# “Odin” Console Implementation Notes



# Software Structure

## Diagram



## Concept for Operation

The serial queue to / from the PC will use normal Arduino library code. On TX, it will simply transfer the requested data. On RX, the CAT messages will be decoded and passed to the appropriate handlers.

The pushbutton, and encoder handlers will simply provide events to the CAT handlers. Each will be assigned by config variables to one CAT handler, so they know where to pass data to. LEDs will provide their ids to CAT handlers from that data; thereafter the CAT handlers will set them lit or not.

The display will have several screens, changed under its control. It can also originate commands to the handlers. It is known to be inefficient to write data to the display that isn’t on the current screen: so some awareness of which screen is active is needed in the Arduino. “redraw” and “update” may be relevant for each screen.

An operation could be as follows:

* Volume up encoder event
* Encoder informs the “AF gain” handler of a +1 click event
* AF gain handler requests current gain
* When AF gain handler gets current gain from PC, it increments it and sends a new request

# List of Control Types

(Taken from the original specification document)

The list of functions that needs to be assignable to controls is as follows:

|  |  |
| --- | --- |
| Pushbuttons (including encoder “press”)   * A/B VFO select * MOX * TUNE * AF MUTE * Filter reset * Band + * Band – * Mode + * Mode – * AGC speed * NB step * NR step * SNB on/off * ANF on/off * RIT on/off * RIT + * RIT – * A>B * B>A * A/B swap * Split * CTUNE * Lock * Radio Start/Stop * Squelch on/off * Attenuation Step * VOX on/off * Diversity fast/slow step * Compander on/off * Puresignal on/off * Puresignal two tones test * Puresignal single cal * MON on / off | Indicators (including illuminated pushbuttons & LCD)   * MOX * TUNE * RIT on * Split selected * CTune selected * Lock selected * NB off/on * NR off/on * SNB off/on * ANF off/on * Squelch on/off * VFO A/B * Compander on/off * Puresignal on/off * LED lit if encoder 2nd function selected   Encoders   * AF channel gain * Master AF gain * AGC * Filter high cut * Filter low cut * Drive * Mic Gain * VFO A tune * VFO B tune * VOX gain * VOX delay * CW sidetone * CW speed * Squelch * DiversityGain * DiversityPhase * Multifunction * Compander threshold * RX1 AF gain * RX2 AF gain * RX1 RF gain (atten) * RX2 RF gain (atten) |

# Initial Settings for Controls

All controls, other than the VFO encoder, can be reallocated by the user to any function. The “factory default” assignment, noting which push switches are illuminated, is as follows:

## Encoder functions

|  |  |  |
| --- | --- | --- |
| **Encoder** | **Main function** | **2nd Function** |
| 2A | AF Gain | AF Gain |
| 2B | AGC threshold | AGC threshold |
| 3A | Filter high | Filter high |
| 3B | Filter low | Filter low |
| 4A | Drive (?to become Diversity?) | Drive |
| 4B | Mic Gain (?Diversity?) | Mic Gain |
| 5A | Multifunction | Multifunction |
| 5B | Drive | Drive |

(This gives the end result of each being single function)

## Indicator/switch functions

|  |  |  |  |
| --- | --- | --- | --- |
| **Switch number** | **Indicator** | **Digital pin** | **Initial function** |
| SW1 | LED1 | 30 | Toggle VFO A / VFO B |
| SW2 | LED2 | 31 | MOX |
| SW3 | LED3 | 32 | TUNE |
| SW4 | LED4 | 33 | Click Tune |
| SW5 | LED5 | 34 | VFO LOCK |
| SW6 |  | 35 | A>B |
| SW7 |  | 36 | B>A |
| SW8 |  | 37 | SPLIT operation |
| SW9 | LED6 | 38 | RIT on |
| SW10 |  | 39 | RIT step up |
| SW11 |  | 40 | RIT step down |
| SW12 |  | 41 | Band down |
| SW13 |  | 42 | Mode down |
| SW14 |  | 43 | Radio start/stop |
| SW15 |  | 44 | Band up |
| SW16 |  | 45 | Mode up |
| Encoder 2 push |  | 6 | AF MUTE |
| SW17 | LED7 | 9 | NR |
| Encoder 3 push |  | 12 | Filter Reset |
| Encoder 4 push |  | 23 | (No function) |
| Encoder 5 push |  | 29 | Encoder action (for multi) |

Note Encoder 1 is the VFO encoder and has no pushbutton)

# CAT Messaging

Ideally I should have an LED to show “console in use” ie successful message exchange with PowerSDR. At power up, could send a “request VFO A frequency” message and await response. Would that sit in the serial queue?

Might want to consider if(Serial())….. or if (TX queue != empty)….. to know that we’ve attempted a connection.

Need to allow for the possibility of messages getting corrupted, and needing timeouts / retry

## Information Display

There needs to be a periodic scan to update the console for any settings that are displayed (either by LCD or LED). Strictly that might only be needed for those things currently on the display. Aim to update everything every 2-5 seconds?

LED display of TUNE and MOX should be for locally initiated commands. The reason being: a MOX initiated by CAT can’t be cancelled from the PC end; so an indication that “I’ve initiated it from here” would be useful.

CAT 1.5 second update sequence, 1 message per 220ms:

Frequency

S meter/TX power

VFO combined status

Frequency

S Meter/TX power

RX Combined status

S meter/TX power

Mode

Band, AGC threshold – request on demand

There needs to be protection against data that has been requested from the PC but not received yet overwriting new data from the console. Essentially we need to cancel an unactioned request that would result in the arrival of state data. Proposed solution: when the data is periodically requested – set a bool flag; when the data arrives, only process it if that bool flag is active. And when we send new CAT data eg from a pushbutton, clear that flag for that datatype. At the moment this applies to:

* Frequency (ZZFA/FB);
* VFO status (ZZXV);
* RX status ((ZZXN/XO);
* Mode (ZZMD).

# Frequency value as a text field

A frequency value arrives from CAT as an 11 digit text field, zero added meaning the frequency in Hz. We need to display a frequency in MHz, and the user can enter a frequency in MHz on the touchscreen. We need to be able to convert between them!

If > 10MHz:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Frequency from CAT | 0 | 0 | 0 | 1 | 4 | 3 | 2 | 4 | 5 | 6 | 7 | X | (X=terminating zero) |
| Displayed | 1 | 4 | . | 3 | 2 | 4 | 5 | 6 | 7 | X |  |  |

If < 10MHz:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Frequency from CAT | 0 | 0 | 0 | 0 | 3 | 6 | 2 | 4 | 5 | 6 | 7 | X |
| Displayed | 3 | . | 6 | 2 | 4 | 5 | 6 | 7 | X |  |  |  |

When the string has been edited by the user “enter frequency” screen We can’t assume any format. There is too much to go wrong treating this as an ASCII text manipulation problem - so need to convert as a floating point number.

Suggested approach:

1. For CAT received frequencies:
   1. Start with CAT string as above
   2. Convert to integer, Hz
   3. Convert to float; divide by 1E6; convert to text. Send to display. ftoa()
2. For a frequency step (VFO encoder):
   1. Send the step command to CAT
   2. Update the integer Hz frequency & store.
   3. Convert to float; divide by 1E6; convert to text. Send to display. ftoa()
3. For a user entered frequency:
   1. Start with a text value in MHz
   2. Convert to float; multiply by 1E6; store Hz frequency. atof()
   3. Create zero padded string for CAT message.
   4. Convert to float; divide by 1E6; convert to text. Send to display.

To speed things up: we could give the display both the string and the frequency in Hz. It could use the integer to decide if the number displayed is different, and the string needs to be updated. It will still need persistent storage of the string so it can redraw itself.

ftoa(float input, char\* buffer, int numdecimalplaces)

float atof(char\* buffer) returns 0.0 if no number found

## CAT Commands To be Parsed

|  |  |  |  |
| --- | --- | --- | --- |
| **Control effect** | **CAT message** | **Notes** | **Response Case** |
| Set Master AF gain | Get: ZZAG;  Set: ZZAGnnn; | nnn=000 to 100; meaning a percentage value.  ZZAG065; sets to 65% | 2 |
| Set A, B AF gain | Get RX1: ZZLA;  Get RX2: ZZLE;  Set RX1: ZZLAnnn;  Set RX2: ZZLEnnn; | nnn=000 to 100; meaning a percentage value. | 2  Modify for direct A,B controls |
| Set/display A, B attenuation | Get RX1: ZZPA;  Get RX2: ZZPB;  Set RX1: ZZPAn;  Set RX2: ZZPBn; | 10dB steps only, & settings h/w dependent  n=0: -20dB n=1: 0dB  n=2: -10dB; n=4: -30dB  do a “get” after setting to find outcome. | 2 |
| Set/display A/B AGC threshold | Get RX1: ZZAR;  Get RX2: ZZAS;  Set RX1: ZZARnnnn;  Set RX2: ZZASnnnn; | nnnn=-020 to +120 (with mandatory sign) | 3 |
| Set/display A/B AGC speed | Get RX1: ZZGT;  Get RX2: ZZGU;  Set RX1: ZZGTn;  Set RX2 ZZGUn; | n=0: fixed; n=1: long; n=2: slow; n=3: med;  n=4: fast; n=5: custom | 2 |
| Set filter low (possible display) | Get RX1: ZZFL;  Get RX2: ZZFS;  Set RX1: ZZFLnnnnn;  Set RX2: ZZFSnnnnn; | nnnnn=-9999 to +9999 (in Hz, with sign) | 3 |
| Set filter high cut (possible display) | Get RX1: ZZFH;  Get RX2: ZZFR;  RX1: ZZFHnnnnn;  RX2: ZZFRnnnnn; | nnnnn=-9999 to +9999 (in Hz, with sign) | 3 |
| Set drive | Get: ZZPC;  Set: ZZPCnnn; | nnn = 000 to 100 | 2 |
| Set mic gain | Get: ZZMG;  Set: ZZMGnnn; | nnn= -50 to 070. No sign for +. | 4 |
| Set / display VFO A/B frequency | Get RX1: ZZFA;  Get RX2: ZZFB;  Set:  RX1: ZZFAnnnnnnnnnnn;  RX2: ZZFBnnnnnnnnnnn; | nnnnnnnnnnn: 11 digit frequency in Hz  14.379123MHz = 00014379123 | 2 |
| Increment VFO A/B frequency by ± N steps | (no get)  Set VFO A +: ZZAFnn;  Set VFO A -: ZZAEnn;  Set VFO B +: ZZBFnn;  Set VFO B -: ZZBEnn; | nn=0-99 steps | n/a |
| Set VOX gain | Get: ZZVG;  Set: ZZVGnnnn; | nnnn=0-1000 | 2 |
| Set VOX delay | Get: ZZXH;  Set: ZZXHnnnn; | nnnn = 0-4000 | 2 |
| Vox On/Off | Get: ZZVE;  Set: ZZVEn; | n=0: VOX OFF; n=1: VOX ON; | 2 |
| Set CW sidetone freq | Get: ZZCL;  Set: ZZCLnnnn; | nnnn=0200 to 2250 (units Hz) | 2 |
| Set CW speed | Get: ZZCS;  Set: ZZCSnn; | nn=01 to 60 | 2 |
| Set/display MOX state | Get: ZZTX;  Set: ZZTXn; | n=0: RX; n=1: TX | 2 |
| Set display TUNE state | Get: ZZTU;  Set: ZZTXn; | n=0: RX; n=1: TX | 2 |
| Set/display A/B band | Set RX1 down: ZZBD;  Set RX1 up: ZZBU;  Get RX1: ZZBS;  Set RX1: ZZBSnnn;  Set RX1 down: ZZBA;  Set RX1 up: ZZBB;  Get RX2: ZZBT;  Set RX2: ZZBTnnn; | BD/BU step down/up in frequency  BA/BB step down/up  nnn: 160,080,060, 040, 030, 020, 017, 015, 012, 010, 006, 002, 888 (gen) 999 (WWV)  transverters could report V01 through V13, but don’t necessarily appear in step list if not enabled.  after doing a mode up/down, need to do a “get” to check what was selected as a consequence! | 5  (transverter response has non numerical digit) |
| Set/display A/B mode | Get RX1: ZZMD;  Set RX1: ZZMDnn;  Get RX2: ZZME;  Set RX2: ZZMEnn; | nn = 00 (LSB) 01 (USB) 02 (DSB) 03 (CWL) 04 (CWU) 05 (FM) 06 (AM) 07 (DIGU) 08 (SPEC) 09 (DIGL) 10 (SAM) 11 (DRM) | 2 |
| Set/display RIT state | Get: ZZRT;  Set: ZZRTn; | n=0: same freq; n=1: RIT active | 2 |
| Set RIT tune offset up/down | (no Get)  Set+: ZZRU;  Set-: ZZRD;  Set: ZZRDnnnnn; | With no params, a “set” increments or decrements by 10Hz  ZZRDnnnnn or ZZRUnnnnn both set to -9999 to +9999 Hz | 3 |
| Set/display SPLIT state | Get: ZZSP;  Set: ZZSPn; | n=0: no split; n=1: SPLIT active | 2 |
| Set/display CTUN state | Get RX1: ZZCN;  Set RX1: ZZCNn;  Get RX2: ZZCO;  Set RX2: ZZCOn; | n=0: no CTUN; n=1: CTUN active | 2 |
| Set/display LOCK state | Get VFO A: ZZUX;  Set VFO A: ZZUXn;  Get VFO B: ZZUY;  Set VFO B: ZZUYn; | n=0: no lock; n=1: LOCK active | 2 |
| Display S meter | Get RX1: ZZSM0;  Get RX2: ZZSM1;  Show RX1: ZZSM0nnn;  Show RX2: ZZSM1nnn; | nnn=000 to 260  (nnn/2-140) = value in dBm  Example values:  powerSDR says -89dBm: ZZSM0122;  powerSDR says -109dBm: ZZSM00074; | 5  (0/1 digit) |
| Display TX power indication or ALC | Get: ZZRMn;  Show: ZZRMnxxxxxxx  xxxxxxxxxxxxx;  ZZRM4: ALC  ZZRM5: fwd power  ZZRM7: rev power  ZZRM8: VSWR | Response is h/w dependent. When Alex selected in h/w options:  ZZRM4-20.0 dB;  ZZRM50 W;  ZZRM70 W;  ZZRM81.0 : 1;  when Alex not selected: ZZRM50.00 W;  (I presume all are zero padded) | 5  (complex response) |
| Set/display NR mode | Get RX1: ZZNR;/ZZNS;  Get RX2: ZZNV;/ZZNW;  Set RX1: ZZNRn;/ZZNSn;  Set RX2: ZZNVn;/ZZNWn; | NR off: ZZNR0; ZZNS0;  NR: ZZNR1; ZZNS0;  NR2: ZZNR0; ZZNS1;  (RX2 - similarly ZZNV/ZZNW) treat as a pair | 2 |
| Set/display NB mode | Get RX1: ZZNA;/ZZNB;  Get RX2: ZZNC;/ZZND;  Set RX1: ZZNAn; ZZNBn;  Set RX2: ZZNCn; ZZNDn; | NB off: ZZNA0; ZZNB0;  NB: ZZNA1; ZZNB0;  NB2: ZZNA0; ZZNB1;  (RX2 - similarly ZZNC/ZZND) treat as a pair | 2 |
| Set/display SNB mode | Get: RX1: ZZNN;  Get: RX2: ZZNO;  Set: RX1: ZZNNn;  Set: RX2: ZZNOn; | n=0: SNB off; n=1: SNB on | 2 |
| Set/display ANF mode | Get RX1: ZZNT;  Get RX2: ZZNU;  Set RX1: ZZNTn;  Set RX2: ZZNUn; | n=0: ANF off; n=1: ANF on | 2 |
| Get Combined RX Status | Get RX1: ZZXN;  Get RX2: ZZXO;  RX1 Ans: ZZXNnnnn;  RX2 Ans: ZZXOnnnn;  nnnn=0 to 8191 | Combines reporting of NB1/2, NR1/2, SNB, ANF, AGC, Atten, Squelch  Bits 2-0: AGC Speed (see ZZGT/GU)  Bits 5-3: Attenuation (see ZZPA/PB)  Bit 6: Squelch on/off (see ZZSO/SV)  Bit 7: NB0 (see ZZNA/NC)  Bit 8: NB1 (see ZZNB/ND)  Bit 9: NR0 (see ZZNR/NV)  Bit 10: NR1 (see ZZNS/NW)  Bit 11: SNB (see ZZNN/NO)  Bit 12: ANF (see ZZNT/NU) |  |
| Combined VFO Status | Get: ZZXV;  Ans: ZZXVNNN;  NNN = 0 - 255 | Combines reporting of RIT, LOCK, SPLIT, CTUNE, MOX and TUNE status  Bit 0: RIT on/off (see ZZRT)  Bit 1: VFO A LOCK status (see ZZUX)  Bit 2: VFO B LOCK status (see ZZUY)  Bit 3: SPLIT status (see ZZSP)  Bit 4: VFO A CTUNE status (see ZZCN)  Bit 5: VFO B CTUNE status (see ZZCO)  Bit 6: MOX status (see ZZTX)  Bit 7: TUNE status (see ZZTU) |  |
| Set/clear A/B mute | Get RX1: ZZMA;  Get RX2: ZZMB;  Set RX1: ZZMAn;  Set RX2: ZZMBn; | n=0: no mute; n=1: MUTE on | 2 |
| Radio START | Get: ZZPS;  Set: ZZPSn; | n=0: radio OFF; n=1: radio ON | 2 |
| Reset filters to defaults | No new message – just send out new low, high |  |  |
| Squelch level | Get RX1: ZZSQ;  Get RX2: ZZSX;  Set RX1: ZZSQnnn;  Set RX2: ZZSXnnn; | nnn= 160-0; it means -160 to 0 | 2 |
| Squelch on/off | Get RX1: ZZSO;  Get RX2: ZZSV;  Set RX1: ZZSOn;  Set RX2: ZZSVn; | n=0: squelch OFF; n=1: squelch ON | 2 |
| VFO copy/swap | (not get)  Set: ZZVSn; | n=0: A>B; n=1: B>A; n=2: swap | 2 |
| Get VFO tuning step | Get: ZZAC;  Set: ZZACnn; | nn=0 to 24, encoding a step size that will need a table lookup |  |
| Diversity on/off | Get: ZZDE;  Set: ZZDEn; | N=0: diversity off; n=1: diversity on. |  |
| Diversity RX1 gain | Get: ZZDG;  Set: ZZDGnnnn; | nnnn=0 to 5000, for 0.000 to 5.000 |  |
| Diversity RX2 gain | Get: ZZDC;  Set: ZZDCnnnn; | nnnn=0 to 5000, for 0.000 to 5.000 |  |
| Diversity phase | Get: ZZDD;  Set: ZZDDnnnnnn; | nnnnnn=-18000 to +18000, with mandatory sign. Meaning -180.00 to +180.00 degrees. |  |
| Diversity reference source | Get: ZZDB;  Set: ZZDBn; | n=0: receiver 2; n=1: receiver 1 |  |
| Diversity receiver source | Get: ZZDH;  Set: ZZDHn; | n=0: RX1 + RX2  n=1: RX1  n=2: RX2 |  |
| Set A, B RF gain (atten) | Get RX1: ZZRX;  Get RX2: ZZRY;  Set RX1: ZZRXnn;  Set RX2: ZZRYnn; | nn=00 to 31; meaning an attenuation value in dB. | Similar to A/B AF gain? |
| Compander on/off | Get: ZZCP;  Set: ZZCPn; | n=0: off  n=1: on | Sim to diversity ref ZZDB |
| Compander threshold | Get: ZZCT;  Set: ZZCTnn; | nn is the threshold value, in dB (0-20) | Sim to diversity RX1 gain ZZDG |
| Puresignal on/off | Get: ZZLI;  Set: ZZLIn; | n=0: off  n=1: on | Sim to diversity ref |
| Puresignal two-tone test | Get: ZZUT;  Set: ZZUTn; | n=0: off  n=1: on | Sim to diversity ref |
| Puresignal single cal | Set: ZZUS; | No parameters | Sim to VFO swap zzvs |
| MON on/off | Get: ZZMO;  Set: ZZMOn; | n=0: off  n=1: on | Sim to diversity ref |

* There are few commands with no “get” but the control code should know them.
* For parsing there are 5 cases:

1. No parameters (send only – never happens for messages to console);
2. Unsigned parameters with a known number of digits;
3. Signed parameters, known number chars, with a sign always present;
4. Signed parameters, known number of chars, with a sign present only for negative.
5. “special cases” eg ZZRM & band display

# Handler Algorithm

### Type 1 - Set Relative eg Gain Set:

(We need a recent value to be able to send the new setting; recent = 3 seconds)

When encoder turned:

If (recent value available)

{

Calculate new gain

Send message

Update the local value

Restart recent counter

}

Else

{

Increment /decrement the stored step count

Send gain request command

Start timeout count

}

Received msg handler()

Clear timeout

Parse current gain value

Set recent count

If (there is a stored click count)

{

Add/subtract step count

Clip result

Send message

Store new value

}

// this should run after the RX message handler

Timeout tick()

If (timeout active)

{

Decrement count

If (count == 0)

{

re-send command

restart timeout

}

}

### Type 2 - Set Absolute, One way (eg VFO steps):

(we can send the new setting straight away, and no response needed)

When encoder turned:

{

Send VFO step command

}

### Type 3 – Set Absolute, Data also displayed (eg NR setting)

(we can send the new value, but we also need periodic updates)

When data changed:

{

Send CAT command to set new value

Store value for local use

Set recency count

}

Periodically re-request data

If a request is active when data sent

don’t store that data when it is received

### Type 4-Display only (eg S Meter)

If (StaleCountExpired)

{

Reload stale count;

Send request message;

}

Else

Decrement stale count;

When message arrives:

{

Store data

Offer to display

}

## Non-persistent parameters

Some parameters are persistently stored, and frequently updated via continual CAT “polling”. This includes Frequency, RX settings, VFO settings, S meter and mode. This creates a lot of message traffic, so only frequently needed data is in this category. Others need to be requested on demand.

### Band Setting

The “band” value is only needed for the display when the band screen is opened; so it can be called on demand. The band can also be set by “band up” and “band down” pushbutton commands. Those result in a visible response (ie frequency change) but formally the band itself isn’t reported as a CAT message. To send a band up or band down does not require the current band to be known. There’s no immediately obvious reason why we would have a “recent” value so the concept of having a “recent” band value seems wrong.

For a button event this looks quite simple:

|  |  |  |  |
| --- | --- | --- | --- |
| **Button / encoder** | **Display** | **CAT handler** | **10ms Tick** |
| Band “+” button pressed | No action | Send “band up” CAT | No action |
| Band “-” button pressed | No action | Send “band down” CAT | No action |

For a display event this might be more involved:

|  |  |  |
| --- | --- | --- |
| **Display** | **CAT handler** | **10ms Tick** |
| “band” screen opened. No buttons are set. |  |  |
| Callback code invoked; requests current band | Initiate a “get band” CAT message with timeout | If timeout happens, re-request |
|  | When get CAT reply: send to display |  |
| Display lights up a button |  |  |
| A different button click callback occurs: call “set band” | CAT handler sends new CAT “set band” message |  |

The display code should not have persistent storage of the band balue!

### Mode Display

This is a slightly different use case: the data is in the periodic request list, so we always have an up-to-date value.

|  |  |  |  |
| --- | --- | --- | --- |
| **Button / encoder** | **Display** | **CAT handler** | **10ms Tick** |
|  |  |  | Periodically send “Get mode” |
|  |  | When CAT msg arrives: store locally; send to display |  |
|  | If different from current: store value; update display |  |  |
| Mode “+” button press |  | Mode++, with wrap  Send CAT message. Store locally.  Send to display. |  |
| Mode “-” button press |  | Mode--, with wrap  Send CAT message. Store locally.  Send to display. |  |
|  | “Mode” screen opens. Callback sets one button from local data. |  |  |
|  | New button callback.  Send CAT update. Set local display value. | Send new CAT message. |  |

### AGC Threshold

This is needed for the display, but infrequently so never held persistently. It can be set by the display, or by an encoder; if the latter, the display may or may not be showing the appropriate page. The concept of “recent data” is relevant because a rapidly turning encoder would otherwise lead to a lot of “get value” requests.

|  |  |  |  |
| --- | --- | --- | --- |
| **Button / encoder** | **Display** | **CAT handler** | **10ms Tick** |
|  | Display RF screen opens. Callback executed. Sends threshold request. | If recent != 0: send to display.  If Recent == 0, get CAT & start timeout. | If timeout expires, re-request. |
|  |  | When CAT response: clear timeout, store locally, set “recent”  send to display | Decrement “recent” count till reaches zero (no other action) |
|  | When data made available, send to screen |  |  |
|  | When screen slider moved: send new value to CAT handler | CATSetAGCThreshold()  Store locally, send CAT set “recent” count. | Decrement “recent” count till reaches zero (no other action) |
| Encoder up/down click:  Set, increment or decrement click count |  | If recent != 0: calc new value, CATSetAGCThreshold() [store locally, send CAT, set “recent” count],  clear click count  Send to Display |  |
|  |  | If recent == 0:  If timeout == 0: get CAT & start timeout. | If timeout decrements to 0, get CAT & start timeout. |
|  |  | When CAT response: clear timeout, store locally, set “recent”  if click count != 0: calc new value, CATSetAGCThreshold() [store locally, send CAT, set “recent” count],  clear click count.  send to display | Decrement “recent” count till reaches zero (no other action) |

No persistent storage needed at the display.

### AGC Gain

This is never displayed, but can be set from an encoder.

(note we use timeout as an indication that there is a request “in flight”)

|  |  |  |  |
| --- | --- | --- | --- |
| **Button / encoder** | **Display** | **CAT handler** | **10ms Tick** |
| Encoder up/down click:  Set, increment or decrement click count |  | If recent != 0: calc new value, store locally, send CAT, set “recent” count, clear click count |  |
|  |  | If recent == 0:  If timeout == 0, get CAT & start timeout. | If timeout decrements to 0, get CAT & start timeout. |
|  |  | When CAT response: clear timeout, store locally, set “recent”  if click count != 0: calc new value, store locally, send CAT, set “recent” count, clear click count. |  |

### Diversity Gain

Diversity gain has a further complication: we need first to find out which RX is used as the diversity reference source by using ZZDB.

(note we use timeout as an indication that there is a request “in flight”)

|  |  |  |  |
| --- | --- | --- | --- |
| **Button / encoder** | **Display** | **CAT handler** | **10ms Tick** |
| Encoder up/down click:  Set, increment or decrement click count |  | If recent != 0: calc new value, store locally, send gain CAT, set “recent” count, clear click count, send to display |  |
|  |  | If recent == 0:  If RX source timeout == 0, get RX source CAT & start timeout. | If RX source timeout decrements to 0, get RX source CAT & start timeout. |
|  |  | When RX source CAT response:  clear RX source timeout; Store result;  get gain CAT & start timeout | If gain timeout decrements to 0, get gain CAT & start timeout. |
|  |  | When gain CAT response: clear timeout, store locally, set “recent”  if click count != 0: calc new value, store locally, send CAT, set “recent” count, clear click count, send to display |  |

Variables & functions used for encoder actions:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Control** | **Local Storage** | **Recent** | **Timeout** | **Request** | **Update** |
| consts |  | Load VRECENTTHRESHOLD | Load VGETTIMEOUT |  |  |
| AGC Threshold | GCatAGCThreshold | GAGCThresholdRecent | GAGCThresholdTimeout | CatRequestAGCThreshold | SendAGCThresholdClicks |
| Filter Low Cut | GCatFilterLow | GFilterLowRecent | GFilterLowTimeout | CatRequestFilterLow | SendFilterLowClicks |
| Filter High Cut | GCatFilterHigh | GFilterHighRecent | GFilterHighTimeout | CatRequestFilterHigh | SendFilterHighClicks |
| Squelch Level | GCatSquelchLevel | GSquelchLevelRecent | GSquelchLevelTimeout | CatRequestSquelchLevel | SendSquelchLevelClicks |
| ~~Channel AF Gain~~ | ~~GCatChanAFGain~~ | ~~GChanAFGainRecent~~ | ~~GChanAFGainTimeout~~ | ~~CatRequestChanAFGain~~ | ~~SendChanAFGainClicks~~ |
| RX1,2 AF Gain | GCatRX1AFGain  GCatRX2AFGain | GRX1AFGainRecent  GRX2AFGainRecent | GRX1AFGainTimeout  GRX2AFGainTimeout | CatRequestRX1AFGain  CatRequestRX2AFGain | SendRX1AFGainClicks  SendRX1AFGainClicks |
| Master AF Gain | GCatMastAFGain | GMastAFGainRecent | GMastAFGainTimeout | CatRequestMastAFGain | SendMastAFGainClicks |
| Drive | GCatDriveLevel | GDriveLevelRecent | GDriveLevelTimeout | CatRequestDriveLevel | SendDriveLevelClicks |
| Mic Gain | GCatMicGain | GMicGainRecent | GMicGainTimeout | CatRequestMicGain | SendMicGainClicks |
| VOX Gain | GCatVoxGain | GVoxGainRecent | GVoxGainTimeout | CatRequestVoxGain | SendVoxGainClicks |
| VOX Delay | GCatVoxDelay | GVoxDelayRecent | GVoxDelayTimeout | CatRequestVoxDelay | SendVoxDelayClicks |
| CW Sidetone | GCatCWTone | GCWToneRecent | GCWToneTimeout | CatRequestCWTone | SendCWToneClicks |
| CW speed | GCatCWSpeed | GCWSpeedRecent | GCWSpeedTimeout | CatRequestCWSpeed | SendCWSpeedClicks |
| Diversity phase | GCatDiversityPhase | GDiversityPhaseRecent | GDiversityPhaseTimeout | CatRequestDiversityPhase | SendDiversityPhaseClicks |
| Diversity gain | GCatDiversityGain | GDiversityGainRecent | GDiversityGainTimeout | CatRequestDiversityGain | SendDiversityGainClicks |
| DiversitySource | GCatDiversitySource |  | GDiversitySourceTimeout | CatRequestDiversitySource |  |
| RX1,2 RF Gain | GCatRX1RFAtten  GCatRX2RFAtten | GRX1RFAttenRecent  GRX2RFAttenRecent | GRX1RFAttenTimeout  GRX2RFAttenTimeout | CatRequestRX1RFAtten  CatRequestRX2RFAtten | SendRX1RFAttenClicks  SendRX1RFAttenClicks |
| Compander on/off | GCatCompanderEnable |  | GCompanderEnableTimeout | CatRequestCompanderEnable |  |
| Puresignal on/off | GCatPuresignalEnable |  | GPuresignalEnableTimeout | CatRequestDiversityEnable |  |
| Puresignaltwotone test | GCatPuresignalTwoTone |  | GPuresignalTwoToneTimeout | CatRequestPuresignalTwoTone |  |
| MON on/off | GCatMonEnable |  | GMonEnableTimeout | CatRequestMonEnable |  |
| Compander Threshold | GCatCompThreshold | GCompThresholdRecent | GCompThresholdTimeout | CatRequestCompThreshold | SendCompThresholdClicks |

Variables and functions used for pushbutton actions:

|  |  |  |  |
| --- | --- | --- | --- |
| **Pushbutton** | **CAT data variable** | **CAT send function** | **Display Show function** |
| NB (step values) | GCatStateNB | CATSetNBState(ENBState) | DisplayShowNBState  (ENRState z) |
| NR (step values) | GCatStateNR | CATSetNRState(ENRState) | DisplayShowNRState  (ENRState z) |
| SNB (toggle) | GCatStateSNB | CATSetSNBState(bool) | DisplayShowSNBState(bool z) |
| ANF (toggle) | GCatStateANF | CATSetANFState(bool) | DisplayShowANFState(bool z) |
| Squelch (toggle) | GCatStateSquelch | CATSetSquelchOnOff(bool) | (not displayed) |
| Atten (step values) | GCatStateAtten | CATSetAttenuation(EAtten) | DisplayShowAtten(EAtten x) |
| AGC speed (step) | GCatStateAGCSpd | CATSetAGCSpeed(EAGCSpeed) | DisplayShowAGCSpeed  (EAGCSpeed x) |
| SPLIT (toggle) | GCatStateSplit | CATSetSplitOnOff(bool) | displayShowSplit(bool x) |
| CTUNE A/B (toggle) | GCatStateACTune  GCatStateBCTune | CATSetCTuneOnOff(bool) | (not displayed) |
| LOCK A/B (toggle) | GCatStateALock  GCatStateBLock | CATSetVFOLock(bool) | DisplayShowLockState(bool x) |

(Note this list is incomplete!)

Remember if(Pressed) {} for each!

Red- need to be written!

# Display, LED Handling

The display and LED code will have “update” timer tick handlers

They should get the current required state from the CAT handlers and update where the information is displayed.

## Nextion Display Coding

* To change between pages in the Nextion itself: just add event handlers “page n” to go to page n
* Only send settings to objects that are visible on the current page. To know the page, on each page, add a pre-initialise event with code “printh 65 <page number> 00 00 01 FF FF FF”. A NexPage object will trap this and note the new page number
* To change pages from the Arduino: use the show function of the nexpage object, eg page0.show(). As far as I can see the preinitialise event from the display does NOT happen under those circumstances.

|  |  |
| --- | --- |
| Page 0: base page  The S meter requires a full size background image. I’ve drawn a 120x120 image using visio but had to use gimp to move the image to top right of a 400x240 image.  Set the display background image. Set the gauge to “crop image” AND set its image to the SAME background image (ie far larger than the gauge). That’s the only way not to have a compile error!  (Presumably I can change both image id values to call up a TX power meter image?)  CAT Information shown on display:  A/B; A/B Frequency; A/B Mode; MOX state; LOCK state; TUNE state; SPLIT state; S meter/power meter |  |
| Page 2: About page  Encoder, pushbutton and indicator buttons call up the same “editing” page. To identify the correct target, they set a global variable on the editing page.  (no CAT data displayed) |  |
| Page 3: Frequency entry  (accessed by clicking the frequency box)  Frequency is edited as a string. The “enter” or “set” button will save the value to be acted upon.  The editing functionality is entirely within the Nextion display. The decimal point has a piece of code to allow it to add characters to the string only if the string doesn’t already have a decimal point.  (no CAT data displayed) |  |
| Page 4: Band select  The band buttons are all dual state buttons. When clicked, they will set the state of all the others to zero (ie unclicked). There is Nextion code to collect into an enum integer variable the current selected band. Programmed an Arduino click handler for ONE button and send that “click” string in the event code for other buttons. The event handler will then read the variable rather than query the button state.  (CAT data: A/B Band needed when screen opens) |  |
| Page 5: Mode select  The mode buttons are all dual state buttons. When clicked, they will set the state of all the others to zero (ie unclicked). There is Nextion code to collect into an enum integer variable the current selected band. Programmed an Arduino click handler for ONE button and send that “click” string in the event code for other buttons. The event handler will then read the variable rather than query the button state.  (CAT data: A/B Mode needed when screen opens) |  |
| Page 6: Noise settings  SNB, ANF are simple dual state buttons. NR and NB need multiple buttons; similar logic to above.  (CAT data: A/B NR, A/B NB, A/B SNB, A/B ANF needed when screen opens) |  |
| Page 7: RF settings  AGC speed and atten are both groups of dual state buttons; similar logic to above.  AGC threshold is a slider. Value range 0 to 140 (CAT value range is -20 to +120, so add/subtract a fixed offset)  (CAT data: A/B AGC speed, A/B AGC threshold, A/B atten needed when screen opens) |  |
| Page 8: General settings  Use dual state buttons again; same logic concept. |  |
| Page 9: Configure  There are 3 variants:  Event handlers needed for +/- buttons. Most of the logic executed in the Arduino.  When I/O +/- buttons clicked to select a new LED/encoder/button, Arduino increments the number and changes the displayed string in the “function” box.  When function +/- buttons clicked, Arduino sends next/previous function to the text box.  When accept clicked, current settings saved in the Arduino.  Function 2 visibility set to not visible unless encoder being edited. |  |
| Page1: I/O test  All logic is at the Arduino end.  When an indicator dual state button clicked: Arduino event handler queries the state then sets LED on/off.  When pushbutton or encoder clicked: a pushbutton or encoder text box has its background colour changed to Green until released. There is a “send command” for this – no methods to the class.  Encoder turn increments the displayed number. |  |

## Encoder Action Texts

There are texts available to show the functions of encoders 1,3,5 & 7. The idea is:

* They show the encoder function where you have two functions per encoder
* If you have single shaft encoders for encoders 1,3 & 5: the text should show main function or 2nd function depending on which is active
* If you have dual shaft encoders: you don’t need texts as you wouldn’t have 3 functions per encoder
* The right hand text is for encoder 7, which is single shaft and assumed to be “multi”
* There ought to be a way to turn them off in 2 groups.

When they are on:

* For the non multi encoder: show main or second function. When 2nd function is activated or deactivated, change the display.
* For the multi encoder: change the display when 2nd function used to change the assigned function. Modify the display with M: preceding the function.

# IF Filter Display

There needs to be a display of the variable IF filter settings. There are several ways in which it could be done.

* Show the -10KHz to +10KHz region, with pixels coloured for the region used. The disadvantage is that for SSB only an eighth of the pixels would be displayed; for CW, only a tiny number.
* Show something like twice the “correct” bandwidth, and let the “correct” values decide where that is centred. That would allow a reasonable number of pixels to be lit.
* A complication is that it would be useful to do this is the audio domain (ie if you hear an interfering LF signal, you want to move the left edge). That would mean the controls need to be reversed for LSB and CWL modes.
* I have access to the “correct” values because of the array of “filter reset” values. Could also add a “display width” to that structure (but then we’d need to vary the green part shown on a mode dependent basis)

|  |  |
| --- | --- |
| CWL, LSB, DIGL |  |
| CWU, USB, DIGU |  |
| Symmetrical  DSB, FM, AM, SAM, SPEC, DRM |  |

I’ve created a simple spreadsheet calculator:



# I/O Pin Allocation



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Pin Name** | **Function** | **Connector** | **Pin Name** | **Function** | **Connector** |
| Digital 0 / RX0 | Reserved USB | PWML | Digital 31 | SW 2 | Digital |
| Digital 1/ TX0 | Reserved USB | PWML | Digital 32 | SW 3 | Digital |
| Digital 2 | Encoder 1 (VFO) A | PWML | Digital 33 | SW 4 | Digital |
| Digital 3 | Encoder 1 (VFO) B | PWML | Digital 34 | SW 5 | Digital |
| Digital 4 | Encoder 2A pin B | PWML | Digital 35 | SW 6 | Digital |
| Digital 5 | Encoder 2A pin A | PWML | Digital 36 | SW 7 | Digital |
| Digital 6 | Encoder 2 PUSH | PWML | Digital 37 | SW 8 | Digital |
| Digital 7 | Encoder 2B pin B | PWML | Digital 38 | SW 9 | Digital |
| Digital 8 | Encoder 2B pin A | PWMH | Digital 39 | SW 10 | Digital |
| Digital 9 | SW17 | PWMH | Digital 40 | SW 11 | Digital |
| Digital 10 | Encoder 3A pin B | PWMH | Digital 41 | SW 12 | Digital |
| Digital 11 | Encoder 3A pin A | PWMH | Digital 42 | SW 13 | Digital |
| Digital 12 | Encoder 3 PUSH | PWMH | Digital 43 | SW 14 | Digital |
| Digital 13 LED | Reserved LED | PWMH | Digital 44 | SW 15 | Digital |
| Digital 14 TX3 | Encoder 3B pin B | communication | Digital 45 | SW 16 | Digital |
| Digital 15 RX3 | Encoder 3B pin A | Communication | Digital 46 | LED 1 (SW1) | Digital |
| Digital 16 TX2 | Encoder 5B pin A | Communication | Digital 47 | LED 2 (SW2) | Digital |
| Digital 17 RX2 | Encoder 4A pin B | Communication | Digital 48 | LED 3 (SW3) | Digital |
| Digital 18 TX1 | Display RXD | Communication | Digital 49 | LED 4 (SW4) | Digital |
| Digital 19 RX1 | Display TXD | Communication | Digital 50 | LED 5 (SW5) | Digital |
| Digital 20 SDA | Reserved SDA | Communication | Digital 51 | LED 6 (SW9) | Digital |
| Digital 21 SCL | Reserved SCL | Communication | Digital 52 | Opto PTT in | Digital |
| Digital 22 | Encoder 4A pin A | Digital | Digital 53 | LED 7 (SW17) | Digital |
| Digital 23 | Encoder 4 PUSH | Digital | Analog 0 |  | Analog In |
| Digital 24 | Encoder 4B pin B | Digital | Analog 1 |  | Analog In |
| Digital 25 | Encoder 4B pin A | Digital | Analog 2 |  | Analog In |
| Digital 26 | Encoder 5B pin B | Digital | Analog 3 |  | Analog In |
| Digital 27 | Encoder 5A pin B | Digital | Analog 4 |  | Analog In |
| Digital 28 | Encoder 5A pin A | Digital | Analog 5 |  | Analog In |
| Digital 29 | Encoder 5 PUSH | Digital | Analog 6 |  | Analog In |
| Digital 30 | SW1 | Digital | Analog 7 |  | Analog In |

This supports:

* VFO encoder (encoder 1)
* 7 normal encoders
  + 4 dual encoders, (encoders 2A/2B, 3A/3B, 4A/4B, %A/5B)
  + Note encoder A is the upper control, with the push switch)
  + (single encoders can be used for 2-5; a second function can be activated by clicking the encoder)
* 4 encoder push switches (2, 3, 4, 5)
* 17 normal push switches;
* 7 LEDs

(Note the VFO encoder (encoder 1) has no push action)

# Arduino Libraries

Arduino Due has the Atmel SAM3X8E ARM Cortex-M3 processor. Any input can have an interrupt and it may be possible to select the h/w input debounce. But needs some specific libraries

* “DueFlashStorage” library is an EEPROM equivalent library for Due
* Timer – Due specific “DueTimer”
* Serial – there seem to be several. SerialUSB is probably the “native” port.
* Nextion
* There is a LiquidCrystal\_I2C library by MarcoSchwarz. There is also newliquidcrystal.

The programming port is “Serial” and the “native” port is “SerialUSB”. Consider using SerialUSB for CAT connection, retaining the other for debug?

# Rotary Encoders

It seems that interrupt driven code is poor at debouncing. It does work well with bounce-free optical encoders.

Zacsketches/quadrature works well for the VFO: I’m getting 2400 steps per revolution.

ClickEncoder works well for the other “mechanical” encoders.

# Nextion Interface

3.3V TTL serial. Requires 5V power supply.

# Issues List

|  |  |
| --- | --- |
| **Issue** | **Resolution** |
| 1. Add external PTT code | Completed |
| 1. Add diversity gain, phase controls | Completed |
| 1. Add configurable VFO speed steps (2/4/8) | Completed |
| 1. Add display of filter shift & width | Completed |
| 1. Add PTT latch/not latch depending on how long pressed for |  |
| 1. Modify PowerSDRmrx to allow step attenuator setting |  |
| 1. Kjell’s new commands |  |
| 1. Consider ballistic tracking for VFO encoder |  |