ADS-B Decoding Guide

Release 0.3

Junzi Sun

January 30, 2017

This is a small research project conducted by Junzi Sun at TuDelft. While we were trying to work with ADS-B data collected from our receiver, we notice that there are very few documents available which can explain the ADS-B data comprehensively. So, we created this guide, along with a decoder written in python (https://github.com/junzis/pyModeS). Have Fun!

The main focus of the guide is on reading different types of messages, understanding the information in the message, and decoding/computing aircraft status.

CHAPTER 1

Introduction

1.1 ADS-B

ADS-B is short for Automatic Dependent Surveillance–Broadcast. it is a satellite based survillance system. Aircraft position, velocity, together with identification are transmitted through Mode-S Extended Squitter (1090 MHz).

Majority of the aircraft nowadays are broadcasting ADS-B messages constantly. There are many ways you can set up you own receiver and antenna to start tapping into those signals (DVB-T usb stick, ModeSBeast, Raspberry Pi, RadarScape, etc).

An ADS-B message is 112 bits long, and consist of 5 parts:

```
+--------+--------+-----------+--------------------------+---------+
|  DF 5  |  ** 3  |  ICAO 24  |          DATA 56         |  PI 24  |
+-----------------+-----------+--------------------------+---------+
```

This table lists the key bits of a message:

nBits	Bits	Abbr.	Name
5	1 - 5	DF	Downlink Format (17)
3	6 - 8	CA	Capability (additional identifier)
24	9- 32	ICAO	ICAO aircraft address
56	33 - 88	DATA	Data
	[33 - 37]	[TC]	Type code
24	89 - 112	PI	Parity/Interrogator ID

Example:

```
Raw message in hexadecimal:
8D4840D6202CC371C32CE0576098



-----+------------+--------------+-------------------------------+--------------
HEX  | 8D         | 4840D6       | 202CC371C32CE0                | 576098
-----+------------+--------------+-------------------------------+--------------
BIN  | 10001  101 | 010010000100 | [00100]0000010110011000011011 | 010101110110
     |            | 000011010110 | 10001110000110010110011100000 | 000010011000
-----+------------+--------------+-------------------------------+--------------
DEC  |  15    5   |              | [4] .......................   |
-----+------------+--------------+-------------------------------+--------------
        DF    CA     ICAO           [TC] ------ DATA ----------    PI
```

Any ADS-B must start with the Downlink Format 17 (10001 in binary code) for the first 5 bits. Bits 6-8 are used as additional identifier, which has different meanings within different types of ADS-B message.

1.2 ADS-B message types

To identify what information is contained in a ADS-B message. We need to take a look at the `Type Code` of the message, indicated at bits 33 - 37 of the ADS-B message (or first 5 bits of the `DATA` segment)

Following are the relationship between each `Type Code` and its information contained in the `DATA` segment:

TC	Content
1 - 4	Aircraft identification
5 - 8	Surface position
9 - 18	Airborne position (w/ Baro Altitude)
19	Airborne velocities
20 - 22	Airborne position (w/ GNSS Height)
23 - 31	Reserved for other uses

1.3 ADS-B Checksum

ADS-B uses cyclic redundancy check to validate the correctness of received message, where the last 24 bits are the parity bits. Following pseudo-code describes the CRC process:

```
GENERATOR = 1111111111111010000001001

MSG = binary(8D4840D6202CC371C32CE0576098)    # 112 bits

for i from 0 to 88:                           # 112 bits - 24 parity bits
  if MSG[i] is 1:
    MSG[i:i+24] = MSG[i:i+24] ^ GENERATOR

CRC = MSG[-24:]                               # last 24 bits of result

IF CRC not 0:
  MSG is corrupted
```

For the implementation of CRC encoder in python, refer to the pyModeS library function: `pyModeS.util.crc()`

A comprehensive documentation on Mode-S parity coding can be found:

```
Gertz, Jeffrey L. Fundamentals of mode s parity coding. No. ATC-117.
MASSACHUSETTS INST OF TECH LEXINGTON LINCOLN LAB, 1984. APA
```

Aircraft Identification

An aircraft identification message has `DF: 17`, and `TC: 1 to 4`, the 56-bit `DATA` field is configured as follows:

```
+---------+---------+---------+---------+---------+---------+---------+---------+---------+---------
|  TC (5) |  EC (3) |  C1 (6) |  C2 (6) |  C3 (6) |  C4 (6) |  C5 (6) |  C6 (6) |  C7 (6) |  C8 (6)
+---------+---------+---------+---------+---------+---------+---------+---------+---------+---------


TC: Type code
EC: Emitter category
C*: Charactor
```

For decode charactors, a lookup table is needed for mapping numbers to characters. It is defines as follows, where the # is not used, and _ represents a sepration.

```
#ABCDEFGHIJKLMNOPQRSTUVWXYZ#####_###############0123456789######
```

In summary, characters and there decimal reprsentations are:

```
A - Z :   1 - 26
0 - 9 :  48 - 57
    _ :  32
```

The `EC` value in combination with `TC` value defines the category of the aircraft (such as: heavy, large, small, light, glider, etc.). When `EC` is set to zeros, such information is not avaiable.

2.1 Example

For example:

```
8D4840D6202CC371C32CE0576098
```

The structure of the message is:

```
     DF--- CA-  ICAO--  DATA------------------  PI----
HEX: 8   D      4840D6  2   0     2CC371C32CE0  576098
BIN: 10001|101  ******  00100|000 ************  ******
DEC: 17   |4            4     0
                        TC    *
```

Note that `Type Code` is inside of the DATA frame (first 5 bits). With `DF=17` and `TC=4`, we can confirm this is a aircraft identification message. Aircraft `callsign` then can be decoded.

In previous example message, it is easy to decode the `Data` segment:

```
HEX: 20         2CC371C32CE0
BIN: 00100000 | 001011 001100 001101 110001 110000 110010 110011 100000
DEC:          |   11     12     13     49     48     50     51     32
LTR:          |   K      L      M      1      0      2      3      _
```

So the final aircraft callsign decoded is: **KLM1023_**

For detailed codes in python, refer to the pyModeS library function: `pyModeS.adsb.callsign()`

CHAPTER 3

Airborne Positions

An aircraft position message has `DownlinkFormat: 17` and `TypeCode: from 9 to 18`.

Messages are composed as following:

MSG bits	# bits	Abbr	Content
1-5	5	DF	Downlink format
33-37	5	TC	Type code
38-39	2	SS	Surveillance status
40	1	NICsb	NIC supplement-B
41-52	12	ALT	Altitude
53	1	T	Time
54	1	F	CPR odd/even frame flag
55-71	17	LAT-CPR	Latitude in CPR format
72-88	17	LON-CPR	Longitude in CPR format

Decoding the positions of the aircraft is a bit complicated. Naturally, we would expect to read latitude and longitude directly from the data frame. Unfortunately, it's not that simple...

Two different types of the position messages (odd and even frames) are needed to find out the LAT and LON of the aircraft. The position is described in the Compact Position Reporting (CPR) format. The advantage of CPR is that we can use fewer bits to encode position with higher resolution. However this results the complexity of decoding process.

3.1 First, "odd" or "even" message?

For each frame, bit 54 determines whether it is an "odd" or "even" frame:

```
0 -> Even frame
1 -> Odd frame
```

For example, the two following messages are received:

```
8D40621D58C382D690C8AC2863A7
8D40621D58C386435CC412692AD6

|    | ICAO24 |       DATA      |   CRC   |
|----|--------|-----------------|--------|
| 8D | 40621D | 58C382D690C8AC | 2863A7 |
| 8D | 40621D | 58C386435CC412 | 692AD6 |
```

Convert both messages to binary strings:

```
| DF    | CA  | ICAO24 ADDRESS             | DATA                                    ->
|=======|=====|==========================|========================================
|                                          | TC    | SS | NICsb | ALT          | T | ->
|-------|-----|--------------------------|-------|----|-------|--------------|---| ->
| 10001 | 101 | 010000000110001000011101 | 01011 | 00 | 0     | 110000111000 | 0 | ->
| 10001 | 101 | 010000000110001000011101 | 01011 | 00 | 0     | 110000111000 | 0 | ->


  ->  Data (cont.)                                | CRC                      |
  ->  ============================================|==========================|
  ->  | F | LAT-CPR           | LON-CPR           |                          |
  ->  |---|-------------------|-------------------|--------------------------|
  ->  | 0 | 10110101101001000 | 01100100010101100 | 001010000110001110100111 |
  ->  | 1 | 10010000110101110 | 01100010000010010 | 011010010010101011010110 |
```

In both messages we can find `DF=17` and `TC=11`, with the same ICAO24 address `40621D`. So, those two frames are valid for decoding the positions of this aircraft.

3.2 CPR parameters and functions

First, we denotes some of the parameters and common functions used in the decoding process here.

3.2.1 NZ

Number of geographic latitude zones between equator and a pole. It is set to `NZ = 15` for Mode-S CPR encoding

3.2.2 floor(x)

the floor function `floor(x)` defines as the greatest integer value k, such that `k<=x`, for example:

```
floor(5.6) = 5
floor(-5.6) = -6
```

3.2.3 mod(x, y)

the modulus function `mod(x, y)` return:

$$x - y \cdot floor(\frac{x}{y})$$

where `y` can not be zero

3.2.4 NL(lat)

Denotes the "number of longitude zones" function, given the latitude angle `lat`. The returned integer value is constrained within `[1, 59]`, calculated as:

$$\mathrm{NL}(lat) = floor\left(\frac{2\pi}{\arccos(1 - \frac{1-\cos(\frac{\pi}{2\cdot\mathrm{NZ}})}{\cos^2(\frac{\pi}{180}\cdot\mathrm{lat})})}\right)$$

For latitudes that are close to equator or poles, following value is returned:

```
lat = 0      ->    NL = 59
lat = +87    ->    NL = 2
lat = -87    ->    NL = 2
lat > +87    ->    NL = 1
lat < -87    ->    NL = 1
```

3.3 Latitude/Longitude calculation

There are a few technical documents that explain in detail the math behind the CPR. For example: a document from Eurocontrol.

Let's first separate the CPR latitude and longitude bits in both messages. The steps after will guide you to calculate the LAT/LON of the aircraft.

```
| F | CPR Latitude       | CPR Longitude      |
|---|--------------------|--------------------|
| 0 | 10110101101001000 | 01100100010101100 |  -> newest frame received
| 1 | 10010000110101110 | 01100010000010010 |
```

3.3.1 Step 1: Convert the binary string to decimal value

```
LAT_CPR_EVEN: 93000 / 131072 -> 0.7095
LON_CPR_EVEN: 51372 / 131072 -> 0.3919
LAT_CPR_ODD:  74158 / 131072 -> 0.5658
LON_CPR_ODD:  50194 / 131072 -> 0.3829
```

Since CPR latitude and longitude are encoded in 17 bits, 131072 (2^17) is the maximum value. The resulting values from the calculations represent the percentages of that maximum value.

3.3.2 Step 2: Calculate the latitude index j

Use the following equation:

$$j = floor\left(59 \cdot Lat_{cprE} - 60 \cdot Lat_{cprO} + \frac{1}{2}\right)$$

```
j = 8
```

3.3.3 Step 3: Latitude

First, two constants will be used:

$$DLat_E = \frac{360}{4 \times NZ} = \frac{360}{60}$$
$$DLat_O = \frac{360}{4 \times NZ - 1} = \frac{360}{59}$$

Then we can use the following equations to compute the relative latitudes:

$$Lat_E = DLat_E * (mod(j, 60) + Lat_{cprE})$$
$$Lat_O = DLat_O * (mod(j, 59) + Lat_{cprO})$$

For southern hemisphere, values will fall from 270 to 360 degrees. we need to make sure the latitude is within range `[-90, +90]`:

$$Lat_E = Lat_E - 360 \quad \text{if } (Lat_E \geq 270)$$
$$Lat_O = Lat_O - 360 \quad \text{if } (Lat_O \geq 270)$$

Final latitude is chosen depending on the time stamp of the frames–the newest one is used:

$$Lat = \begin{cases} Lat_E & \text{if } (T_E \geq T_O) \\ Lat_O & \text{else} \end{cases}$$

In the example:

```
Lat_EVEN = 52.25720214843750
Lat_ODD  = 52.26578017412606
Lat = Lat_EVEN = 52.25720
```

3.3.4 Step 4: Check

Compute `NL(Lat_E)` and `NL(Lat_O)`. If not the same, two positions are located at different latitude zones. Computation of a global longitude is not possible. exit the calculation and wait for new messages.

If two values are the same, we proceed to longitude calculation.

3.3.5 Step 5: Longitude

If the even frame come latest `T_EVEN > T_ODD`:

$$ni = max\left(NL(Lat_E), 1\right)$$
$$DLon = \frac{360}{ni}$$
$$m = floor\left(Lon_{cprE} \cdot [NL(Lat_E) - 1] - Lon_{cprO} \cdot NL(Lat_E) + \frac{1}{2}\right)$$
$$Lon = DLon \cdot (mod(m, ni) + Lon_{cprE})$$

In case where the odd frame come latest `T_EVEN < T_ODD`:

$$ni = max\left(NL(Lat_O) - 1, 1\right)$$
$$DLon = \frac{360}{ni}$$
$$m = floor\left(Lon_{cprE} \cdot [NL(Lat_O) - 1] - Lon_{cprO} \cdot NL(Lat_O) + \frac{1}{2}\right)$$
$$Lon = DLon \cdot (mod(m, ni) + Lon_{cprO})$$

if the result is larger than 180 degrees:

$$Lon = Lon - 360 \quad \text{if } (Lon \geq 180)$$

In the example:

```
Lon:  3.91937
```

Here is a Python implemented: https://github.com/junzis/pyModeS/blob/faf4313/pyModeS/adsb.py#L166

3.4 Altitude Calculation

The altitude of the aircraft is much easier to compute from the data frame. The bits in the altitude field (either odd or even frame) are as following:

```
1100001 1 1000
        ^
       Q-bit
```

This Q-bit (bit 48) indicates whether the altitude is encoded in multiples of 25 or 100 ft (0: 100 ft, 1: 25 ft).

For Q = 1, we can calculate the altitude as following:

First, remove the Q-bit

```
N = 1100001 1000 => 1560 (in decimal)
```

The final altitude value will be:

$$Alt = N * 25 - 1000 \text{ (ft.)}$$

In this example, the altitude at which aircraft is flying is:

```
1560 * 25 - 1000 = 38000 ft.
```

Note that the altitude has the accuracy of +/- 25 ft when the Q-bit is 1, and the value can represent altitude from -1000 to +50175 ft.

3.5 The final position

Finally, we have all three components (latitude/longitude/altitude) of the aircraft position:

```
LAT: 52.25720 (degrees N)
LON:  3.91937 (degrees E)
ALT:    38000 ft
```

Airborne Velocity

There are two different types of messages for velocities, determined by 3-bit subtype in the message. With subtype 1 and 2, surface velocity (ground speed) is reported. And in subtype 3 and 4, aircraft airspeed are reported.

In real world, very few of the subtype 3 or 4 messages are reported. In our setup, we only received **0.3%** of these message with regard to subtype 1 (subtype 2 seems never used).

An aircraft velocity message has `DF: 17`, `TC: 19`. and the subtype code are represented in bits 38 to 40. Now, we can decode those messages.

4.1 Decoding message subtype 1 or 2

Subtype 1 (subsonic) or 2 (supersonic), are broadcast when ground velocity information are available. The aircraft velocity contains speed and heading information. The speed and heading are also decomposed into North-South, and East-West components.

Note: The following decoding only apply to Subtype 1 (subsonic).

For example, following message is received:

```
Message: 8D485020994409940838175B284F

|    | ICAO24 |      DATA      |  CRC   |
|----|--------|----------------|--------|
| 8D | 485020 | 99440994083817 | 5B284F |

Convert into binary:

| HEADER                                 |  DATA                     ->
|-------|-----|--------------------------|-------|-----|----|-------- ->
| DF    | CA  | ICAO24 ADDRESS           |  TC   | ST  | IC | RESV_A  ->
|-------|-----|--------------------------|-------|-----|----|-------- ->
| 10001 | 101 | 010010000101000000100000 | 10011 | 001 | 0  | 1       ->

 ->  DATA (cont.)                                                 ->
 -> |-----|------|------------|------|------------|-------|------ ->
 -> | NAC | S-EW | V-EW       | S-NS | V-NS       | VrSrc | S-Vr  ->
 -> |-----|------|------------|------|------------|-------|------ ->
 -> | 000 | 1    | 0000001001 | 1    | 0010100000 | 0     | 1     ->

 ->  DATA (cont.)                          | CRC                      |
 -> |-----------|--------|-------|---------|--------------------------|
 -> | Vr        | RESV_B | S-Dif | Dif     | CRC                      |
```

```
-> |-----------|--------|-------|---------|--------------------------|
-> | 000001110 | 00     | 0     | 0010111 | 010110110010100001001111 |
```

There are quite a few parameters in the the velocity message. From left to rights, the number of bits indicate the following contents:

MSG Bits	N bits	Abbr	Content
33-37	5	TC	Type code
38-40	3	ST	Subtype
41	1	IC	Intent change flag
42	1	RESV_A	Reserved-A
43-45	3	NAC	Velocity uncertainty (NAC)
46	1	S-WE	East-West velocity sign
47-56	10	V-WE	East-West velocity
57	1	S-NS	North-South velocity sign
58-67	10	V-NS	North-South velocity
68	1	VrSrc	Vertical rate source
69	1	S-Vr	Vertical rate sign
70-78	9	Vr	Vertical rate
79-80	2	RESV_B	Reserved-B
81	1	S-Dif	Diff from baro alt, sign
82-88	7	Dif	Diff from baro alt

4.1.1 Horizontal Velocity

For calculating the horizontal speed and heading we need four values, East-West Velocity `V(ew)`, East-West Velocity Sign `S(ew)`, North-South Velocity `V(ns)`, North-South Velocity Sign `S(ns)`. And pay attention on the directions (signs) in the calculation.

```
S-NS:
    1 -> flying North to South
    0 -> flying South to North
S-EW:
    1 -> flying East to West
    0 -> flying West to East
```

In mathematical representation, the Speed (v) and heading (h) can be computed as following:

$$V(we) = \begin{cases} -1 * [V(ew) - 1] & \text{if } (s(ew) = 1) \\ V(ew) - 1 & \text{if } (s(ew) = 0) \end{cases}$$

$$V(sn) = \begin{cases} -1 * [V(ns) - 1] & \text{if } (s(ns) = 1) \\ V(ns) - 1 & \text{if } (s(ns) = 0) \end{cases}$$

$$v = \sqrt{V_{we}^2 + V_{sn}^2}$$

$$h = arctan(\frac{V_{we}}{V_{sn}}) * \frac{360}{2\pi} \quad \text{(deg)}$$

In case of an negative value here, we will simply add 360 degrees.

$$h = h + 360 \quad (\text{if } h < 0)$$

So, now we have the speed and heading of our example:

```
V-EW: 0000001001 -> 9
S-EW: 1
V-NS: 0010100000 -> 160
S-NS: 1

V(we) = - (9 - 1) = -8
V(sn) = - (160 - 1) = -159

v = 159.20 (kn)
h = 182.88 (deg)
```

4.1.2 Vertical Rate

The direction of vertical movement of aircraft can be read from `S(Vr)` field, in message bit-69:

```
0 -> UP
1 -> Down
```

The actual vertical rate `Vr` is the value of bits 70-78, minus 1, and then multiplied by 64 in **feet/minute** (ft/min). In our example:

```
Vr-bits: 000001110 = 14
Vr: (14 - 1) x 64 => 832 fpm
S-Vr: 0 => Down / Descending
```

So we see a descending aircraft at 832 ft/min rate of descend.

The Vertical Rate Source (VrSrc) field determine whether if it is a measurement in barometric pressure altitude or geometric altitude:

```
0 ->  Baro-pressure altitude change rate
1 ->  Geometric altitude change rate
```

4.2 Decoding message subtype 3 or 4

Subtype 3 (subsonic) or 4 (supersonic), are broadcast when ground speed information are NOT available, while airspeed is available. Subtype 3 or 4 messages are rare. As stated previously, we only received about 0.3% of those messages from all the velocity reports. However, the information contains airspeed of aircraft, which can be an interesting parameter in some researches. The structure of the message is similar to previous one. Let’s take a close look at an example for decoding here.

Note: The following decoding only apply to Subtype 3 (subsonic).

```
Message: 8DA05F219B06B6AF189400CBC33F

|    | ICAO24 |      DATA       |  CRC   |
|----|--------|-----------------|--------|
| 8D | A05F21 | 9B06B6AF189400 | CBC33F |

Convert into binary:

| HEADER                                | DATA
|-------|-----|--------------------------|-------|-----|----|--------
| DF    | CA  | ICAO24 ADDRESS           | TC    | ST  | IC | RESV_A
|-------|-----|--------------------------|-------|-----|----|--------
```

```
| 10001 | 101 | 101000000101111100100001 | 10011 | 011 | 0  | 0

  ->   Data (cont.)                                              ->
  -> |-----|------|------------|------|------------|-------|------ ->
  -> | NAC | H-s  | Hdg        | AS-t | AS         | VrSrc | S-Vr  ->
  -> |-----|------|------------|------|------------|-------|------ ->
  -> | 000 | 1    | 1010110110 | 1    | 0101111000 | 1     | 1     ->

  ->   Data (cont.)                        | CRC                      |
  -> |-----------|--------|-------|---------|--------------------------|
  -> | Vr        | RESV_B | S-Dif | Dif     | CRC                      |
  -> |-----------|--------|-------|---------|--------------------------|
  -> | 000100101 | 00     | 0     | 0000000 | 110010111100001100111111 |
```

The detail bits representations are:

MSG Bits	N bits	Abbr	Content
33-37	5	TC	Type code
38-40	3	ST	Subtype
41	1	IC	Intent change flag
42	1	RESV_A	Reserved-A
43-45	3	NAC	Velocity uncertainty (NAC)
46	1	H-s	Heading status
47-56	10	Hdg	Heading (proportion)
57	1	AS-t	Airspeed Type
58-67	10	AS	Airspeed
68	1	VrSrc	Vertical rate source
69	1	S-Vr	Vertical rate sign
70-78	9	Vr	Vertical rate
79-80	2	RESV_B	Reserved-B
81	1	S-Dif	Difference from baro alt, sign
82-88	7	Dif	Difference from baro alt

4.2.1 Heading

`H-s` makes the status of heading data:

```
0 -> heading data not available
1 -> heading data available
```

10-bits `Hdg` is the represent the proportion of the degrees of a full circle, i.e. 360 degrees. (Note: 0000000000 - 1111111111 represents 0 - 1023)

$$heading = Decimal(Hdg)/1024 * 360^o$$

in our example

```
1010110110 -> 694
heading = 694 / 1024 * 360 = 243.98 (degree)
```

4.2.2 Velocity (Airspeed)

To find out which type of the airspeed (TAS or IAS), first we need to look at the `AS-t` field:

```
0 -> Indicated Airspeed (IAS)
1 -> True Airspeed (TAS)
```

And the the speed is simply a binary to decimal conversion of `AS` bits (in knot). In our example:

```
0101111000 -> 376 knot
```

4.2.3 Vertical Rate

The vertical rate decoding remains the same as subtype 1 or 2 messages.

NIC / NAC

NIC, NAC, NUC, and SIL, those acronyms do sound confusing. They are measurement for the integrity, accuracy, or uncertainties of the position measurement from GPS unit.

- NIC - Navigation Integrity Category
- NUC - Navigation Uncertainty Category
- NAC - Navigation Accuracy Category
- SIL - Surveillance/Source Integrity Level

Two of those values are more interesting for us, `NICp` and `NACv`, represent the position integrity and velocity accuracy respectively.

Before dive into decoding and interpolation, let's introduce two parameters:

- `Rc`: Horizontal Containment Radius Limit, interpolated from `NICp` number
- `HFOM`: Horizontal Figure of Merit, interpolated from `NACv` number

5.1 NIC and Rc

Bring back the message from position decoding previously:

```
MSG: 8D40621D58C382D690C8AC2863A7

|    | ICAO24 |      DATA      |        |
|----|--------|----------------|--------|
| 8D | 40621D | 58C382D690C8AC | 2863A7 |
```

Convert both messages to binary strings:

```
| DF    | CA  | ICAO24 ADDRESS           | DATA                                    ->
|                                        | TC    | SS | SBnic | Altitude     | T  ->
|-------|-----|--------------------------|-------|----|-------|--------------|--- ->
| 10001 | 101 | 010000000110001000011101 | 01011 | 00 | 0     | 110000111000 | 0  ->

  ->  Data (cont.)                                    | CRC                      |
  -> | CPR-F | CPR Latitude      | CPR Longitude      |                          |
  -> |-------|-------------------|-------------------|--------------------------|
  -> | 0     | 10110101101001000 | 01100100010101100 | 001010000110001110100111 |
```

Not the *2 field (`bit-40`), where we have the NIC Supplement-B (S[B]) in combination with TC number, we are able to determine the NIC value.

The relation of TC, NIC, and Rc are as follow:

TC	SBnic	NIC	Rc
9	0	11	< 7.5 m
10	0	10	< 25 m
11	1	9	< 74 m
	0	8	< 0.1 NM (185 m)
12	0	7	< 0.2 NM (370 m)
13	1 *	6	< 0.3 NM (556 m)
	0		< 0.5 NM (925 m)
	1 **		< 0.6 NM (1111 m)
14	0	5	< 1.0 NM (1852 m)
15	0	4	< 2 NM (3704 m)
16	1	3	< 4 NM (7408 m)
	0	2	< 8 NM (14.8 km)
17	0	1	< 20 NM (37.0 km)
18	0	0	> 20 NM or Unknown

- * NIC Supplement-A = 0
- ** NIC Supplement-A = 1

In our example:

```
TC -> 11
SBnic -> 0

We have:
  NIC -> 8
```

So, what happened to the NIC Supplement-A and C? Those two bits are broadcast in Aircraft Operational Status Message (TC=31, see Introduction page). For Surface Position Message, you will need the combination of A and C to determine the NIC number (note: Rc values are different from Airborne Position Messages). However, with Supplement-B bit we are already able to decode the NIC and Rc for airborne positions.

5.2 NAC and HFOM

NAC is reported in the Airborne Velocity Message.

CHAPTER 6

Mode S Enhanced Surveillance (EHS)

Let's hack into the EHS messaged too! more information on aircraft air speeds.

[under editing]

For a complete Python implementation: https://github.com/junzis/pyModeS/blob/master/pyModeS/ehs.py

The Mode-S Enhanced Surveillance (EHS) provides air traffic controller more information that what is included in the ADS-B (a.k.a Mode-S Elementary Surveillance). It responds to ATC Secondary Surveillance Radar, and broadcast specific parameters non-independently. Hence it is only available in the area where ATC presents.

There are quite a few very interesting data contained within various types of the EHS messages. Such as: airspeeds (IAS, TAS, Mach), roll angles, track angles, track angle rates, selected altitude, magnetic heading, vertical rate, etc..

There are a few challenges to decode those information:

- Which aircraft does one message come from?
- What is the type of one message (a.k.a. which BDS code) most likely to be?
- How confident is the information that has been decoded?

6.1 Downlink Format and message structure

DF 20 and DF 21 are used for downlink messages.

The same as ADS-B, in all Mode-S messages, the first 5 bit contains the Downlink Format. The same identification process can be used for discover EHS messages. So the EHS messages starting bits are:

```
DF20 - 10100
DF21 - 10101
```

The message is structured as following, where the digit represents the number of binary digits:

```
+--------+--------+--------+--------+-------------+------------------+------------+
|  DF 5  |  FS 3  |  DR 5  |  UM 6  |    AC 13    |       MB 56      |  AP/DP 24  |
+-----------------+--------+---------------------------------------+------------+
|    <---------+       32 bits      +-------->     |


DF:     downlink format
FS:     flight status
DR:     downlink request
UM:     utility message
ID:     identity
```

```
MB:      message, Comm-B
AP/DP:   address/parity or data/parity
```

Except the DF, the first 32 bits does not contain useful information for decode the message. The exact definitions can be found in ICAO annex 10 (Aeronatutical Telecommunications).

6.2 Parity and ICAO address recovery

Unlike ADS-B, the ICAO address is not broadcast along with the EHS messages. One would have to "decode" the ICAO address before decoding the message.

ICAO is hidden in the message and checksum. For DF 4, 5, 20, 21, message parity field is produced by XOR ICAO bits with the checksum of data bits from CRC, as following. So to recover the ICAO bits, simply reverse XOR process as following:

```
+--------------------------------+  +------------------+
|   DATA FIELD (32 OR 88 BIT)    |  |   PARITY BITS    |
+---------------+----------------+  +------------------+
                |
                |                          XOR
                v
+---------------+-----------+       +------------------+
|          ENCODER          |  +--> | CHECKSUM (24BIT) |
+---------------------------+       +------------------+
                                            ||
                                    +------------------+
                                    |   ICAO (24BIT)   |
                                    +------------------+
```

Take an example message:

```
Message:      A0001838CA380031440000F24177

Data:         A0001838CA380031440000
Parity:       F24177

Encode data:  CE2CA7

ICAO:    [F24177] XOR [CE2CA7] => [3C6DD0]
```

For the implementation of CRC encoder, refer to the pyModeS libaray `pyModeS.util.crc(msg, encode=True)`

6.3 BDS (Comm-B Data Selector)

In simply words, BDS is a number (usually a 2-digit hexadecimal) that defines the type of message we are looking at. Both ADS-B messages and other types of Mods-S messages are all assigned their distinctive BDS number. However, it is **no where** to be found in the messages.

When SSR interrogates aircraft, a BDS code is included in request message (Uplink Format - UF 4, 5, 20, or 21). This BDS code are then used by the aircraft transponder to register the type of message to be sent. But when the downlink message is transmitted, its BDS code is not included in the message (because the SSR knows what kind message it requested). Good new for them, but challenges for us.

Here are some BDS codes that we are interested, where additional parameters about aircraft can be found:

```
BDS 2,0   Aircraft identification
BDS 2,1   Aircraft and airline registration markings
BDS 4,0   Selected vertical intention
BDS 4,4   Meteorological routine air report
BDS 5,0   Track and turn report
BDS 6,0   Heading and speed report
```

6.4 BDS 2,0 (Aircraft identification)

Similar to ADS-B aircraft identification message, the callsign of aircraft can be decode in the same way. For the 56 MB (message, Comm-B) field, information decodes as follows:

```
+--------------+---------+---------+---------+---------+---------+---------+---------+---------+
|  BDS2,0  (8) |  C1 (6) |  C2 (6) |  C3 (6) |  C4 (6) |  C5 (6) |  C6 (6) |  C7 (6) |  C8 (6) |
+--------------+---------+---------+---------+---------+---------+---------+---------+---------+
  0010 0000      6 bits
```

Here, 8 bits are 0010 0000 (2,0 in hexadecimal) and the rest of chars are 6 bits each. To decode the chars, the same char map as ADS-B is used:

```
'#ABCDEFGHIJKLMNOPQRSTUVWXYZ#####_###############0123456789######'
```

Example:

```
MSG:  A000083E202CC371C31DE0AA1CCF
DATA:         202CC371C31DE0

BIN:  0010 0000 001011 001100 001101 110001 110000 110001 110111 100000
HEX:     2    0
DEC:             11     12     13     49     48     49     55     32
CHR:             K      L      M      1      0      1      7      _

ID:   KLM1017
```

6.5 BDS 4,0 (Selected aircraft intention)

In BDS 4,0, information such as aircraft select altitude and barometric pressire settings are given. The 56-bit MB filed is structure as following:

```
 FIELD                                   START  N-BITS
                                         (END)
+---------------------------------------+------+------+
| Status                                |  1   |  1   |
+---------------------------------------+------+------+
| MCP/FCU selected altitude             |  2   |  12  |   **
|                                       |      |      |
| range = [0, 65520] ft                 |      |      |
|                                       |      |      |
| LSB: 16 ft                            |  13  |      |
+---------------------------------------+------+------+
| Status                                |  14  |  1   |
+---------------------------------------+------+------+
| FMS selected altitude                 |  15  |  12  |   **
|                                       |      |      |
```

```
| range = [0, 65520] ft                 |      |      |
|                                       |      |      |
| LSB: 16 ft                            |  26  |      |
+---------------------------------------+------+------+
| Status                                |  27  |  1   |
+---------------------------------------+------+------+
| Barometric pressure setting           |  28  |  12  |   **
|   -> Note: actual value minus 800     |      |      |
|                                       |      |      |
| range = [0, 410] mb                   |      |      |
|                                       |      |      |
| LSB: 0.1 mb                           |  39  |      |
+---------------------------------------+------+------+
| Reserved                              |  40  |  8   |
|   -> set to ZEROS                     |      |      |
|                                       |  47  |      |
+---------------------------------------+------+------+
| Status                                |  48  |  1   |
|   -> next 3 fileds                    |      |      |
+---------------------------------------+------+------+
| Mode: VNAV                            |  49  |  1   |
+---------------------------------------+------+------+
| Mode: Alt hold                        |  50  |  1   |
+---------------------------------------+------+------+
| Mode: Appraoch                        |  51  |  1   |
+---------------------------------------+------+------+
| Reserved                              |  52  |  2   |
|   -> set to ZEROS                     |  53  |      |
+---------------------------------------+------+------+
| Status                                |  54  |  1   |
+---------------------------------------+------+------+
| Target alt source                     |  55  |  2   |
|   -> 00: Unkown                       |      |      |
|   -> 01: Aircraft altitude            |      |      |
|   -> 10: FCU/MCP selected altitude    |      |      |
|   -> 11: FMS seleted altitude         |  56  |      |
+---------------------------------------+------+------+
```

An example:

```
MSG:  A000029C85E42F313000000007047D3
MB:           85E42F31300000

---------------------------------------------------------------------------------
MB BIN:   1 000010111100 1 000010111100 1 100010011000 00000000 0 0 0 0 00 0 00
---------------------------------------------------------------------------------
STATUS:   1
MCP:        188 (x16)
---------------------------------------------------------------------------------
STATUS:                  1
FMS:                       188 (x16)
---------------------------------------------------------------------------------
STATUS:                                 1
BARO:                                     2200 (x0.1 + 800)
---------------------------------------------------------------------------------
FINAL:      3008 ft        3008 ft        1020 mb
---------------------------------------------------------------------------------
```

6.6 BDS 4,4 (Meteorological routine air report)

CHAPTER 7

Apendix

7.1 Documents, code, and data

This guide document is shared on GitHub and ReadTheDoc. Please feel free to help us improving it.

Links to this guide document:

- (GitHub) https://github.com/junzis/pyModeS
- (Document) http://adsb-decode-guide.readthedocs.org/

You can download from GitHub the python decoder, as well as some data samples we collected:

- https://github.com/junzis/py-adsb-decoder

7.2 Contact

Feel free to drop me a messages at: **j.sun-1[at]tudelft.nl**

7.3 About us

We are a group at TuDelft working on aircraft operations and controls.

- Junzi Sun, PhD Student
- Jacco Hoekstra, Prof.dr.ir
- Joost EllerBroek, Dr.ir

7.4 References

Some good source of documents:

- RTCA/EUROCAE: Minimum Operational Performance Standards for 1090 MHz Extended Squitter Automatic Dependent Surveillance – Broadcast (ADS-B) and Traffic Information Services – Broadcast (TIS-B)
- ICAO: Technical Provisions for Mode S Services and Extended Squitter
- ICAO ADS-B Guide

- Dump1090 Project
- A Very Simple ADSB Receiver,