The FT8 decoder of the JTDX CE3TSK fork

How it works, what changed, the presets - with the classic unit (P9) and the light pipeline (P10)

Tihomir Sokcevic CE3TSK

September 6, 2026

By Tihomir Sokcevic CE3TSK

How the decoder works, what this project changed in it, the three approaches that are new - the alternate-approach pass, ensemble decoding and the pipeline ensemble \- and the presets that put them in the menu. Written 2026-08-26 at the end of the work for the WW Digi contest; the measurements are those of the plan documents named in each section, all taken with the file mode on the benchmark capture `wav/full_band_16.wav` (a crowded 20 m band, 104 reference messages) and reproducible to the message.

1. FT8 in brief

An FT8 transmission is 79 symbols of 8-FSK, 6.25 Hz apart (a 50 Hz wide signal), 0.16 s each - 12.64 s of a 15 s period. Three 7-symbol Costas arrays (tones 3,1,4,0,6,5,2) at symbols 0-6, 36-42 and 72-78 are the synchronisation pattern; the 58 data symbols carry 174 bits: a 77-bit message, its 14-bit CRC and 83 parity bits of an LDPC(174,91) code. Decoding is therefore: find where a signal is (frequency and start time, "DT"), measure its 58 symbols as soft bits, and run the error-correcting decoder until a codeword with a valid CRC comes out - or not. Everything a decoder does better than another is in *which* candidates it tries, *how well* it measures them, and *what* it does with the strong signals that cover the weak ones.

2. The decoding chain as JTDX runs it

JTDX decodes in a separate process, `jtdxjt9`, fed by the GUI through shared memory: the GUI's `dataSink` accumulates the period's audio (12000 samples/s, `dd8`, 180000 samples = 15 s), and at half-symbol 49 (14.1 s; 51 = 14.7 s in SWL mode, 48 with "early start") removes a `.lock` file; the decoder, polling for it, reads the block and decodes. Decoded lines go back through a pipe, the GUI prints them and, when `<DecodeFinished>` arrives, the auto-sequencer chooses the reply. The chain inside the decoder (`decoder.f90` → `ft8_decode.f90` → `sync8.f90` → `ft8b.f90`), per period:

2.1 Passes

A decode is a sequence of *passes* over the band - the "decoding cycles" of the menu (3-9) - and in each pass every candidate found by the sync search is tried. Between the passes the signals decoded so far are subtracted from the audio, so the next pass sees what was under them. The pass number sets the sync threshold (`syncmin` 1.225 / 1.5 / 1.1 in a repeating triple when *low thresholds* or SWL mode is on, 1.5 otherwise), whether the pass subtracts (all but the last of a 3-cycle plain decode, and none after the fifth), and two data tricks: at pass 4 the audio is replaced

by the average of adjacent samples (a half-sample delay and a low-pass, which changes every downstream measurement), at pass 7 by the average shifted the other way. **SWL mode** ("the 6-pass decoder") uses the SWL cycle count, a wider DT search (+-3.5 s instead of +-2.5 s), its own sync thresholds and more decoding sub-passes per candidate \- and its own subtraction window (section 3.7).

2.2 Candidate search - `sync8`

Symbol spectra: the audio is cut into 1920-sample symbols stepped by a quarter symbol (0.04 s), each windowed and transformed (3840-point FFT: 3.125 Hz bins, two bins per FT8 tone). Over the bins of the decoded range and every DT lag of the window, the Costas pattern is correlated: the power at the seven sync tones against the power beside them, for each of the three arrays and for the arrays plus the message's own "CQ tail" positions, in two variants (all three arrays; the last two only, for late starts). The result is a sync surface `sync2d(frequency, lag)`. Per frequency bin the best lag is kept - and, since this project, also the best lag within +-0.52 s of nominal, so a weak signal whose peak is hidden by a stronger co-channel one at another DT gets its own candidate. The surface is normalised by a baseline, the 40th percentile of the per-bin peaks over the *wide span* (section 3.6), and every bin above `syncmin` (1.1 within 3 Hz of the QSO frequency) with a plausible DT becomes a candidate, up to 2000 (was 450), sorted by sync. In JTDX the sync surface is computed per pass and per slice (section 2.6) - about a tenth of the decoder's thread time.

2.3 One candidate - `ft8b`

- **Downsample**: the band is mixed so the candidate's frequency sits at baseband and

decimated to 200 samples/s - 3200 complex samples, 32 per symbol (`ft8_downsample`, cached per pass).

- **Fine sync** (`sync8d`): the DT is refined by half-sample steps and the frequency by

small tweaks (`twkfreq1`), maximising the Costas correlation on the downsampled data; a candidate that does not reach the sync gates is dropped here - this is where most of the 2000 die.

- **Symbol measurement**: 79 32-point FFTs give each symbol's power in its 8 tones; from

them soft bits (log-likelihood ratios) for the 174 code bits - computed with one, two or three symbols taken jointly (`nsym` 1/2/3: the joint versions resolve tones that a single symbol's noise blurs) and a fourth variant of the metric (`llrd`).

- **Decoding sub-passes**: each LLR set goes to the LDPC belief-propagation decoder

(`bpdecode174_91`); when it fails, to ordered-statistics decoding (`osd174_91`, order 3 - order 4 with *OSD order 2 for every candidate* / "deep OSD" - the algorithm's order counts differently from the menu's wording). Then the **a-priori (AP) sub-passes**: the QSO state tells the decoder what part of the message must be (`CQ ...`, `MyCall ...`, `MyCall DxCall ...`, `MyCall DxCall RRR/73/RR73`: 32, 61 or 77 of the 77 bits), those bits are fixed and the rest decoded - the table `naptypes(QSO progress, sub-pass)` says which hypotheses to try when, and SWL mode runs more of them. AP is what makes a reply to *us* decode several dB below anything else; the decodes it produces are marked in the output.

- **Checks**: the CRC-14 must match; `unpack77` turns the bits into text (and since

2026-08-28 rejects a report field no transmitter encodes, `irpt` > 105 - the signature of an a-priori false decode such as `CE3TSK JW1GPY/R 216`; and rejects type 3 RTTY Roundup exchanges and

type 0.5 telemetry altogether behind the source constants LRTTYRU_DECODE and LTELEMETRY_DECODE, neither used in FT8 on HF and both only ever seen as false decodes); chkfalse8 rejects implausible callsigns (structure and the prefix-vs-grid table; the ALLCALL7.TXT known-call lookup is switched off in the source since 2026-08-28, LALLCALL7_FILTER in chkflscall.f90 - with the 2024 file it cost ~45 real decodes an hour for one or two false ones), grids and hash inconsistencies; an SNR is estimated; the message is compared with the period's list (allmessages) so a re-decode of a known message never prints twice.

- **Subtraction** (subtractft8): the decoded message is re-encoded into its 79-tone

waveform (gen_ft8wave), its complex amplitude against the audio is estimated symbol by symbol and smoothed with the cwfilter window, and the fitted signal is subtracted from the band copy, so the next pass sees what was under it. Since this project the tones are also stored for the residual unit (section 4.3).

2.4 Hashes, hints and stored signals

Non-standard callsigns travel as hashes; fillhash after each pass makes the callsigns decoded so far resolvable in the next. "Hint" decodes (lft8s / lft8sd, the * and ° marks) are messages the QSO thread decodes with the hypothesis that the other station sent what the sequence expects, on the QSO frequency; the stored CQ / MyCall / QSO signals of pass 1 feed the next period's hypotheses, and the DT average of the period's decodes (avexdt) centres the next period's DT window.

The hint lists reach four same-parity periods back (P11, the project notes): JTDX kept one, so a single missed period broke a station's chain of hint decodes; two minutes of memory measured +6 % decodes on air for one or two stale hint decodes an hour. A band or mode change empties the lists, which JTDX kept across it.

FT4 had no hint pass at all in JTDX. Since 2026-08-29 it has the same four-period memory (ft4_mod1.f90): a message decoded in one of the last four same-parity 7.5 s periods at a candidate's frequency and DT is tried once more as one extra pass with all 77 message bits fixed - the RR73 a-priori type 6 mechanism, so the acceptance test is JTDX's own - after every ordinary pass failed in all three DT segments. Marked ^ like FT8's hints; +4.7 % messages on 73 periods of the WW Digi FT4 traffic, nothing lost, ~1 % of the decode time (IMPROVEMENTS_WW_DIGI_2026.pdf item 37; the ft4hint.sh check of the measurement harness).

Since 2026-08-30 FT4 is threaded the way FT8 is, and the two decoders now share the same shapes: the band is cut on an anchored slice grid, each slice decodes its own copy of the period, and the results are merged in slice order (the project notes, IMPROVEMENTS_WW_DIGI_2026.pdf items 51 and 52). Two differences are worth knowing. FT4's second slicing pass defaults **on** where FT8's is off, because FT4's slice boundaries cut into far more of its 90 Hz-wide signals than FT8's do into its 50 Hz ones. And FT4 saturates at 8 threads whatever the machine, because the 417 Hz grid cells make 100-3300 Hz into 8 slices, not one per thread.

One asymmetry now runs the other way, and is written up in the project notes: FT4 holds whole-line content reproducible run to run, while FT8 holds only the message set. FT8 still emits each decode from inside the parallel region (ft8_decode.f90:321-358), so when a signal near a slice boundary is decodable by two slices the copy that prints is whichever thread arrived first

- measured as SNR differences up to 6 dB on the same message, and one marginal background decode

appearing in one run of five. Nothing is lost by it today, but FT8 is the production decoder and the fix is the merge FT4 already has.

The threading also exposed how much of the per-slice bookkeeping FT8 had already had to solve: `osd174_91`'s enumeration state, `unpack77`'s window of recently seen callsigns and `fillhash`'s harvest are all keyed by a slice index, and FT4 was passing a literal `1` to every one of them from its single-threaded days. Correcting that, and giving each thread its own copy of the a-priori CQ/RRR/73/RR73 bit patterns, recovered decodes at every threaded setting as well as making the output reproducible.

2.5 The output

Every decode goes through one callback (`ft8_decoded`): UTC, SNR, DT, frequency, the message and a mark column (`*` hint, ° LoTW hint, `?` doubtful, • LoTW; since this project | a TX-background decode and ⊞ a hint made there). `<DecodeFinished>` closes the period with the DT average and the candidate count; `<BackgroundFinished>` (section 4) closes the background phase.

2.6 Threads and slices

With more than one thread the band is cut into slices decoded in parallel, each slice on its own private copy of the audio (`dd8` is thread-private), each recording the delta its subtractions made; the deltas are merged in slice order into the band that later passes and units start from. Because the merge order is by slice index and not by which thread finished first, the result is identical at any thread count (section 3.3); a single thread decodes the whole band in one piece. The *second slicing pass* repeats the pass list on a grid shifted by half a slice, so a signal on a slice edge is seen whole once.

3. What this project changed in the decoder

The full list with numbers is `IMPROVEMENTS_WW_DIGI_2026.pdf` (25 items); the campaign that produced them, step by step with every measurement, is the project notes. In short, grouped:

3.1 A way to measure

The file mode (`jtdxjt9 -8 -d 3 <options> file.wav`): every GUI setting became a command-line option (`-W` SWL, `-K`/`-C` cycles, `-E` sensitivity, `-O` deep OSD, `-z` second slicing, `-X` alternate pass, `-M n` members, `-B` background and its `-I -D -F -J -P -U -V -Y -Z -k` controls, `-L`/`-H` range, `-j` threads, `-Q` AGC metric), with diagnostics (`JTDX_DUMP_PARAMS`, `JTDX_MEMO_STATS`, `JTDX_ENSEMBLE_TRACE`, `JTDX_BG_LOCK`). A reference set of 104 messages was built from many runs of JTDX, WSJT-X and averaging, classified by SNR tier and crowding (`reference_set.tsv` and the project notes of the measurement harness): the 17 at or below -16 dB are "the weakest". Every later change was judged against it.

3.2 Correctness

Threaded decoding was non-deterministic - the same file gave different sets run after run - and some of the reasons were races: the shared audio buffer that all threads subtracted into, a C++ downsample cache, the OSD initialisation, the FFTW plan cache publication, `gen_ft8wave` tables, the decoded-message counter. All fixed (thread-private buffers, double-checked initialisation, a critical section). A latent trap (FFTW MEASURE plans in place) is documented, not changed.

3.3 Determinism and machine independence

The band grid was tied to the thread count (one hard-coded section per thread, 23 of them); it became one grid of slices that threads pick up dynamically, with the delta merge in slice order - so a recipe's set is the same at 1, 3, 6 or 12 threads and only the time changes (the regression rows pin this). Every run of the file mode is now reproducible bit for bit, which is what made the sweeps of this project possible.

3.4 Speed

The OSD's Gaussian elimination rewritten on packed 64-bit words (3 words per row instead of 174 bytes; equivalence proved on 23,000 seeded cases), `sync8`'s window sums tabulated once per pass, the long-FFT plan cache thread-private. The exhaustive recipe went from about 6 s to 2.6 s at 12 threads; every preset fits the period from 2 threads up.

3.5 Candidates

The second DT peak per bin (2.2), the candidate cap raised from 450 to 2000 (a pileup's adjacent-bin echoes filled the old one), and the decode range decoupled from the waterfall width (the decoder used to see only what the waterfall showed).

3.6 The decoded range no longer changes the result

A range narrower than the full 0-5000 Hz used to decode worse (default 71 $\rightarrow$ 61, exhaustive 85 $\rightarrow$ 80 at 100-3100 Hz) although the signals were all there, for two reasons found in this project (the project notes): the sync baseline was taken over the decoded range, where a busy band puts the percentile on signals, and the slice grid was laid over the range, making 250 Hz slices that stop subtraction at their edges. Now the baseline uses a fixed 0-5000 Hz span and the grid is anchored to the full band (cells of 417 Hz from 0 Hz; a range gets the cells it intersects), so 100-3100 Hz decodes as the full band does inside it (70 / 77 / 84 against 71 / 78 / 85, the difference being the one message at 78 Hz). The runtime the narrow range "saved" was fewer candidates tried.

3.7 The subtraction window follows the unit

`cwfilter` builds the window that smooths the subtraction's amplitude estimate - a 3400-tap cosine in SWL mode, a 4000-tap cosine-squared otherwise - and built it once per decode from the period's SWL flag. After a plain-cycles RX phase every SWL member and the SWL background recipe therefore subtracted with the wrong window and the pipeline stopped 9 messages short (91 against 100). Each unit now sets the window for its own SWL flag; the "pipeline run" preset went from 93 to 99 on the full band. (Forcing one window on everything is measurably worse either way: the windows are recipe-specific.)

3.8 Findings that are settings, not code

AGC compensation costs 5 messages on a busy band (it switches `sync8` to another metric; keep it off). The decoder sensitivity setting is inert under SWL mode. "Aggressive" is never read by the FT8 decoder. OSD order 2 is not a weak-signal lever on a crowded band. Sensitivity 2 ("low thresholds + subpass") is the cheapest gain in the menu.

4. The three new approaches

4.1 The alternate-approach pass

The observation. Every recipe on the benchmark stalled at 79-81 messages and 7 of the 17 weakest, whichever knob was turned - because every recipe is one path through the same chain of threshold decisions (which candidates, which DT window, which sync thresholds, which AP hypotheses, which subtraction order), and the messages it misses are the ones that path rejects at some gate.

The method (`-X`, the project notes step 10): after the recipe's own passes, decode the band once more with the *opposite* recipe - a plain 7-cycle decode with OSD order 2 after an SWL decode - starting from the band with every subtraction so far merged in. The second path has other thresholds, another DT window and other AP sub-passes, and it runs on a band from which the strong signals are already gone; what it finds is printed if new, its subtractions are merged too.

The result. SWL-5 alone 78, with the alternate pass **85 messages and 8 of the 17 weakest** in 2.6 s at 12 threads (3.1 s on 100-3100 Hz), zero false decodes - the "exhaustive" recipe, and the base of everything after it. It is part of the ensemble and of the TX background recipe, and available as a switch in both submenus.

4.2 Ensemble decoding

The observation (the project notes). Before the races were fixed, the threaded decoder sometimes reached "over 100" messages on the capture; deterministic, the best recipe gave 85. The non-determinism had been *sampling* the chain of threshold decisions: a different timing gave a different subtraction order, and some of those samples got past gates the deterministic path did not. The gain was real; the mechanism was luck.

The method (`src/lib/ft8ensemble.f90`, `-M n`): take the same samples on purpose and reproducibly. A *member* decodes the pristine band again through a fixed, tiny perturbation - a 128-sample delay (shifts every FFT window by a tenth of a symbol), a 3 % dither (a seeded xorshift64 noise, sum of 12 uniforms, added to the audio), or a sub-bin frequency shift (+-0.8, +1.2 Hz via the analytic signal) - with a fixed recipe (SWL-5, some with the alternate pass), and only messages not yet on the list are printed, their DT and frequency corrected back. The members share the period's duplicate arrays and hash lists and start from the untouched band, so they are independent samples of the decision chain, not passes of one. Eight members are defined; the RX phase runs up to three, gated by the thread count, or the number the *ensemble effort* control names.

The result. Exhaustive 85 $\rightarrow$ **98 with three members** (7.9 s at 12 threads), 101 with five, every gained message in the reference set, none false; on 100-3100 Hz the Ensemble preset gives 97 with 13 of the 17 weakest. On a sparse band the members find nothing extra and invent nothing. One member on the cheap 5-cycle base is the best second second of decoding there is (+11 messages for 1.2 s), which is what the Max decodes preset is.

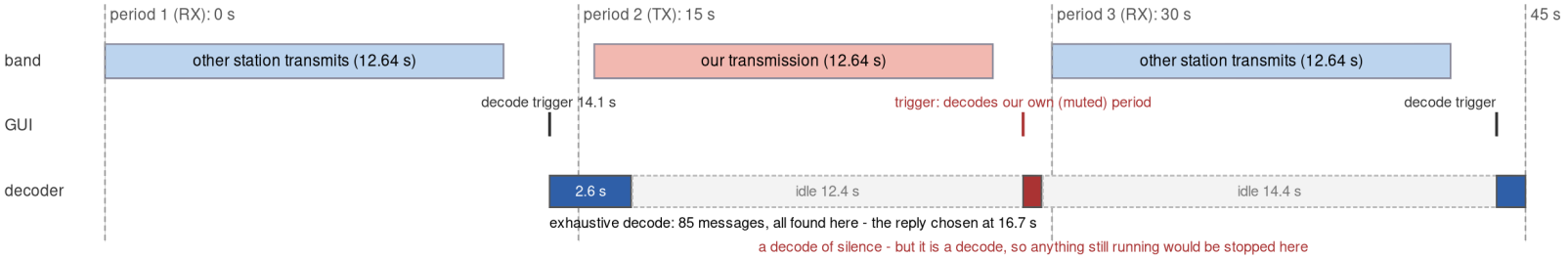
4.3 The pipeline ensemble

The observation (the project notes). The decoder process was idle about 12 of every 15 s - in RX periods after the decode, and in our own TX periods entirely \- while the reply decision is taken about a second after the decode starts, so anything slower than that was already "late" for the reply and there was no reason to make the *reply* wait for the members at all.

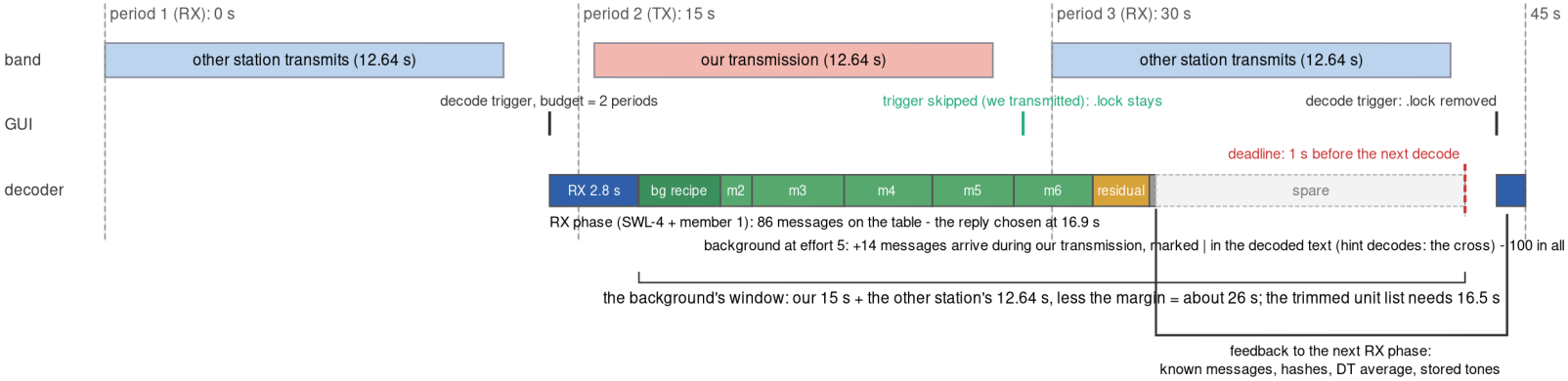
The method. The period's decode becomes a fast **RX phase** (its result is on the table when the reply is chosen), and a **background phase** follows in the same process: more units on the

retained band, in the decoder's idle time, printing only new messages with the period's UTC. The unit list, in order: the *TX background recipe* (the TX background submenu's own settings: default SWL-5 + alternate pass, a fresh decode of the pristine band with the duplicates carried), the **classic unit** (P9: the plain 6-cycle non-SWL decode - the pass sequence no SWL unit runs, \~0.5 s; measured over 207 on-air periods it adds 6 messages the rest of the pipeline never gets and loses none; on air the P9 build then out-decoded the old plain decoder 9201 to 8540 over 1323 periods, the project notes), the ensemble members the RX phase did not run, the **residual pass** (every known decode subtracted from the pristine band through its stored tones, then SWL-5 with the sync threshold lowered to 0.8 of normal: the unit that finds what the members miss), and the *retry unit* (the LDPC/OSD failures decoded again on dithered data, measured to find nothing: the failures are noise candidates, a result that settled where the members' gains come from, the candidate/subtraction stage). Every unit checks between its passes whether the GUI removed `.lock` for the next decode - or created the `.bgabort` sentinel, which a band or mode change does, so the previous band's audio is never printed under the new dial - and stops within a pass if so; before each unit the learned cost table decides whether it still fits the deadline. After an abort the decoder goes straight to the next decode only if `.lock` is really gone (the GUI's request); after a sentinel abort with `.lock` still in place it waits as usual - jumping back unconditionally re-decoded the same retained audio in a loop when Monitor was off (found on air 2026-08-27, fixed in `jt9a.f90`). The decodes carry `|`, hint decodes made there ⊞; the decoder label reads "/D -" while the period decodes, "/D" when done, "/(D+N=S) |" while the background runs, "/(D+N=S)" when it is done; `<BackgroundFinished>` closes it with units, messages, seconds, abort.

Before: one decode per period, the decoder idle the rest of the time



Pipeline ensemble: a fast RX phase, then the background phase in the idle time - on through our own TX



The units of the background phase (measured, 100-3100 Hz, 12 threads, after the SWL-4 + member RX phase)

- background recipe: SWL-5 + alternate pass (3 new, 2.6 s)
- ensemble members m2-m6: 3 % dither / -0.8 Hz / +1.2 Hz / delay 128 with alt / +0.8 Hz (3, 3, 2, 1, 0 new; the +0.8 Hz one pays on sparse bands)
- residual pass: every known decode subtracted, SWL-5 at a lowered sync threshold (2 new)
- retry unit: the LDPC/OSD failures again on dithered data (0 new - instrumentation)
- members 7-8 (dither seed 2, 8-sample delay) never found a message and are skipped at effort 5; background effort "auto" still runs them

The classic cycle against the pipeline: three periods, the GUI's triggers, the decoder's work

The drawing (FT8_PIPELINE.svg): above, the classic cycle - one decode when the period's audio is in, the decoder idle for the other 12-14 s of every period, and a pointless decode of our own muted TX period; below, the pipeline - the RX phase decides the reply just as early, and the unit list runs on through our transmission until a second before the next decode, feeding the next RX phase.

The window (P7, the project notes). JTDX used to decode its own TX periods (the rig's muted RX audio), which aborted the background 14 s into our transmission and replaced the retained band with an empty period. With the TX background enabled the GUI now skips that decode and hands the decoder a two-period budget before our TX: our 15 s plus the other station's 12.64 s, about 27 s, in which the whole unit list completes (20 s) every period. The RX budget the operator allows is about 2.7 s; the pipeline presets' RX phase fits it with a member.

The result. On 100-3100 Hz the *pipeline ensemble* preset has 86 messages on the table at 2.8 s and ends at **100 - every message any recipe finds on that range** (the union of 140 runs), 15 of the 17 weakest; *pipeline run* keeps the reply decision at \~0.8 s and ends at 97. On the full band 78 + 24 = 102. Deterministic, the same set at 3 and 12 threads.

4.4 The two-phase recipe in the menu

Decode → FT8 decoding: *Presets* (green), then one *expert* submenu (since 2026-08-28, P12) holding the *RX* (blue) and *TX background* (red) submenus with the same controls each - SWL mode, decoding cycles, SWL cycles, QSO RX frequency sensitivity, decoder sensitivity, OSD order 2, second slicing pass, alternate-approach pass, ensemble effort - all persistent, presets setting both phases; then early start, wideband DX Call search, and the *decode bandwidth* submenu (waterfall width, 100-3300, 100-3200 ... 200-2400, 300-3300, 50-3650, 0-5000 Hz "testing only"). Applying any preset also switches early start off and the wideband DX Call search on (PRESET_EARLY_START, PRESET_WIDE_DXCALL_SEARCH) - the environment every table here was measured in; the two are not part of the preset identification. The preset in force shows on the main window as the *Preset X* lamp under the Contest lamp (P13: 3 P V R E B M O in the submenu's order, "Custom" otherwise, lit in the preset's dot colour, plain text in a grey box for default / ensemble / custom, greyed outside FT8). Two effort controls: the RX ensemble effort (auto by thread count; *budget auto* - members while the next one's learned cost fits the \~2.7 s reply budget, so a quiet band gets the depth its spare reply time can hold at reply time; off; 1-5) and the TX background's (auto \= as many as fit, 0-5).

5. The presets

A preset sets the ordinary controls of both phases; what persists is the controls, the preset shown is derived from them (Custom when they match none). Measured on full_band_16.wav, 100-3100 Hz, 12 threads (this machine's "threads = auto"), one deterministic run each (preset_data.tsv of the measurement harness). RX times are file-mode times - the GUI's persistent decoder is about 0.5 s faster; TX background times are the whole unit list to completion, which the two-period window allows; total = RX + TX background.

preset	RX phase	RX msgs (weakest of 17)	RX time	TX background	TX msgs added	TX time	total msgs	total time
Default	3 cycles	70 (4)	0.95 s	off	-	-	70 (4)	0.95 s
Max efficiency	5 cycles, sensitivity 2	76 (7)	1.2 s	off	-	-	76 (7)	1.2 s
Max decodes	5 cycles, sensitivity 2, 1 member	87 (10)	2.4 s	off	-	-	87 (10)	2.4 s
Pipeline max decodes light	the same	87 (10)	2.4 s	plain 6-cycle pass, members 2-3, residual	+7	6.3 s	94 (12)	8.7 s
Ensemble	SWL-5 + alt pass + 3 members	97 (13)	9.1 s	off	-	-	97 (13)	9.1 s
Pipeline ensemble	SWL-4 + 1 member	86 (8)	2.8 s	SWL-5 + alt, classic 6, effort 5 (members 2-6), residual, retry	+14	17.2 s	**100 (15)**	20.0 s
Pipeline ensemble full	SWL-4 + 1 member	86 (8)	2.8 s	as above, every member (effort auto)	+14	19.6 s	100 (15)	22.4 s
Pipeline run	5 cycles, sensitivity 2	76 (7)	1.3 s	same, effort auto	+21	20.9 s	97 (14)	22.2 s

- **Default** - JTDX's own: 3 cycles, sensitivity 1 (low thresholds), no passes.
- **Max efficiency** - the knee of the decodes-per-second curve: +6 messages for 0.3 s over the default (43 $\rightarrow$ 51 on a 3 dB noisier capture); everything beyond costs far more per message. Also the pipeline run's RX phase.
- **Max decodes** - the most a 2.7 s reply budget can hold: the member adds 11 messages and 3 weakest for 1.2 s. One member from 8 threads (2.4 s there, 2.9 s at 6, 4.3 s at 3); below 8 threads it is the max-efficiency recipe. It retired the earlier "maximum decodes" (SWL-6 + second slicing: 81 in 7.4 s), "weakest signals" (5 cycles + OSD: 79 in 4.9 s) and "exhaustive" (SWL-5 + alt: 84 in 3.1 s), all dominated - and there is no separate weakest-signal preset because within the budget the same recipe wins both aspects (10 of 17 is the in-budget ceiling: eight variants tie or lose) and on a sparse band no recipe moves pure sensitivity at all.
- **Pipeline max decodes light** - Max decodes at reply time and a fixed ~6 s background (the plain 6-cycle pass first, members 2 and 3, the residual pass; no classic unit, the pass is that decode): the light pipeline for a 15 s cycle with TX (P10).
- **Ensemble** - the exhaustive recipe plus members by the thread count (3 from 12, 2 from 6, 1 from 3), no time budget: max effort RX only and no TX background. Since 2026-09-05 it has no entry in the presets submenu (the pipelines beat it at every effort); the recipe stays for file mode and the tests, and the lamp still shows E when the controls happen to match it.

- **Pipeline ensemble** - an SWL RX phase (SWL-4) with a member from 8 threads, because an SWL RX phase is what reaches the range's ceiling with the background; the TX background recipe at effort 5 - with the RX member and the member table's order that is every unit that ever found anything, plus the classic unit (P9) - done in ~17 s instead of ~20 (the two samples that never found a message are skipped; *auto* still runs them). The most decodes in all, and the most weakest (15), arriving during our TX.
- **Pipeline run** - for running a pileup: the reply decided at ~0.8 s, every reply an on-time start; the same background afterwards. (SWL mode in the RX phase would move the decode trigger 0.6 s later and turn replies into late restarts.)

Thread gates: the second slicing and alternate passes from 2 threads, the RX member from 8, the Ensemble members 3/2/1 from 12/6/3, a single thread runs the base recipes and has no background phase. On a machine with fewer threads the presets degrade by these rules; the RX effort control overrides them by hand.

On air the light preset ran an hour (240 periods) beside the Sep-2025 decoder at 9 cycles + sensitivity 2 on the same audio: 5739 decodes to 5100, 674 real finds only it made against 36, net +12.5 % (the project notes). The same hour replayed through every preset (section 13; the classical 9-cycle recipe standing in for Default and pinned at JTDX's one-period hint memory; rows 2-8 on the production decoder with the four-period hint memory of P11; RX and TX background seconds per period, decoder-measured; the sets are machine independent, the seconds are not - AMD Ryzen 7 5800H, 8 cores / 16 threads, decoder at 12, powersave governor, Linux Mint 22.3, gfortran 13.3 -O3). That hour is downloadable, with the script that reproduces these rows, at <https://ce3tsk.com/download/wav/>:

preset	decodes (vs stock)	RX s	TX bg s	total s (vs stock)	not in stock	stock only
Stock JTDX v2.2.159: 9 cycles, sensitivity 2, without ALL-CALL7.TXT, 12 threads - the baseline	4869	1.40	-	1.4	-	-
Stock JTDX v2.2.159: 9 cycles, sensitivity 2, ALL-CALL7.TXT present, 12 threads (its GUI's auto on this machine)	4783 (-1.8 %)	1.39	-	1.4 (-1 %)	40	126
Stock JTDX v2.2.159: 9 cycles, sensitivity 2, without ALL-CALL7.TXT (single thread)	4866 (-0.1 %)	2.04	-	2.0 (+45 %)	31	34
Stock JTDX v2.2.159 as shipped: 9 cycles, sensitivity 2, ALL-CALL7.TXT present (single thread)	4802 (-1.4 %)	2.03	-	2.0 (+45 %)	30	97
Stock JTDX v2.2.159 at its deepest: 9 cycles, sensitivity 2, SWL mode, 12 threads, without ALL-CALL7.TXT	4793 (-1.6 %)	1.55	-	1.6 (+11 %)	148	224
Stock JTDX v2.2.159 at its deepest: 9 cycles, sensitivity 2, SWL mode, 12 threads, ALL-CALL7.TXT present	4767 (-2.1 %)	1.56	-	1.6 (+11 %)	151	253
This decoder at JTDX's own recipe: 9 cycles, sensitivity 2, one-period hint memory	5249 (+7.8 %)	0.76	-	0.8 (-46 %)	448	68
Max efficiency	5417 (+11.3 %)	0.40	-	0.4 (-71 %)	624	76
Max decodes	5678 (+16.6 %)	1.14	-	1.1 (-18 %)	860	51
Pipeline max	6091 (+25.1	1.13	4.2	5.3 (+278 %)	1229	7

The

baseline is stock v2.2.159 at its honest best - its ALLCALL7.TXT lookup off (the operator's call is not in it), its GUI's 12 threads on this machine, the real binary through the file-mode patch of `test/decode/stock/`: 4869 messages at 1.4 s a period. The light preset takes 91 % of the pipeline ensemble's gain over stock at 42 % of its time (1222 of 1339 messages above 4869; 5.3 s against 12.7 s). This decoder at JTDX's own recipe (9 cycles, sensitivity 2, the one-period hint memory) already stands +7.8 % over stock at 0.8 s a period - the deterministic decoder and its threading - so of the recommended preset's +25.1 % about a third is that and the rest the pipeline. WSJT-X 3.0.2 on the same hour (its ini's multithreaded FT8 decoder, 9 passes, sensitivity 3, AP, 12 threads): 5171, +6.2 % over stock; its standard decoder at depth 3: 4782 (-1.8 %); WSJT-X improved 3.2.0 with the same ini settings: 5163 (+6.0 %). The union of all seventeen sets is 6450; the full pipelines reach 96 % of it, stock 75 %, JTDX's own recipe on this decoder 81 %. The gains are the weak end: the SNR distribution of the project notes (taken against this decoder at JTDX's own recipe) puts about a fifth of what each recipe adds at -24 dB or below, half at -23..-21 dB and a quarter at -20..-18 dB, next to nothing above -17 dB (the light preset turns that recipe's 152 decodes at $\leq$ -24 dB into 346, its 677 at -23..-21 into 1046).

Three presets, three efforts (section 13's recommendation): *Max decodes* is the best bang for the buck (+16.6 % over stock for 1.1 s per period, no background), *Pipeline max decodes light* the best medium effort (+25.1 % for 1.1 s at reply time plus 4.2 s of background in the idle half of the cycle - the production choice), *Pipeline ensemble* the best result (+27.2 % for 1.3 + 11.4 s; run and full within 40 messages of it, chosen by their RX phase). The presets submenu marks them: bold, a coloured dot and the tier with its figures in front of the name - orange "best value" for Max decodes, green "best medium effort" for the light preset, blue "best results" for Pipeline ensemble, purple "most results" for Pipeline run, red "max effort" for Pipeline ensemble full (every member sample, the longest background), and grey "best power" for Max efficiency (+3.1 % for 0.4 s: the least CPU and battery, for a portable or a hot day). The light preset is additionally marked as the recommendation: underlined, with "recommended:" in front of its label; Pipeline ensemble is underlined too.

FT4 on the recorded hours

The same experiment for FT4 (`test/decode/onair_ft4/`, 2026-09-05): every FT4 preset, the real stock JTDX v2.2.159 FT4 decoder with and without its ALLCALL7.TXT (the patched file-mode build, single-threaded - stock FT4 has no thread path), this decoder without its hint memory (JTDX's FT4 recipe on the fork's code), and the two WSJT-X engines, over the 240-period night hour (14.080 MHz from 23:38 UTC 2026-08-29) and the 73-period day hour (14:00 UTC the same day), 100-3100 Hz, one chain per row, 12 threads, `JTDX_STOPHINT=1` as the FT8 rows. The night hour is downloadable beside the FT8 one at <https://ce3tsk.com/download/wav/>. Percentages are against stock at its honest best - its ALLCALL7.TXT known-call lookup switched off, since the lookup itself rejects 169 real decodes on the night hour; the second row is stock as it ships. Stock and WSJT-X print no per-period time: their column is wall time per period.

Night hour:

preset	decodes (vs stock)	RX s	TX bg s	total s (vs stock)	not in stock	stock only
Stock JTDX v2.2.159 FT4 without ALL-CALL7.TXT (single thread) - the baseline	1525	1.40	-	1.4	-	-
Stock JTDX v2.2.159 FT4 as shipped, ALL-CALL7.TXT present (single thread)	1356 (-11.1 %)	1.35	-	1.4 (-3 %)	1	170
This decoder without the hint memory (JTDX's FT4 recipe on the fork's code)	1552 (+1.8 %)	0.09	-	0.1 (-94 %)	79	52
Default: deep, four passes	1737 (+13.9 %)	0.10	-	0.1 (-93 %)	255	43
Best power: background 3	1822 (+19.5 %)	0.10	0.3	0.4 (-73 %)	316	19
Recommended (best value): background 6 + deep OSD + alternate pass + residual	1881 (+23.3 %)	0.10	2.1	2.2 (+59 %)	370	14
Most at reply time: six members + the background's extras	1878 (+23.1 %)	0.48	0.5	1.0 (-30 %)	365	12
Max effort: budget-auto members + low thresholds; everything + low thresholds in the background	1888 (+23.8 %)	0.59	0.6	1.2 (-14 %)	374	11
WSJT-X 3.0.2 FT4 decoder, depth 3, AP, 12 FFT threads	1561 (+2.4 %)	0.14	-	0.1 (-90 %)	99	63
WSJT-X improved 3.2.0 FT4 decoder, depth 3, AP, 12 FFT threads	1688 (+10.7 %)	0.22	-	0.2 (-85 %)	207	44

Day

hour (73 periods - the sparse-versus-busy check, too short to rank recipes two decodes apart):

preset	decodes (vs stock)	RX s	TX bg s	total s (vs stock)	not in stock	stock only
Stock JTDX v2.2.159 FT4 without ALL-CALL7.TXT (single thread) - the baseline	507	1.49	-	1.5	-	-
Stock JTDX v2.2.159 FT4 as shipped, ALL-CALL7.TXT present (single thread)	494 (-2.6 %)	1.48	-	1.5 (-1 %)	0	13
This decoder without the hint memory (JTDX's FT4 recipe on the fork's code)	516 (+1.8 %)	0.10	-	0.1 (-93 %)	15	6
Default: deep, four passes	543 (+7.1 %)	0.11	-	0.1 (-92 %)	43	7
Best power: background 3	553 (+9.1 %)	0.11	0.3	0.4 (-72 %)	50	4
Recommended (best value): background 6 + deep OSD + alternate pass + residual	561 (+10.7 %)	0.11	2.5	2.6 (+72 %)	55	1
Most at reply time: six members + the background's extras	560 (+10.5 %)	0.52	0.6	1.1 (-28 %)	56	3
Max effort: budget-auto members + low thresholds; everything + low thresholds in the background	560 (+10.5 %)	0.59	0.6	1.2 (-17 %)	56	3
WSJT-X 3.0.2 FT4 decoder, depth 3, AP, 12 FFT threads	515 (+1.6 %)	0.23	-	0.2 (-84 %)	23	15
WSJT-X improved 3.2.0 FT4 decoder, depth 3, AP, 12 FFT threads	536 (+5.7 %)	0.32	-	0.3 (-79 %)	40	11

The tables in full: reply-phase and background decodes, and their times

The compact tables above give one number per engine. The runs behind them separate what the decoder found *before the reply decision* from what it found *afterwards, in the idle time*, and time the two phases separately. The full tables, regenerated by the same `analyse.py` run as the compact ones (the originals live in the project notes sections 13 and 13a):

- **decodes** - distinct messages per period, summed over the periods (a message printed twice in a period, say by the RX phase and again with a marker, counts once); the percentage is against the baseline row.
- **RX msgs** - the messages decoded in the RX phase, i.e. on the table when the reply is decided. **TX bg msgs** - the messages the TX background added afterwards, while the station transmits; they arrive too late for that period's reply but feed the hint memory and the next period's decode, and they are what the | marker shows in the decoded-text window. For engines without a background the two columns are the total and "-".
- **RX s mean (max)** - the RX phase's wall seconds per period, measured by the decoder itself (`<rxs>` on its `<DecodeFinished>` line), mean over the periods and the worst period. This is the number to hold against the reply deadline: FT8 must decide its reply about 2.7 s into the 15 s period (the presets aim at 1.1-1.3 s), FT4 within 1.36 s of its 7.5 s period. A recipe whose *max* exceeds the deadline loses replies on its worst periods even when its mean is fine.
- **TX bg s mean (max)** - the background phase's wall seconds per period (`<secs>` on its `<BackgroundFinished>` line). Its window is the rest of the period, or two periods when the next one is our own transmission (P7), less a margin (1.0 s; 0.5 s for FT4's max effort). In file mode the background is unclocked, so these are the phase's *full* costs; on air the clock stops a phase that would not fit, so a mean above the window means "the clock is cutting it", not "the reply is late".
- **total s/period** - RX mean + background mean, the CPU the recipe takes from the period.
- **only it, not baseline** / **baseline only** - the set differences against the baseline row over the same periods; **periods better** / **worse** - how many periods the row out-decoded the baseline and how many it fell short.
- **Engines without per-period times** (stock JTDX, WSJT-X) show the run's wall time divided by the periods in the RX column, with no max. Stock JTDX is single-threaded in FT4 and runs its own section threading in FT8; WSJT-X runs its multi-thread FT8 decoder on 12 threads and its FT4 decoder single-threaded. Compare their message counts with ours, not their seconds.

FT8, the on-air hour (240 periods, 100-3100 Hz, 12 threads; baseline = stock v2.2.159 without its ALLCALL7.TXT lookup at its GUI's 12 threads - the real binary through the file-mode patch of `test/decode/stock/` - which is what the percentages of the preset menu are calibrated against since 2026-09-05; the row "this decoder at JTDX's own recipe" is what the tables measured against before that date):

preset	decodes (vs stock)	RX msgs	TX bg msgs	RX s mean (max)	TX bg s mean (max)	total s/period (vs stock)	only it, not stock	stock only	periods better / worse
Stock JTDX v2.2.159: 9 cycles, sensitivity 2, without ALL-CALL7.TXT, 12 threads - the baseline	4869	4869	-	1.40 (wall/period)	-	1.4	-	-	-
Stock JTDX v2.2.159: 9 cycles, sensitivity 2, ALL-CALL7.TXT present, 12 threads (its GUI's auto on this machine)	4783 (-1.8 %)	4783	-	1.39 (wall/period)	-	1.4 (-1 %)	40	126	20 / 72
Stock JTDX v2.2.159: 9 cycles, sensitivity 2, without ALL-CALL7.TXT (single thread)	4866 (-0.1 %)	4866	-	2.04 (wall/period)	-	2.0 (+45 %)	31	34	22 / 20
Stock JTDX v2.2.159 as shipped: 9 cycles, sensitivity 2, ALL-CALL7.TXT present (single thread)	4802 (-1.4 %)	4802	-	2.03 (wall/period)	-	2.0 (+45 %)	30	97	17 / 59
Stock JTDX v2.2.159 at its deepest: 9 cycles, sensitivity 2, SWL	4793 (-1.6 %)	4793	-	1.55 (wall/period)	-	1.6 (+11 %)	148	224	60 / 104

union of all 17 sets is 6450 messages; Stock JTDX v2.2.159 reaches 75.5 %, This decoder at JTDX's own recipe reaches 81.4 %, Pipeline max decodes light reaches 94.4 %, Pipeline ensemble full reaches 96.2 %.

Reading it: the recommended preset spends 1.13 s deciding the reply and 4.2 s of the remaining 14 s on the background, and 1229 of its messages stock never finds; the pipeline ensemble's 11-13 s background only fits when the next period is our own transmission, which is why its presets are "best results" and "max effort" rather than the recommendation. Stock's 2.0 s a period at one thread (1.4 s at twelve) is all reply-phase time, since it has no background.

FT4, the night hour (240 periods of 7.5 s, 100-3100 Hz, 12 threads, baseline = stock without its ALLCALL7.TXT lookup):

preset	decodes (vs stock)	RX msgs	TX bg msgs	RX s mean (max)	TX bg s mean (max)	total s/period (vs stock)	only it, not stock	stock only	periods better / worse
Stock JTDX v2.2.159 FT4 without ALL-CALL7.TXT (single thread) - the baseline	1525	1525	-	1.40 (wall/period)	-	1.4	-	-	-
Stock JTDX v2.2.159 FT4 as shipped, ALL-CALL7.TXT present (single thread)	1356 (-11.1 %)	1356	-	1.35 (wall/period)	-	1.4 (-3 %)	1	170	1 / 39
This decoder without the hint memory (JTDX's FT4 recipe on the fork's code)	1552 (+1.8 %)	1552	-	0.09 (0.18)	-	0.1 (-94 %)	79	52	51 / 29
Default: deep, four passes	1737 (+13.9 %)	1737	-	0.10 (0.67)	-	0.1 (-93 %)	255	43	138 / 12
Best power: back-ground 3	1822 (+19.5 %)	1761	61	0.10 (0.19)	0.3 (0.6)	0.4 (-73 %)	316	19	167 / 6
Recommended (best value): back-ground 6 + deep OSD + alter-nate pass + resid-ual	1882 (+23.3 %)	1763	118	0.10 (0.19)	2.1 (4.9)	2.2 (+59 %)	370	14	182 / 3
Most at reply time: six mem-bers + the back-ground's extras	1878 (+23.1 %)	1837	41	0.48 (1.08)	0.5 (1.1)	1.0 (-30 %)	365	12	185 / 2
Max effort: budget-auto	1888 (+23.8 %)	1855	33	0.59 (1.24)	0.6 (1.2)	1.2 (-14 %)	374	11	187 / 2

The

union of all 10 sets is 2008 messages; Stock JTDX v2.2.159 FT4 without ALLCALL7.TXT (single thread) - the baseline reaches 75.9 %, Recommended (best value) reaches 93.7 %, Max effort reaches 94.0 %.

FT4, the day hour (73 periods):

preset	decodes (vs stock)	RX msgs	TX bg msgs	RX s mean (max)	TX bg s mean (max)	total s/period (vs stock)	only it, not stock	stock only	periods better / worse
Stock JTDX v2.2.159 FT4 without ALL-CALL7.TXT (single thread) - the baseline	507	507	-	1.49 (wall/period)	-	1.5	-	-	-
Stock JTDX v2.2.159 FT4 as shipped, ALL-CALL7.TXT present (single thread)	494 (-2.6 %)	494	-	1.48 (wall/period)	-	1.5 (-1 %)	0	13	0 / 4
This decoder without the hint memory (JTDX's FT4 recipe on the fork's code)	516 (+1.8 %)	516	-	0.10 (0.16)	-	0.1 (-93 %)	15	6	9 / 4
Default: deep, four passes	543 (+7.1 %)	543	-	0.11 (0.49)	-	0.1 (-92 %)	43	7	27 / 3
Best power: back-ground 3	553 (+9.1 %)	544	9	0.11 (0.17)	0.3 (0.5)	0.4 (-72 %)	50	4	32 / 2
Recommended (best value): back-ground 6 + deep OSD + alter-nate pass + resid-ual	561 (+10.7 %)	545	16	0.11 (0.17)	2.5 (5.0)	2.6 (+72 %)	55	1	36 / 1
Most at reply time: six mem-bers + the back-ground's extras	560 (+10.5 %)	557	3	0.52 (0.89)	0.6 (1.1)	1.1 (-28 %)	56	3	36 / 1
Max effort: budget-auto	560 (+10.5 %)	557	3	0.59 (1.00)	0.6 (1.2)	1.2 (-17 %)	56	3	36 / 1

The

union of all 10 sets is 583 messages; Stock JTDX v2.2.159 FT4 without ALLCALL7.TXT (single thread) - the baseline reaches 87.0 %, Recommended (best value) reaches 96.2 %, Max effort reaches 96.1 %.

Reading them: every FT4 tier decides its reply in 0.10 s except the two that spend members at reply time (0.48 and 0.59 s mean, 1.1 and 1.2 s worst - inside the 1.36 s deadline on every period of both hours); the recommended tier's 2.1 s background is the largest idle-time spend and finds 118 of its 1881 late; max effort's budget-auto count admitted all six members on every period here. Stock FT4 takes 1.4 s a period on one thread, which is its whole reply phase, against the deadline of 1.36 s.

6. Reproducibility and tests

Every number here comes from the file mode and is reproducible to the message, and the audio behind them is published: both recorded hours and the two crowded-band files, converted to 16 bit, with a cross-platform script that replays them through the standalone `jtdxjt9` preset by preset and prints these same columns, at **<https://ce3tsk.com/download/wav/>**. It prints its numbers beside this machine's, so another machine can be measured against the tables below before a preset is chosen for it.

The full measurement harness is not part of the published source repository. In it, the regression `regress.sh` (43 rows: presets, thread counts, 16/32-bit, the AGC metric, slice overrides, feedback across periods, ensemble, pipeline, the 100-3100 Hz rows, the experiment hooks; every message printed once) and `bgabort.sh` (abort, one- and two-period budgets, a short window, restore, markers) run in a few minutes; the C++ harness (`test_preset`, `test_budget`, `test_range`, `test_label`, `test_marker`, ...) and the Fortran unit tests (OSD equivalence, the perturbations, packing) cover the rules and the pieces; its two READMEs describe them.

7. Operating it (WW Digi)

Decode bandwidth 100-3100 Hz; AGC compensation off; TX background ensemble effort *auto*; *pipeline run* when running, *pipeline ensemble* for search and pounce; *write decoded debug* on for the first evening, so ALL.TXT shows "Decode skipped: own TX period" and the background lines through the TX period. The operating notes are in the project notes.

7a. Why nothing decodes for a period after a band or mode change

Asked on the air 2026-08-30: the waterfall paints signals on the new band at once, but the decoder prints nothing for a period, sometimes two - "it feels like it needs to fill up something". It is the opposite: something is deliberately emptied, and this is stock JTDX behaviour, not a change of this project.

The GUI records how many seconds into the current period the change happened (`m_nsecBandChanged`, mainwindow.cpp ~7917 - set for a **mode** change as well as a band change, the test is `oldband != newband || m_oldmode != m_mode`). The decoder then blanks that much audio off the front of its buffer (decoder.f90 159-170):

```
nsamplesdel = params%nsecbandchanged * 12000
FT8: if nsec > 14 -> dd8 = 0.   else dd8(1:nsamplesdel) = 0.
FT4: if nsec > 6  -> dd4 = 0.   else dd4(1:nsamplesdel) = 0.
```

Those samples were recorded on the old dial and decoding them yields garbage and false decodes. Because an FT8 transmission needs its full 12.6 s, a period with any of its front blanked usually decodes nothing at all; **FT4 is harsher still** - past 6 s of its 7.5 s period the whole buffer goes, so nearly any FT4 change costs the entire period.

The waterfall is unaffected because it is drawn from `ss`/`savg`, the spectrum taken straight off the incoming audio, which is never blanked - only `dd8`/`dd4`, the decoder's copy, are. Hence the asymmetry: signals visible, nothing printed.

Measured on the mode change of 2026-08-30 02:10:49 (FT4 → FT8): period 021030 gave 16 decodes, period 021045 gave **none at all** - it is simply absent from ALL.TXT - and 021100 was back to 16.

The period *after* is clean audio but runs without the hint memory and the hashed-callsign tables, both cleared as stale on a band or mode change (decoder.f90 143, `ft4hint_clear`, and JTDX's `ihash22=-1`): they are keyed to the old band's frequencies. That costs the hint contribution - on FT4 the +12.5 % of item 48 - until `fillhash` and the lists refill over the next few periods. That part really is "filling up", just not a buffer.

No notice is printed for this: it happens on every band and mode change, and the operator keeps a clean log.

8. The documents behind this one

the project notes (the campaign, steps 1-15), the project notes, `IMPROVEMENTS_WW_DIGI_2026.pdf` (the list), and the harness's `preset_data.tsv`, `reference_set.tsv` and the project notes.