ToADS HF: A Digital Layercake That Nobody Ordered

What I learned from building a new digital mode

Ben Glick KN6UBF

7/11/2025

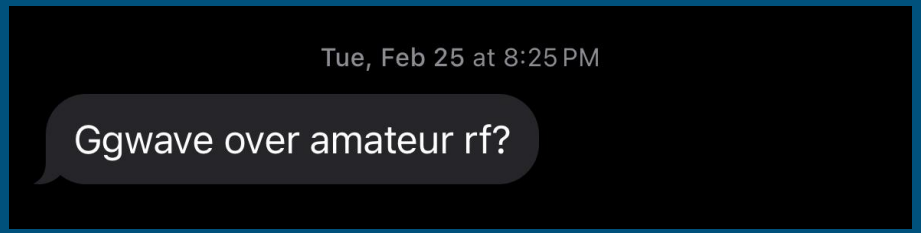
Agenda

- Background
- Text over Sound case study: GGWave
- How does ToAD work
- Demo
- Experiments
- Learnings
- Next Steps
- Conclusion

Goal: Demystify digimodes by learning how to build one from scratch

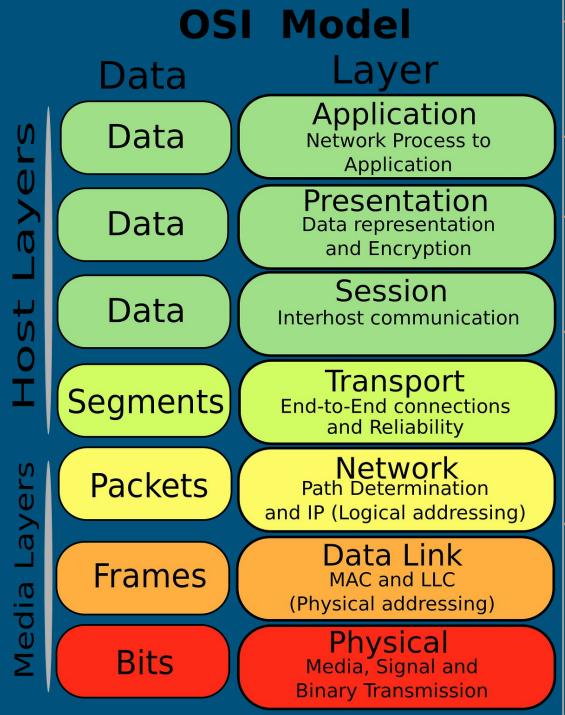
Where did this start?

- I’m an FT8/FT4 user
- I didn’t like that you have to sync externally
- I wanted to learn more about how digimodes work
- It seemed like fun
- I’d been told about ggwave
- I did exactly ZERO research about what it takes to make a good digital mode



Layering

- OSI model for computer networks
- Abstraction is the key value-add
- ToAD is concerned with 1, 2, 6

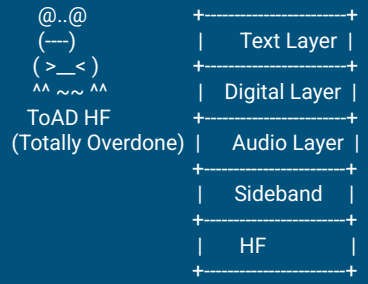


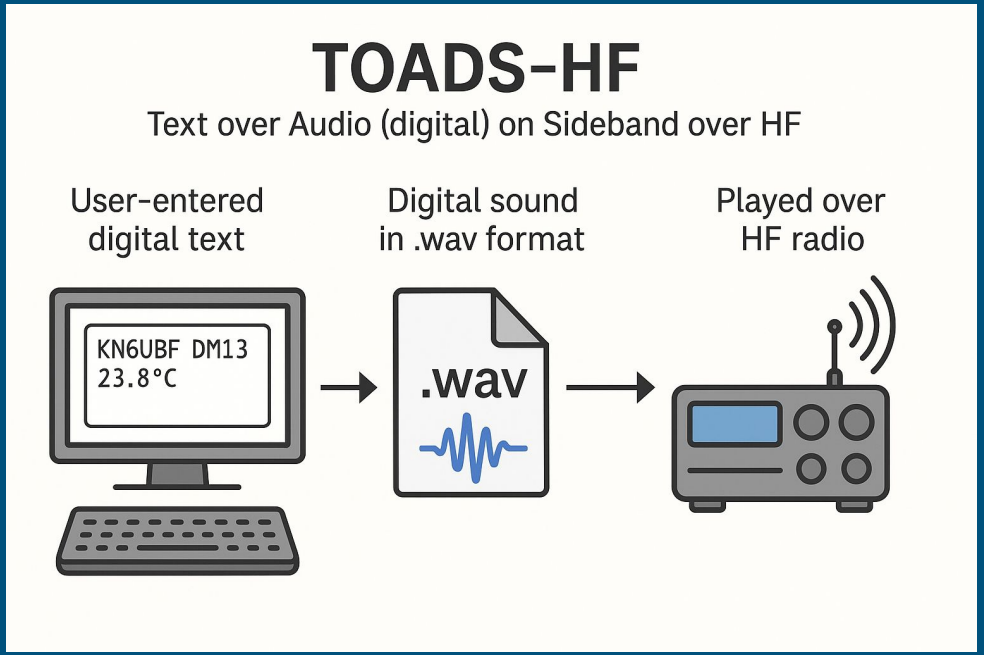
“Ham OSI” Model

Application 7	Human decides what to send “CQ KN6UBF”
Presentation 6	Encoding text into something useful
Session 5	Doesn’t really exist.
Transport 4	Ionosphere is inherently lossy
Network 3	Always done by the receiver, like original Ethernet (“Is that for me?”)
Data Link 2	Mode-specific. FT8 has ECC, ToAD has redundancy, CW has operator skill.
Physical 1	RF Carrier, audio, magic smoke on your workbench

🐸 Why's it called ToAD?

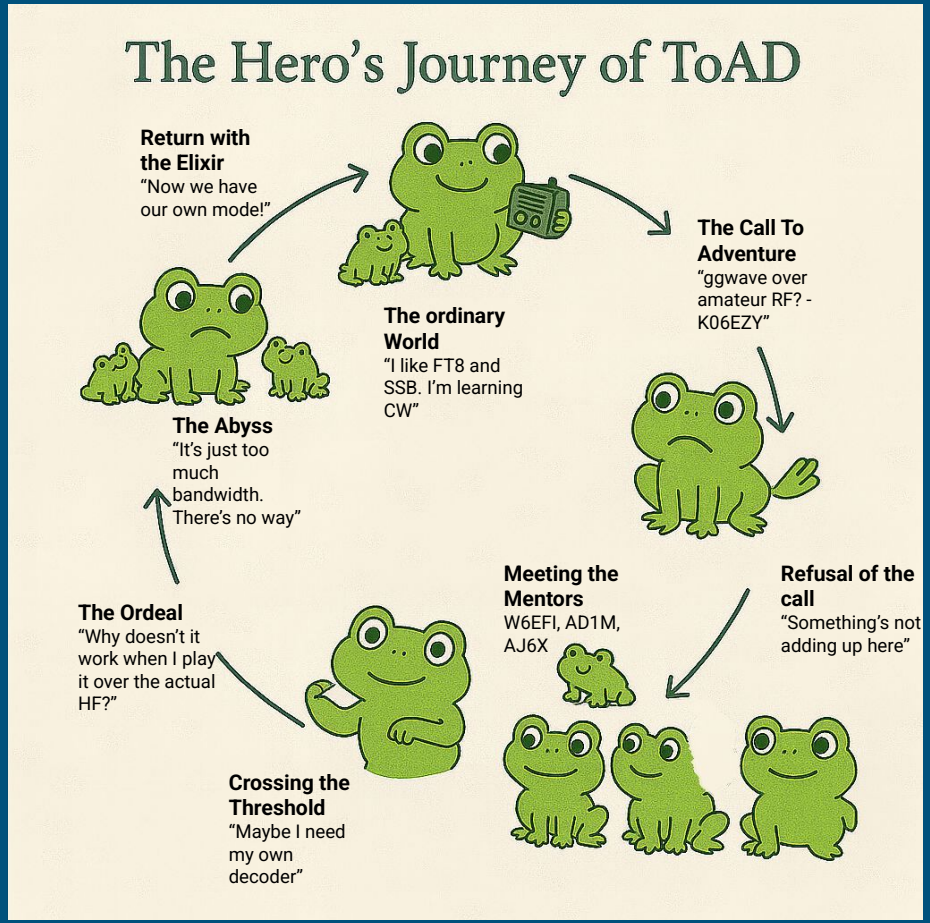
- Text over Audio (Digital) over Sideband on HF
- ToADF-VHF is possible too (ToAD over FM on VHF)





The Journey of ToADHF

- The wrong turns mattered
- They taught me what HF needs
- They led to the right design
- And it was more fun this way



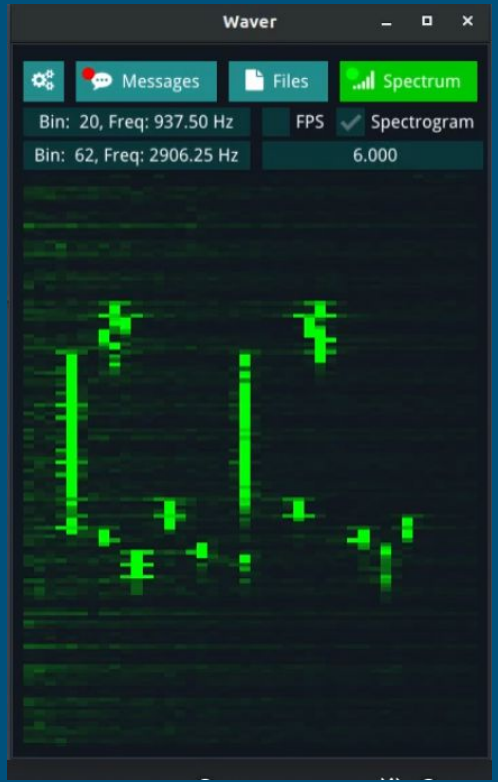
1-Slide GGWave Primer

Appeals because:

- Popular Open-source “Data over Audio” library
- Encodes text into sound
- Designed for ultrasonic/audible short-range comms over audio equipment
- Built-in error correction
- Large community

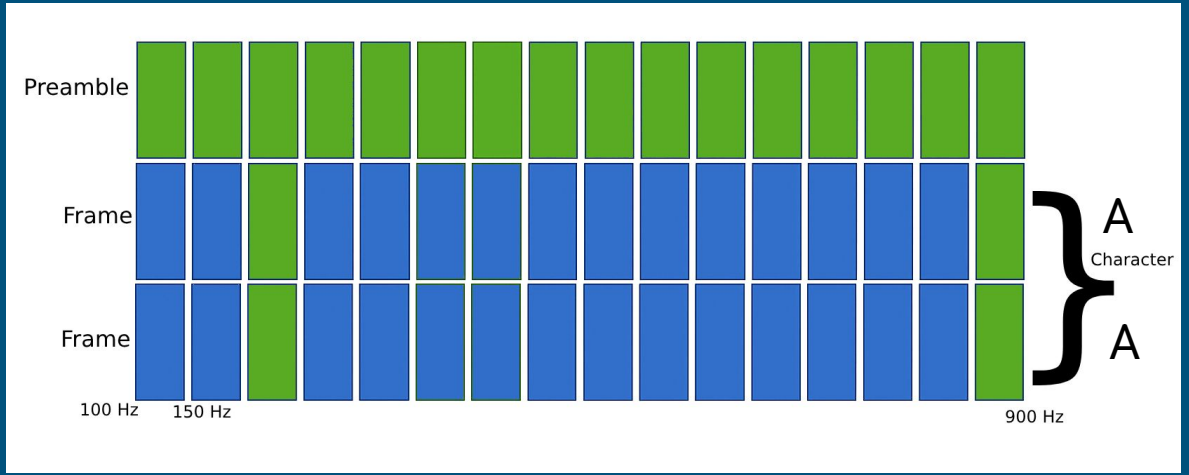
However:

- Not designed for narrow channels (+6KHz B/W)
- Reference decoder is extremely sensitive to QRM, AGC, Fading
- Not suitable especially for AM-based modes
- Not tolerant of small frequency buckets

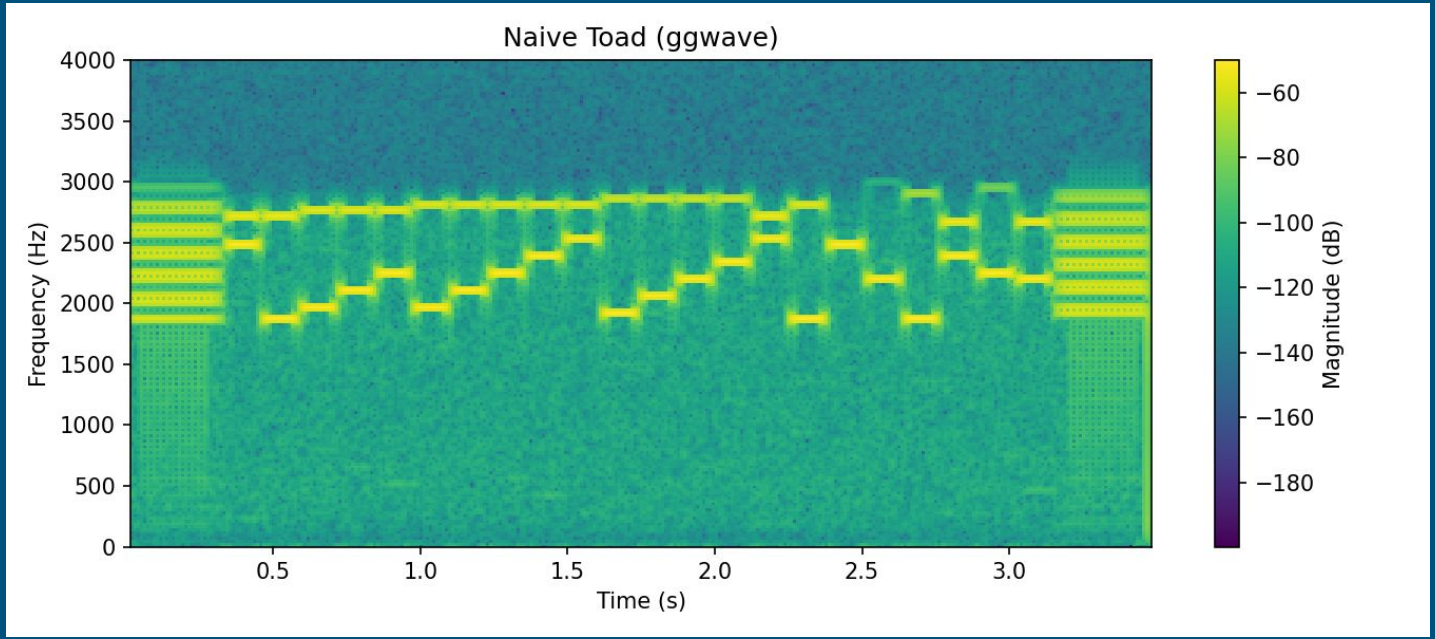


Intro to ToAD

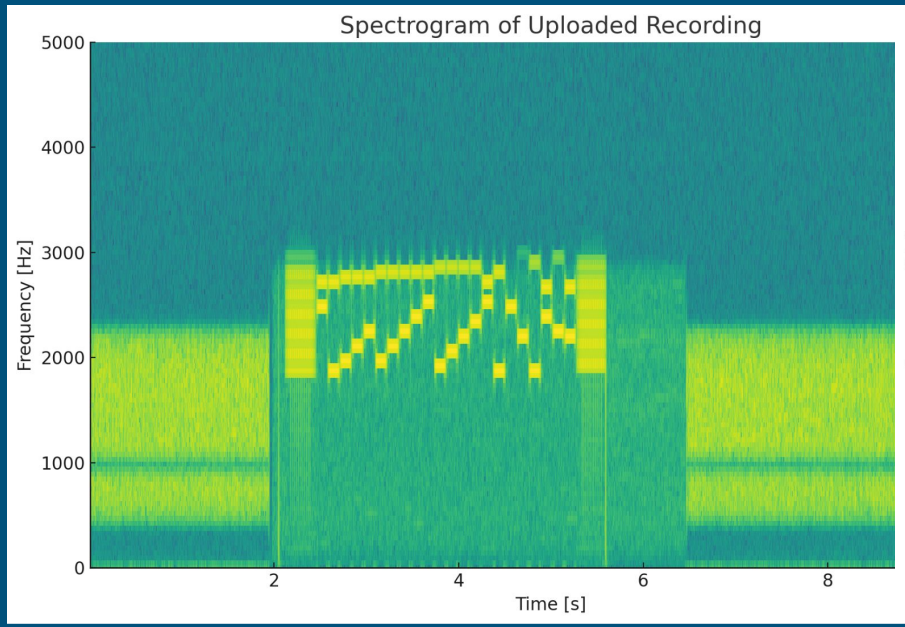
- FSK “16-choose-2”
  - 7ish bits per symbol
- 45-character alphabet
- Asynchronous (preamble/postamble)
- Redundancy
- 900 Hz bandwidth
  - Base tone: 100 Hz
  - 50 Hz wide tone bins
  - 16 bins

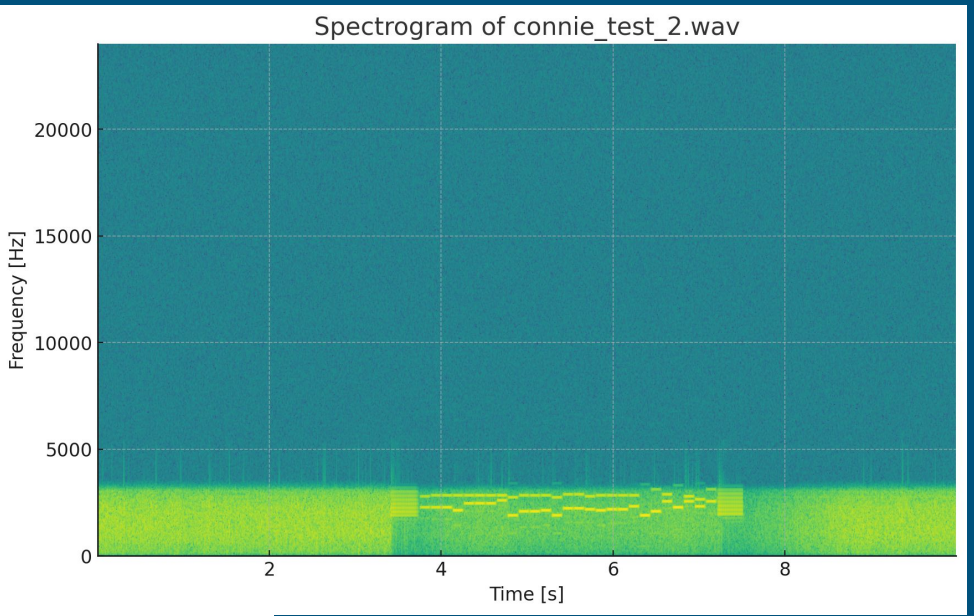


First Trial


Naive Toad (ggwave)
Frequency (Hz)
4000
3500
3000
2500
2000
1500
1000
500
0
0.5
1.0
1.5
2.0
2.5
3.0
Time (s)
−60
−80
−100
−120
−140
−160
−180
Magnitude (dB)

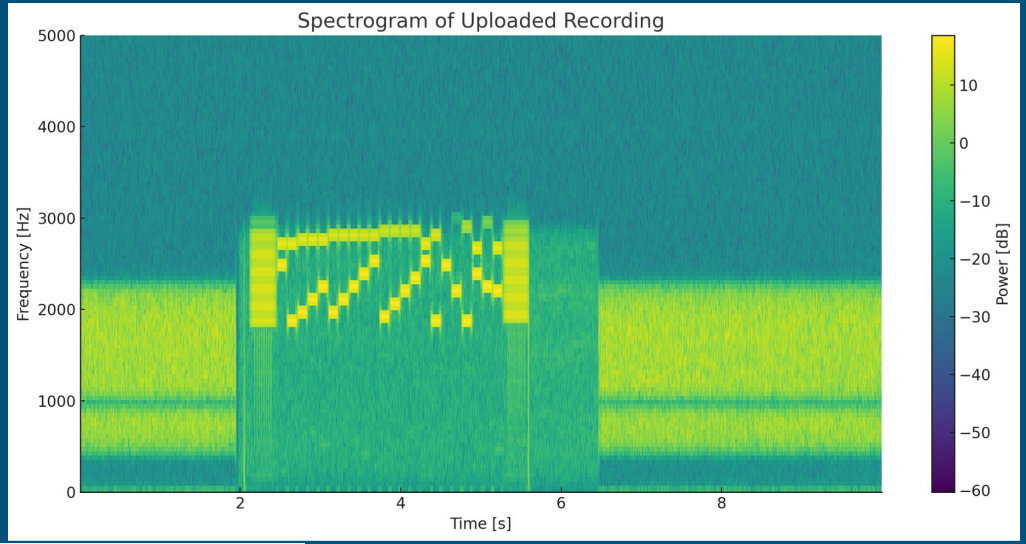
Wrong Turns - Bandwidth too high!

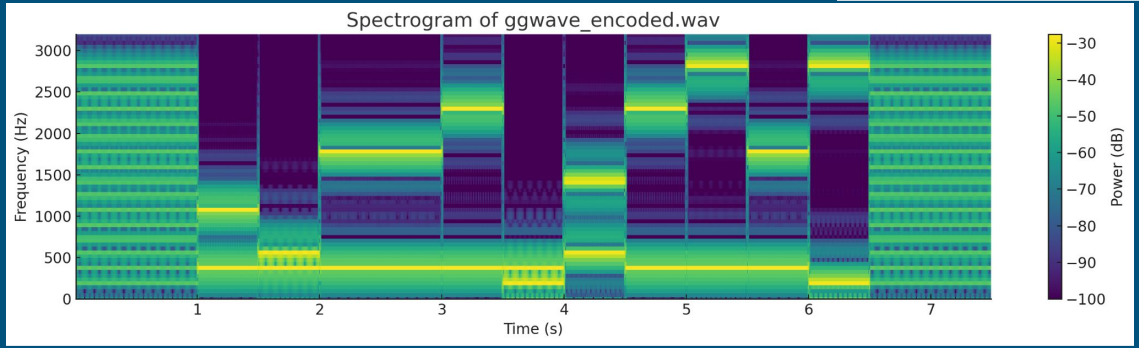




Encoded vs Played

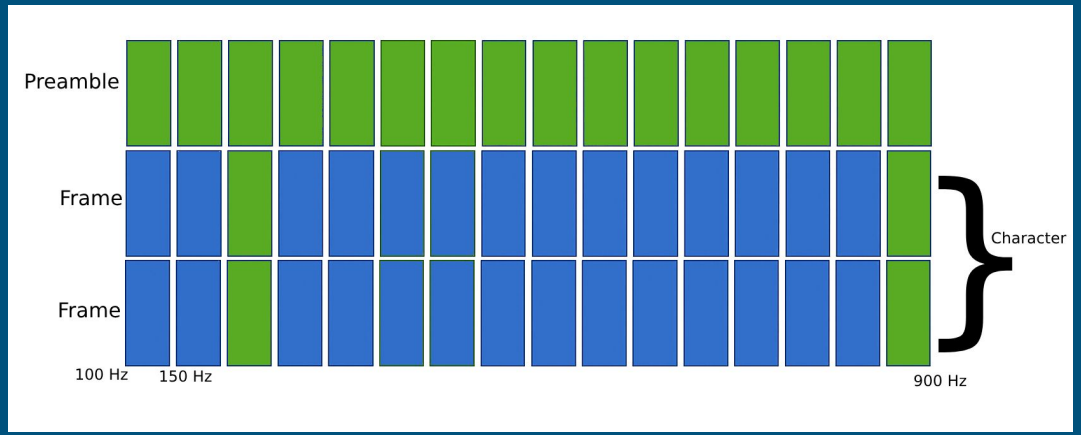
- Notice my noise floor
- Notice harmonics
- Notice artifacts from switching





After that, I

- Reduced the alphabet size (to 45)
- This made 16 bins doable
- I fiddled with the bin width:
  - First, 175 Hz (overkill)
  - Then settled on 50 Hz
- That’s ToAD!:
  - Less than 1 KHz BW
  - 16 Bins, FSK



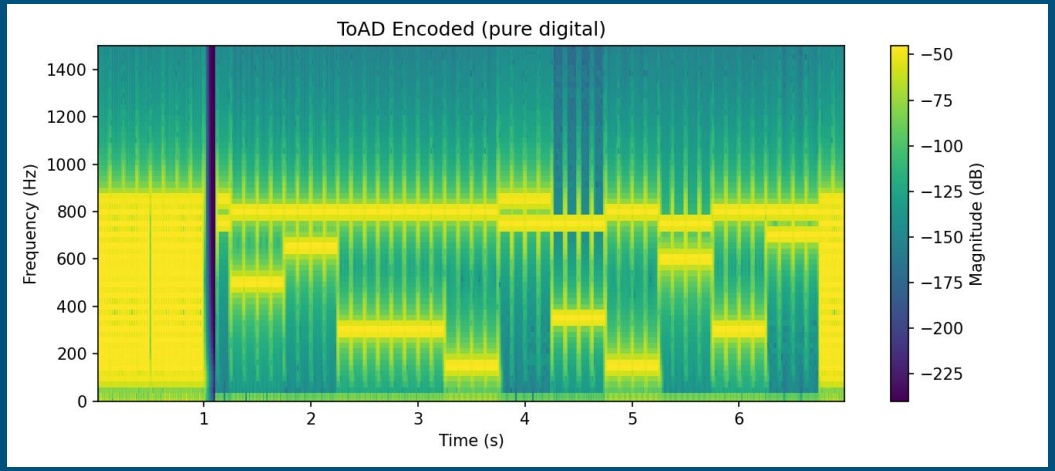
Alphabet

- 16 bins
- 2 active
- Theretically, 120 character options
- Try to keep them as far apart as possible (hamming distance)
- 45 characters in use today
- SFD/EFD are “all ones”

```
TEXT_TO_TOAD: dict[str, str] = {
  " ": "0000000000000101",
  "0": "0000000000001001",
  "1": "0000000000010001",
  "2": "0000000000100001",
  "3": "0000000001000001",
  "4": "0000000010000001",
  "5": "0000000100000001",
  "6": "0000001000000001",
  "7": "0000010000000001",
  "8": "0000100000000001",
  "9": "0001000000000001",
  "A": "0010000000000001",
  "B": "0100000000000001",
  "C": "1000000000000001",
  "D": "0000000000001010",
  "E": "0000000000010010",
  "F": "0000000000100010",
  "G": "0000000001000010",
  "H": "0000000010000010",
  "I": "0000000100000010",
  "J": "0000001000000010",
  "K": "0000010000000010",
  "L": "0000100000000010",
  "M": "0001000000000010",
  "N": "0010000000000010",
  "O": "0100000000000010",
  "P": "1000000000000010",
  "Q": "0000000000010100",
  "R": "0000000000100100",
  "S": "0000000001000100",
  "T": "0000000010000100",
  "U": "0000000100000100",
  "V": "0000001000000100",
  "W": "0000010000000100",
  "X": "0000100000000100",
  "Y": "0001000000000100",
  "Z": "0010000000000100",
  "-": "0100000000000100",
  ",": "1000000000000100",
  ".": "0000000000101000",
  "!": "0000000001001000",
  "?": "0000000010001000",
  "@": "0000000100001000",
  "$": "0000001000001000",
  "#": "0000010000001000",
  "^": "1111111111111111",
  "MARKER_HI": "1111111100000000",
  "MARKER_LO": "0000000011111111",
}
```

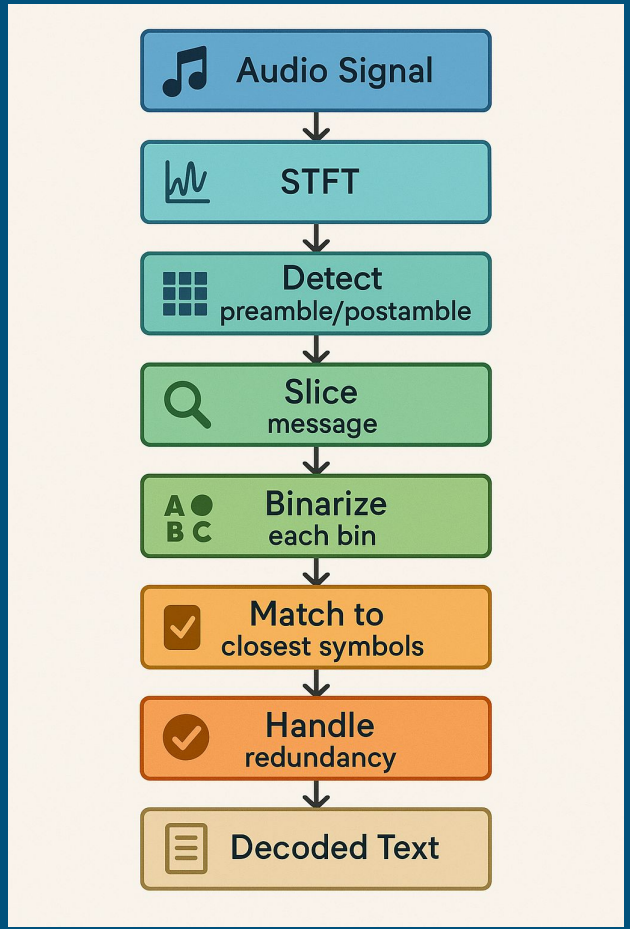
Encoding

1. Text is mapped to ToAD Symbols (16-choose-2)
2. Symbol is turned into a tone pattern (2 pure-sine waves)
3. Tones are cosine-smoothed to make transmissions cleaner
4. Preamble and postamble is added (all 16 tones on)
5. 1 KHz bandpass filter is applied to clean up rough edges



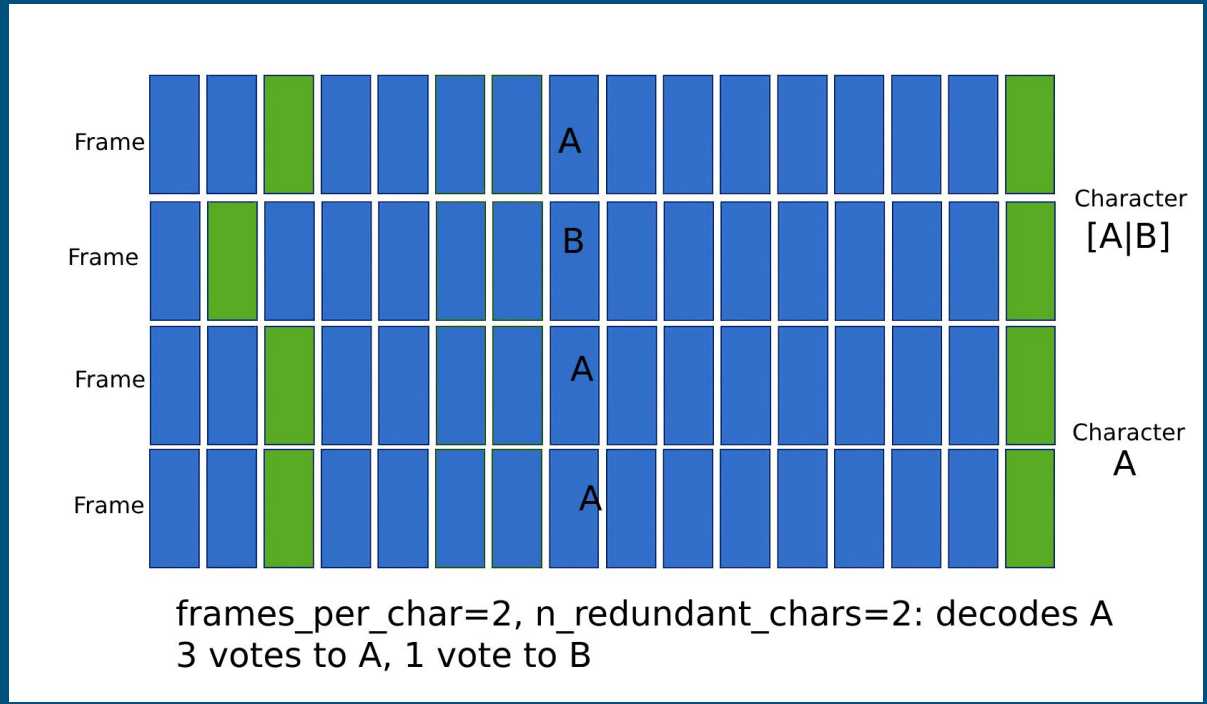
Decoding

- Take a Fourier Transform of the signal
  - Convert the signal to frequencies
- Find the 16 bins we need (100 Hz -> 900 Hz)
- Detect the preamble/postamble
- Slice the message between the preamble and postamble
- Binarize each of the bins to either “on” or “off” and compare to the alphabet
- Match the binary strings to closest symbols
- Handle redundancy (more details later)



Error Correction / Redundancy

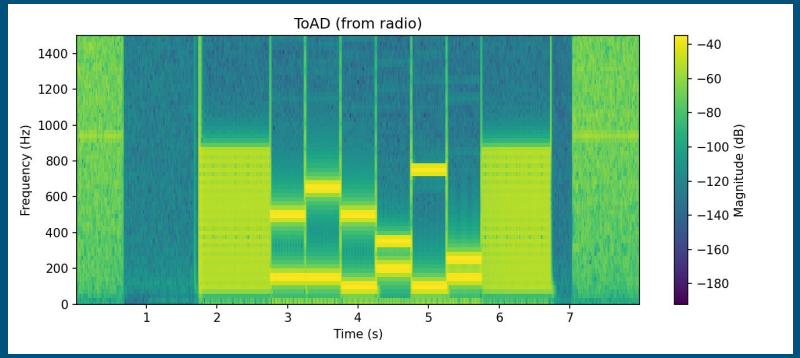
- N frames per character
- M characters per redundant group
- Two-level voting
- Resolve ties by sending both to user
- Ignore 4-way or more ties.



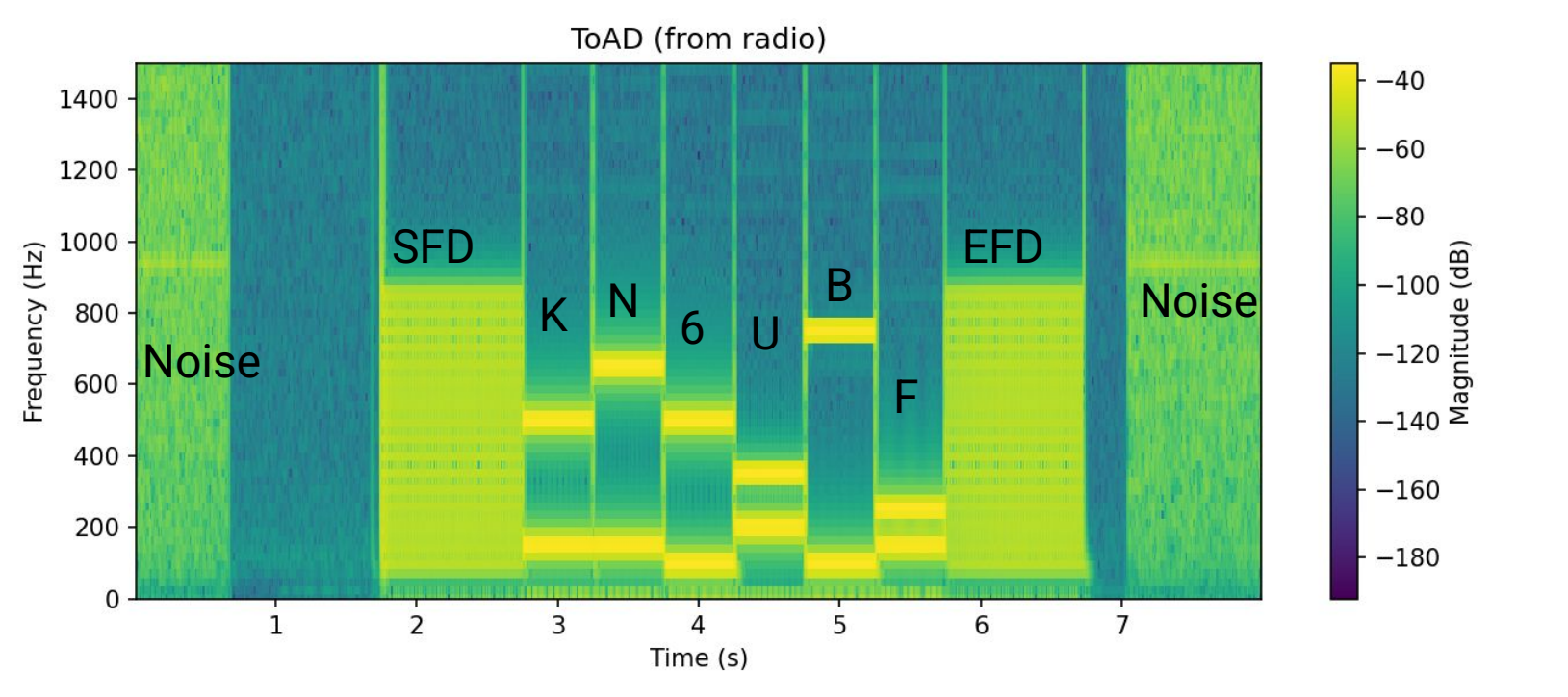
Framing

- Right now it's really Naive
  - All 1s for _n_ frames
  - Played with N, amplitude, etc.
  - Played for max energy - is this a problem?
- This is causing me **huge** problems
  - When I test, I can usually visually see the symbols and decode manually
  - But my SFDs almost always become unreadable
- Reducing the threshold helps, but causes us to decode noise
  - Wiping out 4+ way ties helps a bit…
- I now totally understand the desire for modes with external clocks!

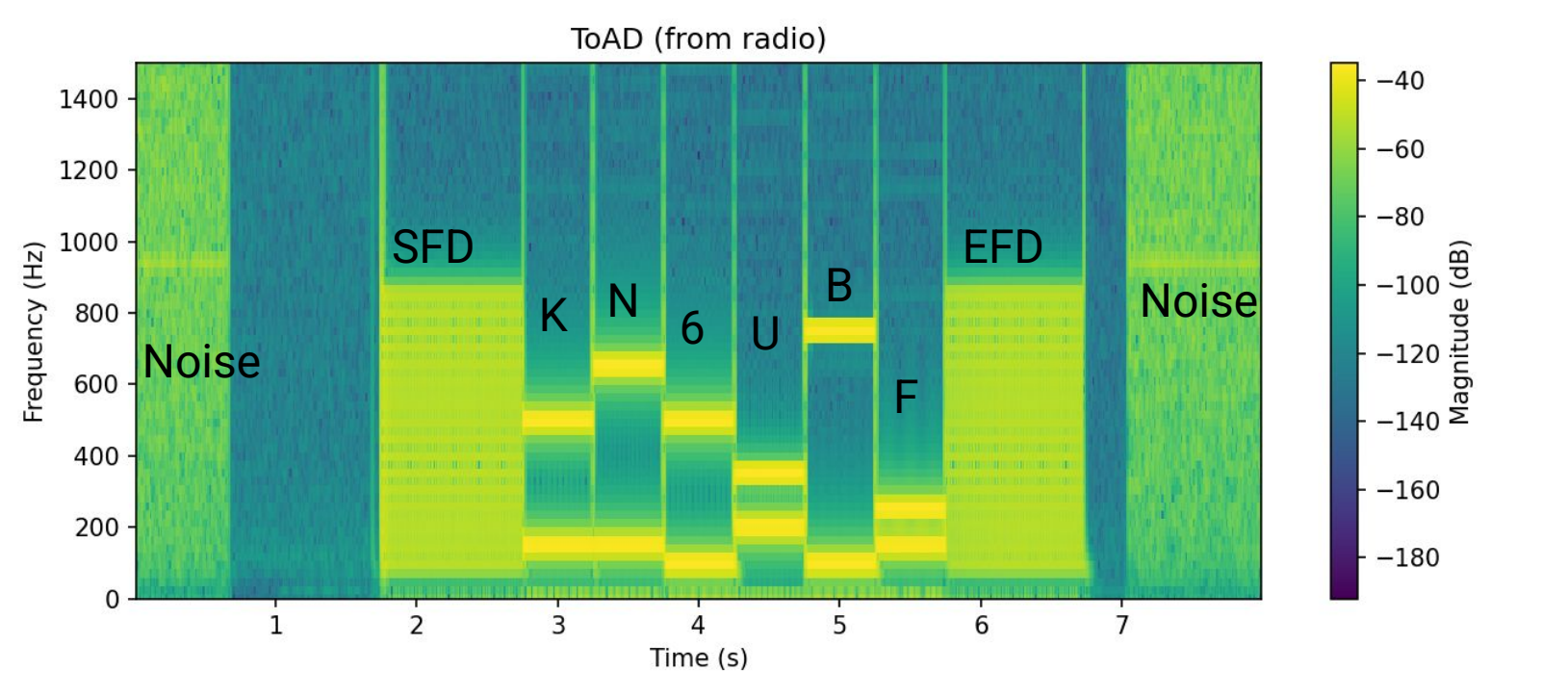
```
(radio_venv) glick@glicklab:~/Desktop/CATpack$ toad_terminal
Enter operating frequency in KHz: 14060
[ToAD] Listening on <IC 7300 port=/dev/ttyUSB0 audio_in=USB Audio CODEC audio_out=USB Audio CODEC> ('USB Audio CODEC')
[SEND] > kn6ubf
[RECV] KN6UBF
[RECV] EV
[SEND] > kn6ubf
[RECV] 3[ |0]3 [0|D]
[RECV] KN6UBF
[RECV] #
[RECV] 03D9[ |Q][?|R]0E[2|?]
[RECV] 1
[RECV] 4 T [D|E|Q][.|1]1[D|E|Q].[ |0|D][1|4]H[D|R]
[RECV] [1|E|Q]11?
[RECV] QR.[ |9]
[RECV] 4HE[G|N]T[ |1]D
[RECV] Q
[RECV] [,|C|P]
[RECV] ?
[RECV] ?
[RECV] F6IKE.Q.[6|F]I[0|R][E|U]!I[.|I][K|R]VE!
[RECV] J[5|T][D|V]?F?T
[SEND] > kn6ubf
[RECV] KN6UBF
[SEND] > kn6ubf
[RECV] KN6UBF
```



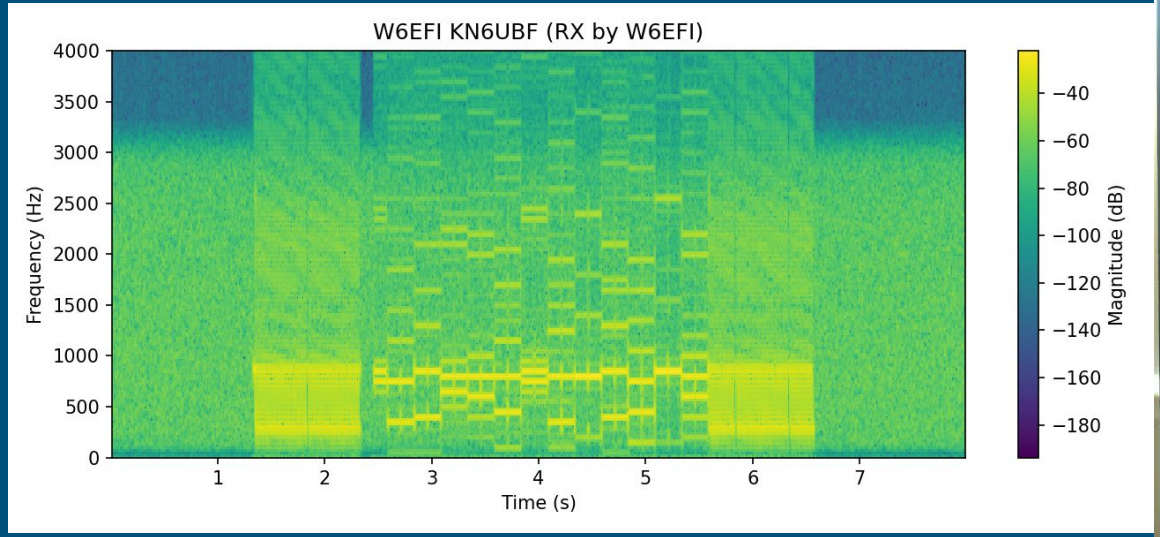
Example


ToAD (from radio)
Frequency (Hz)
Time (s)
Magnitude (dB)
Noise
SFD
K
N
6
U
B
F
EFD
Noise

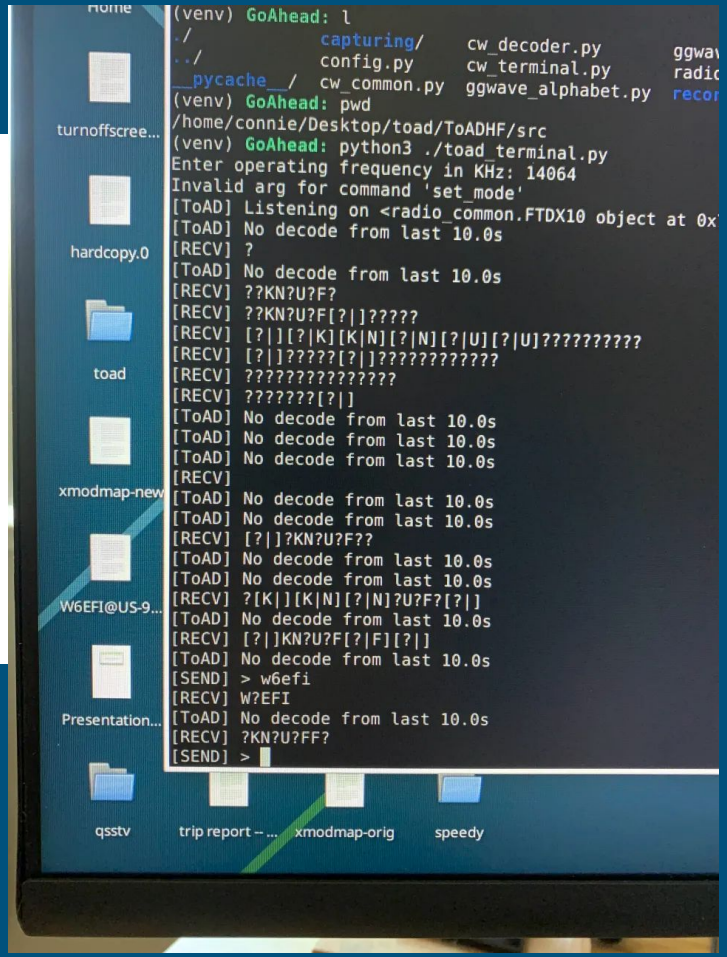
Demo


ToAD (from radio)
Frequency (Hz)
Time (s)
Magnitude (dB)
Noise
SFD
K
N
6
U
B
F
EFD
Noise

More live examples




```
1120: 0000000001000001
1121: 0010111001111111
1122: 0000000000011000
1123: 0000000000010001
1124: 0000000000000110
1125: 0000000000000110
1126: 0000000001000010
[RECV] W6EFI
[SEND] >
[ToAD] Shutting down...
```



Unique features of ToAD

[Assuming we sort out this SFD issue]

- Asynchronous
  - Leave your time source at home
- Debuggable/decodeable by ear/eye
  - At least at low symbol rates
- Flexible payload length
- Theoretically could push “high” symbol rate
  - Target 30 baud / 150 b/sec
    - With FEC / in good conditions
  - Today: 8 baud / 48 b/sec
  - Minus redundancy

Support

Radios:

- Icom IC7300
- Elecraft K3s
- FTDX10 (de W6EFI)
- TRuSDX (de AD1M)

Modulation:

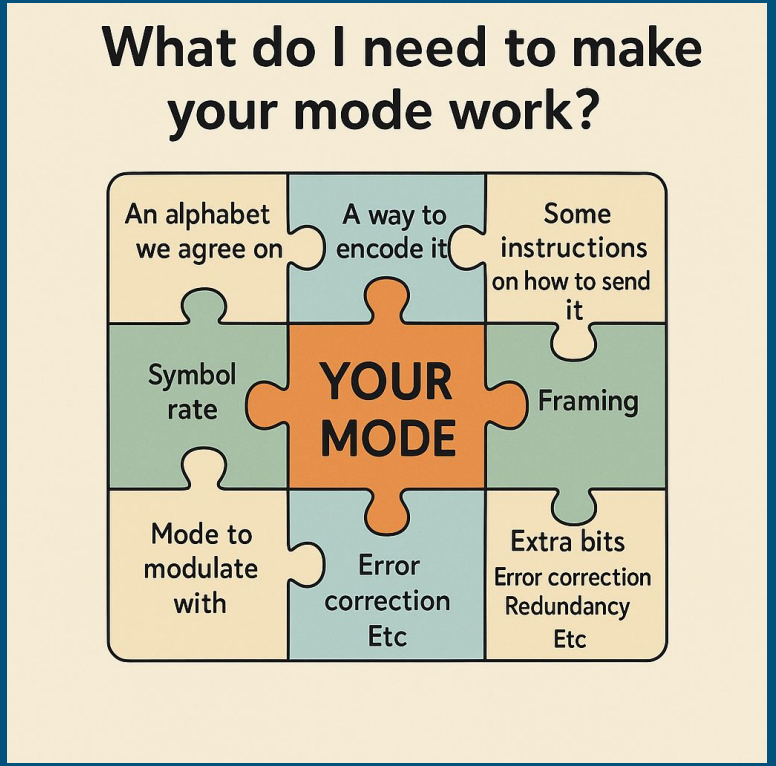
- SSB
- FM

Tested on bands:

- 20m
- 10m
- To Do: 2m (ToAD on N6NFI?)

What do I need to make a digital mode work?

- An alphabet we agree on
- A way to encode it
- Some instructions on how to send it
  - Symbol Rate
  - Mode to modulate with (fsk/psk, how to send data)
- Framing (this one is hard!!!!)
  - Start/stop keys (like OLIVIA / ToAD)
  - External clock (Like FT8/FT4) (with sync)
- Extra bits
  - Error correction
  - Redundancy
  - Checksum, CRC
- Lots of patience and work



Learnings

Technical:

- The ggwave decoder cannot handle any kind of noise at all
- Synchronization is hard and preamble detection is harder
- RFI/QRM can turn a 1 into a 0 just as well as it can turn a 0 into a 1
- On the IC7300, 0% power output is decidedly NOT 0 watts
- It’s easy to consume a lot of bandwidth on accident

Other:

- Sometimes it’s a good idea to try something somewhat impractical/useless

Next Steps

ToAD Improvements

- Low Density Parity Check codes for FEC
- GPU-accelerated DSP steps
- Better tone-tone transitions (filter before FFT)
- Better preamble/postamble
  - If you have ideas, please let me know :)
- Once the reference system works, write an actual spec about the protocol so others can implement it

Using ToAD

- Test at higher baud rates
- ToAD POTA activation
- ToADpeater
- Other demo apps using ToAD as a data transport layer rather than just for keyboard to keyboard

Acknowledgement

- W6EFI, AD1M, AJ6X for testing, ideas, feedback, additional radio support
- K06EZY for inspiration
- KC6TYD for encouraging me to share here

Conclusion/Questions

- Thanks for your attention
- ToAD lives at https://github.com/benhg/ToADHF
- Please get in touch : glick@glick.cloud / KN6UBF OTA
- I’m happy to answer any questions here!