What is FIS-B 978?

FIS-B 978 (‘fisb-978’) is a set of programs that demodulates and error corrects FIS-B (Flight Information System - Broadcast) and ADS-B (Automatic Dependent Surveillance - Broadcast) messages transmitted on 978 Mhz, mostly in the United States.

It performs the same function as FlightAware’s 978, but is spread out over several programs that are piped together. It’s main advantage over FlightAware’s 978 and other ‘dump-978’ clones is that for moderate to weaker signals it can provide much higher packet decode rates, since it takes into account that samples at the Nyquist sampling rate may not be optimal and will shift them to more optimal levels.

For strong signals, ‘dump-978’ and ‘fisb-978’ will perform similarly.

‘fisb-978’ is composed of two main parts and one optional part:

- **demod_978** is a C language program which takes raw data from an SDR program at 978 Mhz and sample rate of 2.083334 Mhz (with 16-bit complex integer output (CS16)) and will demodulate the data, detect sync words, and output packets along with attributes of the packet to standard output. Typically, ‘ec-978.py’ will read this data.
- **ec_978.py** takes the data from ‘demod_978’ and will use Reed-Solomon error correction to produce an output packet as hex bytes (similar to ‘dump-978’). It uses a number of techniques to provide a higher packet decode rate. The output is sent to standard output, where is can be used as is, or processed by ‘server_978’.
- **server_978.py** takes the output from ‘ec_978.py’ and will provide a TCP server where clients can connect and receive the data. It is optional if you don’t want to serve the data via TCP.

The program is broken into parts for modularity and speed. C is very fast for searching through the input stream and finding sync packets (numpy is great at the demodulation part, but horrid at searching for sync codes). Python using numpy (and a Reed-Solomon library linked to a C library), is very quick at error correcting the packets and manipulating them to provide more optimum bit levels for better error correction rates.

A few things to consider before using:

- Since ‘demod_978.c’ uses type-punning, *a compiler that is friendly to that is required*. GCC is such a compiler. *All code expects little-endian byte order*. This will work on most common architectures in use today. If needed, big-endian can be added as a future feature.
- ‘server_978.py’ uses a `select()` statement using both sockets and file I/O. As such, *this will usually not work on Windows* (it should work fine if you are using the ‘Linux Subsystem for Linux’).
- ‘fisb-978’ may improve the number of packets you can error correct, but it should be low on your list of improvements that actually matter. Nothing beats a good antenna, and a good

978MHz LNA and filter. Also, better quality radios are always a plus.

FIS-B 978 is not designed, nor intended, to be used for any purpose other than fun exploration.

Warning: FIS-B 978 is **NOT** intended for actual flight use, nor to be used as a component of a system for any flight related operations (flight planning, etc). It is strictly a fun hobby program with many bugs. It is based loosely on draft standards, not actual standards, and has not undergone any formal testing. **DO NOT USE OTHER THAN FOR FUN– NOT FLIGHT!!**

Getting things running

There is not much setup to do. For `decode_978` just cd to the cloned directory and type `make`. You should see something like:

```
$ make
gcc -c -o demod_978.o demod_978.c -I. -O3 -Wall -funroll-loops
gcc -o demod_978 demod_978.o -I. -O3 -Wall -funroll-loops
```

There is nothing to do for `server_978.py`. It should work out of the box.

`ec_978.py` requires numpy, a C library, and the Reed-Solomon interface to the C library.

There are lots of ways to install numpy. Go to numpy.org if you need help. If you have `pip3`, just type:

```
pip3 install numpy
```

For Reed-Solomon, visit https://pypi.org/project/pyreedsolomon/. The best way to install this is **not** with pip3. Make a clone as follows:

```
# Do the following line only if you have it already installed
# or think you might.
sudo pip3 uninstall pyreedsolomon

git clone --recursive  https://github.com/epeters13/pyreedsolomon.git
cd pyreedsolomon
sudo python3 setup.py install
sudo ldconfig
```

`ec_978.py` should now have all the prerequisites installed.

Sample usage script

Here is an example of a script I use for normal decoding (based on the *SDRplay RSP1A*). It can be found in `scripts/sdrplay-demod`:

```
# Script used for using rx_sdr with sdrplay to capture raw data
# and send them to demod_978, ec_978, and server_978.
#
# You can change the first portion to reflect your SDR radio, but use the
# following settings:
#   frequency:          978 MHz
#   sample rate:        2083334
#   output type:        Complex 16 bit integer (usually called CS16)
#
# If you just want to see decoded output printed, leave off '| ./server_97
#
rx_sdr -d driver=sdrplay -t biasT_ctrl=true,rfgain_sel=1 -g 25 -F CS16 \
 -f 978000000 -s 2083334 - | ./demod_978 | ./ec_978.py | ./server_978.py
```

You will need to substitute your SDR program and settings. The settings must include the frequency, sample rate, and output type (*CS16*) shown above. You just pipe the raw output through `demod` and `ec_978.py`. This will give you the decoded hex strings for FIS-B and ADS-B. To serve it remotely, pipe that output to `server_978.py`.

For an 8-bit *RTLSDR* or *RadarBox 978 FlightStick*, the above command can be modified as (also found in `scripts/rtlsdr-demod`):

```
rx_sdr -d driver=rtlsdr -F CS16 -g 40 \
 -f 978000000 -s 2083334 - | ./demod_978  | ./ec_978.py | ./server_978.py
```

You won't get the performance out of an 8-bit SDR as you will with something with a higher bit ADC. Also, for all radios, a good filtered preamp is the next most important thing after a good antenna. I have found the Uputronics 978MHz UAT filtered preamp to be an excellent performer. In the United States you can get them at AIRSPY.us (disclosure: I am not sponsored by any product, nor do I have affiliate links).

Explanation of program output

Output from `ec_978.py` will show three types of packets: FIS-B, ADS-B long, and ADS-B short. A FIS-B packet will look like:

```
+38f18185534cb2c01a0000fc308083e0c1070517040314530423220706 0f060514
03044b041b2a070e07050c0b0c6302032a0e0f0614030413042b041b321a0000fc3
0808443331801022112010 2012a23040b0518090a0912033a231c030910010a095a
1b04031c09100100094a010a1b04031c1a0000fc3080846d6040010402090305040
3020b040e050302900103120b0d060403029801020302140d0b0298010b02040504
0b09281a0000fc3080840067330c130a1b05061d0413040203022304030c4305041
```

```
504031c83020403042d0c3b0405242b1c050e0d140314031a0000fc30808436e508
190a13040304051c0b1c0b021b08190a1b340b050b1a1b0809000112130c5b110a0
30809000a230c5b1a03021a0000fc308083e28303050e0706052c03042b044b040d
0e050c031c4b0423020b0406050f05344b041b0a03020d060f051405047b221a000
0fc308083ed0d0211320902010a01580100020902015a013001000100110201 0a01
52091811080100110201620920110001081100000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000
;rs=0/01:02:01:01:00:03/007;ss=3.76;t=1639224615.144
```

Note that in reality, all packets are a single line. The examples are broken up for clarity. The '**+**' at the beginning indicates a FIS-B packet. ADS-B packets start with '**-**'.

The actual message is the set of hex characters that follow.

After that, separated by ';', are three items:

- `rs=` is about Reed Solomon error correction and how many sync bits in the sync preamble didn't match.

  In this case `rs=0/` means that all bits in the sync word didn't match. This number will be from 0 to 4. Each FIS-B and ADS-B packet is preceded by a 36-bit sync word. 32 of those bits have to be correct for the sync word to be considered a match.

  `01:02:01:01:00:03` represents the number of Reed-Solomon errors corrected in each FIS-B block. FIS-B messages are made up of six parts, each with their own set of error correction bits. Each block can have up to 10 errors before it is considered uncorrectable. If a packet has more than 10 errors, the number of errors will be listed as `99`. You will see this when printing errors and very commonly in the form of: `04:99:99:99:99:99`. This is what you will see with an empty packet. `ec_978.py` looked at the packet and determined, by only looking at block 0, that this packet is empty. In that case it doesn't even look at the other blocks.

  The last component of the `rs=` string in our example (`/007`) represents the total number of times we had to call the Reed-Solomon error corrector. For a normal packet (one that we don't short circuit decoding because it is an empty packet), the smallest number this can be is 6, or one for each block. If a block needed to try additional shift values to decode, the number will be higher than 6. If we were able to detect a packet that ended early, the number of Reed-Solomon decodes can be less than 6. You will occasionally see numbers over 500. This means that we could not obtain a result with the original packet, and needed to shift one bit over to make a new packet and try the process again. Subtracting 500 from the number gives you the number of Reed-Solomon attempts for the new packet.

- `ss=3.76` is the signal strength. It has no units and isn't related to anything. It is just a relative indication of the signal strength of the sync word. When data is read by `demod_978`, the demodulated data is a set of signed integers. The program keeps a running average of the absolute value of the last 72 bits (i.e. a 36 bit sync word with a set of bits in between because we are sampling at two samples per bit). In order for the program

to even check for sync, this value must be larger than some threshold. By default this is 0.9, but it can be changed with the `-l` argument in `demod_978`. If the running average is above 0.9, we will attempt to match a sync word, and if we do, this value is recorded as the signal strength.

- `t=1639224615.144` is the time in UTC seconds past Linux EPOCH with the number of milliseconds attached. This value is calculated as follows: `demod_978` records the time every time it reads from the disk (that usually happens 10 times a second). Whenever we decode a sync word, we calculate the time by adding 0.48 microseconds per sample for each bit from the time the disk was read, minus 0.48 * 72 bits so the time is reflected back to when the sync word was started.

Failed FIS-B and ADS-B messages will look something like:

```
#FAILED-FIS-B 1/99:08:99:08:10:99 ss=1.66 t=1639224737.098
1639224737.098328.F.01663540.1

#FAILED-ADS-B 2/99 ss=1.66 t=1639229048.478
1639229048.478000.A.01658613.2
```

Again, each will be a single line. The `#` in the front indicates A comment. `fisb-decode` will ignore this. The data on the line is similar to what we just discussed. The last portion of the line is the attribute string that `demod_978` passed to `ec_978.py` and is used as part of the filename in case errors are being saved for further study. In order to get failed error messages, you must supply the `--ff` (FIS-B) or `--fa` (ADS-B) arguments to `ec_978.py`.

A long ADS-B message will look like:

```
-0b28c0ee3879938546c605d6100600c01105eded2ded2d0ad2740300000000000000
;rs=0/1/002;ss=3.29;t=1639226996.293
```

It starts with a dash. The format is similar to FIS-B except the `rs=0/1/002` reflects 0 sync code errors (as in FIS-B), 1 Reed-Solomon error was corrected, and two attempts at Reed-Solomon error detection were required. There is only one Reed-Solomon block in ADS-B messages, so you will only see a single number. Not six as in FIS-B.

A short ADS-B message is just like a long one, but shorter!

```
-00a97c0d3868cd856ac6076910ac2c602800;rs=1/2/005;ss=3.56;t=1639228834.048
```

Theory of operation

demod_978

`demod_978` receives raw FSK data from an SDR radio at the Nyquist limit of twice the bit rate. With a bit rate of 1.041667 Mhz, the sample rate is 2.083334 Mhz. Each sample is a complex IQ value with the real and complex parts being 16 bit integers.

Demodulation is accomplished using the formula:

```
         (I[n-2] * Q[n]) - (I[n] * Q[n-2])
sample = ---------------------------------
                I^2[n] + Q^2[n]

where n is the current sample and n-2 is the sample 2 samples before
      the current sample.
```

This formula is the equivalent of taking the arctangent and differentiating it for time. It's simple and fast and doesn't require any arctangent tables or arctangent calculations. This technique is from Richard Lyons in *Understanding Digital Signal Processing, Second Edition*. You can find an explanation of this technique here.

If you were taking more samples per second, you would want something other than `n-2`. For our bit rate, `2` produces the best results.

The denominator of this equation is for scaling. For our calculations we ignore it. Empirically, you will get slightly more decodes with scaling, but none that can't be corrected in `ec_978.py`.

After demodulating the signal we need to match the sync codes. The sync codes are 36 bit codes and we need to match 32 (or more) out of the 36 bits (32 isn't a magic number– it just represents a reasonable value between too many and too few sync code matches) The sync code for FIS-B is `0x153225b1d` and `0xeacdda4e2` for ADS-B. They are actually inverses of each other, so you could calculate the sync for one, and you would know if the other matched too. Unfortunately, this technique is much slower than using Brian Kernighan's algorithm for calculating 1 bits separately for each sync code. The sync candidate is XOR'ed with the sync word and the one bits counted. If you get more than 4 ones, you can stop– it didn't match.

One quick note: searching for the sync word is very slow using numpy, and is the reason we have a separate program in C. Numpy is quick for all other operations including demodulation.

Before we even try to match a sync word, we take the additional step of maintaining a 72 bit running total of the absolute values of the samples. Sample values when signal is present are much higher than when only noise is present. In order to even attempt to match a sync word, we must have a value greater than some number. In our case, the default (empirically derived) is 900000. To keep things simpler, all values are presented to the user in millionths. So 900000 is denoted as 0.9. This value probably does not apply to other SDR setups or amplifications. The `demod_978` program will let the user set this with the `-l` argument. It is probably best to set this to `-l 0.0` and look at the results to set the best level. An improvement would be to sample for lowest noise values and use that to set a cutoff.

Once we have matched a sync code, we will send 8835 32-bit signed integers for a FIS-B packet and 771 32-bit signed integers for an ADS-B packet. These numbers include all the bits required for the message, plus the bits in between the sample bits, plus one extra sample at the beginning and two extra samples at the end. This will allow `ec_978.py` to try some weighted averages to find better sampling points.

The packets are preceded by a 30 character string which tells `ec_978` information about the packet to follow. This includes the type (FIS-B or ADS-B), signal strength, time the packet arrived, and number of mismatched sync bits. The `demod_978.c` documentation contains details on the format of this string. The string is important so that `ec_978.py` will know how many bytes to read for the packet.

We send a single length packet for both ADS-B short and ADS-B long packets. Technically, we could guess at the type since the first five bits of an ADS-B short packet are zero, but we haven't done error correction yet, so we might be wrong.

What we don't do, and might be a future enhancement, is that once we match a sync code, we send the data, and then start looking for the next sync after the end of the packet, not with the next bits. For FIS-B, this isn't an issue, but might be for ADS-B. One case that is quite common is that one set of bits may match a sync code, and the one right next to it (i.e. the 'other' sample in 'every other sample') will match too. We take care of this by sending enough bits in the packet so `ec_978.py` can check the current sample, as well as the sample right after it.

The last thing `demod_978.py` does is to send the 30 character string and packet.

A couple of caveats. This program is written for speed. It uses type-punning to convert between bytes and various size integers. It needs a compiler that allows this, such as GCC. It also assumes little-endian architectures.

ec_978.py

`ec_978.py` receives the fixed length string and reads the appropriate number of bytes for the actual packet. It then turns this into a numpy array. This array is processed slightly differently for ADS-B and FIS-B because FIS-B packets contain six different error correction blocks. I will explain the process for FIS-B packets because ADS-B packets are just a subset.

We take each block of a FIS-B message and try to apply Reed-Solomon error correction to it. We do this by taking the packet and turning it into three packets. One packet is the original packet, one is the set of bits before each bit of the current packet, and one is the set of bits after the current packet. The routine that does this also handles deinterleaving the blocks. This applies only to FIS-B. FIS-B packets are interleaved to help minimize the effect of burst errors.

The first task is to try to decode the original packet without any help from the bits before or after. This works most of the time. But if that doesn't work, we switch to method two.

If you are sampling at many times the bit rate, there is a good chance that one of your samples is close to optimum. When you are sampling at the Nyquist limit of 2 samples per bit at twice the bit frequency, there is a good chance that neither of your samples are near optimum. Both are probably some shade of ‘meh’.

If you have a very strong signal, that means that the one and zero points are widely separated and almost any sampling point will work.

If you don’t have a strong signal, the one and zero points are closer together and you will run into problems if the sampling points are far off from optimum.

What we do is to use the bits-before and the bits-after to create essentially a weighted average. We do this for the entire packet and then try to error correct again. For example, assume we are using the bits-after at a level of 90%. This means we take each bit in bit-after, multiply it by 0.9, and add it to the corresponding bit in the original sample then divide by two. After we do this for all bits, we try to error correct again. At any given time, we are using either bits-before or bits-after and a fixed percentage to calculate a new packet. We are essentially nudging the sample bits toward either bits-before or bits-after to find a better sampling point.

After lots of experiments, a table was derived ordering the percentages and whether they are bits-before or bits-after in an order which will decode a packet the quickest. For FIS-B, if we decode a packet at a particular shift level, we will start with that shift level for the next block.

There are other techniques we could do, but are not currently using (mostly because what we do now works and is fast enough) such as using zero crossing to estimate a guess on the percentage to use.

It is very uncommon that it takes more than two or three attempts to decode a packet if the packet is going to decode at all. But some decode down in the weeds, so we try anyway. Given two minutes worth of data to decode from a file, a run may take 5 seconds or so, so we are not approaching any CPU limits.

If we didn’t decode the block, we repeat the process by using the next set of bits. In other words, bits-after becomes the current bits, the current bits become the bits-before, and the bits after the original bits-after becomes the new bits-after. This will result in a small number of additional decodes.

Anytime we decode block 0, we check it to see if it is an empty packet, or it ends somewhere in block 0. If it does, we are done and can just fill all the other blocks with zero.

If we fail decoding, we call this same routine to check for early packet ending, but for blocks beyond block 0. This checking doesn’t apply to ADS-B.

If we decode all six blocks we create an output string and send it to standard output.

If the packet doesn't decode, we can send an error message to standard output if the user wants us to. We also have the option of saving errored out packets to a directory for further study.

server_978.py

Nothing fancy here. Just takes standard input and sends it to any connected socket. It is send only. The only wrinkle is that we use `select()` not only for sockets, but also for standard input. This might not work on native Windows, but most likely would work with *Windows Subsystem for Linux*.

Individual program usage

demod_978

`demod_978` reads raw SDR I/Q data from standard input at frequency of 978Mhz. Assumes samples of 2 samples per data bit or 2,083,334 samples/sec. Samples should be complex int 16 (CS16).

SDR samples are demodulated into packets of signed 32-bit integers. Attributes of each packet (whether FIS-B or ADS-B, arrival time, and signal strength) are stored in a string and sent to standard output. This is followed up with the actual packet data as signed 32-bit integers. These values are then received and processed by the standard input of `ec_978.py`.

The decoding is divided between two programs since searching for sync words in a large amount of data isn't what numpy is best at, but C is amazingly fast at this. Likewise, python, using numpy, is super fast at decoding data packets and trying various approaches to decode data that is at the Nyquist limit.

```
usage: <sdr-program 2083334 CS16> | demod_978 <arguments>

Read samples from SDR and capture FIS-B and ADS-B packets.

Arguments:

 -a
    Process ADS-B packets only. If neither -a or -f are specified,
    both ADS-B and FIS-B are processed. You cannot specify both
    -a and -f at the same time.

 -f
    Process FIS-B packets only. If neither -a or -f are specified,
    both ADS-B and FIS-B are processed. You cannot specify both
    -a and -f at the same time.

 -l <float>
    Set the noise cutoff level. Data samples are stronger than the
```

```
    baseline noise level. This sets the minimum value required that
    demod will attempt to process a packet. The default is 0.9. The
    purpose of this is to decrease the number of false packets that
    are extracted from noise. If you are not sure if you are capturing
    all valid packets, set this to 0.0. The default value has no units,
    it was determined by evaluation of empirical data. It may vary
    based on SDR radio used, or SDR program used. Optional.

 -x
    If you are testing by feeding a file of already captured raw data
    in a file, set this argument. 'demod_978' attempts to get the correct
    timing when a packet arrived, so will figure out how many
    microseconds past the time the sample was read to provide a correct
    value. This works fine for real-time data, but when dumping a file,
    it won't work. The -x argument will make sure the times on the packet
    filename will sort correctly and make sense. Optional.
```

ec_978.py

```
usage: ec_978.py [-h] [--ff] [--fa] [--se SE] [--re RE]

ec_978.py: Error correct FIS-B and ADS-B demodulated data from 'demod_978

Accepts FIS-B and ADS-B data supplied by 'demod_978' and send any output t
standard output. By default will not produce any error messages for
bad packets.

The '--se' argument requires a directory where errors will be stored.
This directory should be used by the '--re' argument in a future run to re
the errors. When the '--se' argument is given, you need to supply either
'--ff' or both to indicate the type of error(s) you wish to save.

When errors are reprocessed with '--re', the '--ff' and '--fa' arguments
are automatically set, and any '--se' argument is ignored.

optional arguments:
 -h, --help  show this help message and exit
 --ff        Print failed FIS-B packet information as a comment.
 --fa        Print failed ADS-B packet information as a comment.
 --se SE     Directory to save failed error corrections.
 --re RE     Directory to reprocess errors.
```

server_978.py

`server_978.py` justs takes the standard output from `ec_978.py` and serves it to whoever connects. It is run as

```
./server.py --port 3333
```

The default port is `3333` so you can omit the port argument if that is one you want.

Building Documentation

If you want to build the documentation, install sphinx. On Ubuntu 20.04 you can do this with:

```
sudo apt install python3-sphinx
```

Next, install the Python requirements from the `bin` directory as:

```
pip3 install -r ../misc/requirements-sphinx.txt
```

Next come the tricky parts. You will need to install doxygen and breathe on your system. The setup for these is system dependent, so you need to read and follow the documentation.

At a minimum, you will need to edit the file `docs/source/conf.py` and edit the line with `breathe_projects` and change the provided path to reflect where `doxygen/xml` will be on your system. This will normally be the path to where you cloned fisb-978 (i.e. `~/fisb-978/doxygen/xml`). The xml directory won't exist at first, but will be created the first time you build the documentation. There is a doxygen project file in `doxygen/fisb_978`.

Then (assuming 'fisb-978' was cloned in your home directory):

```
cd ~/fisb-978/docs
./makedocs
```

The html documentation will be found in `fisb-978/docs/build/html`. Load `index.html` in your browser to view. Sphinx is configured to link directly to the source, so this is an easy way to explore the code.