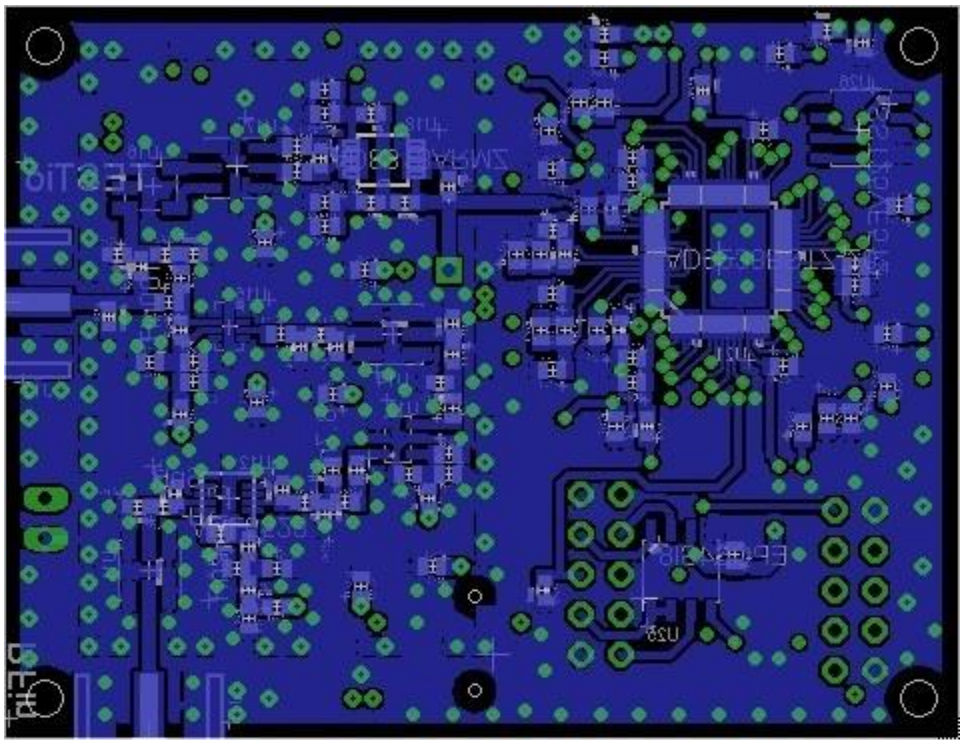


Figure 39 Layer 16 of final PCB layout

Assembly

The RF frontend is the first part installed on the PCB because certain RFICs, such as ADL5523 and AD8354, have a bottom pad which have higher soldering requirement. Thus, the first step of installation is to apply soldering paste on the bottom pad of ADL5523 and AD8354, solder their pins with soldering iron and perform a 240 ℃ reflow soldering. Later other RF components and some decoupling capacitors for digital part can be soldered. When the installation of RF part is done, a measurement is required to ensure RF frontend is functional. If the design for RF frontend is unchanged, a current measurement of RF frontend is enough to verify the installation.

The next step is install FPGA, ADC and their external components. After ADC is soldered on PCB, an extra microscope inspection is needed to check connection because the space between its pins is only 0.22 mm. The same inspection is also recommended for FPGA.

The third step is install all the voltage regulators and power the receiver by an external 3.3 V power supply and measure current. USB interface FT245R and USB socket can be installed if the test is success and USB interface should be recognized by computer when the USB cable is connected.

The last step is to download the firmware into the FPGA configuration flash by connecting USB-Blaster™ to the programming port of the receiver labeled as "AS" and use Active Serial programming mode in Altera Open Programmer for programming. Once the programming is done and USB cable is connected correctly, receiver should be able to send data to the computer.

Receiver Testing

PlanePlotter

The receiver was tested by PlanePlotter which is a commercial mode-S decoding software. This software can perform error detection, frame decoding and display received aircraft information on user interface.

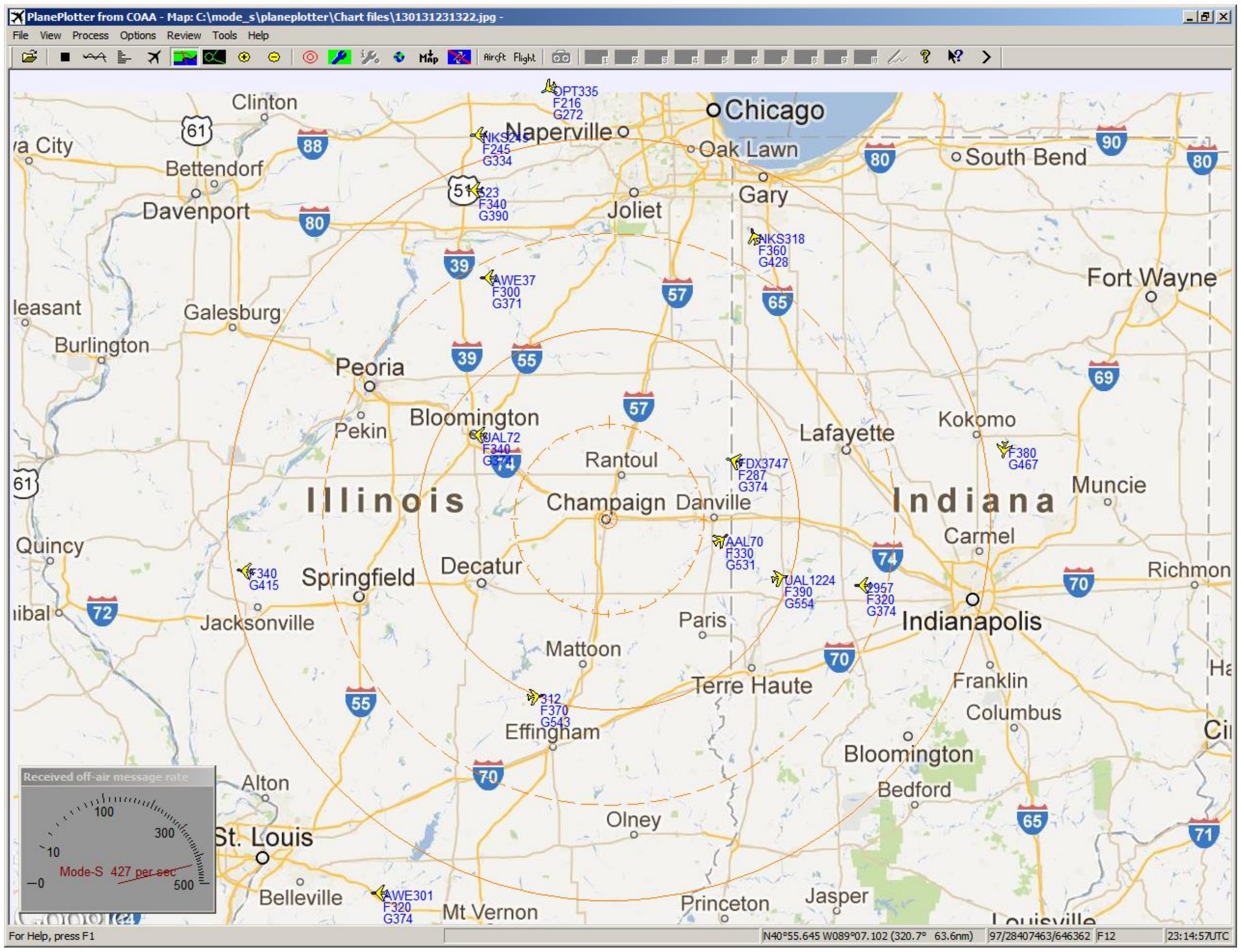

PlanePlotter from COAA - Map: C:\mode_s\planeplotter\Chart files\130131231322.jpg -
File View Process Options Review Tools Help
Received off-air message rate
Mode-S 427 per sec
For Help, press F1
N40°55.645 W089°07.102 (320.7° 63.6nm)
97/28407463/646362
F12
23:14:57UTC
Chicago
Champaign
Indianapolis
Illinois
Indiana

Figure 40 PlanePlotter screenshot in Urbana

Figure 40 is a typical observation screenshot of PlanePlotter, the orange cross in the center (Champaign County) represents the location of receiver's antenna. Once the receiver captures a position report, the corresponding aircraft is plotted on the map. The meter on the bottom-left corner indicates the average input valid frame rate which is 427 frame/second.

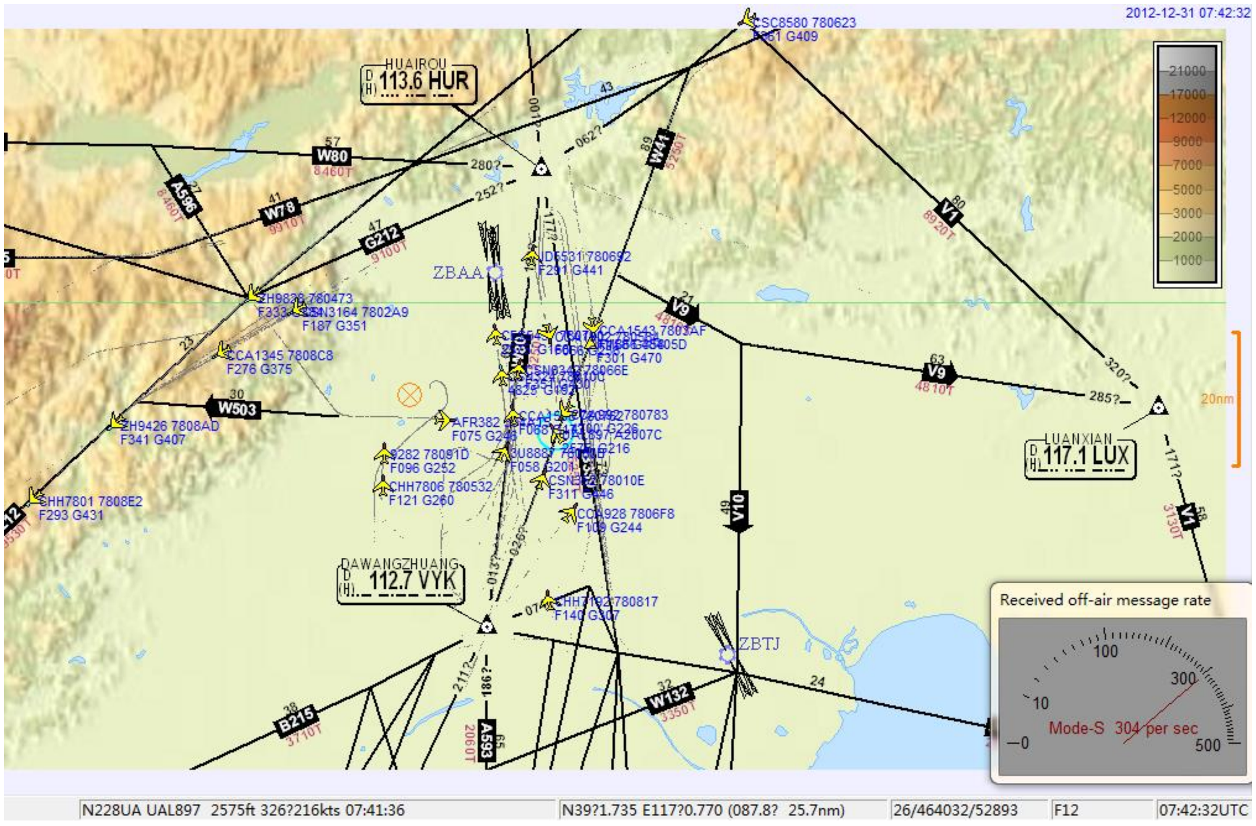
Test in Urbana

Most of the receiver's testing work was done in Urbana with an ICOM AH-7000 wideband discone antenna on the roof of Everitt Lab which gain is around 2 dBi. Antenna output is fed into the receiver by 200 feet LMR-400 coaxial cable with at least 9 dB loss.

The receiver's performance can be measured by running the receiver with PlanePlotter for a certain period of time. By observing the system for a week, the maximum detection distance was 193 nm (357 km) and 2.301% of total received frames contain position information (ADS-B message). The highest frame rate was 463 frame/second and typical frame rate was between 100 and 300. There were usually more than 50 aircrafts in detection range (received a frame within three minutes) and this number can be greater than 100 in some circumstances. Since the received frame rate and number of aircraft in range depend on air traffic density and there was no large airport close to the testing location, those numbers may not reflect the performance accurately.

Test in Beijing

The receiver was tested in Beijing during winter break (December 31, 2012 to January 2, 2013) with an unmatched vertical antenna located on the roof of a three-floor building.



N228UA UAL897 2575ft 326?216kts 07:41:36 | N39?1.735 E117?0.770 (087.8? 25.7nm) | 26/464032/52893 | F12 | 07:42:32UTC

Figure 41 PlanePlotter screenshot in Beijing

As shown in Figure 41, the maximum detection range was around 80 nm (144 km) because the antenna system was badly conditioned. However, a large number of aircrafts have been spotted since the antenna was only 35 km away from Beijing Capital International Airport (marked as ZBAA in the figure) and nearly all the civil aircraft in China broadcast ADS-B information.

3. Radar Antenna Characterization

Overview

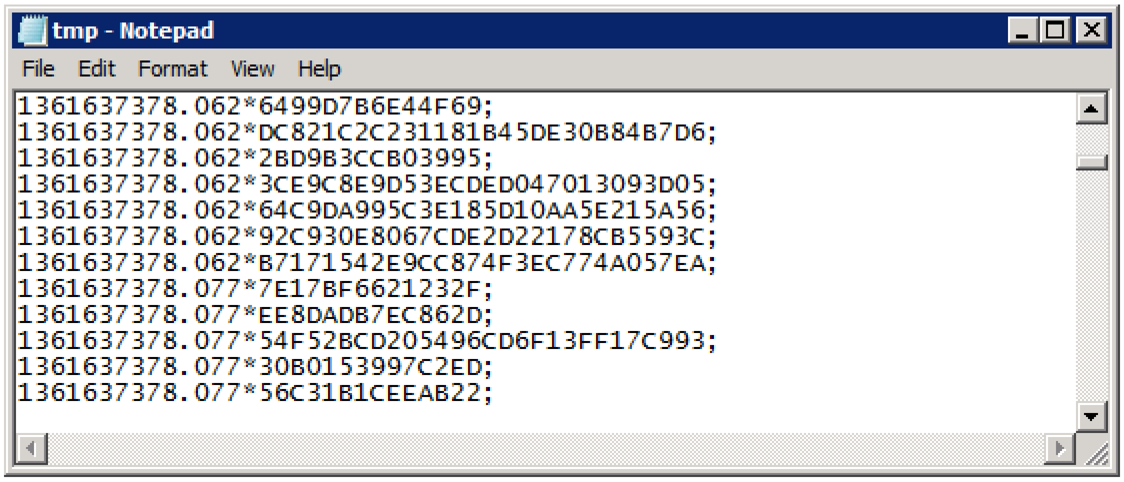
Although the radar was designed to observe field aligned plasma irregularities, it also detects common hard targets, such as aircraft, that fly through its antenna beam. Since the radar antenna array is too large to measure its far field radiation pattern in an anechoic chamber, and mode-S receiver can provide accurate trajectory information of some aircrafts, aircraft echoes can be used to map the antenna pattern.

Mode-S Data Processing

Unless noted, all the raw data processing programs were written and complied in C++ in Windows environment (Windows 7 Ultimate and Windows Server 2003 R2).

Data Collecting

Data collecting software was designed to collect enough raw mode-S data from the receiver with enough timing information for analysis. The main function of this program is attaching time stamp on frames sent by receiver and store the time stamped frames in the hard drive. The data collecting process took 24 days from Dec 25, 2012 to Jan 18, 2013. Approximately 152 GB of raw data was collected comprising 4.2061 billion raw frames.



```
1361637378.062*6499D7B6E44F69;
1361637378.062*DC821C2C231181B45DE30B84B7D6;
1361637378.062*2BD9B3CCB03995;
1361637378.062*3CE9C8E9D53ECDED047013093D05;
1361637378.062*64C9DA995C3E185D10AA5E215A56;
1361637378.062*92C930E8067CDE2D22178CB5593C;
1361637378.062*B7171542E9CC874F3EC774A057EA;
1361637378.077*7E17BF6621232F;
1361637378.077*EE8DADB7EC862D;
1361637378.077*54F52BCD205496CD6F13FF17C993;
1361637378.077*30B0153997C2ED;
1361637378.077*56C31B1CEEAB22;
```

Figure 42 Collected raw mode-S data

Figure 42 shows the output file when opened by Notepad, the number before asterisk is local time stamp when the frame was received from USB port and the hexadecimal string between asterisk and semicolon is the data block of the frame.

CRC Calculation and Data Selection

Since data analysis requires correct and useful mode-S frames such as ADS-B position reports, only valid frames were selected from the raw data.

To verify the data, a cyclic redundant check (CRC) [8] was applied to the message field of frames (first 24-bit for short frame and first 88-bit for long frame) and the calculation result is compared with the parity bits of each frame (last 24-bit). Only frames which passed the CRC calculation were kept.

For data selection, DF17 mode-S frames (extended ADS-B squitter) were extracted for decoding because they contain aircrafts' location information.

ADS-B Frame Decode

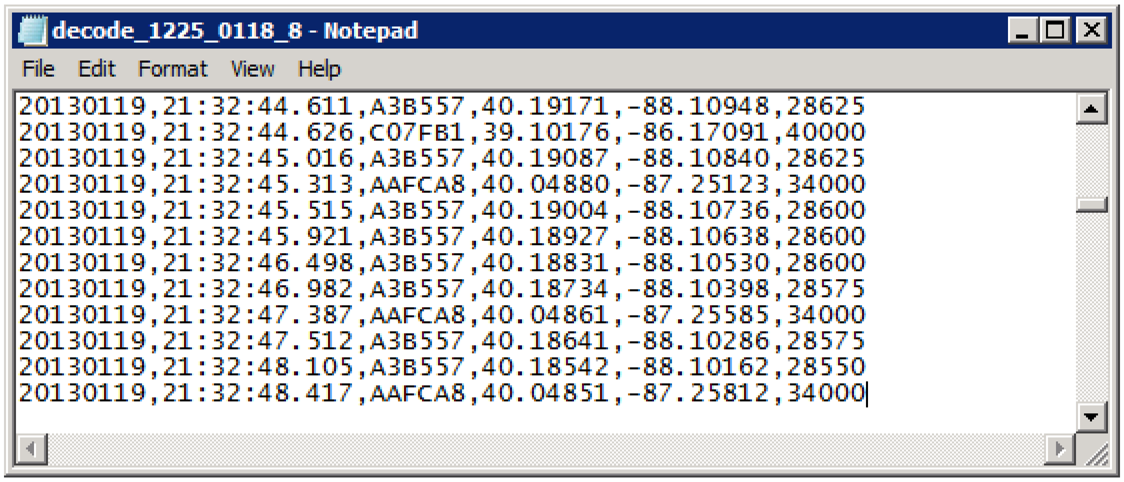
The longitude and latitude in ADS-B airborne position message frames (DF17) were encoded in compact position reporting (CPR) [9] format which requires a pair of frames send by the same aircraft to start decoding. As specified in ICAO manual, those two frames should have different bit 22 and be captured by the receiver within 10 seconds. To decode those frames, a data buffer was developed in the decoding program to store the newest received even frames (bit22=0) and odd frames (bit22=1) from each aircraft. Every time the program receives a new frame, it will check the data buffer and search the newest frame from the same aircraft which has different bit 22. Once a pair of frames has been collected, the program will use CPR algorithm which defined by ICAO to calculate the latitude and longitude of the aircraft.

Altitude Decode

Altitude information was encoded with coordinate together in ADS-B frames which can provide the 3-dimensional information of an aircraft. Moreover, altitude frames are 22 bits long and its resolution is 25 feet. The actual altitude of the aircraft can be calculated by (numerical-1000feet)×25feet.

Output Format

Since only type 9 to 18 of ADS-B frames (mode-S downlink format 17) have been decoded, the decoding output files follow the same format that contain the date, local time stamp, aircraft's ICAO ID, latitude, longitude and altitude of the aircraft. Figure 43 shows the output file when opened by Notepad. Total number of decoded position reports from data collection was 4.109 million, 0.977‰ of total collected mode-S frames.



```
20130119,21:32:44.611,A3B557,40.19171,-88.10948,28625
20130119,21:32:44.626,C07FB1,39.10176,-86.17091,40000
20130119,21:32:45.016,A3B557,40.19087,-88.10840,28625
20130119,21:32:45.313,AAFCA8,40.04880,-87.25123,34000
20130119,21:32:45.515,A3B557,40.19004,-88.10736,28600
20130119,21:32:45.921,A3B557,40.18927,-88.10638,28600
20130119,21:32:46.498,A3B557,40.18831,-88.10530,28600
20130119,21:32:46.982,A3B557,40.18734,-88.10398,28575
20130119,21:32:47.387,AAFCA8,40.04861,-87.25585,34000
20130119,21:32:47.512,A3B557,40.18641,-88.10286,28575
20130119,21:32:48.105,A3B557,40.18542,-88.10162,28550
20130119,21:32:48.417,AAFCA8,40.04851,-87.25812,34000
```

Figure 43 Output file of decoding program

Time Error Correction

After all the data was collected, a time error between the radar and ADS-B data was noticed. This error was caused by the local time error in the computer used to collect raw mode-S data, because the local time in Windows workstations only synchronize with internet time once a week with default settings. Moreover, if a failure occurs during time synchronization, Windows time service will only retry the synchronization a week later. Figure 44 shows the local clock drift for each 30-minute period, the average clock drift was 60.56 ms and standard derivation was 9.093 ms, thus local clock accuracy was 33.65 ppm.

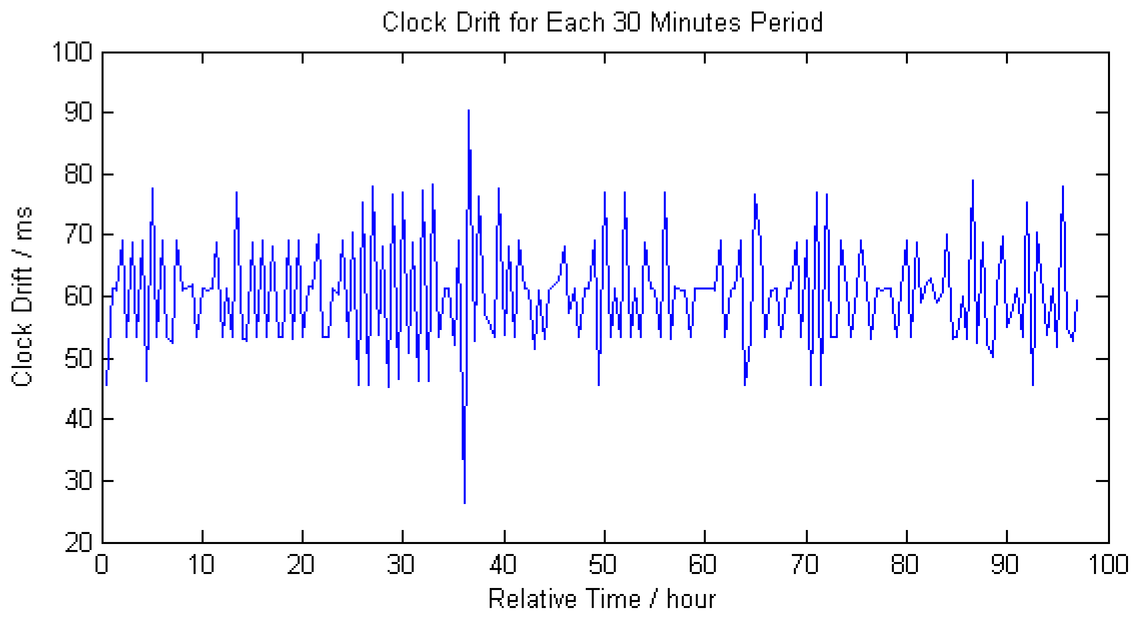


Figure 44 Local PC clock drift

Since the local clock drift was almost linear and the timestamp of when clock synchronization was successfully performed can be found in the Windows system log file, the local time error was corrected by calculating the clock offset value after the last successful clock synchronization and adding it to the time stamp of the decoded ADS-B data.

ADS-B Data Selection

Figure 45 plots all the ADS-B position reports collected in four days of data collection period. The center of the plot is the radar antenna's location and the range of this plot is 500 km in both east-west and north-south direction. The decoded trajectories (ADS-B position reports) were evenly distributed around the receiver, but only a small portion of those position reports were used for antenna pattern analyses. Thus extra data selections were performed.

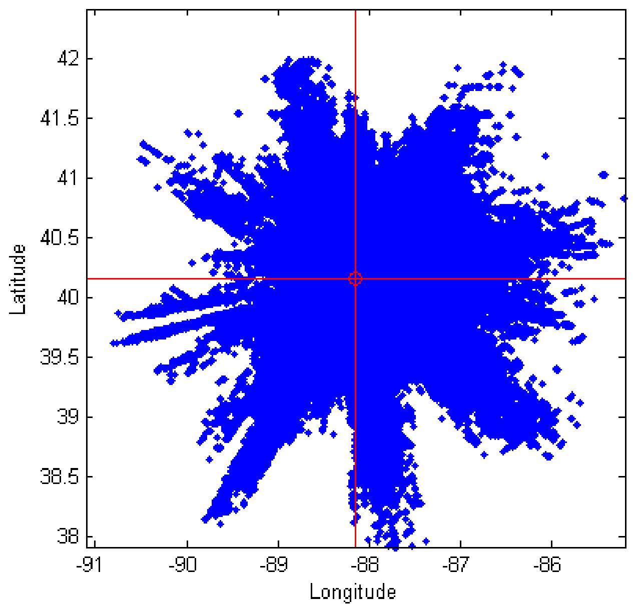


Figure 45 Plotted position reports without selection

Select Cross Beam Record

As discussed above, an aircraft echo only appears on the radar image when an aircraft crosses the radar's main beam. Since the trajectories which crossed the beam were needed for analysis, all the ADS-B position reports were filtered using this criterion.

Select Azimuth

The gain of antenna lobes was limited and cross-beam trajectories were truncated such that only position reports within a certain azimuth range were kept. This selection removed more unwanted data and increased the processing speed for further calculation.

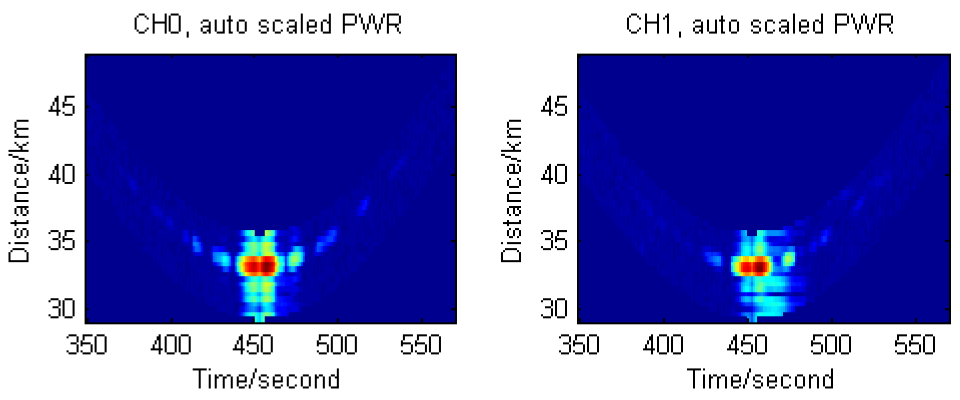


Figure 46 A typical radar image of an aircraft

Figure 46 shows a typical radar image which contains four pairs of visible side lobes. Since the center of fourth side lobe is around ±40° in azimuth plane, the azimuth range was set to ±45° for data selection.

Select Reliable Record

Trajectories which cross the main beam with elevation angle between 7° and 27° were used to characterize the main beam because the main beam of the radar antenna is looking into the north with nominal elevation angle of 17°.

Moreover, multiple ADS-B position reports were needed to calculate a reliable trajectory of the aircraft which requires the number of ADS-B position reports sent by a certain aircraft to be larger than some threshold value. From observation, twelve position reports can usually describe the behavior of the aircraft in the main beam. Therefore, only trajectories that contain no less than twelve position reports were kept.

Trajectories from the South

Since the front and back lobes of radar antennas have similar structure, trajectories from the south of the antenna which fulfill all the above criteria above were selected.

Statistics

Figure 47 shows the selected trajectories which fulfill all the above criteria and the number of points has been reduced dramatically compared with Figure 45. 88 trajectories were selected from the north of the antenna which can be used for front lobe antenna pattern calculation and 307 trajectories were selected from the south of the antenna which can be used for back lobe antenna pattern calculation.

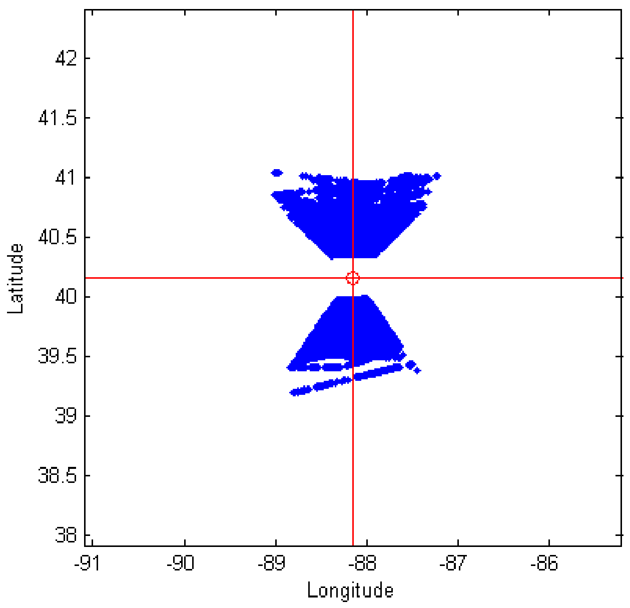


Figure 47 Plotted position reports with selection

Radar Data Extraction

As discussed above, ADS-B frames contain aircraft's information such as coordinate and altitude. The radar images contain aircraft echoes which can provide the distance between aircraft and radar antenna. Therefore, the reflection power of a particular aircraft can be extracted from the radar image and associated with location information provided by the ADS-B frames.

Create Mask by ADS-B Data

Each radar image may contain a large number of radar echoes, but only a few of them are needed for analyzing the radar image. Therefore, useful echoes have to be extracted by applying a mask which blocks all the other unwanted echoes. The mask was generated by ADS-B data since ADS-B data contains positional information of the aircraft and was converted into a time versus distance curve.

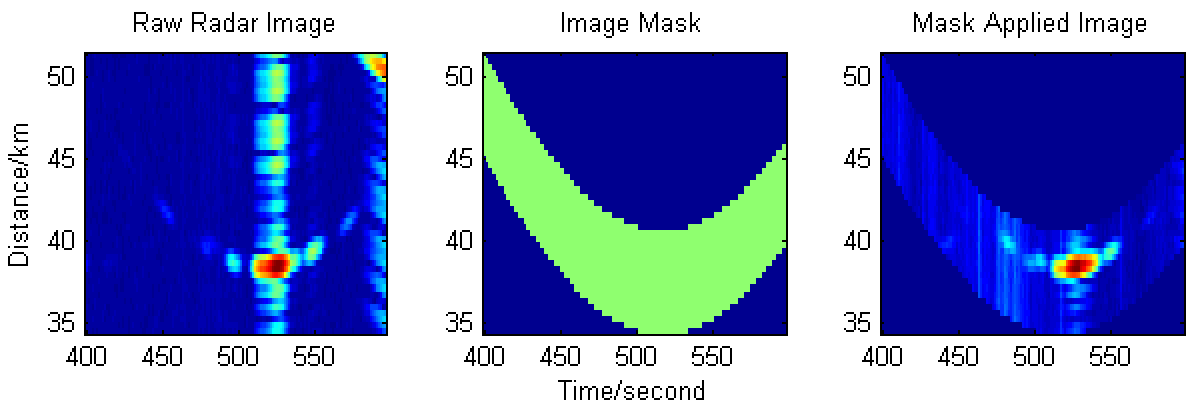


Figure 48 Radar image and mask created by ADS-B data

Figure 48 shows an image mask created by the time versus distance curve from ADS-B position reports with additional ±3 km distance tolerance because the resolution of radar is 750 m and ADS-B data itself may contains some error. When the product of the mask and radar image is formed, only echoes corresponding to the ADS-B data will be kept (Figure 48, mask applied image).

Radar Data Extraction

After the mask was applied to the raw radar image, a new image which only contains the useful echoes was generated. The distance between the aircraft and radar antenna was calculated by finding the maximum reflection power for each second in y-axis and this maximum power is the power detected by the radar antenna. Therefore, aircraft's time versus distance curve and time versus reflection power curve were plotted in Figure 49. Red curve corresponds to the reflection power received by channel 0; blue curve corresponds to the reflection power received by channel 1.

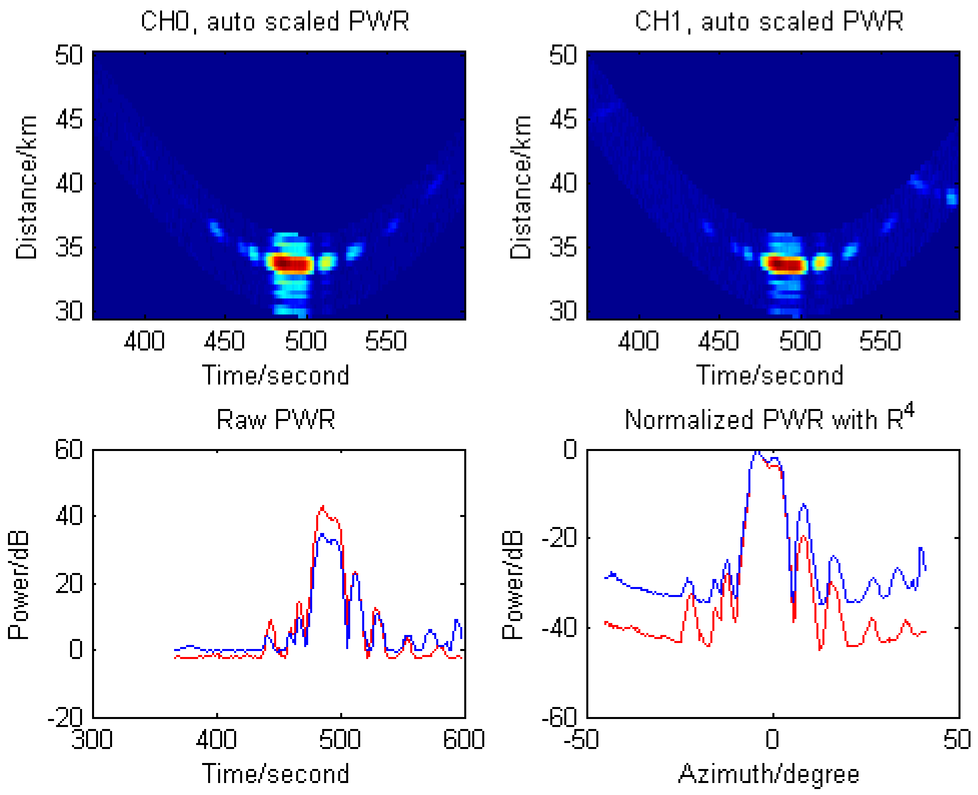


Figure 49 Radar image and reflection power

Evaluation Method

Radar Equation

As given in radar equation, received power $P_r = \frac{P_t G_t G_r \lambda^2 \sigma}{(4\pi)^3 R^4}$ where P_t is transmitting power, G_t is gain of transmit antenna, G_r is gain of receiving antenna, λ is wavelength, σ is radar cross section (RCS) of the target, R is the distance between radar antenna and target [10].

In our situation, received power can be written as $P_r \propto G^2\sigma/R^4$ because the transmitting power and frequency were fixed and transmitter and receiver share the same antenna. Thus the received power was determined by antenna gain, RCS and distance.

To minimize the effect of distance change in the measured reflection power, extracted radar power has been multiplied with R^4 to yield $P_r' \propto G^2\sigma$.

Radar Data with Position Information

The extracted radar data describes the relation between reflection power and time and ADS-B position reports provides aircraft's position information versus time. Therefore the reflection power was mapped to elevation and azimuth angle with respect to of the radar antenna by synthesizing extracted radar data together with ADS-B data.

Radar Data with Antenna Simulation

Extracted radar data was plotted with simulated antenna pattern to compare the influence of RCS. Figure 50 shows a typical radar response for the main (north) lobe with simulated antenna pattern. As can be seen in the figure, the shape of reflection power curve is slightly different from the simulated antenna gain curve due to RCS effect. An offset exists between reflection power and antenna gain versus azimuth curve because of beam tilt.

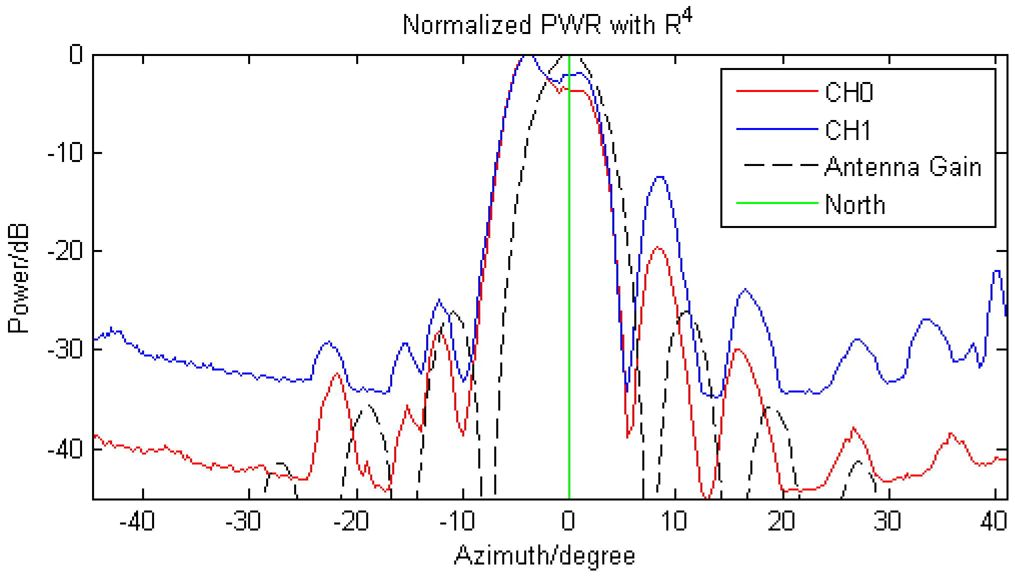


Figure 50 Reflection power vs. azimuth of an aircraft echo

Analyze Result

Beam Tilt

As described above, there is an offset between reflection power and antenna gain versus azimuth curves. From observation, those two curves matched with each other when the correction tilting angle was -2° and this implies the radar antenna arrays were shifted 2° to the west. Figure 51 shows the azimuth-corrected curves and those curves are consistent.

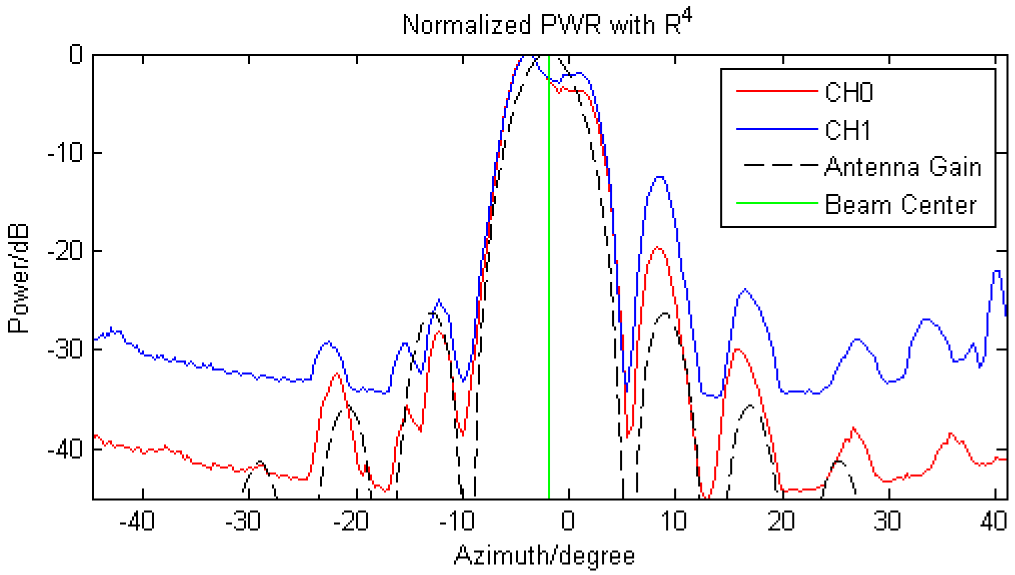


Figure 51 Reflection power vs. azimuth of an aircraft echo with correction

This observation result has been verified by satellite photo from Google Map [11]. Radar antenna in Figure 52 was marked as the red line and the tilting angle calculated from this image was 1.91°.

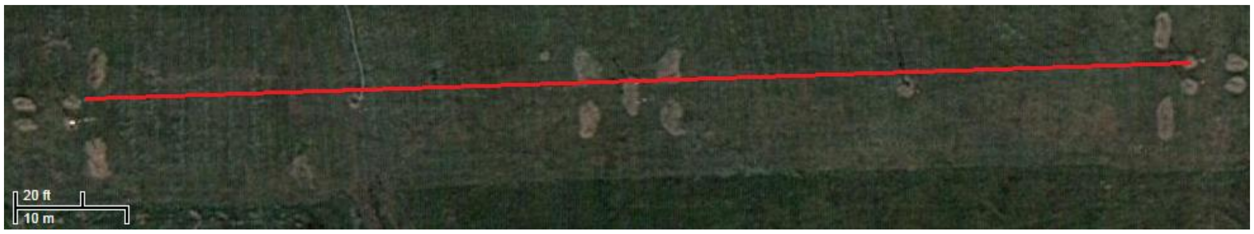


Figure 52 Satellite photo for radar antenna array from Google Map

Gain Calculation

Radar equation $P_r' \propto G^2\sigma$ implies that the two-way gain of antenna can be calculated when target's RCS factor is same, which requires same type of aircraft show up in the same elevation and azimuth angle of the radar antenna. In this particular case, radar equation can be simplified as $P_r' \propto G^2$.

Front Lobe Gain Difference

Since both antenna, channel 0 and 1, are receiving at the same time, the gain difference between those two antennas was calculated by $\Delta gain = \sqrt{\frac{channel\ 0\ power}{channel\ 1\ power}}$. By analyzing all 88 traces in the north, the median of front lobe gain difference between channel 0 and channel 1 ($G_{CH0} - G_{CH1}$) was 3.39 dB

Back Lobe Gain Difference

By analyzing all 307 traces in the south, the median of back lobe gain difference between channel 0 and channel 1 ($G_{CH0} - G_{CH1}$) was -7.96 dB

Front-to-Back Ratio

To calculate the front to back ratio of each antenna, two same type aircraft must show up both in north and south of the antenna with the same elevation and heading angle to form a pair of trajectories. A MATLAB script was used to automatically generate those trajectory pairs and perform front to back ratio calculation. However, the calculated front to back ratio for each pair of trajectories was not a normal distribution in linear scale, thus the overall front to back ratio was sorted by magnitude and evaluated based on median, 20th percent and 80th percent value.

Table 6 Antenna array front to back ratio comparison

	20th percent	Median	80th percent
Antenna Array 0 (CH0)	7.645 dB	10.19 dB	13.15 dB
Antenna Array 1 (CH1)	-3.462 dB	0.547 dB	3.954 dB
Gain Difference	11.11 dB	9.64 dB	9.19 dB

Table 6 shows that channel 0's front to back ratio is always 9.19dB higher than channel 1 which implies channel 0 has a higher performance.

RCS Effect

As shown in Figure 51, antenna gain curve and reflection power curve have the same peak value because they have been normalized. However, the magnitude difference in side lobes of reflection power and antenna gain curve is obvious and this difference was mainly caused by the RCS change when aircraft was moving along its trajectory.

Elevation and Azimuth

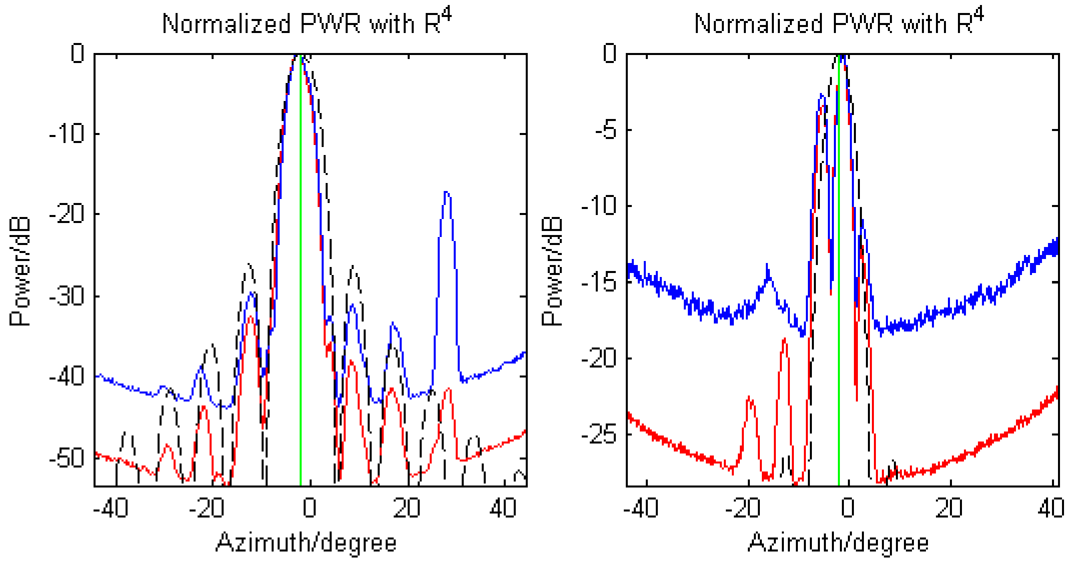


Figure 53 Reflection power vs. azimuth of Boeing 737-800

Figure 53 shows two reflection power curves for trajectories created by the same type of aircraft as it flew through the radar antenna beam with different azimuth and elevation angle.

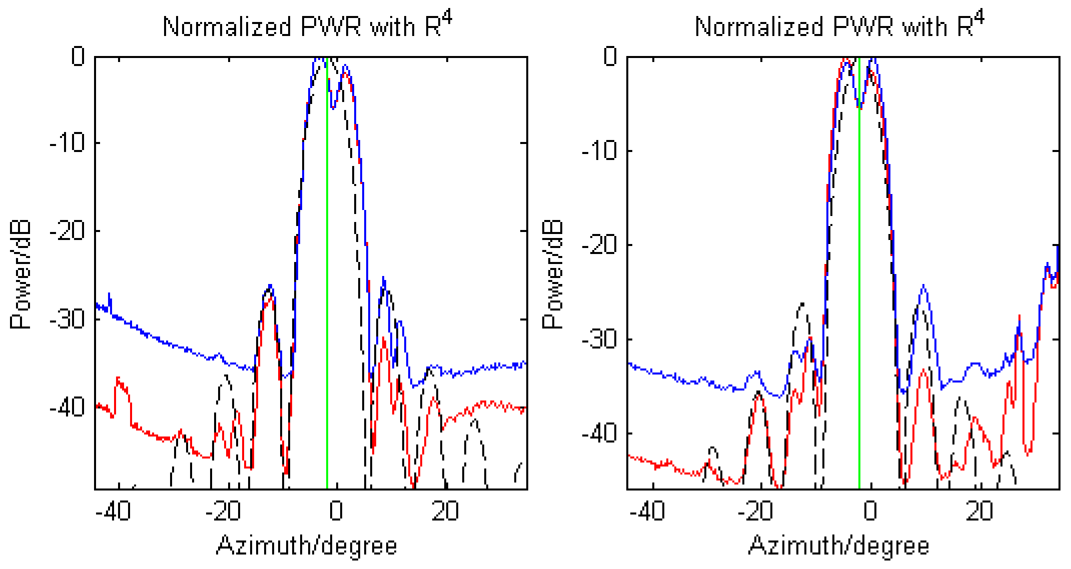


Figure 54 Reflection power vs. azimuth of Airbus 320

Figure 54 shows two reflection power curves for trajectories created by the same type of aircraft that flew through the radar antenna beam with similar azimuth and elevation angle. It is obvious that curves in Figure 54 are more consistent than curves in Figure 53 because the difference caused by the RCS effect is relatively smaller.

Aircraft Type

The type of aircraft is another dominant factor of RCS effect. Since different types of aircrafts have different dimension and shape, the reflection power pattern varies for different types of aircraft even when their azimuth and elevation angle are close.

4. Conclusions and Future Work

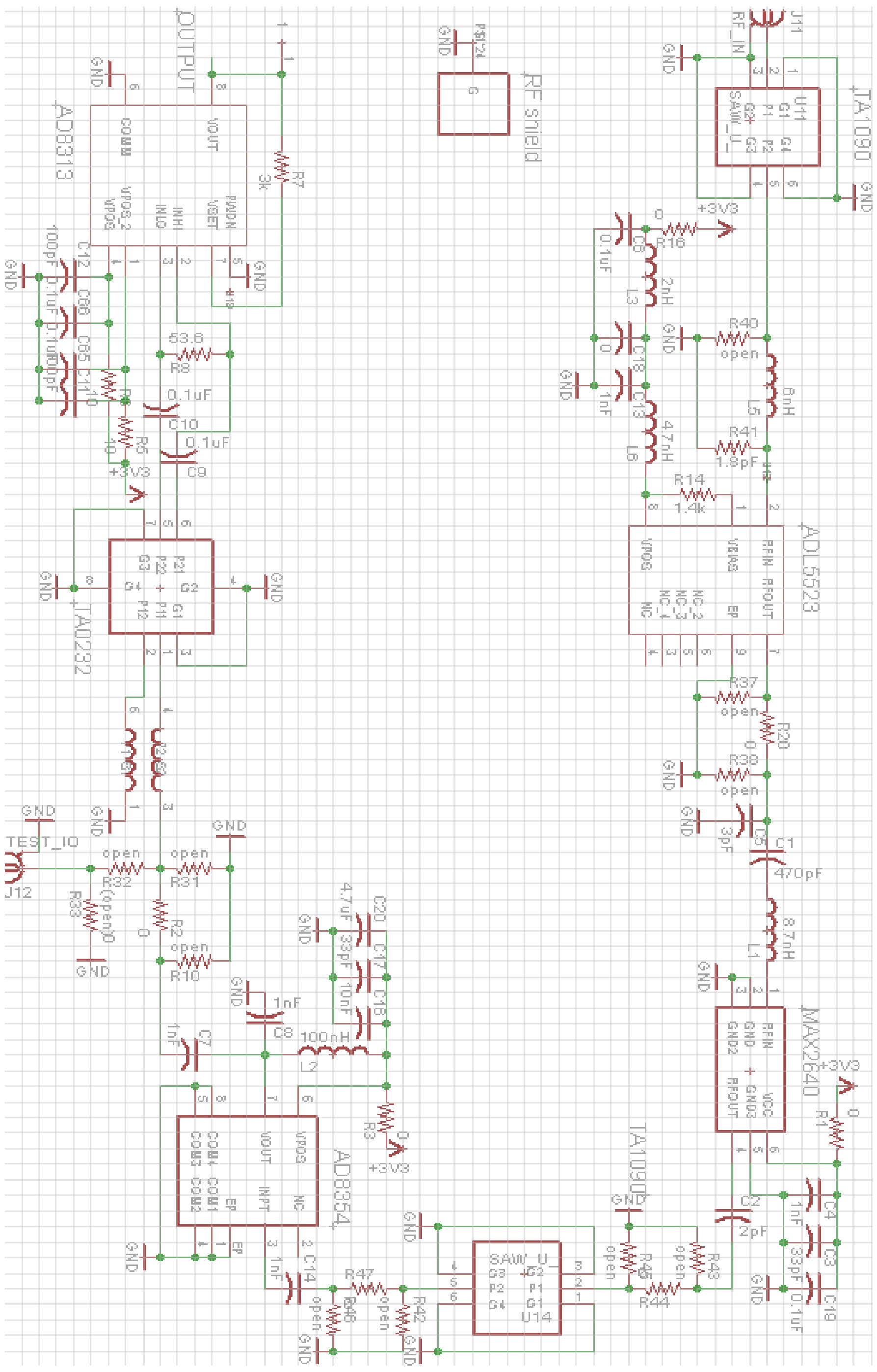
A 1.09 GHz mode-S receiver has been successfully implemented in this project with good performance. Radar antenna analysis was performed with mode-S receiver data and radar data. The analysis shows that the azimuthal beamwidth and sidelobe levels are consistent with expectations based on numerical simulation of the radar antennas. Significant differences were found between the gains and front-to-back ratios of the two radar antenna arrays. The front-to-back ratio shows one radar antenna array was working properly as designed but the phasing cable of another array may be damaged.

The design of RF frontend and firmware of the receiver can be revised in the future to improve performance. Higher accuracy radar antenna pattern analysis can be performed when more data have been collected.

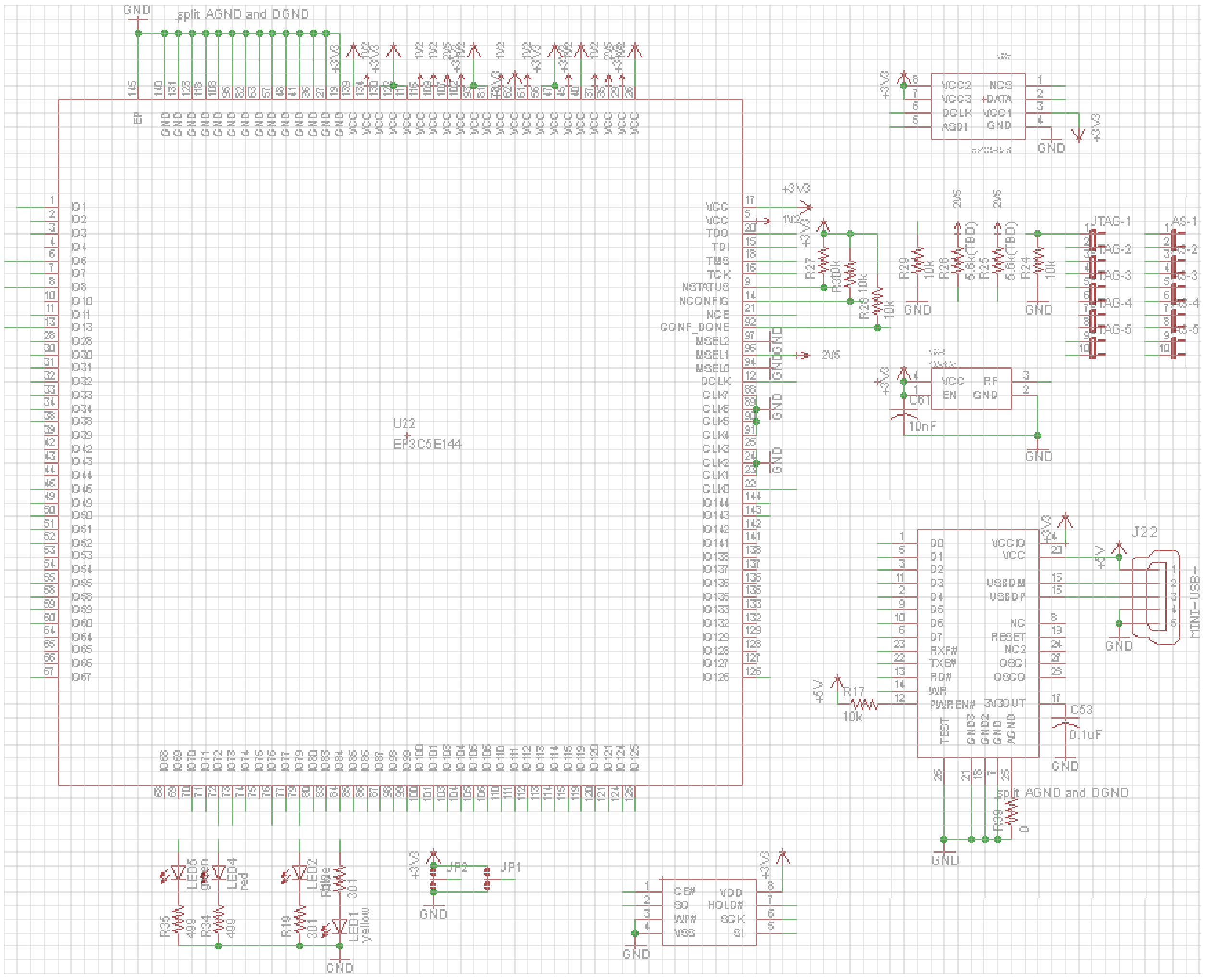
Appendix

Circuit Schematics

RF Frontend



FPGA



ADC (AD9238) and FPGA (EP3C5E144) are connected with invisible connection since direct wiring on this schematic is complicated and Table 7 describes those connections.

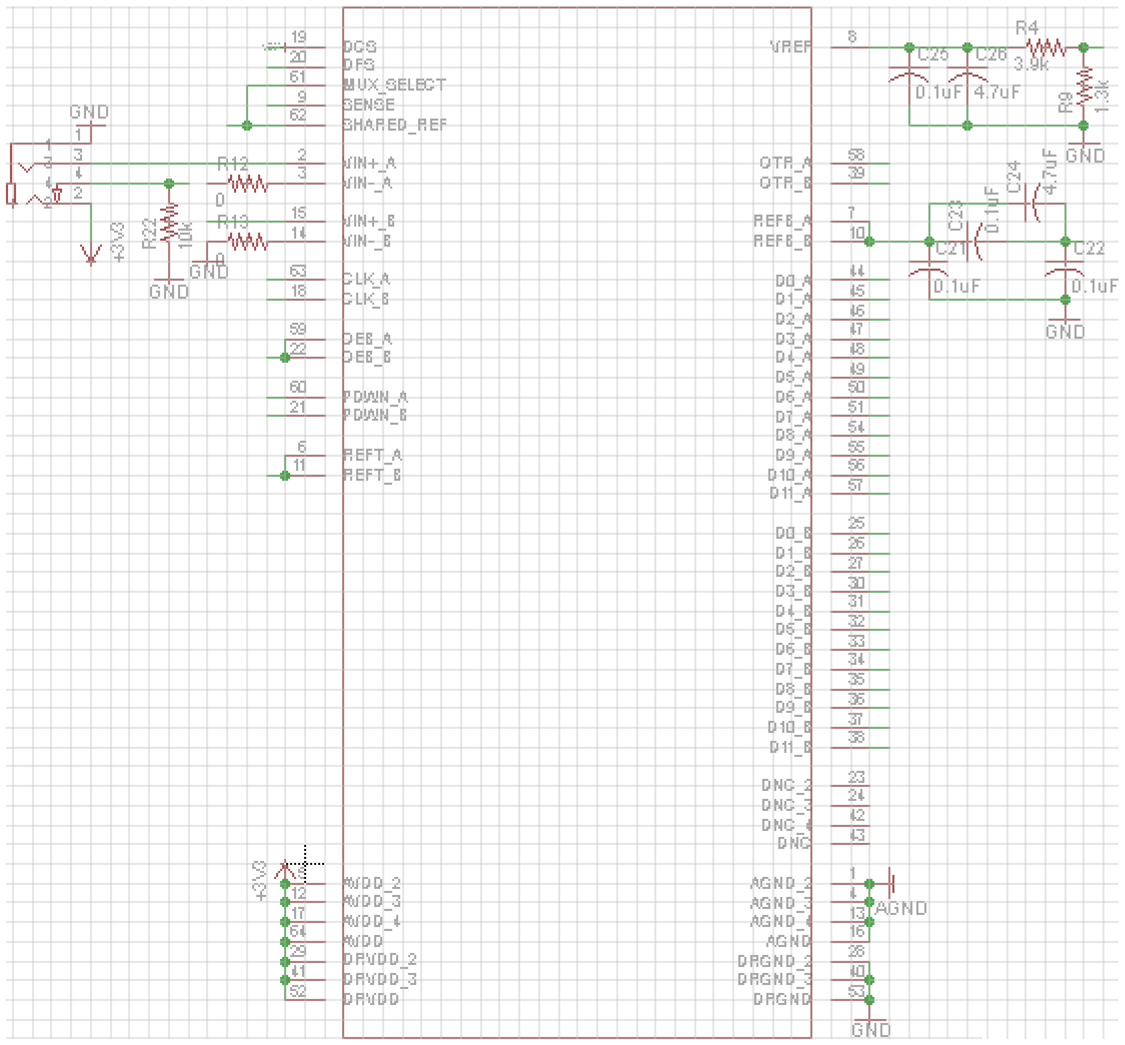
Table 7 FPGA pin assignment

FPGA Pin #	Target Device	Target Pin #	FPGA Pin #	Target Device	Target Pin #	FPGA Pin #	Target Device	Target Pin #
1	Audio Connector	1	34	AD9238	50	87	AD9238	30
2	AD9238	63	38	AD9238	49	92	AS connector	3
3	AD9238	61	46	AD9238	48	98	AD9238	27
3	AD9238	62	49	AD9238	47	99	AD9238	26
6	AS connector	9	50	AD9238	46	100	AD9238	25

6	EPCS4	5	51	AD9238	45	101	AD9238	22
7	AD9238	60	52	AD9238	44	101	AD9238	59
8	AS connector	8	55	AD9238	39	103	AD9238	21
8	EPCS3	1	58	AD9238	38	104	AD9238	20
11	AD9238	58	59	AD9238	37	105	AD9238	19
12	AS connector	1	60	AD9238	36	106	AD9238	18
12	EPCS1	5	67	SST25VF512A	5	110	FT245R	14
13	AS connector	7	70	SST25VF512A	6	111	FT245R	13
13	EPCS2	2	71	SST25VF512A	2	112	FT245R	11
14	AS connector	5	72	LED	0	113	FT245R	10
15	JTAG connector	9	73	LED	1	114	FT245R	9
18	JTAG connector	5	74	SST25VF512A	1	115	FT245R	6
20	JTAG connector	3	76	LED	2	119	FT245R	5
21	AS connector	6	77	SST25VF512A	7	120	FT245R	3
22	Oscillator	3	79	LED	3	121	FT245R	2
28	AD9238	54	80	AD9238	35	124	FT245R	1
30	AD9238	55	83	AD9238	34	125	FT245R	22
31	AD9238	56	84	AD9238	33	126	FT245R	23
32	AD9238	57	85	AD9238	32	143	Jumper	1
33	AD9238	51	86	AD9238	31	144	Jumper	2

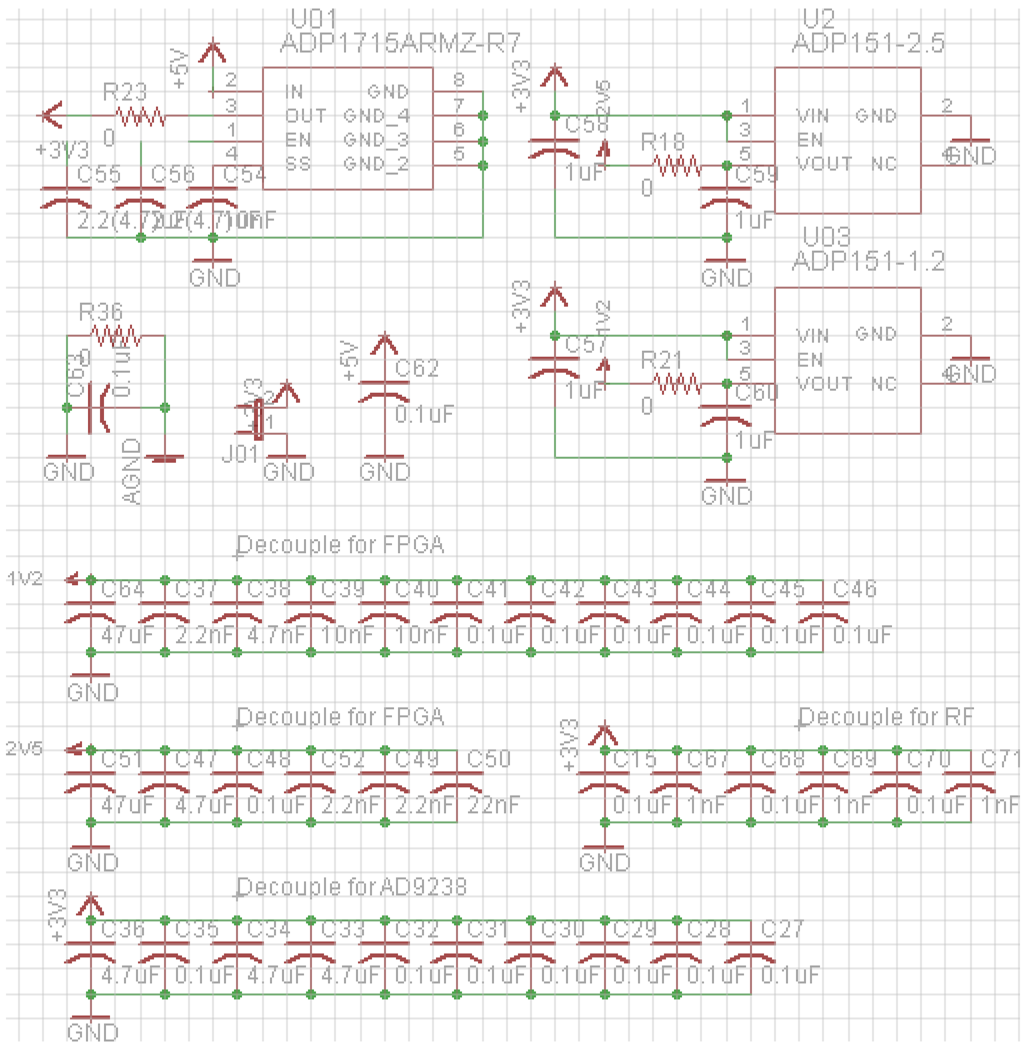
ADC

ADC (AD9238) and FPGA (EP3C5E144) are connected with invisible connection since direct wiring on this schematic is complicated and Table 7 describes those connections.



DCS
DFS
MUX_SELECT
SENSE
SHARED_REF
VIN+_A
VIN-_A
VIN+_B
VIN-_B
CLK_A
CLK_B
OEB_A
OEB_B
PDWN_A
PDWN_B
REFT_A
REFT_B
AVDD_2
AVDD_3
AVDD_4
AVDD
DRVDD_2
DRVDD_3
DRVDD
VREF
OTR_A
OTR_B
REFB_A
REFB_B
D0_A … D11_A
D0_B … D11_B
DNC_2
DNC_3
DNC_4
DNC
AGND_2
AGND_3
AGND_4
AGND
DRGND_2
DRGND_3
DRGND
GND
AGND
+3V3
R12
R13
R22
10k
0
R4
3.9k
R9
1.3k
C25
0.1uF
C26
4.7uF
C21
C22
C23
C24
0.1uF
4.7uF

Power Distribution Network



C++ Code for Data Collection

main.cpp

```
#include <iostream>
#include "rs232.h"
#include "logger.h"

using namespace std;

int main(int argc, char* argv[])
{
	RS232 receiver;
	receiver.port_open();
	receiver.port_config();
	receiver.port_status();
	LOGGER log_file;

	cout << "Reading data from COM port, Ctrl+C to quit."	<< endl;
	log_file.open();
	string frame_tmp;
	for(;;)
	{
		frame_tmp = receiver.frame_get();
		if(frame_tmp.empty() == 1)
		{
			system("pause");
			return(1);
		}
		//cout << frame_tmp << endl;
		log_file.write( frame_tmp.append("\n") );
	}

	receiver.port_close();
	system("pause");
	return(0);
}
```

logger.h

```
#define FILE_SIZE_MAX 5000000

#include <fstream>
#include <iostream>
#include <sstream>

using namespace std;

class LOGGER
{
	public:
		LOGGER(void);
		void open(void);
```

```
            void write(string);
            void close();
        private:
            string name_header;
            ofstream file_stream;
            string file_name;
            unsigned frame_counter;
            int file_counter;
            void file_control(void);
};

LOGGER::LOGGER(void)
{
        frame_counter = 0;
        file_counter = -1;
}

void LOGGER::open(void)
{
        cout << "Please enter the output file name\nNAME>";
        cin >> name_header;
        file_counter = 0;
        file_name.assign(name_header);
        file_name.append("_0.log");
        file_stream.open( file_name.c_str() );
        cout << "Writting to file "    << file_name << endl;
}

void LOGGER::write(string _input)
{
        file_stream << _input;
        frame_counter++;
        if(frame_counter > FILE_SIZE_MAX)
        {
                file_control();
                frame_counter = 0;
        }
}

void LOGGER::file_control(void)
{
        file_stream.close();    // close current file
        file_counter++;
        string _name_suffix;
        _name_suffix.assign("_");
        std::stringstream _name_ss;
        _name_ss << file_counter;
        _name_suffix.append( _name_ss.str() );
        _name_suffix.append(".log");
        file_name.assign(name_header);
        file_name.append(_name_suffix);
        file_stream.open( file_name.c_str() );
        cout << "Writting to file "    << file_name << endl;
}
```

rs232.h

```
#include <iostream>
#include <windows.h>
#include <sys/timeb.h>
#include <sstream>

using namespace std;

class RS232
{
        public:
                int port_open(void);
                int port_config(void);
                void port_status(void);
                string frame_get(void);
                void port_close(void);
        private:
                HANDLE com_port_handle;
                COMMTIMEOUTS com_port_timeout;
                DCB com_port_config;
                string port_buffer;
                string port_time_last, port_time_new;    // the time stame for last read opeartion
                string time_stamp(void);
                int port_read(void);
};
```

```
int RS232::port_open(void)
{
    // get the name of COM port from user
    string _port_number,_port_name="\\\\.\\COM";
    cout << "Please specify serial port: COM";
    cin >> _port_number;
    _port_name.append(_port_number);
    // open COM port
    com_port_handle = CreateFile
    (
        _port_name.c_str(),
        GENERIC_READ|GENERIC_WRITE,
        0,
        NULL,
        OPEN_EXISTING,
        0,
        NULL
    );
    // check status of COM port
    if( com_port_handle == INVALID_HANDLE_VALUE )    // cannot open COM port
    {
        cout << "*** ERROR ***\tCannot open COM port" << endl;
        return(1);
    }
    return(0);
}

int RS232::port_config(void)
{
    // set timeouts
    com_port_timeout = { 1, 100, 1000, 0, 0 };//COMMTIMEOUTS _port_timeout = { 1, 100, 1000, 0, 0 };
    //DCB _port_config;
    if( !SetCommTimeouts(com_port_handle, &com_port_timeout) )
    {
        cout << "*** WARNING ***\tSet Timeout failed" << endl;
    }
    // set DCB
    memset( &com_port_config, 0, sizeof(com_port_config) );
    com_port_config.DCBlength = sizeof(com_port_config);
    com_port_config.BaudRate = 115200;
    com_port_config.fBinary = 1;
```

```
	com_port_config.fDtrControl = DTR_CONTROL_ENABLE;
	com_port_config.fRtsControl = RTS_CONTROL_ENABLE;
	// com_port_config.fOutxCtsFlow = 1;
	// com_port_config.fRtsControl = DTR_CONTROL_HANDSHAKE;
	com_port_config.Parity = NOPARITY;
	com_port_config.StopBits = 1;
	com_port_config.ByteSize = 8;
	if(!SetCommState(com_port_handle,&com_port_config))
	{
		cout << "*** WARNING ***\tSet State failed" << endl;
		return(1);
	}
	return(0);
}

void RS232::port_status(void)
{
	cout << "Please confirm the following settings" << endl;
	cout << "\tBaud Rate: " << com_port_config.BaudRate << endl;
	cout << "\tParity: " << (int)com_port_config.Parity << endl;
	cout << "\tByte Size: " << (int)com_port_config.ByteSize << endl;
	cout << "\tStop Bits: " << (int)com_port_config.StopBits << endl;
}

string RS232::frame_get(void)
{
	// check if there's an entire frame in the buffer
	int _location_start, _location_end, _read_flag, _residue_frag;
	string _frame;
	for(_residue_frag = 0;;)
	{
		_location_start = port_buffer.find("*");	// locate the start of first frame
		_location_end = port_buffer.find(";");	// locate the end of first frame
		if( _location_start == 0 && _location_end > 0 )	// ideal case
		{
			break;
		}
		else if( _location_start > 0 )	// some trash exist in the front
		{
			if(_location_end < 0)	// no end char, we need read more data from port
			{
```

```
                    _residue_frag = 1;
                }
                port_buffer.erase(0, _location_start);    // remove everthing before the frame
                continue;
            }
            else
            {
                port_buffer.clear();
                _read_flag = port_read();    // if the buffer is empty, get some new data from COM port
                if( _read_flag == 0 )
                {
                    cout << "*** ERROR ***\tCannot get new data from COM port"    << endl;
                    _frame.clear();    // return an empty string
                    return(_frame);
                }
            }
        }
        if(_residue_frag == 1)
        {
            _frame.assign( port_time_last );    // add time stamp to the frame
        }
        else
        {
            _frame.assign( port_time_new );    // add time stamp to the frame
        }
        _frame.append( port_buffer, _location_start, _location_end-_location_start+1 );    // copy the first frame to
frame buffer
        port_buffer.erase(0,_location_end+3);    // remove the first frame in the COM buffer
        return(_frame);
}

void RS232::port_close(void)
{
        CloseHandle(com_port_handle);
}

// generate time stamp string for current time
string RS232::time_stamp(void)
{
        timeb _buffer;
        ftime(&_buffer);
```

```
	std::stringstream _buffer_ss;
	_buffer_ss << (long)_buffer.time;
	_buffer_ss << (long)_buffer.millitm;

	std::string _output;
	_output = _buffer_ss.str();
	switch( _output.length() )
	{
		case 13:
		{
			_output.insert( _output.length()-3, "." );
			break;
		}
		case 12:
		{
			_output.insert( _output.length()-2, ".0" );
			break;
		}
		case 11:
		{
			_output.insert( _output.length()-1, ".00" );
			break;
		}
		default: break;
	}
	return(_output);
}

int RS232::port_read(void)
{
	int _read_flag;
	char _buffer_raw[512];
	DWORD _read = 0;
	port_time_last = port_time_new;
	port_time_new = time_stamp();
	_read_flag = ReadFile( com_port_handle, _buffer_raw, sizeof(_buffer_raw), &_read, NULL );	// read is
updated with the number of bytes read
	port_buffer.append( _buffer_raw, 0, sizeof(_buffer_raw) );
	return(_read_flag);
}
```

C++ Code for Mode-S Decoding

main.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include "reader.h"
#include "crc.h"
#include "decoder.h"
#include "writer.h"
#include <time.h>

using namespace std;

int main(int argc, char *argv[])
{
    string str_record, str_time, str_frame;
    unsigned cnt_total, cnt_good, cnt_useful;
    READER data_in;
    WRITER data_out;
    DECODER decode;
    data_in.open();
    data_out.open();
    CRC crc_t;

    for( cnt_total = cnt_good = cnt_useful = 0; ; cnt_total++ )
    {
        str_time = str_frame = "";
        str_record = data_in.get();
        if(str_record.empty() != 0)    // all data have been processed
        {
            break;
        }
        str_time.append(str_record, 0, 14);
        str_frame.append(str_record, 15, str_record.length()-16);
        if( crc_t.check(str_frame) == 0 )
        {
            if( decode.load(str_record) == 0 )
            {
                data_out.write( decode.result().append("\n") );
                cnt_useful++;
            }
            cnt_good++;
        }
    }
    decode.end();
    printf("Total=%d\tGood=%d\tUseful=%d\t%.2f%%.\n", cnt_total, cnt_good, cnt_useful,
(float)cnt_useful/(float)cnt_total*1000 );
    data_in.close();
    data_out.close();
    system("pause");
    return(0);
}
```

reader.h

```
#include <iostream>
#include <fstream>
#include <sstream>

#define FILE_LIST_SIZE 32

using namespace std;

struct FILENAME
{
        string header;
        int index;
};

class READER
{
        public:
                READER(void);
                int open(void);
                int close(void);
                string get(void);
        private:
                FILENAME file_list[FILE_LIST_SIZE];
                string name_header, name;
        ifstream file_stream;
        unsigned group_index, group_index_max, name_index, name_index_max;
        int file_control(void);
};

READER::READER(void)
{
    name_index = -1;
    group_index = 0;
}

int READER::open(void)
{
    int _group_number, _cnt;
    cout << "Please enter the number of file groups>";
    cin >> _group_number;
    group_index_max = _group_number - 1;
    cout << "Please enter the input file header and index of last file, " << FILE_LIST_SIZE << " groups max." << endl;
    cout << "Such as: \"FileHeaderABC 95\"<enter>" << endl;
    for( _cnt = 0; (_cnt < _group_number) && (_cnt < FILE_LIST_SIZE) ; _cnt++ )
    {
        cout << "Group " << _cnt + 1 << ">";
        cin >> file_list[_cnt].header;
        cin >> file_list[_cnt].index;
    }
    return( file_control() );
}

int READER::file_control(void)
{
    if( name_index != -1 || group_index != 0 )    // this is not the very first file
    {
        file_stream.close();
    }
    name_index++;
    if( name_index > file_list[group_index].index )    // swith between file groups
    {
        name_index = 0;
        group_index++;
    }
    if( group_index > group_index_max )    // at the end of list
```

```
        {
                return(1);
        }
        string _suffix = "_";
        stringstream _suffix_ss;
        _suffix_ss << name_index;
        _suffix.append( _suffix_ss.str() );
        _suffix.append(".log");
        name.assign( file_list[group_index].header );
        name.append(_suffix);
        file_stream.open( name.c_str() );
        if( file_stream.good() )
        {
                cout << "Reading file "     << name << endl;
                return(0);
        }
        else return(-1);
}

string READER::get(void)
{
        bool _flag;
        string _buffer;
        _flag = getline(file_stream, _buffer);
        if( _flag != 1 )
        {
                if( file_control() != 0 )
                {
                        _buffer = "";    // return a empty string
                }
                else return( get() );
        }
        return(_buffer);
}

int READER::close(void)
{
        file_stream.close();
}
```

crc.h [8]

```
//#include "common.h"

using namespace std;

class CRC
{
	public:
		CRC(void);
		int check(string);
	private:
		int length;
		unsigned poly, crc_result, pi_raw;
		unsigned frame_char[14];
		void str2char(string);	// convert hex string into char which formed by two hex numbers
		unsigned crc(void);	// calculate CRC
		unsigned str2num(string);
};

CRC::CRC(void)
{
	poly = 0xFFFA0480;
}

int CRC::check(string _frame)
{
	length = _frame.length();
	if( (length != 14) && (length != 28) ) return(-1);	// abort if format is bad
	str2char(_frame);
	crc_result = crc();
	string _frame_pi;
	pi_raw = str2num( _frame_pi.append( _frame, length-6, 6 ) );
	return( crc_result - pi_raw );
}

void CRC::str2char(string _frame)	// e.g. input="2A30", output={ 0x2A, 0x30 }; only support large case letter
{
	int _cnt, _frame_int[28];
	for( _cnt = 0; _cnt < length; _cnt++ )	// convert hex number string into int array
	{
```

```
            _frame_int[_cnt] = _frame[_cnt] - 48;
            if( _frame_int[_cnt] > 15 )
            {
                _frame_int[_cnt] = _frame_int[_cnt] - 7;    // small case use 39, large case use 7
            }
            /*if( _frame_int[_cnt] > 15 )
            {
                _frame_int[_cnt] = _frame_int[_cnt] - 32;
            }*/
        }
        for( _cnt = 0; _cnt < length/2; _cnt++ )    // convert int array into char array
        {
            frame_char[_cnt] = _frame_int[_cnt*2+1] + _frame_int[_cnt*2]*16;
        }
    }

    unsigned CRC::crc(void)    // for 112-bit only
    {
        unsigned data, data1, data2;
        data = frame_char[0]<<24 | frame_char[1]<<16 | frame_char[2]<<8 | frame_char[3];
        data1 = frame_char[4]<<24 | frame_char[5]<<16 | frame_char[6]<<8 | frame_char[7];
        data2 = frame_char[8]<<24 | frame_char[9]<<16 | frame_char[10]<<8;
        unsigned cnt;
        if(length == 28)
        {
            for( cnt = 0; cnt < 88; cnt++ )
            {
                if( (data & 0x80000000) != 0 ) data = data ^ poly;
                data = data << 1;
                if( (data1 & 0x80000000) != 0 ) data = data | 1;
                data1 = data1 << 1;
                if( (data2 & 0x80000000) != 0 )    data1 = data1 | 1;
                data2 = data2 << 1;
            }
        }
        else
        {
            for( cnt = 0; cnt < 32; cnt++ )
            {
                if( (data & 0x80000000) != 0 ) data = data ^ poly;
                data = data << 1;
            }
        }
        return(data>>8);
    }

    unsigned CRC::str2num(string _str)    // e.g. input="2A30", output=0x2A30
    {
        int _cnt;
        unsigned _length, _frame_num[28], _num;
        _num = 0;
        _length = _str.length();
        if( _length > 8 ) return(-1);    // too long to process
        for( _cnt = 0; _cnt < _length; _cnt++ )    // convert hex number string
        {
            _frame_num[_cnt] = _str[_cnt] - 48;
            if( _frame_num[_cnt] > 10 )
            {
                _frame_num[_cnt] = _frame_num[_cnt] - 7;    // small case use 39, large case use 7
            }
            _num += ( _frame_num[_cnt] << 4*( _length - _cnt - 1 ) );
        }
        return(_num);
    }
```

decoder.h [9]

```
#include <iostream>
#include <sstream>
#include <time.h>
#include <math.h>
#include <iomanip>    // adjust output format
#include "database.h"
#include "counter.h"

#define PI 3.14159265358979323846264338327950
#define CPR_NZ 15.0

using namespace std;

class DECODER
{
    public:
        DECODER(void);
        int load(string);
        string result(void);
        int end(void);
    private:
        string raw_stamp, raw_frame, result_str;
        int length;
        int type_main, type_sub;
        unsigned str2num(string);
        string num2str(unsigned);
        // variable and functions for date and time
        string utc_msec_str, utc_str;    // we are actually using local time instead of UTC
        unsigned time_stamp_sec;    // unix time stamp for current frame
        int utc_year, utc_month, utc_day, utc_hour, utc_min, utc_sec, utc_msec;
        int time_stamp(void);
        string format_number(int);
        // decoding function
        DATABASE db_cpr;
        string df17(void);
        double cpr_nl(double);
        double cpr_max(double, double);
        // statistic and log
        COUNTER frame_counter;
};

DECODER::DECODER(void)
{
    length = -1;
    raw_stamp = raw_frame = "";
}

int DECODER::load(string _record)
{
    string _buffer;
    // reset all parameters before start
    utc_year = utc_month = utc_day = utc_hour = utc_min = utc_sec = utc_msec = -1;
    type_main = type_sub = -1;
    result_str = "";
    // set parameters for later process
    raw_stamp = _record.substr(0, 14);
```

```
        raw_frame = _record.substr(15, _record.length()-16 );
        length = raw_frame.length();
        time_stamp();
        type_main = str2num( raw_frame.substr(0,2) ) >> 3;
        // decode frame
        frame_counter.add(type_main);
        if(type_main!=17) return(-1);    // only consider DF17 for this development phase
        // format result
        _buffer = df17();
        if( _buffer.empty() == 0 )
        {
                result_str = utc_str;
                result_str.append( _buffer );
                return(0);
        }
        //cout << utc_str << raw_frame << endl;
}

string DECODER::result(void)
{
        return( result_str );
}

unsigned DECODER::str2num(string _str)    // e.g. input="2A30", output=0x2A30
{
        int _cnt;
        unsigned _length, _frame_num[8], _num;
        _num = 0;
        _length = _str.length();
        if( _length > 8 ) return(-1);    // to long to process
        for( _cnt = 0; _cnt < _length; _cnt++ )    // convert hex number string
        {
                _frame_num[_cnt] = _str[_cnt] - 48;
                if( _frame_num[_cnt] > 10 )
                {
                        _frame_num[_cnt] = _frame_num[_cnt] - 7;    // small case use 39, large case use 7
                }
                _num += ( _frame_num[_cnt] << 4*( _length - _cnt - 1 ) );
        }
        return(_num);
}
```

```
string DECODER::num2str(unsigned _in)    // convert input number into 6-digit hex string
{
    int _cnt;
    char _number;
    stringstream _ss;
    for( _cnt = 5; _cnt >= 0; _cnt-- )
    {
        _number = ( ( _in & ( 0xf << (_cnt*4) ) ) >> (_cnt*4) );
        if( _number > 9 )
        {
            _number += 7;    // display as large case
        }
        _number += 48;
        _ss << _number;
    }
    return( _ss.str() );
}

// we are actually using local time instead of UTC
int DECODER::time_stamp(void)    // calculate date and time for the current frame
{
    // get number value of time stamp
    stringstream _buffer_s, _buffer_ms;
    _buffer_s << raw_stamp.substr(0,10);
    _buffer_s >> time_stamp_sec;
    utc_msec_str = raw_stamp.substr(11,3);    // we need this string format msec later when write file
    _buffer_ms << utc_msec_str;
    _buffer_ms >> utc_msec;
    // convert time stamp into date and time
    struct tm * _time_struct;
    time_t _stamp = time_stamp_sec;
    _time_struct = localtime( &_stamp );
    // set current time
    utc_year = _time_struct->tm_year + 1900;
    utc_month = _time_struct->tm_mon + 1;
    utc_day = _time_struct->tm_mday;
    utc_hour = _time_struct->tm_hour;
    utc_min = _time_struct->tm_min;
    utc_sec = _time_struct->tm_sec;
    // generate output date/time string, e.g. 20130801,07:27:03.093,
    stringstream _utc_ss;
    _utc_ss << utc_year << format_number(utc_month) << format_number(utc_day) << ",";
    _utc_ss << format_number(utc_hour) << ":" << format_number(utc_min) << ":" << format_number(utc_sec)
<< "." << utc_msec_str <<",";
    utc_str = _utc_ss.str();
}

string DECODER::format_number(int _number)
{
    stringstream _buffer_ss;
    string _buffer;
    _buffer_ss << _number;
    if( _number < 10 )
    {
        _buffer = "0";
        _buffer.append( _buffer_ss.str() );
    }
    else
    {
        _buffer = _buffer_ss.str();
    }
    return(_buffer);
}

string DECODER::df17(void)
{
    unsigned _icao, _format, _altitude;
    unsigned _lat_cpr_0, _lat_cpr_1, _lon_cpr_0, _lon_cpr_1;    // 0=even, 1=odd
    string _adsb, _adsb_db, _log;
    stringstream _buffer_ss;
    _icao = str2num( raw_frame.substr(2,6) );
    _adsb = raw_frame.substr(8,14);
    type_sub = str2num( _adsb.substr(0,2) ) >> 3;
    if( type_sub >= 9 && type_sub <= 18 )    // only consider airbone position message at this development phase
    {
        _format = ( str2num( _adsb.substr(5,1) ) & 0x4 ) >> 2;    // get ADS-B format number
        // deode altitude data
        if( ( str2num( _adsb.substr(3,1) ) & 0x1 ) == 1 )
        {
            _altitude = ( ( ( str2num( _adsb.substr(2,2) ) >> 1 ) << 4 ) + str2num( _adsb.substr(4,1) ) ) * 25 - 1000;
        }
```

```
else
{
        frame_counter.add(-2);
        cout << "ALT CODE Q=0" << endl;
}
// search a different CPR format frame in database to form a pair for global decode
_adsb_db = db_cpr.query(time_stamp_sec, utc_msec, _icao, _format, _adsb);
if( _adsb_db.length() != 14 )    // we got nothing from database
{
        return("");
}
// extract CPR formatted lat and lon information
if( _format == 0 )    // current frame is even
{
        _lat_cpr_0 = ( str2num( _adsb.substr(5,5) ) & 0x3fffe ) >> 1;
        _lon_cpr_0 = str2num( _adsb.substr(9,5) ) & 0x1ffff;
        _lat_cpr_1 = ( str2num( _adsb_db.substr(5,5) ) & 0x3fffe ) >> 1;
        _lon_cpr_1 = str2num( _adsb_db.substr(9,5) ) & 0x1ffff;
}
else
{
        _lat_cpr_1 = ( str2num( _adsb.substr(5,5) ) & 0x3fffe ) >> 1;
        _lon_cpr_1 = str2num( _adsb.substr(9,5) ) & 0x1ffff;
        _lat_cpr_0 = ( str2num( _adsb_db.substr(5,5) ) & 0x3fffe ) >> 1;
        _lon_cpr_0 = str2num( _adsb_db.substr(9,5) ) & 0x1ffff;
}
// start decode CPR format data
double yz0 = (double) _lat_cpr_0;
double yz1 = (double) _lat_cpr_1;
double xz0 = (double) _lon_cpr_0;
double xz1 = (double) _lon_cpr_1;
double dlat0 = 360.0/( 4.0*CPR_NZ );
double dlat1 = 360.0/( 4.0*CPR_NZ-1.0 );
double j = floor( (59.0*yz0-60.0*yz1)/pow(2.0,17.0) + 0.5 );
double rlat0 = dlat0*( fmod(j,60.0)+yz0/pow(2.0,17.0) );
double rlat1 = dlat1*( fmod(j,59.0)+yz1/pow(2.0,17.0) );
if( rlat0 > 270.0 )
{
        rlat0 = rlat0 - 360.0;
        rlat1 = rlat1 - 360.0;
}
```

```
        if( cpr_nl(rlat0) != cpr_nl(rlat1) )
        {
            frame_counter.add(-1);
            cout << "CPR decode ERROR" << endl;
        }
        double n0 = cpr_max( cpr_nl(rlat0), 1.0);
        double n1 = cpr_max( cpr_nl(rlat1)-1.0, 1.0);
        double dlon0 = 360.0/n0;
        double dlon1 = 360.0/n1;
        double m = floor( ( xz0*(cpr_nl(rlat0)-1) - xz1*cpr_nl(rlat1) )/pow(2.0,17.0) + 0.5 );
        double rlon0 = dlon0*( fmod(m,n0) + xz0/pow(2.0,17.0) );
        double rlon1 = dlon1*( fmod(m,n1) + xz1/pow(2.0,17.0) );
        if( rlon0 > 270.0 )
        {
            rlon0 = rlon0 - 360.0;
            rlon1 = rlon1 - 360.0;
        }
        // generate result string
        _buffer_ss << num2str( _icao ) << ",";
        _buffer_ss << fixed;
        if( _format ==0 )    // select the LAT and LON of the current frame instead of the one from database
        {
            _buffer_ss << setprecision(5) << rlat0 << "," << setprecision(5) << rlon0 << "," << _altitude;
        }
        else
        {
            _buffer_ss << setprecision(5) << rlat1 << "," << setprecision(5) << rlon1 << "," << _altitude;
            //printf("ICAO=%x\tLAT=%f\tLON=%f\tALT=%d\n",_icao,rlat1,rlon1,_altitude);
        }
        return( _buffer_ss.str() );
    }
    return("");
}

double DECODER::cpr_nl(double _lat)
{
    if( _lat < -90.0 || _lat > 90.0 )
    {
        return(-1.0);
    }
    return( floor( 2.0*PI / acos( 1.0 - (1.0-cos(PI/2.0/CPR_NZ))/pow(cos(PI/180.0*fabs(_lat)),2.0) ) ) );
```

```
}

double DECODER::cpr_max(double _i0, double _i1)
{
	if( _i0 > _i1 )
	{
		return(_i0);
	}
	return(_i1);
}

int DECODER::end(void)
{
	frame_counter.print();
}
```

database.h

```
#define MAX_TIME_DIFFERENCE 15    // ICAO requires less than 10 seconds for a 1000 knot aircraft
#define MAX_ITEM_NUMBER 512    // shold be twice as the total max number of aircraft in range

using namespace std;

struct DATABASE_ITEM
{
	unsigned time_sec, time_msec, icao, format;
	string adsb;
};

class DATABASE
{
	public:
		DATABASE(void);
		string query(unsigned, unsigned, unsigned, unsigned, string);
		void status(void);
	private:
		int end_pointer, end_pointer_max;    // pointer of the last record
		DATABASE_ITEM item[MAX_ITEM_NUMBER];
		unsigned format_not(unsigned);
		int search(unsigned, unsigned);
		int remove(int);
```

```
        int update(unsigned, unsigned);
};

DATABASE::DATABASE(void)
{
    int _cnt;
    end_pointer = -1;
    end_pointer_max = -1;
    for(    _cnt = 0; _cnt < MAX_ITEM_NUMBER; _cnt++ )
    {
        item[_cnt].time_sec = -1;
        item[_cnt].time_msec = -1;
        item[_cnt].icao = -1;
        item[_cnt].format = -1;
        item[_cnt].adsb = "";
    }
}

string DATABASE::query(unsigned _sec, unsigned _msec, unsigned _icao, unsigned _format, string _adsb)
{
    int _adr_diff, _adr_same;
    update( _sec, _msec );
    // find an old frame which has same format and update it
    _adr_same = search(_icao, _format);
    if( _adr_same > -1 )
    {
        item[_adr_same].time_sec = _sec;
        item[_adr_same].time_msec = _msec;
        item[_adr_same].icao = _icao;
        item[_adr_same].format = _format;
        item[_adr_same].adsb = _adsb;
    }
    else    // if this is the first one in this format, save it
    {
        if( end_pointer < MAX_ITEM_NUMBER-1 )    // if list is full, just rewrite the last record
        {
            end_pointer++;
        }
        item[end_pointer].time_sec = _sec;
        item[end_pointer].time_msec = _msec;
        item[end_pointer].icao = _icao;
```

```
            item[end_pointer].format = _format;
            item[end_pointer].adsb = _adsb;
        }
        // find a frame which has different format, e.g. _format=odd, find a even frame
        _adr_diff = search(_icao, format_not(_format) );
        if( _adr_diff > -1 )    // if founded, return it and update record
        {
            return( item[_adr_diff].adsb );
        }
        return("NULL");
}

void DATABASE::status(void)
{
        if( end_pointer > MAX_ITEM_NUMBER*0.9 )
        {
            cout << "\t*** WARNING ***\tDatabase is almost full." << endl;
        }
        if( end_pointer > end_pointer_max )
        {
            end_pointer_max = end_pointer;
        }
        cout << "\t*** USAGE ***\tnow=" << end_pointer+1 << "\tmax=" << end_pointer_max+1 << endl;
}

unsigned DATABASE::format_not(unsigned _in)
{
        if( _in == 0 ) return(1);
        if( _in == 1 ) return(0);
        return(_in);
}

int DATABASE::search(unsigned _icao, unsigned _format)
{
        int _cnt;
        for( _cnt = 0; _cnt <= end_pointer; _cnt++ )
        {
            if( item[_cnt].icao == _icao && item[_cnt].format == _format )
            {
                return(_cnt);
            }
        }
        return(-1);
}

int DATABASE::remove(int _adr)
{
        int _cnt;
        if( _adr < 0 || _adr > end_pointer || _adr >= MAX_ITEM_NUMBER )
        {
                return(-1);
        }
        for( _cnt = _adr; _cnt <= end_pointer; _cnt++ )
        {
                item[_cnt] = item[_cnt+1];
        }
        end_pointer--;
        item[_cnt+1].time_sec = -1;
        item[_cnt+1].time_msec = -1;
        item[_cnt+1].icao = -1;
        item[_cnt+1].format = -1;
        item[_cnt+1].adsb = "";
        return(0);
}

int DATABASE::update(unsigned _sec, unsigned _msec)
{
        int _cnt;
        for( _cnt = 0; _cnt < end_pointer; _cnt++ )
        {
                if( _sec - item[_cnt].time_sec > MAX_TIME_DIFFERENCE +1 )
                {
                        remove(_cnt);
                        _cnt--;
                }
        }
        return(0);
}
```

counter.h

```
#include <iostream>
```

```
#include <iomanip>    // adjust output format

using namespace std;

class COUNTER
{
        public:
                COUNTER(void);
                int add(int);
                void print(void);
                int reset(void);
        private:
                int type[32],error[32];
                int total;
};

COUNTER::COUNTER(void)
{
        for( int _cnt=0; _cnt<32; _cnt++ )
        {
                type[_cnt] = 0;
        }
        for( int _cnt=0; _cnt<32; _cnt++ )
        {
                error[_cnt] = 0;
        }
        total = 0;
}

int COUNTER::add(int _item)
{
        if( _item > -1 && _item < 33 )
        {
                type[_item]++;
                total++;
                return(0);
        }
        else if( _item > -33 && _item < 0 )
        {
                error[ abs(_item) - 1 ]++;
                total++;
                return(0);
        }
        else return(1);
}

void COUNTER::print(void)
{
        int _cnt, _sum = 0;
        float _percent;
        cout << endl << endl << "*** STATISTICS ***" << endl;
        cout << "* ERRORS COUNT"    << endl;
        cout << "CPR_ERROR=" << error[0] << "\tALT_Q_0=" << error[1] << endl;
        cout << endl << "* DF TYPES COUNT"    << endl;
        cout << "type\t" << "count\t" << "percent" << endl;
        for( _cnt=0; _cnt<32; _cnt++ )
        {
                _percent = (float)type[_cnt]/(float)total*100;
                cout << fixed;
                cout << _cnt << "\t" << type[_cnt] << "\t" << setprecision(2) << _percent << endl;
        }
        cout << endl;
}
```

writer.h

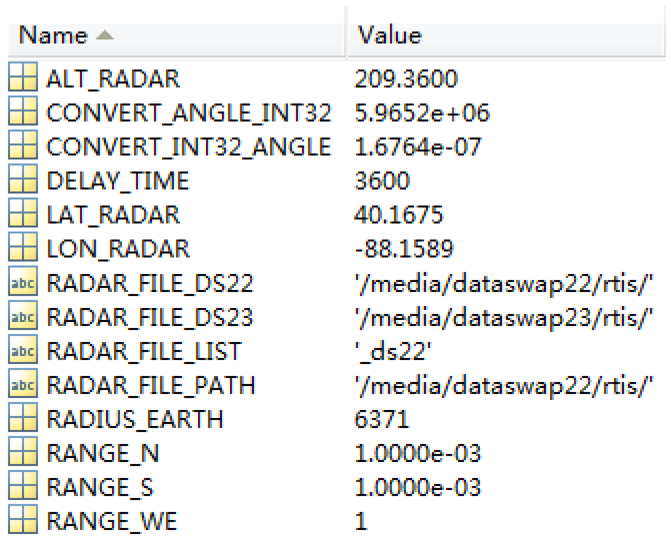
```
#define FILE_SIZE_MAX 500000 //5000000

#include <fstream>
#include <iostream>
#include <sstream>

using namespace std;

class WRITER
{
    public:
        WRITER(void);
        void open(void);
        void open(int);
        void write(string);
        void close();
    private:
        int flag_mode;
        string name_header;
        ofstream file_stream;
        string file_name;
        unsigned frame_counter;
        int file_counter;
        int file_control(void);
};

WRITER::WRITER(void)
{
    frame_counter = 0;
    file_counter = -1;
    flag_mode = 0;
}

void WRITER::open(void)
{
    cout << "Please enter the output file name\nNAME>";
    cin >> name_header;
    file_counter = 0;
    file_name.assign(name_header);
    file_name.append("_0.log");
    file_stream.open( file_name.c_str() );
    cout << "Writting to file "    << file_name << endl;
}

void WRITER::open(int _mode)
{
    flag_mode = _mode;    // in this mode (mode=1), nothing will print on the screen
    cin >> name_header;
    file_counter = 0;
    file_name.assign(name_header);
    file_name.append(".log");
    file_stream.open( file_name.c_str() );
}

void WRITER::write(string _input)
{
    file_stream << _input;
```

```
        frame_counter++;
        if( ( frame_counter > FILE_SIZE_MAX ) && ( flag_mode == 0 ) )
        {
                file_control();
                frame_counter = 0;
        }
}

int WRITER::file_control(void)
{
        file_stream.close();    // close current file
        if( flag_mode == 1 )    // in log mode, only save to one file
        {
                return(1);
        }
        file_counter++;
        string _name_suffix;
        _name_suffix.assign("_");
        std::stringstream _name_ss;
        _name_ss << file_counter;
        _name_suffix.append( _name_ss.str() );
        _name_suffix.append(".log");
        file_name.assign(name_header);
        file_name.append(_name_suffix);
        file_stream.open( file_name.c_str() );
        cout << "Writting to file "    << file_name << endl;
}

void WRITER::close(void)
{
        file_stream.close();
}
```

MATLAB Code for Data Analyze

Receiver and Radar Data Processing

Receiver_const.mat



Name	Value
ALT_RADAR	209.3600
CONVERT_ANGLE_INT32	5.9652e+06
CONVERT_INT32_ANGLE	1.6764e-07
DELAY_TIME	3600
LAT_RADAR	40.1675
LON_RADAR	-88.1589
RADAR_FILE_DS22	'/media/dataswap22/rtis/'
RADAR_FILE_DS23	'/media/dataswap23/rtis/'
RADAR_FILE_LIST	'_ds22'
RADAR_FILE_PATH	'/media/dataswap22/rtis/'
RADIUS_EARTH	6371
RANGE_N	1.0000e-03
RANGE_S	1.0000e-03
RANGE_WE	1

receiver_file_dz.m

```
%% Read log file generated by PlanePlotter
% output array fomatfor each row in the array
% data_date_int, data_time_normal, data_icao_int, data_lat_int, data_lon_int, data_alt, -1,
data_time_msecond_int, -1, -1, -1
% -1 are reserver position for storing future calculation result

function [data_output, length, file_name] = receiver_file_dz()

load receiver_const;

file_name = input('Please specify log file> ', 's');
file_id = fopen(file_name, 'r');
if(file_id == -1)
        fprintf('\n*** ERROR ***\nCannot open file.\n\n');
        clear LAT_RADAR LON_RADAR ALT_RADAR RANGE_S RANGE_WE file_name file_id;
        data_output = -1;
        return;
end;
```

```
fprintf('\nReading file...\n');
% loop for read data from log file
cnt_frame = 0; length = 0;
for cnt = 1:1e7,    % change range
        if(mod(cnt, 1e4) == 0)
                fprintf('\tReading line %d0, 000\n', cnt/1e4);
        end;
        str_0 = fgetl(file_id);
        if(str_0 == -1)
                break;
        end;
        if(isempty(str_0))
                continue;
        else
                str_data = textscan(str_0, '%s %s %s %f %f %d', 'delimiter', ', ');
                cnt_frame = cnt_frame+1;
                % get useful data from string
                length = length+1;
                data_date = char( str_data{1,1} ); data_date_int = int32( str2double(data_date) );
                data_time = char( str_data{1,2} );
                data_icao = char( str_data{1,3} );data_icao_int = int32( hex2dec(data_icao) );
                data_lat = ( str_data{1,4} ); data_lat_int = int32( data_lat*CONVERT_ANGLE_INT32 );    % normalized to
90 degrees
                data_lon = ( str_data{1,5} ); data_lon_int = int32( data_lon*CONVERT_ANGLE_INT32 );
                data_alt = ( str_data{1,6} );
                if( data_lon<-93 || data_lon>-83 || data_lat>44 || data_lat<36 || data_alt>60000)    % drop bad data
                        continue;
                end;
                data_time_hour = str2double( data_time(1, 1:2) );
                data_time_minute = str2double( data_time(1, 4:5) );
                data_time_second = str2double( data_time(1, 7:8) );
                data_time_msecond = str2double( data_time(1, 10:12) );
                data_time_msecond_int = int32( data_time_msecond );
                % convert time to "time stamp"
                data_time_normal = data_time_hour*3600+data_time_minute*60+data_time_second;
                data_output(length, :) = ...
                        [data_date_int, data_time_normal, data_icao_int, data_lat_int, data_lon_int, data_alt,
data_time_msecond_int];    % remove reserve spot for speed in 201302021359
        end;
end;
```

```
fclose(file_id);
% show statistics
fprintf('\tNumber of record is %d, number of ADS-B record is %d\n', cnt_frame, length);
```

receiver_revise_beam.m

```
%% Only keep records in the defined (beam) area
function [data_output,length_output]=receiver_revise_beam(data_raw, mode)

switch nargin;
    case 1
        mode = 0;
    case 2
    otherwise
        fprintf('*** ERROR *** Wrong input @ receiver_revise_beam()\n');
        return;
end;

fprintf('Removing out of beam record...\n');
load receiver_const;
size_input=size(data_raw); length_input=size_input(1,1);

% only keep frames in the beam (assume beam is a rectangle area)
lat_range_s = int32(abs((LAT_RADAR+RANGE_S)*CONVERT_ANGLE_INT32));
lat_range_n = int32(abs((LAT_RADAR-RANGE_N)*CONVERT_ANGLE_INT32));
lon_range_w = int32((LON_RADAR-RANGE_WE)*CONVERT_ANGLE_INT32);
lon_range_e = int32((LON_RADAR+RANGE_WE)*CONVERT_ANGLE_INT32);
length_output = 0;
if( mode == 0 )    % working on north side of radar
    for cnt = 2 : length_input,
        if( data_raw(cnt,4) > lat_range_s && data_raw(cnt,5) < lon_range_e && data_raw(cnt,5) > lon_range_w)
            length_output=length_output+1;
            data_output(length_output,:)=data_raw(cnt,:);
        else
            continue;
        end;
    end;
else    % working on south side of radar
    for cnt = 2 : length_input,
        if( data_raw(cnt,4) < lat_range_n && data_raw(cnt,5) < lon_range_e && data_raw(cnt,5) > lon_range_w)
            length_output=length_output+1;
```

```
                data_output(length_output,:)=data_raw(cnt,:);
            else
                continue;
            end;
        end;
end;
fprintf('\tNumber of in-beam record is %d, %2.2f%% of total
record.\n',length_output,length_output/length_input*100);
```

receiver_revise_cross.m

```
%% Find record which cross the radar beam
function [data_output,length_output] = receiver_revise_cross(data_raw, mode)

switch nargin;
    case 1
        mode = 0;
    case 2
    otherwise
        fprintf('*** ERROR *** Wrong input @ receiver_revise_cross()\n');
        return;
end;

fprintf('Removing record which does not cross the beam...\n');
load receiver_const;
size_input=size(data_raw); length_input=size_input(1,1);

length_output=0;
lon_radar_int=int32((LON_RADAR)*CONVERT_ANGLE_INT32);
cnt=1;
for cntf=1:length_input,
    flag_break=0;
    for pnt=cnt:length_input,
        % find end of this aircraft's record, trigger by: different aircraft, same aircraft travel back, end of file
        % if we observe the same aircraft again after DELAY_TIME, we treat it as a new trace
        if(data_raw(pnt,3)~=data_raw(cnt,3) || (data_raw(pnt,2)-data_raw(cnt,2))>DELAY_TIME ||
pnt==length_input)
            % this record will cross the beam
            if((data_raw(cnt,5)>lon_radar_int && data_raw(pnt-1,5)<lon_radar_int) ||...
                    (data_raw(cnt,5)<lon_radar_int && data_raw(pnt-1,5)>lon_radar_int))
                data_output((length_output+1):(length_output+pnt-cnt),:)=data_raw(cnt:(pnt-1),:);
                length_output=length_output+pnt-cnt;
            end;
            cnt=pnt-1;    % next cycle focus on another aircraft
            if(pnt==length_input)    % we are at the end of file
                flag_break=1;
            end;
            break;
        end;
    end;
    if(flag_break==1)
        break;
    end;
    if(cnt==length_input)
        break;
    end;
    cnt=cnt+1;
end;

if(exist('data_output','var'))
    fprintf('\tNumber of cross-beam record is %d, %2.2f%% of total
record.\n',length_output,length_output/length_input*100);
else
    fprintf('\n*** CAUTION ***\nNo cross-beam record.\n');
    data_output=[];
end;
```

receiver_speed.m

```
%% Calculate the speed of aircraft
function [data_output,length_output] = receiver_speed(data_raw)

load receiver_const;
size_input = size(data_raw); length_input = size_input(1,1);

length_output = length_input;
data_output = data_raw;

cnt = 1;
for cntf = 1:length_input,
    flag_break = 0; distance_total = 0; time_total = 0;
    for pnt = cnt:(length_input-1),
        % find end of this aircraft's record, trigger by: different aircraft, same aircraft travel back, end of file
        % if we observe the same aircraft again after DELAY_TIME, we treat it as a new trace
        if(data_raw(pnt+1,3) ~= data_raw(cnt,3) || (data_raw(pnt+1,2)-data_raw(cnt,2)) > DELAY_TIME || pnt+1
== length_input)
            % calculate the average speed
            speed_aircraft = distance_total / time_total * 3600;    % unit of speed is km/hour
            data_output(cnt:pnt,8) = int32(speed_aircraft);    % changed from 7 to 8 in 201302021358
            % calculat heading
            heading_aircraft = azimuth( double(data_raw(cnt,4))*CONVERT_INT32_ANGLE,
double(data_raw(cnt,5))*CONVERT_INT32_ANGLE,...
                double(data_raw(pnt,4))*CONVERT_INT32_ANGLE,
double(data_raw(pnt,5))*CONVERT_INT32_ANGLE );
            data_output(cnt:pnt,9) = int32( heading_aircraft * CONVERT_ANGLE_INT32 );
            %fprintf('*** GS=%fknot\tHDG=%f\trange=%d~%d\n', speed_aircraft/1.8, heading_aircraft, cnt,
pnt);
            cnt = pnt;    % next cycle focus on another aircraft
            if(pnt == length_input)    % we are at the end of file
                flag_break = 1;
            end;
            break;
        end;
        % record information for this record
        % calculate and store speed for this aircraft
        lat_data_0 = double(data_raw(pnt,4)) * CONVERT_INT32_ANGLE;
        lon_data_0 = double(data_raw(pnt,5)) * CONVERT_INT32_ANGLE;
        lat_data_1 = double(data_raw(pnt+1,4)) * CONVERT_INT32_ANGLE;
        lon_data_1 = double(data_raw(pnt+1,5)) * CONVERT_INT32_ANGLE;
        angle = distance('gc',[lat_data_0,lon_data_0],[lat_data_1,lon_data_1]);
        distance_current = ( (angle/360*2*pi*RADIUS_EARTH).^2 + ( ( double(data_raw(pnt+1,6)) -
double(data_raw(pnt,6)) ) * .3048 / 1000 ).^2 ) .^ .5;
        time_current = double( data_raw(pnt+1,2) - data_raw(pnt,2) );
        distance_total = distance_total + distance_current;
        time_total = time_total + time_current;
        %fprintf('*** lat0 = %f\tlon0 = %f\tlat1 = %f\tlon1 = %f\n',lat_data_0,lon_data_0,lat_data_1,lon_data_1);
        %fprintf('*** d = %fkm\tt = %ds\tD = %fkm\tT
= %ds\n',distance_current,time_current,distance_total,time_total);
    end;
    if( flag_break == 1 || cnt == length_input )
        break;
    end;
    cnt = cnt+1;
```

```
end;
```

receiver_beam_time.m

```
%% Calculate the time (and LON, LAT, ALT) which each aircraft cross the beam
% Input data format: [DATE, TIME_STAMP, ICAO_ID, LAT, LON, ALT]
function [data_output,length_output]=receiver_beam_time(data_raw)

fprintf('Calculating cross-beam time stamp...\n');
load receiver_const;
size_input=size(data_raw); length_input=size_input(1,1);
lon_radar_int=int32(LON_RADAR*CONVERT_ANGLE_INT32);

%
length_output=0;
for cnt=2:length_input,
    data_lon_0=double(data_raw(cnt-1,5))*CONVERT_INT32_ANGLE;
data_lon_1=double(data_raw(cnt,5))*CONVERT_INT32_ANGLE;
    if( (data_raw(cnt-1,3)==data_raw(cnt,3)) &&...
            ( ( (data_lon_0>LON_RADAR) && (data_lon_1<LON_RADAR) ) ||...
            ( (data_lon_0<LON_RADAR) && (data_lon_1>LON_RADAR) ) ) )
        % raw data from input
        data_time_0=double(data_raw(cnt-1,2)); data_time_1=double(data_raw(cnt,2));
        data_lat_0=double(data_raw(cnt-1,4))*CONVERT_INT32_ANGLE;
data_lat_1=double(data_raw(cnt,4))*CONVERT_INT32_ANGLE;
        data_alt_0=double(data_raw(cnt-1,6)); data_alt_1=double(data_raw(cnt,6));
        devide_factor=abs(LON_RADAR-data_lon_0)/abs(data_lon_1-data_lon_0);
        % calculate the time between point A, B, C by using t_ab=AB*T/AC (assume constant speed)
        time_stamp=devide_factor*(data_time_1-data_time_0)+data_time_0;
        time_stamp_int=int32(time_stamp);
        % calculate latitude and height when crosee the beam*CONVERT_INT32_ANGLE
        data_lat=devide_factor*(data_lat_1-data_lat_0)+data_lat_0;
        data_lat_int=int32(data_lat*CONVERT_ANGLE_INT32);
        data_alt=devide_factor*(data_alt_1-data_alt_0)+data_alt_0;
        data_alt_int=int32(data_alt);
        % calculate speed when cross the beam
%          lat_data_0=double(data_raw(cnt-1,3))*CONVERT_INT32_ANGLE;
lon_data_0=double(data_raw(cnt-1,4))*CONVERT_INT32_ANGLE;
%          lat_data_1=double(data_raw(cnt,3))*CONVERT_INT32_ANGLE;
lon_data_1=double(data_raw(cnt,4))*CONVERT_INT32_ANGLE;
%          angle=distance('gc',[lat_data_0,lon_data_0],[lat_data_1,lon_data_1]);
```

```
%
distance_aircraft=((angle/360*2*pi*RADIUS_EARTH).^2+((double(data_raw(cnt-1,6))-double(data_raw(cnt,6)))*.30
48/1000).^2).^.5;
%                   speed_aircraft=distance_aircraft/(double(data_raw(cnt,2))-double(data_raw(cnt-1,2)))*1000;
                data_spd_int=-1;%data_raw(cnt,7);%int32(speed_aircraft);
                length_output=length_output+1;
                data_output(length_output,:) = [ data_raw(cnt,1), time_stamp_int, data_raw(cnt,3), data_lat_int,
lon_radar_int, data_alt_int,...
                        data_raw(cnt,7), data_spd_int, data_raw(cnt,9), data_raw(cnt,10),
double(data_raw(cnt,11))*CONVERT_INT32_ANGLE*1e3, data_raw(cnt,12) ];
        else
                continue;
        end;
end;
```

receiver_time_correct.m

```
%% plot trace for all the cross-beam contacts
% parameter: LON_RADAR,LAT_RADAR,RANGE_WE,length_input,data_raw
function data_correct = receiver_time_correct(data_raw)

switch nargin;
        case 1;
        otherwise;
                fprintf('*** ERROR *** Invalid input @ receiver_time_correct()\n');
                return;
end;
size_input = size(data_raw);

% this table is extracted from server's system log
% format: sync_date, sync_time, time_error
% 20130116 43164.68223 -3.35362E-05
% 20121231 26227.2039 -3.25977E-05
% 20121225 39273.12768 -2.42223E-05
% 20121221 50457.37942

data_correct = data_raw;
for cnt = 1 : size_input(1,1);
        % get date and time for current record
        date_current = double( data_raw(cnt,1) );
        time_current = double( data_raw(cnt,2) ) + double( data_raw(cnt,7) ) / 1000;
```

```
    % 24 * 3600 = 86400
    if( date_current > 20121231 )
        duration = (date_current - 20130100) * 86400 + time_current - 26227.2039;
        error_correct = duration * 3.35362E-05;
    elseif( date_current == 20121231 && time_current > 26227.2039 )
        duration = time_current - 26227.2039;
        error_correct = duration * 3.35362E-05;
    elseif( date_current > 20121225 || ( date_current == 20121225 && time_current > 39273.12768 ) )
        duration = (date_current - 20121225) * 86400 + time_current - 39273.12768;
        error_correct = duration * 3.25977E-05;
    elseif( date_current > 20121221 || ( date_current == 20121221 && time_current > 50457.37942 ) )
        duration = (date_current - 20121221) * 86400 + time_current - 50457.37942;
        error_correct = duration * 2.42223E-05;
    else
        fprintf('*** WARNING *** Wrong record time @ receiver_time_correct()\n');
        continue;
    end;
    time_current = time_current + error_correct;
    if( time_current > 86400 )    % each day should only have 86400 seconds
        time_current = time_current - 86400;
        date_current = date_current + 1;
    end;
    time_int = floor(time_current);
    time_frc = (time_current - time_int) * 1000;
    data_correct(cnt,1) = int32(date_current);
    data_correct(cnt,2) = int32(time_int);
    date_correct(cnt,7) = int32(time_frc);
end;
```

receiver_record_select.m

```
%% Select and sort the display record's sequence in rr_analyse()
function [ data_select, time_select ] = receiver_record_select(data_raw, time_raw, mode)

load receiver_const;
% process input data
switch nargin;
    %case 2
    case 3
    otherwise;
        fprintf('*** ERROR *** Invalid input parameter @ receiver_record_select()\n');
        return;
end;
%mode_h = floor( mode/10 );
%mode = mod(mode,10);

time_select = [];
switch mode
    % the time information from input data haven't been corrected before
    case 0
        fprintf('*** INFO *** Mode %d: process with raw data\n', mode);
        data_select = data_raw;
        time_select = time_raw;
    case 1
        fprintf('Please select the parameter which use for sort, options are...\n');
        fprintf('1:date, 2:time, 3:ICAO, 4:latitude, 6:alitude, 8:speed, 9:heading,\n10: distance, 11:elevation,
17:aircraft_length\n');
        sort_column = input('parameter>');
        time_select = sortrows(time_raw, sort_column);
        data_select = data_raw;
    case 2
        fprintf('Please select the parameter which use for sort, options are...\n');
        fprintf('1:date, 2:time, 3:ICAO, 4:latitude, 6:alitude, 8:speed, 9:heading,\n10: distance, 11:elevation,
17:aircraft_length\n');
        sort_column = input('1st parameter>');
        time_select = sortrows(time_raw, sort_column);
        sort_column = input('2nd parameter>');
        time_select = sortrows(time_select, sort_column);
        data_select = data_raw;
    case 4   % only keep records which fullfill certain requirment
        time_select = receiver_lib_select(time_raw);
        data_select = data_raw;
    case 5   % two criterias
        time_select = receiver_lib_select(time_raw);
        time_select = receiver_lib_select(time_select);
        data_select = data_raw;
    case 6   % three criterias
        time_select = receiver_lib_select(time_raw);
        time_select = receiver_lib_select(time_select);
        time_select = receiver_lib_select(time_select);
        data_select = data_raw;
    case 99
```

```
            fprintf('*** INFO *** Mode %d: correct time error before process\n', mode);
            data_select = receiver_time_correct(data_raw);
            time_select = receiver_time_correct(time_raw);
        otherwise    % correct time error in data files
            fprintf('*** ERROR *** Mode select error @ receiver_record_select()\n');
            return;
end;


%% internal function for data selection
function data_select = receiver_lib_select(data_raw)

load receiver_const;
data_select = [];
fprintf('Please select the parameter which use for select, options are...\n');
fprintf('1:date, 2:time, 3:ICAO, 4:latitude, 6:alitude, 8:speed, 9:heading,\n10: distance, 11:elevation,
17:aircraft_length, 99:aircraft_type\n');
column = input('parameter number>');
if( column < 50 )    % input is number
        fprintf('min for parameter %d', column);
        select_min = input('>');
        fprintf('max for parameter %d', column);
        select_max = input('>');
else
        select_str = input('string>', 's');
end;

% format user input, e.g. convert degree into INT32 format
switch column
        case 4    % latitude
            select_min = double( select_min ) * CONVERT_ANGLE_INT32;
            select_max = double( select_max ) * CONVERT_ANGLE_INT32;
        case 6    % altitude
            select_min = double( select_min ) / 0.3048 * 1000;
            select_max = double( select_max ) / 0.3048 * 1000;
        case 9    % heading
            select_min = double( select_min ) * CONVERT_ANGLE_INT32;
            select_max = double( select_max ) * CONVERT_ANGLE_INT32;
        case 11    % elevation
            select_min = double( select_min ) * 1000;
            select_max = double( select_max ) * 1000;
```

```
end;

% select by numerical range
if( column < 50 )
        for cnt = 1 : lib_length(data_raw)
                if( double( data_raw(cnt,column) ) >= select_min && double( data_raw(cnt,column) ) <= select_max )
                        data_select = [ data_select; data_raw(cnt,:) ];
                end;
        end;
        if( isempty(data_select) )
                fprintf('*** WARNING *** no record fullfill current criteria @ receiver_record_select() |
receiver_lib_select() \n');
                return;
        end;
        data_select = sortrows(data_select,column);
        % select by string match
else
        for cnt = 1 : lib_length(data_raw)
                [~, buf_type] = receiver_aircraft_query( data_raw(cnt,3), 1);
                if( buf_type == select_str )
                        data_select = [ data_select; data_raw(cnt,:) ];
                end;
        end;
        if( isempty(data_select) )
                fprintf('*** WARNING *** no record fullfill current criteria @ receiver_record_select() |
receiver_lib_select() \n');
                return;
        end;
end;
```

Antenna Analyze - rr_analyse.m

```
% Compare the trace from radar and ADS-B data
function [good_data, good_time, good_table, save_data, save_time, save_table] = rr_analyse(data_file_name,
mode)

% load and define constants
load receiver_const;
MASK_TIME_MAX = 60*5;
MASK_DISTANCE_MAX = 3;
HEADING_RANGE = 15;
```

```
BEAM_SHIFT_AZI = 2;
% over some default settings when running this code on flash drive
RADAR_FILE_DS22 = 'radar/';
RADAR_FILE_DS23 = 'radar/';
file_name = '';

% process input data
switch nargin;
        case 1
                mode = 1;
        case 2
        otherwise
                frpintf('*** ERROR *** Wrong input parameter @ rr_analyse()\n');
                return;
end;
mode_h = floor( mode/10 );
mode = mod(mode,10);

load(data_file_name);
if( mode < 5 )    % 0~4 is processing traces on the south of the radar
        direction = 0;    % looking into north
        AZIMUTH_RANGE = 45;
else    % 5~9 is processing traces on the south of the radar
        mode = mode - 5;
        direction = 180;    % looking into south
        AZIMUTH_RANGE = 35;
        data_5(:,12) = lib_azimuth_convert( data_5(:,12) );
end;
[data_raw, time_raw] = receiver_record_select(data_5, time_3, mode_h);

size_time = size(time_raw); length_time = size_time(1, 1);
size_data = size(data_raw); length_data = size_data(1, 1);

% create output array to keep good record so we don't need return this code
good_data = []; good_time = []; good_table = []; cnt_good = 0;
save_data = []; save_time = []; save_table = []; cnt_save = 0;

% calculate elevation angle for all records
[distance_radar, elevation_angle] = lib_distance( time_raw(:,4), time_raw(:,5), time_raw(:,6) );
% print header for table
fprintf('\n#\tDate&Time\tElevation\tDistance\tALT\tHDG\tSPD\tConsistency\tGain\n');
```

```
% get corresponding radar file information for contect
[radar_file_name, radar_file_time] = radar_file(time_raw, 2);

% process each contact point
pointer_next =    - 1;
for cnt = 1 : length_time,
        data_current = [];
        if( pointer_next ~=    - 1 )    % we are moving to a designate record which requested by user
                cnt = pointer_next;
                pointer_next =    - 1;
        end;
        if( radar_file_time(cnt) ==    - 1 )    % skip record which radar data doesn't exist
                continue;
        end;

        for cntl = 1 : length_data,
                % extract data for current contact record from ADS-B database
                if( time_raw(cnt,1) == data_raw(cntl,1) && time_raw(cnt,3) == data_raw(cntl,3) &&...
                                ( abs(time_raw(cnt,2) - data_raw(cntl,2) ) < MASK_TIME_MAX ) && ...
                                abs( double( data_raw(cntl,12) ) * CONVERT_INT32_ANGLE - direction ) < AZIMUTH_RANGE
&&...
                                data_raw(cntl,2) > radar_file_time(cnt) && data_raw(cntl,2) < ( radar_file_time(cnt) + 600 ) )
                        data_current = [ data_current; data_raw(cntl,:) ];
                end;
        end;% record should contain at least one sidelobe
        size_current = size(data_current);
        if( size_current(1,1) < 10 )      % only analyse data which has at least 10 points
                continue;
        end;
        % only keep records which contain at least one sidelobe
        % skip record which outside the main lope
        if( max( double( data_current(:,12) ) * CONVERT_INT32_ANGLE ) - direction < 15 ||...
                min( double( data_current(:,12) ) * CONVERT_INT32_ANGLE ) - direction > -15 ||...
                elevation_angle(cnt,1) < 7 || elevation_angle(cnt,1) > 27 )
                continue;
        end;
        % get heading for the aircraft and only analyse heading E or W aircraft
        heading_angle = double( data_current(1,9) ) * CONVERT_INT32_ANGLE;
        if( ( heading_angle > (90 - HEADING_RANGE) && heading_angle < (90 + HEADING_RANGE) ) ||...
                        ( heading_angle > (270 - HEADING_RANGE) && heading_angle < (270 + HEADING_RANGE) ) )
        else
```

```
            continue;
        end;

        % get distance vs time from previous calculation
        distance_receiver_current = double( data_current(:,10) ) / 1000;
        % select correct radar path to read radar data
        if( time_raw(cnt,1) > 20130102 || ( time_raw(cnt,1) == 20130102 && time_raw(cnt,2) > 64760 ) )
            radar_file_path = RADAR_FILE_DS23;    % start from 20130102 64770
        else
            radar_file_path = RADAR_FILE_DS22;
        end;
        load( strcat(radar_file_path, radar_file_name(cnt, :) ) );

        % generate a mask which act as a filter only keep radar data for current contact
        COMPARE_DISTANCE_step = round( MASK_DISTANCE_MAX / ( hts(2, 1) - hts(1, 1) ) );
        radar_mask = zeros(size(rti0));

        % use ploy fit to get a approximate time vs distance curve
        current_time_double = double( data_current(:,2) ) + double( data_current(:,7) ) / 1000;
        current_time_ployfit = double( min(data_current(:,2)) : max(data_current(:,2)) );
        warning('off');
        distance_receiver_polyfit = polyval( polyfit(current_time_double, distance_receiver_current, 4),
    current_time_ployfit );
        distance_receiver_polyfit_step = round(distance_receiver_polyfit/( hts(2, 1) - hts(1, 1) ) );
        % use poly fit to get a approximate time elevation/azimuth curve
        elevation_double = double( data_current(:,11) ) * CONVERT_INT32_ANGLE;
        elevation_polyfit = polyval( polyfit(current_time_double, elevation_double, 4), current_time_ployfit );
        azimuth_double = double( data_current(:,12) ) * CONVERT_INT32_ANGLE;
        azimuth_polyfit = polyval( polyfit(current_time_double, azimuth_double, 4), current_time_ployfit );
        warning('on');
        current_time_min = min(current_time_ployfit);
        current_time_max = max(current_time_ployfit);

        % creating mask
        continue_flag_in = 0;
        for cnt_time = current_time_min : current_time_max,
            relative_time = cnt_time - radar_file_time(1, cnt);
            if( relative_time < 1 || relative_time > 600 )    % skip this record since some thing going wrong
                continue_flag_in = 1;
                break;
            end;
            pointer = cnt_time - min(current_time_ployfit)+1;
            pointer_distance_step = distance_receiver_polyfit_step(1, pointer);
            radar_mask(relative_time,...
                (pointer_distance_step - COMPARE_DISTANCE_step) :
(pointer_distance_step+COMPARE_DISTANCE_step) ) = 1;
        end;
        if( continue_flag_in == 1 )
            continue;
        end;
        % select radar data for this contact by applying mask
        data_ch0 = (rti0 .* radar_mask)';
        data_ch1 = (rti1 .* radar_mask)';
        % extract useful radar data
        distance_polyfit_step_min = min(distance_receiver_polyfit_step) - COMPARE_DISTANCE_step;
        distance_polyfit_step_max = max(distance_receiver_polyfit_step) + COMPARE_DISTANCE_step;
        relative_time_min = current_time_min - radar_file_time(1, cnt);
        relative_time_max = current_time_max - radar_file_time(1, cnt);
        data_radar_0 = data_ch0( distance_polyfit_step_min : distance_polyfit_step_max, relative_time_min :
relative_time_max );
        data_radar_1 = data_ch1( distance_polyfit_step_min : distance_polyfit_step_max, relative_time_min :
relative_time_max );
        secs_radar = secs(relative_time_min:relative_time_max, 1);
        hts_radar = hts(distance_polyfit_step_min:distance_polyfit_step_max, 1);

        % get reflected signal strength and distance
        [radar_pwr_raw_0, radar_distance_pointer_0] = max(data_radar_0, [], 1);
        radar_distance_pointer_0 = radar_distance_pointer_0 - 1;
        radar_distance_0 = radar_distance_pointer_0*.375+(distance_polyfit_step_min - 1)*.375;
        radar_pwr_normal_0 = radar_pwr_raw_0/max(radar_pwr_raw_0);
        radar_pwr_normal_distance_0 = radar_pwr_raw_0.*distance_receiver_polyfit.^4;    % normalize with
distance^4
        radar_pwr_normal_distance_0 = radar_pwr_normal_distance_0/max(abs(radar_pwr_normal_distance_0));    %
normalize with max value
        [radar_pwr_raw_1, radar_distance_pointer_1] = max(data_radar_1, [], 1);
        radar_distance_pointer_1 = radar_distance_pointer_1 - 1;
        radar_distance_1 = radar_distance_pointer_1*.375+(distance_polyfit_step_min - 1)*.375;
        radar_pwr_normal_1 = radar_pwr_raw_1/max(radar_pwr_raw_1);
        radar_pwr_normal_distance_1 = radar_pwr_raw_1.*distance_receiver_polyfit.^4;    % normalize with
distance
        radar_pwr_normal_distance_1 = radar_pwr_normal_distance_1/max(abs(radar_pwr_normal_distance_1));    %
normalize with max value
```

```
        % evaluate the consistency and gain of two channels
        power_min = min( min(radar_pwr_normal_0), min(radar_pwr_normal_1) );
        power_max = max( max(radar_pwr_normal_0), max(radar_pwr_normal_1) );
        power_range = power_max - power_min;
        power_compare = abs( radar_pwr_normal_0 - radar_pwr_normal_1 ) / power_range;
        consistency = sum(power_compare);
        for cntp = 2 : length(azimuth_polyfit),
                if( ( azimuth_polyfit(cntp-1) > direction-BEAM_SHIFT_AZI && azimuth_polyfit(cntp)
<direction-BEAM_SHIFT_AZI ) ||...
                        ( azimuth_polyfit(cntp-1) < direction-BEAM_SHIFT_AZI && azimuth_polyfit(cntp) >
direction-BEAM_SHIFT_AZI ) )
                radar_pwr_beam_0 = ( radar_pwr_raw_0(cntp-1) + radar_pwr_raw_0(cntp) ) / 2 *
( .5*distance_receiver_polyfit(cntp-1) + .5*distance_receiver_polyfit(cntp) ).^4;
                radar_pwr_beam_1 = ( radar_pwr_raw_1(cntp-1) + radar_pwr_raw_1(cntp) ) / 2 *
( .5*distance_receiver_polyfit(cntp-1) + .5*distance_receiver_polyfit(cntp) ).^4;
                end;
        end;
        relative_gain = 10*log10( max(radar_pwr_raw_0) / max(radar_pwr_raw_1) );

        if( consistency > 15 ) % || relative_gain < 2 % drop bad radar data
                continue;
        end;

        % save the current trace and extracted data into an array for future use
        cnt_good = cnt_good + 1;
        good_data = [ good_data; data_current ];
        good_time = [ good_time; time_raw(cnt,:) ];
        good_table = [good_table; double(cnt), double(time_raw(cnt,1)), double(time_raw(cnt,2)),...
                elevation_angle(cnt,1), distance_radar(cnt,1), double(time_raw(cnt,6))*.3048/1000,...
                heading_angle, double(data_current(1,8)), consistency, relative_gain];

        % print table
        fprintf('%d\t%d\t%d\t%f\t%f\t%.3f\t%3.3f\t%4.2f\t%f\t%f', ...
                cnt, time_raw(cnt,1), time_raw(cnt,2), elevation_angle(cnt,1), distance_radar(cnt,1),...
                double(time_raw(cnt,6))*.3048/1000, heading_angle, double(data_current(1,8)), consistency,
relative_gain );
        fprintf('\t%f\t%f\t%d', radar_pwr_beam_0, radar_pwr_beam_1, time_raw(cnt,3));
        [bird_length, bird_type] = receiver_aircraft_query(time_raw(cnt,3), 1);
        if( bird_length ~= -1 )
                fprintf('\t%f\t%s', bird_length, bird_type);
```

```
end;
fprintf('\n');

if( mode == 0 )
        continue;
end;

% plot extracted ADS-B information
figure(551);
subplot(221);
plot(secs_radar, radar_distance_0, 'r'); hold('on');
plot(secs_radar, radar_distance_1);
plot(secs_radar, distance_receiver_polyfit, 'black');
lib_plot_axis( time_raw(cnt,2) - radar_file_time(1,cnt), radar_distance_0, radar_distance_1, 'green');
xlabel('Time/second'); ylabel('Distance/km');
title( sprintf('Distance vs. time, %s', file_name) ); hold('off');
subplot(222);
receiver_plot_record(data_current(:,4), data_current(:,5));
xlabel('Longtitude'); ylabel('Latitude');
title('Trace of current record');
subplot(223);
plot( current_time_double - radar_file_time(1,cnt), elevation_double, 'r'); hold('on');
plot( secs_radar, elevation_polyfit );
lib_plot_axis(time_raw(cnt,2) - radar_file_time(1,cnt), elevation_double, elevation_polyfit, 'green');
xlabel('Time'); ylabel('Elevation/degree'); title('Elevation vs Time'); hold('off');
subplot(224);
plot( current_time_double - radar_file_time(1,cnt), azimuth_double, 'r'); hold('on');
plot( secs_radar, azimuth_polyfit );
lib_plot_axis(time_raw(cnt,2) - radar_file_time(1,cnt), azimuth_double, azimuth_polyfit, 'green');
xlabel('Time'); ylabel('Azimuth/degree'); title('Azimuth vs Time'); hold('off');

% plot radar information
figure(561);
subplot(221);
image(secs_radar, hts_radar, 50*log10(data_radar_0), 'CDataMapping', 'scaled');
axis('xy'); xlabel('Time/second'); ylabel('Distance/km');
title('CH0, auto scaled PWR');
subplot(222);
image(secs_radar, hts_radar, 50*log10(data_radar_1), 'CDataMapping', 'scaled');
axis('xy'); xlabel('Time/second'); ylabel('Distance/km');
title('CH1, auto scaled PWR');
```

```
        subplot(223);
        plot(secs_radar, 10*log10(radar_pwr_raw_0), 'r'); hold('on');
        plot(secs_radar, 10*log10(radar_pwr_raw_1) );
        lib_plot_axis(time_raw(cnt,2) - radar_file_time(1,cnt), 10*log10(radar_pwr_raw_0),
10*log10(radar_pwr_raw_1), 'green');
        xlabel('Time/second'); ylabel('Power/dB');
        title('Raw PWR'); hold('off');
        subplot(224);
        plot( azimuth_polyfit, 10*log10(radar_pwr_normal_distance_0), 'r'); hold('on');
        plot( azimuth_polyfit, 10*log10(radar_pwr_normal_distance_1) );
   %    plot(azimuth_polyfit, -3 + azimuth_polyfit-azimuth_polyfit, 'green');    % -3dB line
        plot( azimuth_polyfit, 2 * ant_pattern(azimuth_polyfit + BEAM_SHIFT_AZI, elevation_polyfit, 1), '--black');
        lib_plot_axis(direction-BEAM_SHIFT_AZI, 10*log10(radar_pwr_normal_distance_0),
10*log10(radar_pwr_normal_distance_1), 'green', azimuth_polyfit);
        xlabel('Azimuth/degree'); ylabel('Power/dB'); title('Normalized PWR with R^4'); hold('off');

        if( mode == 1 )
            continue;
        elseif( mode == 2 )
            pause;
        elseif( mode == 3 )    % get user control command
            user_control = input('command>');
            if( user_control > 0 )
                pointer_next = user_control;
            elseif( user_control == 0 )    % save data mamauly
                cnt_save = cnt_save + 1;
                save_data = [ save_data; data_current ];
                save_time = [ save_time; good_time(cnt_good,:) ];
                save_table = [ save_table; good_table(cnt_good,:) ];
            else
                pointer_next =    - 1;
            end;
        elseif( mode == 4 )    % truncate mode
            cut_start = input('FR>');
            cut_end = input('TO>');
            if( cut_start < min(secs_radar) || cut_start > max(secs_radar) ||...
                    cut_end < min(secs_radar) || cut_end > max(secs_radar) || cut_start > cut_end)
                fprintf('*** WARNING *** Wrong input @ rr_analyse()\n');
                continue;
            end;
            cut_start = cut_start + radar_file_time(1, cnt);
            cut_end = cut_end + radar_file_time(1, cnt);
            for cnt_cut = 1 : length(data_current),
                if( cut_start <= data_current(cnt_cut,2) && cut_end >= data_current(cnt_cut,2) )
                    save_data = [ save_data; data_current(cnt_cut,:) ];
                end;
            end;
            save_time = [ save_time; good_time(cnt,:) ];
            save_table = [ save_table; good_table(cnt,:) ];
    end;
end;

fprintf('\nNote: unit of elevation is degree; unit of distance and altitude is km\n');
```

Other

lib_distance.m

```
%% Calculate distance
% input: latitude(int32/degree), longtitude(int32/degree), altitude(feet)
% error is high when input coordinate are close enough
function [dis, ele_output, azi_output] = lib_distance(lat_0, lon_0, alt_0, model, lat_1, lon_1, alt_1, direction)

load receiver_const;
% process input data
switch nargin;
    % convert signle input array into lat, lon, alt array
    case 1
        size_lat_0 = size(lat_0);
        if( size_lat_0(1,2) ~= 3 )
            fprintf('*** ERROR *** Invalid input parameter @ lib_distance()\n');
            return;
        else
            lon_0 = lat_0(:,2);
            alt_0 = lat_0(:,3);
            lat_0 = lat_0(:,1);
        end;
        direction = 0;
    case 3
        direction = 0;
    case 4
        direction = 0;
    case 7
        alt_1 = alt_1 * 0.3048;    % convert input feet into meter
        direction = 0;
    case 8
    otherwise;
        fprintf('*** ERROR *** Invalid input parameter @ lib_distance()\n');
        return;
end;
if( nargin == 1 || nargin == 3 )
        lat_1 = LAT_RADAR;
        lon_1 = LON_RADAR;
        alt_1 = ALT_RADAR;
        alt_0 = alt_0 * 0.3048;    % convert input feet into meter
        model = 0;    % use default long distance model for calculation
end;

% input format is int32 instead of double, convert here
if( lat_0(1,1) > 180 || lon_0(1,1) > 180 )
    lat_0 = double(lat_0) * CONVERT_INT32_ANGLE;
    lon_0 = double(lon_0) * CONVERT_INT32_ANGLE;
    alt_0 = double(alt_0);
end;
% make sure size of two input groups are the same
if( length(lat_0) ~= length(lat_1) && length(lat_1) ~= 1 )
    fprintf('*** ERROR *** Invalid input size @ lib_distance()\n');
    return;
elseif( length(lat_0) > length(lat_1) )
    balance_array = ones( length(lat_0), 1 );
    lat_1 = lat_1 .* balance_array;
    lon_1 = lon_1 .* balance_array;
    alt_1 = alt_1 .* balance_array;
end;

% calculate elevation and azimuth
angle_core = distance('GC', [lat_0,lon_0], [lat_1,lon_1]) / 180 * pi;    % may have LICENSE ISSUE
% angle_core = lib_lib_angle(lat_0, lon_0, lat_1, lon_1);
ele_output = elevation( lat_0, lon_0, alt_0, lat_1, lon_1, alt_1 ) * -1;
ele = ele_output / -180 * pi;
azi_output = azimuth(lat_1, lon_1, lat_0, lon_0);
if( direction == 0 )
    azi_output = azi_output - 360 * floor(azi_output/180);    % transform range from 0~360 to -180~180
```

```
end;
% accurate distance calculation
if( model == 0 )    % long distance model
        angle_radar = ele + pi/2;
        angle_air = pi - angle_radar - angle_core;
        dis = sin(angle_core) ./ sin(angle_air) * RADIUS_EARTH;
else    % short distance model
end;

% format output, set distance to meters and angle to int32
% note azimuth is convert to 2^29 instead of 2^31
if( nargin == 1 )
        dis = int32( [dis*1000, ele_output*CONVERT_ANGLE_INT32 , azi_output*CONVERT_ANGLE_INT32] );
        %clear ele_output azi_output;
end;

%% Calculate the central angle between two points
% reference: http://en.wikipedia.org/wiki/Great-circle_distance
% input: lat_0, lon_0, lat_1, lon_1
% output: central angle in radius
function ca = lib_lib_angle(lat_0, lon_0, lat_1, lon_1)
switch nargin;
        % convert signle input array into lat, lon, alt array
        case 4
        otherwise;
                fprintf('*** ERROR *** Invalid input parameter @ lib_distance() | lib_lib_angle()\n');
                return;
end;
dlon = abs(lon_0 - lon_1);
% dlat = abs(lat_0 - lat_1);
ca = atan( ( ( cos(lat_1).*sin(dlon) ).^2 + ( cos(lat_0).*sin(lat_1) - sin(lat_0).*cos(lat_1).*cos(dlon) ).^2 ).^0.5 /...
        ( sin(lat_0).*sin(lat_1) + cos(lat_0).*cos(lat_1).*cos(dlon) ) );
```

lib_azimuth_convert.m

```
%% convert -180~180 azimuth back to 0~359
function azi = lib_azimuth_convert(azi_raw)

load receiver_const;
% process input data
switch nargin;
```

```
        case 1
        otherwise;
                fprintf('*** ERROR *** Invalid input parameter @ lib_azimuth_convert()\n');
                return;
end;

% determine the format is INT32 or degree
if( max(azi_raw) > 360 )
        format = 0;
        azi_raw = double(azi_raw) * CONVERT_INT32_ANGLE;
else
        format = 1;
end;

azi = azi_raw + 180 * ( 1 - sign(azi_raw) );

if( format == 0 )
        azi = int32( azi * CONVERT_ANGLE_INT32 );
end;
```

lib_plot_axis.m

```
%% Calculate the speed of aircraft
function lib_plot_axis(position, data_0, data_1, config, data_x)

switch nargin;
        % convert signle input array into lat, lon, alt array
        case 2
                range_min = min(data_0);
                range_max = max(data_0);
                config = '';
        case 3
                config = '';
        case 4
        case 5
        otherwise;
                fprintf('*** ERROR *** Invalid input parameter @ lib_plot_axis()\n');
                return;
end;

if( nargin ~= 2 )
```

```
            range_min = min( min(data_0), min(data_1) );
            range_max = max( max(data_0), max(data_1) );
end;

line = [range_min,range_max];
plot([position,position], line, config);

if( nargin == 5 )    % rescale
      axis([ min(data_x), max(data_x), range_min, range_max ]);
end;
```

References

[1] M. E. Serafy. (2009) Secondary Radar. [Online]. http://www.radartheory.8m.com/secondaryradar.html

[2] Sarasota Avionics International. (2009) Sarasota Avionics International. [Online]. http://sarasotaavionics.com/avionics/kt_76c

[3] L. J. Gertz, "Fundamentals of Mode S Parity Coding," Massachusetts Institute of Technology, Massachusetts, Project Report 1984.

[4] V. A. Orlando and P. R. Drouilhet, "Mode S Beacon System: Functional Description," Massachusetts Institute of Technology, Massachusetts, Project Report 1986.

[5] W. Stamper. (2005, Dec.) Defense Electronics. [Online]. http://rfdesign.com/military_defense_electronics/radio_understanding_mode_technology/index.html

[6] R. D. Grappel and R. T. Wiken, "Guidance Material for Mode S-Specific Protocol Application Avionics," Massachusetts Institute of Technology, Massachusetts, Project Report 2007.

[7] E. J. Cardew. (2010, May) Edward's Homepage. [Online]. http://www.lll.lu/~edward/edward/adsb/VerySimpleADSBreceiver.html

[8] P. Hunt, "CRC Calculation for Mode-S Transponders," EUROCONTROL Experimental Centre, Project Report 1994.

[9] Federal Aviation Administration, "RTCA SC-186 Working Group #3 1090 MHz Extended Squitter MOPS," , Washington, 2006, pp. 1-10.

[10] C. Wolff. (2009, Dec.) Radar Basics. [Online]. http://www.radartutorial.eu/01.basics/rb13.en.html

[11] Google. (2013) Google Map. [Online]. https://maps.google.com/maps?q=40.1675n+88.1589w&hl=en&ll=40.166986,-88.158602&spn=0.000564,0.001321&sll=40.211916,-88.091125&sspn=0.288401,0.676346&t=h&z=20