SOUTH DAKOTA
SCHOOL OF MINES
& TECHNOLOGY

Affordable Automatic Dependent Surveillance – Broadcast (ADS-B)

Prepared by:
Samuel Coffin & Theodore Schoper

Faculty Advisors:
Mr. Scott Rausch
Interim Department Head, Department of Electrical Engineering

Dr. Thomas Montoya
REU Site Director, Department of Electrical Engineering

Dr. Alfred Boysen
Professor, Department of Humanities

Program Information:
National Science Foundation
Award NSF #EEC-1359476

Research Experience for Undergraduates
Summer 2016

South Dakota School of Mines and Technology
501 E Saint Joseph Street
Rapid City, SD 57701

Table of Contents

Abstract

The United States aging air traffic control infrastructure that uses primary surveillance radar (PSR) and secondary surveillance radar (SSR) is being replaced by automatic dependent surveillance - broadcast (ADS-B). As a part of Federal Aviation Administration's (FAA's) plan to implement ADS-B, all aircraft within the National Airspace System must broadcast an ADS-B out signal by 2020. Currently, ADS-B out systems for sale to the general aviation population cost $5,000 to $6,000 [3]. The high cost is making compliance difficult for most pilots. The purpose of this research is to determine if it is possible to develop an ADS-B out system that is cheaper than the current systems on the market and then begin development of a prototype. This prototype will be assembled mainly using software defined radios (SDRs) capable of transmitting, and a Raspberry Pi micro controller. During our research an ADS-B program was developed that successfully track aircraft flying in our local area. Tracked aircraft were plotted on Google Maps and verified using FlightAware.

1. Introduction

The aerospace industry and the services they provide have become an essential part of our modern way of life, whether it's ordering products from across the globe, traveling long distances for business, or providing national defense. A necessary part of this industry consists of knowing what aircraft are currently flying overhead and managing their takeoffs and landings. Currently, the United States regulates its airspace through the Federal Aviation Administration which operates 750 air traffic control (ATC) facilities, 18,000 airports, and 4,500 air navigation facilities [13]. The radar technology known as primary surveillance radar (PSR) and secondary surveillance radar (SSR) are used in a majority of these facilities and has been in use since the 1950s and 60s. [6]

PSR is a passive system that transmits a signal that is reflected off of an aircraft, and the received reflected signal is used to determine aircraft position relative to the transmitting ground station. The major disadvantage of PSR is that any object or atmospheric phenomena can reflect back signals, forcing air traffic control to discern between that and actual aircraft. [13] SSR fixes this disadvantage by equipping aircraft with special equipment that receives interrogation signals from ground stations and transmits a reply that contains a variety of information such as aircraft ID, heading, and altitude. The major benefit of SSR is aircraft can transmit back their reply, and the ground stations consume less power. However, not every aircraft is equipped with a SSR transponder, so it is very common to see both PSR and SSR used. Examples of both are shown in Fig. 1. [13]

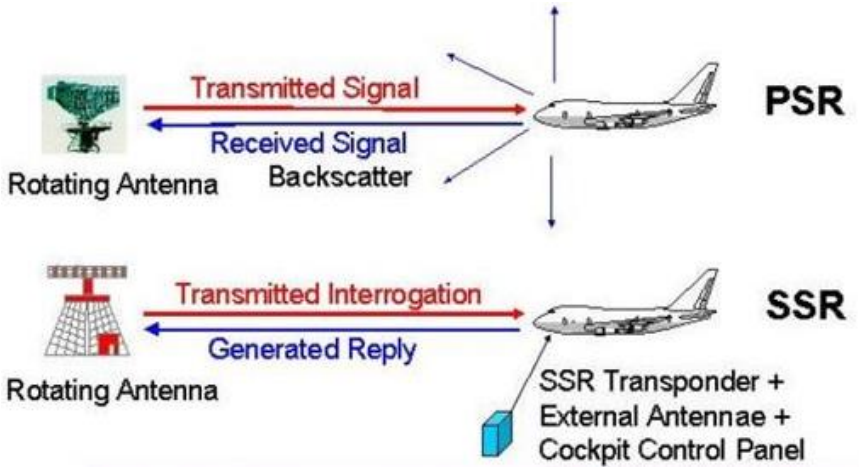

Fig. 1 - Diagram displaying difference between PSR and SSR [9]

Currently, roughly $150 million dollars are spent every year by the FAA to operate and maintain PSR and SSR equipment. This equipment allows air traffic controllers to handle 9 to 15 aircraft at one given time. Both of these factors, coupled with the yearly increase in global commercial air travel is expected to double from its current size by 2035 [12], has put pressure on the FAA to replace this aging infrastructure.

As a part of their decision to update our nation's aerospace management infrastructure, the FAA has decided to implement the Automatic Dependent Surveillance - Broadcast (ADS-B), shown in fig. 2, system nationwide as a replacement for PSR and SSR. On aircraft, ADS-B's main functions are to automatically broadcast a 112 bit message at 1090 MHz containing: aircraft position, altitude, heading, call sign, and aircraft type, [13], along with receiving ADS-B broadcasts from other aircraft or ground stations to be displayed within the cockpit. The data transmitted is similar to Mode S transmission, the current iteration of PSR technology. The navigation data used is given to ADS-B from onboard global navigation satellite system (GNSS), which is typically Global Positioning System (GPS). [13] ADS-B ground stations are used to deliver ADS-B data to ATC, and also re-broadcast ADS-B received, increasing the range of the original broadcast.

There are several reasons ADS-B was chosen to replace SSR and PSR. ADS-B provides better coverage in areas radar could not reach, such as the Gulf of Mexico and parts of Alaska [8]. ADS-B also broadcasts location once a second, compared to five to ten seconds using traditional radar. This allows reduced separations between aircraft from 4500 ft. to 750 ft., which reduces travel time and fuel consumption. With ADS-B pilots can make continuous descent landings instead of using the current stair step process that wastes fuel alternating between descending and leveling off. [13]

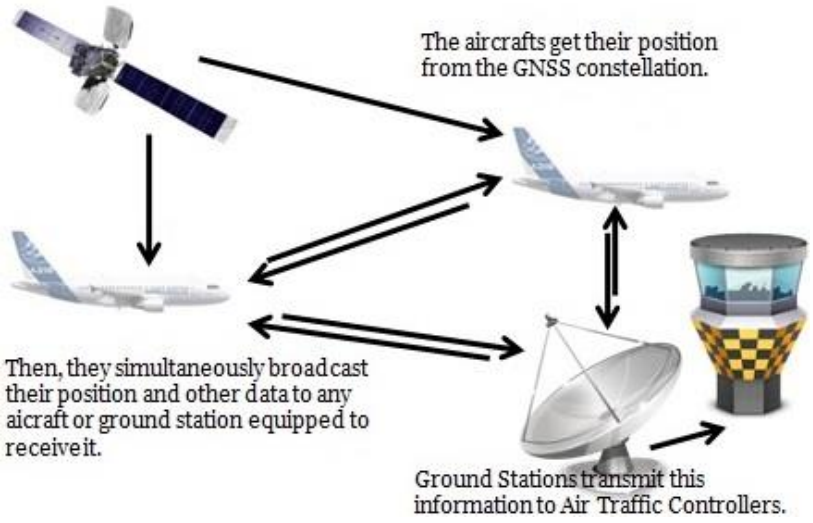

Fig. 2 - Illustration displaying how ADS-B works [5]

The purpose of this research is to familiarize ourselves with ADS-B by developing an ADS-B in system and beginning to develop an ADS-B out system. Progress made will be carried over to a senior design project at South Dakota School of Mines and Technology (SDSM&T). To accomplish these objectives we will be making hardware and software selections, assembling our hardware, writing the necessary code, and finally testing the completed system.

2. Broader Impact

ADS-B already has already made a considerable impact on the aviation community. New aircraft are now manufactured with ADS-B pre-installed during production, and the U.S. Government is investing a large amount in new ground stations ($100,000 - $400,000 per station) [7]. The FAA has also mandated that by 2020 that all aircraft have ADS-B. With the 2020 mandate fast approaching currently only a projected 20,078 [11] out of 188,099 [3] have ADS-B out installed as of Jan. 1, 2016. There is still a large portion of aircraft that must still be equipped with ADS-B. For a large portion of those not equipped, the greatest barrier to entry is the current price to equip an ADS-B out system. Right now the cheapest a consumer could equip ADS-B out is for $2,000 or $3,700 for ADS-B in/out, with an average cost of around $5,000 including installation [11]. To increase the appeal of upgrading to ADS-B out, it's critical to provide a lower cost alternative to the current products available on the market.

3. Procedure

3.1 Equipment

- HackRF One
- Raspberry Pi 3 Model B
- Adafruit Ultimate GPS Breakout
- 1090 MHz Antenna

3.2 Parts

To begin research on developing an embedded system that is capable of broadcasting out data, it was found that similar projects had been completed using a Raspberry Pi and a software defined radio (SDR). However, most of these projects just received radio broadcasts and ADS-B is going to be broadcasting out as well. With that in mind a HackRF One was selected. A HackRF One is a SDR capable of broadcasting and receiving that operates with a frequency range of 1 MHz to 6 GHz, is compatible with many open

source software toolkits, and has a high sample rate of up to 20 million samples per second. In order to use a SDR for ADS-B, a 1090 MHz antenna was used in conjunction with the HackRF One, shown in Fig. 3.

To provide GPS data, an Adafruit Ultimate GPS Breakout V3 was used. This GPS module in particular because it is intended to be used in embedded applications and with the Raspberry Pi. Finally for the microcontroller, the Raspberry Pi 3, shown in Fig. 4, was selected. The Raspberry Pi was chosen because of its small size, and similar projects had great success. With various Raspberry Pi models there is an abundant amount of information available on how to program them, and the built-in Wi-Fi makes remote access easy.

Fig. 3, The HackRF One [10] and 1090 MHz antenna [1]

Fig. 4, Raspberry Pi 3 Model B [15] and GPS module [2]

3.3 Software

The hardware was programmed in a Linux OS environment using GNU Radio and the Python coding language. GNU Radio is an open source visual based programming SDR toolkit that provides various prebuilt functions and utilities to program a SDR, see Fig. 5 for an example. GNU Radio also allows the user to write their own custom programming blocks which was used for this project's ADS-B system. Programs written in GNU Radio can be viewed as a Python file that is then deployed to a remote system such as a Raspberry Pi.

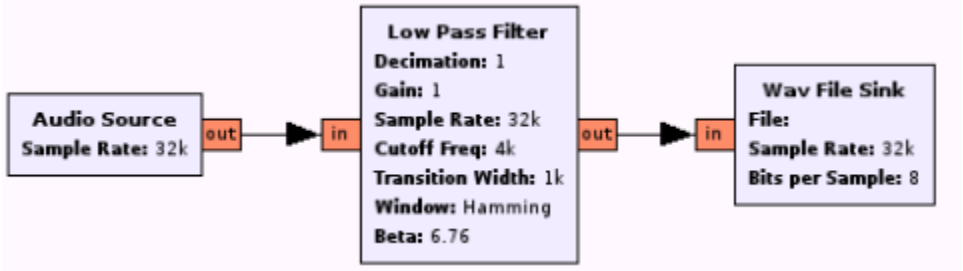

Fig. 5 – A basic example of a GNU Radio flow chart passing an audio signal through a filter to a file sink. [4]

3.4 Assembly and Coding

While the ultimate goal is to develop an ADS-B out system, we decided to start with an in system. There are several reasons why this approach was chosen:

1. ADS-B in systems have already been successfully developed by hobbyist and students, so there a wealth of material on how to do this.

2. This was the group's first experience with programming SDRs and wireless communication, developing an ADS-B in system allowed them to learn the hardware and software with a lower learning curve than developing an out system.

3. The in system can be used to test and benchmark the out system.

For the in system attention was focused on programming and running the system through a Linux OS laptop and GNU Radio. To run the in system the 1090 MHz antenna and HackRF One were directly connected to the laptop, as seen in Fig. 6.

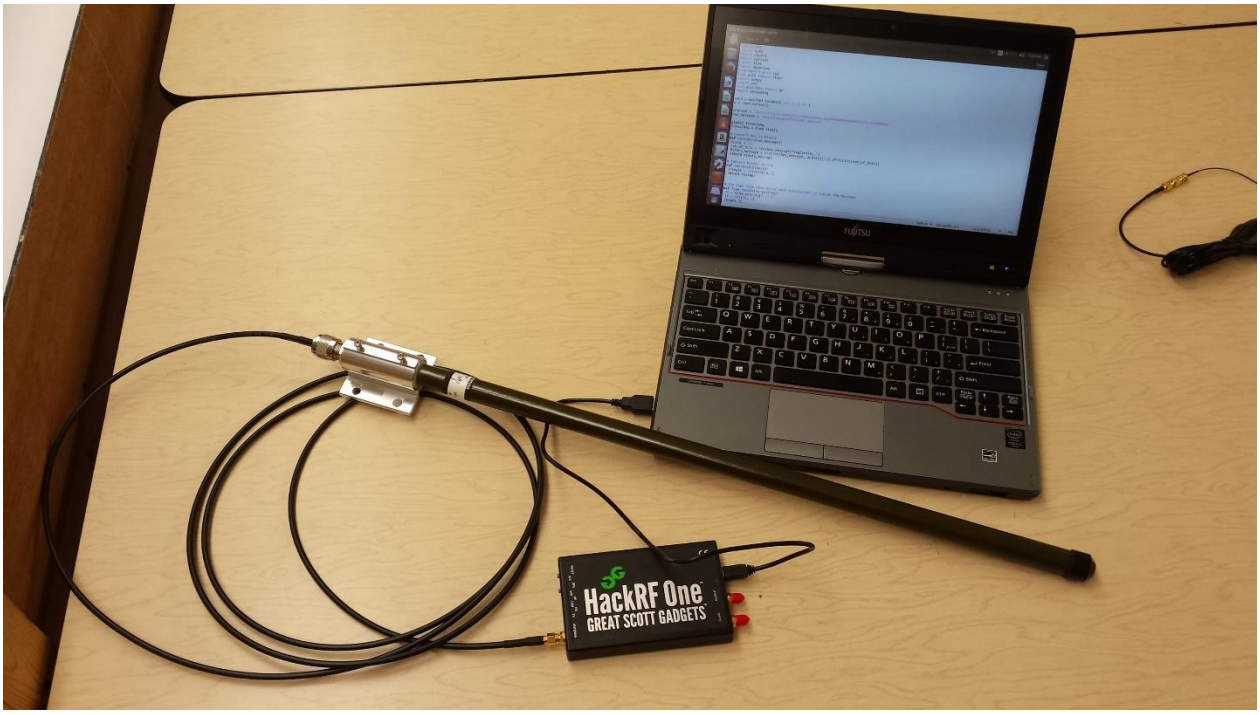

Fig. 6 – A HackRF One equipped with a 1090 MHz antenna, connected to a laptop running the ADS-B program.

4. Results

4.1 GNU Radio Companion (GRC)

Through GRC an ADS-B signal at 1090 MHz was successfully received. GRC is block diagram coding that allows the user to program the radio to receive or transmit a RF signal. Fig. 7 shows a screenshot of the diagram build in GRC. Each block performs a different task listed in Table 1. Receiving a message is just one part of an ADS-B system, and once received needs to be decoded and stored. This was accomplished through the program shown in Appendix A beginning page 16.

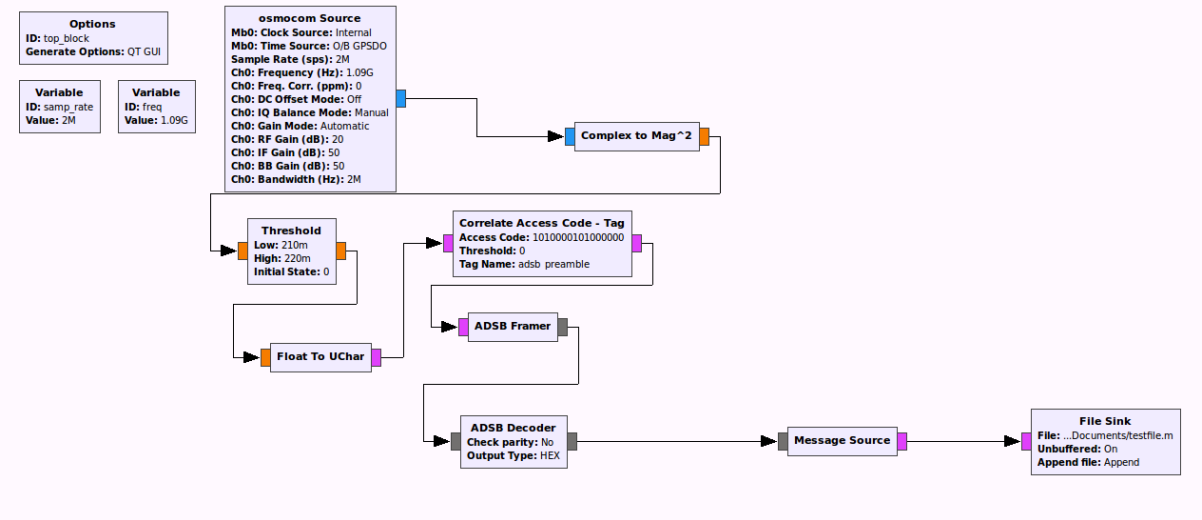

Fig. 7 – A GNU Radio flow graph that decodes ADS-B messages.

Table 1 - Description of blocks shown in Fig. 7

Block Title	**Description**
Oscmocom Source	Defines the HackRF One as our recieve hardware and programs it to receive at 1090 MHz, as well as initial and intermediate filter gains.
Complex to Mag^2	Signals received have real and imaginary components. This block removes the imaginary portion and passes along the real component of the signal.
Threshold	Defines the low and high point of the signal, converting the noisy signal to a signal that is either high (1) or low (0).
Float to UChar	Converts the signal from a series of floating point values to binary values.
Correlate Access Code	Looks for the 16 bit preamble of an ADS-B message. Once a preamble is received the next 112 bits are passed to the ADS-B Framer block.
ADS-B Framer	Compiles the 112 bit message and passes it along the ADS-B Decoder.
ADS-B Decoder	Converts the binary message received from the ADS-B Framer and converts it into a hexadecimal message.
Message Source	Compiles all of the individual bits together and sends that message to the File Sync.
File Sync	Deposits the final hexadecimal message into a specified file that can be opened with GNU Octave.

4.2 ADS-B Message

Before an ADS-B message can be decoded it is important to identify the different parts of a message, see Fig. 8. The first part of the message is called the preamble, which is the first 16 bits of any message and is used to identify the beginning of a message. A simple way to think of a preamble is as a doorkeeper that will only allow the message through if it has the proper code.

Fig. 8 - Diagram displaying the different parts of an ADS-B message [13]

The next part of the message, bits 1-5, is the downlink format that will define what type of message is being received. ADS-B messages' downlink format are designated as 17 or 10001 in binary.
Bits 6-8 describe the capability of what information the aircraft's transmitter is capable of sending. For example a capability of 6 says that this aircraft can transmit both surface and airborne data.

The next part, bits 9-32, detail the specific address of the aircraft. This is also known as the International Civil Aviation Organization (ICAO) 24 bit address. This address is unique for each transponder and allows the message to identify what aircraft is broadcasting. The following 56 bits (33-88) are the actual details of the ADS-B data. The first 5 bits of the data designate the type code. This type code identifies what specific information is in the following 51 bits. Table 2, shown on the following page, identifies what information is associated with each type code. The final 24 bits of the message are a parity check to ensure received messages are not corrupt or erroneous.

Table 2 - List of type codes and their associated ADS-B message type

Type Code	ADS-B Message Type
1-4	Aircraft Identification (Call sign)
5-8	Surface Position Message
9-18	Airborne Position Message with barometric altitude
19	Airborne Velocity Message

4.3 ADS-B Program

The ADS-B program created in Python was intended to help understand the ADS-B message and to create a test bed for an ADS-B out system. The program can decode a message with a downlink format of 17 (ADS-B message). Once a message is received the program will identify and store information against the ICA024 address in a database that our program generates. Using the ICA024 address as a primary key, the program can associate the call sign, airborne position, altitude, and velocity data within the ADS-B message with an individual aircraft.

The aircraft call sign and velocity message types are straightforward in the decoding process. Once the aircraft has been identified using the ICA024 address and the type code has been discerned, the program will decode bits 56-88 and return the aircraft's call sign or velocity depending on type code. The ICA024 address is used to sort the information that is decoded and stored in a database.

The difficult part of an ADS-B message is decoding the aircraft's position. ADS-B messages use a reporting format called compact position reporting (CPR). Without CPR a position message would require a total of 22 and 23 bits to encode/decode the latitude and longitude. The altitude requires an additional 12 bits and the type code of 5 bits. This gives a total of 62 bits and we only have 56 bits of available space. CPR reduces the latitude and longitude message parts to 17 bits with one additional bit to mark the message as an even or odd bit. This is made possible by dividing the world up into a number

of longitude and latitude zones as shown in Fig. 7. Accurate positioning of aircraft require two messages and both messages must lie inside of the same zone index.

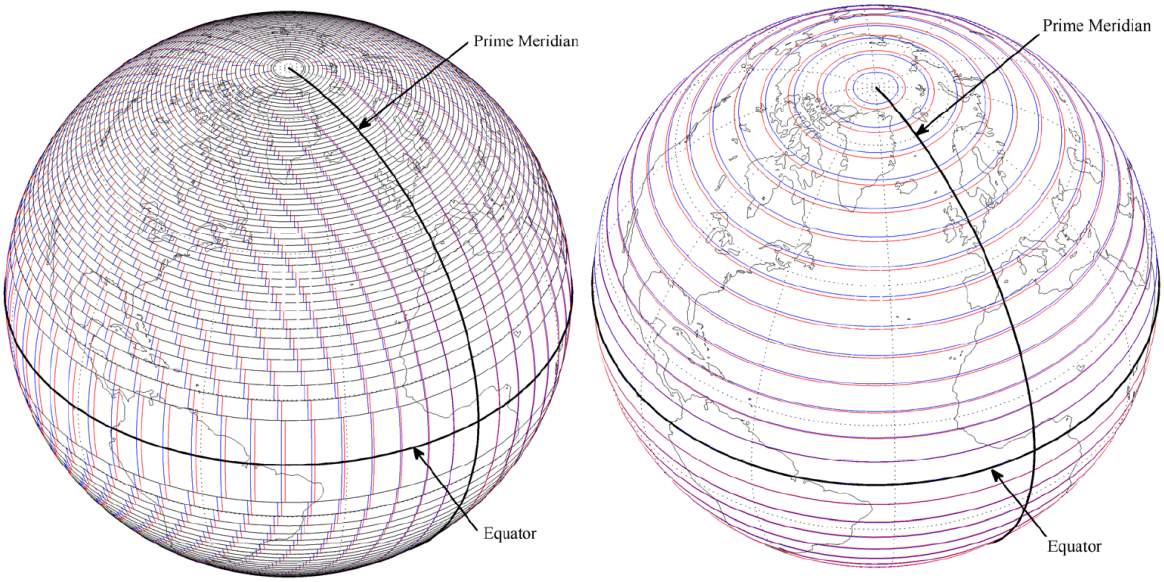

Fig. 7 - Longitude Zone boundaries (left), Latitude Zone boundaries (right) [14]

If the program receives a type code between 9 and 18 (position) it will look into a database and determine three things. Is there a previous position message attached to this aircraft's ICA024 address? Are the two messages within ten seconds of each other? Do the messages have a different CPR odd/even format bit? If all three criteria are not met the program will store the new message and wait for the next position message. If all three criteria are met the program will then send both messages to a decoder will decode the latitude and check to see if they lie inside the same grid block according the CPR guidelines. If they lay inside the same grid then the program will finish decoding the latitude, longitude and altitude and store these values inside the database.

5. Discussion

Initially this REU project was going to design an ADS-B out transmitter that could be installed on general aviation aircraft. However, ADS-B like any avionics equipment is strictly regulated by the FAA. There are many aspects of ADS-B that will need to be tested in order to meet the FAA standards. It is illegal to transmit a ADS-B out signal at 1090 MHz without the permission of the FAA. In order to design an ADS-B out system an accurate test system was developed. After research was conducted, the overall scope of the project shifted from development of an ADS-B out system to an ADS-B in system that could then be used for test purposes.

The program written can read an ADS-B hexadecimal message and is capable of decoding the call sign, position, or velocity depending upon the type code and the previous messages received. This program can be used to test the broadcast messages of an ADS-B out system.

6. Conclusion

6.1 Summary

With the 2020 FAA mandatory swiftly approaching, having aircraft equipped with ADS-B is a pivotal part of the plan to keep United States airspace safe and efficient. The current cost of ADS-B equipment and lack of immediate benefit is making aircraft owners hesitant to upgrade. By developing a cheap ADS-B out system that can meet the compliance standards set in RTCA DO-260 the cost of ADS-B systems can be lowered. With the system developed this summer we now have a greater understanding of ADS-B, along with the capability to develop and test an out system.

6.2 Future Work

The scope of this REU project was to begin development of an ADS-B out system. With the work accomplished this summer, this project can continue towards developing an ADS-B out system as a part of a senior design or REU project with this document serving as a preliminary point. The ADS-B out

system needs to be deployed to a remote Raspberry Pi and utilize the GPS transceiver as the source for aircraft position. Most importantly, when developing an ADS-B out system it is important that the following technical standards be followed:

- TSO-C166b, *Extended Squitter ADS-B and Traffic Information Service - Broadcast (TIS-B) Equipment Operating on the Radio Frequency of 1090 Megahertz*
- RTCA/DO-260B, *Minimum Operational Performance Standards for 1090 MHz Extended Squitter ADS-B and TIS-B*
- 14 CFR 91.227, *ADS-B Out equipment performance requirements*

APPENDIX A: ADS-B in Python Code

```
import csv
import math
import shutil
import sqlite3
import time
import datetime
from math import log
from math import floor
import numpy
import pmt
from gnuradio import gr
import threading

conn = sqlite3.connect('ADS-B_IN.db')
c = conn.cursor()
charset = '#ABCDEFGHIJKLMNOPQRSTUVWXYZ#####_###############0123456789######'
hex_message = "8DA1B0D458CD81E6B20AD16CE21A"
global timestamp
timestamp = time.time()



# Convert Hex to Binary
def hextobin(hex_message):
 scale = 16
 num_of_bits = len(hex_message)*log(scale, 2)
 binary_message = bin(int(hex_message, scale))[2:].zfill(int(num_of_bits))
 return binary_message

# Convert binary to int
def bintoint(binary):
 integer = int(binary,2)
 return integer


# The Type code determines what information is inside the message
def Type_Code(binarystring):
 TC = binarystring[32:37]
 TC = int(TC, 2)
 return TC


#-----------------------------------------------------------------------
# Aircraft Identification
# Type code 1 to 4 - Gives Aircraft Callsign
#-----------------------------------------------------------------------


def ACID(message):

 callsign = ''
 callsign += charset[bintoint(message[0:6])]
 callsign += charset[bintoint(message[6:12])]
```

```
 callsign += charset[bintoint(message[12:18])]
 callsign += charset[bintoint(message[18:24])]
 callsign += charset[bintoint(message[24:30])]
 callsign += charset[bintoint(message[30:36])]
 callsign += charset[bintoint(message[36:42])]
 callsign += charset[bintoint(message[42:48])]

 chars_to_remove = ['_', '#']
 cs = callsign.translate(None, ''.join(chars_to_remove))
 return cs in the module's to

#-------------------------------------------------------------------------
# Aircraft Position:
# Type Code 9 to 18 - Latitude and Longitude uses compact position #reporting
#-------------------------------------------------------------------------


# This function gets the bit to identify if a message is odd or even and
returns the one bit
def odd_even_flag(bin):
 odd_even = bin[53]
 return odd_even

# Checks the Q bit 48 to determine multiplier for aircraft altitude
def Q_bit(bin):
 Q = bin[48]
 return Q

def odd_even_ck(bin1,bin2):

 odd_even1 = odd_even_flag(bin1)

 if odd_even1 == 1:
  lat_odd = bin1
 else: in the module's to
  lat_even = bin1

 odd_even2 = odd_even_flag(bin2)

 if odd_even2 == 0:
  lat_even = bin2
 else:
  lat_odd = bin2

 if odd_even1 == odd_even2:
  return False
 else:
  return True

# Calculates both odd and even latitdue and compares them on CRP lat zone. If
they are
# the same it passes back just the even latitude.  Later on need to compare
the time
# stamps on the received hex signal and return the most current altitude.

def aircraft_latitude(bin1,bin2):
```

```
 odd_even1 = odd_even_flag(bin1)
 if odd_even1 == 1:
  lat_odd = bin1
 else:
  lat_even = bin1

 odd_even2 = odd_even_flag(bin2)
 if odd_even2 == 0:
  lat_even = bin2
 else:
  lat_odd = bin2

 lat_even = bintoint(lat_even[54:71])
 lat_odd = bintoint(lat_odd[54:71])

 lat_even_CPR = (lat_even/131072.0)
 lat_odd_CPR = (lat_odd/131072.0)

 j = abs(int(59 * lat_odd_CPR - 60 * lat_even_CPR + 0.5))

 relative_lat_even = float(360.0/60 * (j % 60 + lat_even_CPR))
 relative_lat_odd = float(360.0/59 * (j % 59 + lat_odd_CPR))

 print relative_lat_even
 print relative_lat_odd

 lat1 = int(num_lat_zone(relative_lat_even))
 lat2 = int(num_lat_zone(relative_lat_odd))

 print lat1
 print lat2

 if lat1 == lat2:
  latitude = relative_lat_even
  return latitude
 else:
  latitude = 0
  return latitude


def aircraft_longitude(bin1,bin2, lat):
 print "longitude"
 odd_even1 = odd_even_flag(bin1)
 if odd_even1 == 1:
  lon_odd = bin1
 else:
  lon_even = bin1

 odd_even2 = odd_even_flag(bin2)

 if odd_even2 == 0:
  lon_even = bin2
 else:
  lon_odd = bin2

 lon_even = bintoint(lon_even[71:88])
 lon_odd = bintoint(lon_odd[71:88])
```

```
 lon_even_CPR = (lon_even/131072.0)
 lon_odd_CPR = (lon_odd/131072.0)

 # compute the longitude index.

 ni = num_lat_zone(lat)
 m = floor(lon_even_CPR * (num_lat_zone(lat) - 1) - lon_odd_CPR *
num_lat_zone(lat) + 0.5)
 lon = 360.0/(ni) * (m % ni + lon_even_CPR)

 return lon

# This function will determine if the latitude is in the same zone accordinng
to CPR
def num_lat_zone(lat):
 #number of lat zones
 NZ = 60
 #gives what lat zone the give latitude is in
 nl = 2 * math.pi / (math.acos(1-(1-math.cos(math.pi * 2 /
NZ))/(math.cos(math.pi / 180 * abs(lat)) ** 2)))
 return nl


# This function calculates the altitude of aircraft depending on the Q bit
[48]
def altitude(alt, binary):

 message = ''
 message += binary [40:48]
 message += binary [49:52]
 message = bintoint(message)

 if alt == 0:
  message = message * 100 - 1000
 else:
  message = message * 25 - 1000
 return message

#-------------------------------------------------------------------------
# Velocity
#-------------------------------------------------------------------------


#-------------------------------------------------------------------------#
Creating and manipulating a table in SQLite
#-------------------------------------------------------------------------

def create_adsb_table():
 c.execute('CREATE TABLE IF NOT EXISTS adsb_in(ICA024 BLOB, Callsign TEXT,
Latitude REAL, Longitude REAL, Altitude REAL, Velocity REAL, Timestamp R)')

def add_data_to_table(AA, Callsign, Lat, Lon, Alt):
 check = find_ICA024(AA)
 if check == True:
  update_table(AA, Callsign, Lat, Lon, Alt)
 else:
  add_new_ICA024(AA, Callsign, Lat, Lon, Alt)
```

```
def update_table(AA, Callsign, Lat, Lon, Alt):
 if Callsign != 0:
  c.execute('UPDATE adsb_in SET Callsign = (?) WHERE ICA024 = (?)',(Callsign,
AA))
  conn.commit()
 else:
  c.execute('UPDATE adsb_in SET Latitude = (?) WHERE ICA024 = (?)',(Lat,AA))
  c.execute('UPDATE adsb_in SET Longitude = (?) WHERE ICA024 = (?)',(Lon,AA))
  c.execute('UPDATE adsb_in SET Altitude = (?) WHERE ICA024 = (?)',(Alt,AA))
  c.execute('UPDATE adsb_in SET Timestamp = (?) WHERE ICA024 =
(?)',(timestamp, AA))
  conn.commit()

def add_new_ICA024(AA, Callsign, Lat, Lon, Alt):
 if Callsign !=0:
  c.execute("INSERT INTO adsb_in (ICA024, Callsign) VALUES (?,?)",(AA,
Callsign))
  conn.commit()
 else:
  c.execute("INSERT INTO adsb_in (ICA024, Latitude, Longitude, Altitude)
VALUES (?,?,?,?)",(AA,Lat,Lon,Alt))
  conn.commit()


def read_from_database(AA):

 c.execute("SELECT Latitude " "FROM adsb_in WHERE ICA024 = ?", (AA,))
 latitude = c.fetchone()
 if latitude != None:
  lat = (''.join(map(str, (latitude))))
 else:
  lat = 0

 c.execute("SELECT Longitude " "FROM adsb_in WHERE ICA024 = ?", (AA,))
 longitude = c.fetchone()
 if longitude != None:
  lon = (''.join(map(str, (longitude))))
 else:
  lon = 0

 if lat == 0 and lon == 0:
  return False
 else:
  return True

def find_ICA024(AA):
 c.execute("SELECT ICA024 " "FROM adsb_in WHERE ICA024 = ?", (AA,))
 for row in c.fetchall():
  return True

#---------------------------------------------------------------------------
#This section will create a database to store the binary code if both an #odd
and even are not availabe
#---------------------------------------------------------------------------
```

```
def create_bin_table():
 c.execute('CREATE TABLE IF NOT EXISTS message_storage(ICA024 BLOB, Hex BLOB,
Timestamp REAL)')

def add_new_bin_table(Bin, AA):
 c.execute("INSERT INTO message_storage (ICA024, Hex, Timestamp) VALUES
(?,?,?)",(AA, Bin, timestamp))
 conn.commit()

def update_bin_table(Bin, AA):
 c.execute('UPDATE message_storage SET Hex = (?) WHERE ICA024 = (?)',(Bin,
AA))
 c.execute('UPDATE message_storage SET Timestamp = (?) WHERE ICA024 =
(?)',(timestamp, AA))
 conn.commit()

def read_from_bin_table(AA):
 Hex = 0
 c.execute("SELECT Hex " "FROM message_storage WHERE ICA024 = ?", (AA,))
 bin_message = c.fetchone()

 c.execute("SELECT Timestamp " "FROM message_storage WHERE ICA024 = ?",
(AA,))
 time_stamp = c.fetchone()



 if bin_message != None:
  Hex = (''.join(map(str, (bin_message))))
  Bin = hextobin(Hex)
 else: in the module's to
  Bin = 0


 if Bin == 0:
  return (False)
 else:
  return (Bin)


#-------------------------------------------------------------------------
# Main
#-------------------------------------------------------------------------
create_bin_table()
create_adsb_table()


callsign = 0
check = False
TC = 0
binary_message = hextobin(hex_message)
ICA024 = hex_message[2:8]
TC = Type_Code(binary_message)
print (TC)
Aircraft_ID = 0
Bar_Alt = 0
```

```
lat = 0
lon = 0
CPR1 = False

# Aircraft Callsign
if TC >= 1 and TC <= 4:
 binmessage = binary_message[40:88]
 callsign = ACID(binmessage)
 print "Aircraft Callsign ",callsign


# Aircraft Position
if TC >= 9 and TC <= 18:
 pos_ck = read_from_database(ICA024)
 if pos_ck == False:
  ck_bin_table = read_from_bin_table(ICA024)
  if ck_bin_table == False:
   add_new_bin_table(hex_message, ICA024)
  else:
    CPR = odd_even_ck(binary_message, ck_bin_table)
    if CPR == False:
     update_bin_table(hex_message, ICA024)
    else:
     alt = Q_bit(binary_message)
     Bar_Alt = altitude(alt, binary_message)
     print "Barametric Altitude = ", Bar_Alt
     lat = aircraft_latitude(binary_message, ck_bin_table)
     print "Latitude = ", lat
     print "blah balh"
     lon = aircraft_longitude(binary_message, ck_bin_table, lat)
     print "Longitude = ", lon
     add_data_to_table(ICA024, callsign, lat, lon, Bar_Alt)
     update_bin_table(hex_message, ICA024)
 else:
  ck_bin_table = read_from_bin_table(ICA024)
  CPR1 = odd_even_ck(binary_message, ck_bin_table)
if TC >= 9 and TC <= 18:
 if CPR1 == False:
  update_bin_table(hex_message, ICA024)
 else:
  alt = Q_bit(binary_message)
  Bar_Alt = altitude(alt, binary_message)
  print "Barametric Altitude = ", Bar_Alt
  lat = aircraft_latitude(binary_message, ck_bin_table)
  print "Latitude = ", lat
  if lat == 0:
   lon = aircraft_longitude(binary_message, ck_bin_table, lat)
   print "Longitude = ", lon
   add_data_to_table(ICA024, callsign, lat, lon, Bar_Alt)
   update_bin_table(hex_message, ICA024)
c.close()
conn.close()
```

References

1. 1090MHz ADS-B - 66cm / 26in. (n.d.). Retrieved August 02, 2016, from https://www.amazon.com/1090MHz-ADS-B-Antenna-66cm-26in/dp/B00WZL6WPO

2. Adafruit Ultimate GPS Breakout - 66 channel w/10 Hz updates. (n.d.). Retrieved July 18, 2016, from https://www.adafruit.com/product/746

3. Air Traffic Services Brief -- Automatic Dependent Surveillance-Broadcast (ADS-B). (2015, June 3). Retrieved July 07, 2016, from https://www.aopa.org/advocacy/advocacy-briefs/air-traffic-services-brief-automatic-dependent-surveillance-broadcast-ads-b

4. Braun, M. (2014, November 24). Core concepts of GNU Radio¶. Retrieved August 02, 2016, from http://gnuradio.org/redmine/projects/gnuradio/wiki/TutorialsCoreConcepts

5. Casanovas, F. (2013, September 29). Low-cost hardware and free software. Retrieved July 13, 2016, from https://ferrancasanovas.wordpress.com/2013/09/
Article translated from Spanish to English using Chrome web browser.

6. Chang, E., Hu, R., Lai, D., Li, R., Scott, Q., & Tyan, T. (2000, December 15). An Overview of Air Traffic Control. Retrieved July 11, 2016, from http://web.mit.edu/6.933/www/Fall2000/mode-s/atc.html

7. Dunstone, Greg. (2012). ADS-B Introduction [Power Point Slides]. Retrieved from http://www.icao.int/APAC/Meetings/2012_SEA_BOB_ADSB_WG8/SP01_AUS%20-%20ADS-B%20Basics.pdf

8. Fact Sheet – Automatic Dependent Surveillance-Broadcast (ADS-B). (2010, June 24). Retrieved July 14, 2016, from http://www.faa.gov/news/fact_sheets/news_story.cfm?newsid=7131

9. Ferrer, A. (2009, January 21). Frequently Asked Questions (IX) - The Radar (3/3). Retrieved July 13, 2016, from http://surcandoloscielos.es/blog/frequently-asked-questions-ix-el-radar-3parte/
Article was translated from Spanish to English using Chrome web browser.

10. HackRF One. (n.d.). Retrieved July 18, 2016, from https://greatscottgadgets.com/hackrf/

11. Hughes, D. (n.d.). *Got ADS-B?* (United States, FAA, NextGen Peformance and Outreach). https://www.faa.gov/nextgen/media/AEA_ADS-B_Article.PDF

12. Jones, C. (2015, June 08). Air travel demand projected to double in 20 years. Retrieved August 02, 2016, from http://www.usatoday.com/story/travel/flights/2015/06/08/demand-to-fly-will-likely-double-but-industry-may-not-be-ready/28680637/

13. Magazu, D. (2012). *Exploiting the Automatic Dependent Surveillance-Broadcast system via false target injection* (Master's thesis, Air Force Institute of Technology). Wright-Patterson Air Force Base: Department of the Air Force.

14. *Minimum operational performance standards (MOPS) for 1090 MHz extended squitter automatic dependent surveillance - broadcast (ADS-B) and traffic information services - broadcast (TIS-B)*. (2011). Washington, DC: RTCA.

15. Raspberry Pi 3 Model B Motherboard. (n.d.). Retrieved July 20, 2016, from https://www.amazon.com/Raspberry-Pi-RASP-PI-3-Model-Motherboard/dp/B01CD5VC92/ref=sr_1_3?ie=UTF8

Acknowledgements

The funding for this research came from the National Science Foundation REU Site: Bringing Us Together, Improving Communications and Lives through award number EEC-1359476. Special thanks to our advisor, Mr. Rausch, for his guidance and assistance with the project, and Dr. Boysen from the English department for his critique in writing and speaking. Finally, we would like to thank SDSM&T President Heather Wilson for her generous donation of $500 to the project.