# Aries ATU

# Aries Concept

The concept is for an ATU to be installed into Andromeda. Kjell suggested the standard L-C architecture, and can lend me an AT-11 as the basis for the prototype of the RF section.

## Rationale

|  |  |
| --- | --- |
| **Aries Idea** | **Why** |
| Aries is controlled by Thetis using CAT commands | Provides tighter user interface and display through Thetis of tuner status |
| Aries is connected to the radio TX path, not in the RX path | We do not need to switch the ATU out of circuit for RX. |
| Aries will use the well known “L match” arrangement with 8 inductors and 8 capacitors | It works, and appropriate component values are well known |
| Aries stores L/C tuning solutions with a “grid” of 10KHz steps | Fine enough grid for most antennas (but possibly not magnetic loops) |
| Aries stores separate solutions for each of the 3 antenna outputs | So that 3 different antennas can be connected, with separate tuning for each |
| Aries is sent the tuned TX frequency by CAT command.  A new command is sent every time the frequency moves to a new frequency step (~10KHz) | So that it “knows” frequency as the radio is tuned and can give feedback straightaway that a stored solution is or isn’t available.  Also so that it doesn’t have to measure TX frequency for an SSB signal, which would be problematic. |
| Aries measures VSWR itself | It needs a new VSWR measurement every 10-16ms; it would be too slow to send VSWR reports from Thetis during tuning. |
| Aries can be user selected to be active or bypassed for each antenna individually | To allow it to be switched out if ANT3 is connected to an external linear, for example |
| There needs to be a way for the user to tell Aries to clear the Tuning solutions for any of the 3 antennas individually | So that if an antenna is changed or modified, Aries doesn’t try to use “old” solutions |
| Aries searches for a new tuning solution when TUNE is selected on the radio |  |

## Interface to Thetis

|  |  |
| --- | --- |
| **THETIS action** | **Aries action** |
| At Startup:  Find the band and hence antenna in use.  Find if Aries enabled for that band. Send enabled/not enabled message.  For initial TX frequency: send frequency.  Initialise ATU display symbol – either “off” or “no solution” |  |
| When an antenna is changed, Thetis sends an enable or bypass command to Aries and new antenna number  If Aries enabled, Thetis displays a symbol on the display  If a tuning solution is reported as available, the green LED is lit and the ATU display symbol is highlighted. | Aries searches for a tuning solution: on the frequency specified, then up to ±50KHz away. If it finds a solution it selects it ready for use; it not it selects “bypass”. Note the relays are not changed – it just prepares to set these relay values.  A message is sent back to Thetis with solution available or not available |
| When a band is changed, Thetis sends a new frequency message to Aries  When the radio is tuned by more than 10KHz from the last frequency, a new frequency message is sent to Aries.  If a tuning solution is reported as available, the green LED is lit and the ATU display symbol is highlighted. | Aries searches for a tuning solution: on the frequency specified, then up to ±50KHz away. If it finds a solution it selects it ready for use; it not it selects “bypass”. Note the relays are not changed – it just prepares to set these relay values.  A message is sent back to Thetis with solution available or not available. |
| When TX is initiated | If the frequency has been changed, Aries drives the new relay selection for the new tuning solution from memory to the relays. If no solution was available, it selects bypass.  **This needs to happen quickly – driven by PTT** |
| When TX removed | The relays are left in the same position ready for the next operation |
| When TUNE is selected: the red LED is lit. A “tune now” message sent to Aries.  Thetis stays in the TUNE state, and indicates success/no success using the green LED and display symbol highlight. (?option to stay in for fine tune, or just exit) | Aries begins its algorithm to find a new solution. When complete, if a good solution was found it is stored in EEPROM.  Aries sends a message back saying “tune complete” and “successful/not successful”. |
| If user requests settings for ANT1/2/3 to be cleared  Display “erasing…”  Change display to “Done” | All tuning solutions for that antenna are erased from EEPROM. If that antenna is selected, Aries enters bypass state. (note this takes around 5 seconds I think)  ARIES sends a response message when complete If selected antenna == erased antenna, ARIES sends “not successful” |



Erasing….Done

H/W 2; S/W 17

CAT port

Com4

Solutions

X

3

2

Erase

Erase

Erase

Ant

ATU enabled

X

X

1

Enabled

f/w version

X

ATU

Figure 1: Suggested setup tab



ATU

Figure 2: Andromeda display having an “ATU” symbol:

## Interface with External Linear Amplifier

At minimum, it must be possible to bypass the ATU if an external linear amplifier is attached.

A “full interface” would allow the Tune/ATU tuning solution to be fully meshed with Thetis. This would be amplifier dependent. This could mean further Thetis additions:

* User interface Setup to identify the amplifier type
* User interface in setup to identify which Antenna the amplifier is connected to
* User interface in setup to enable/disable the external ATU
* Messaging to inform external ATU about TUNE selected
* Messaging to accept tune complete and fail/success from external ATU

## CAT messages required

Re-use existing CAT messages where possible

|  |  |  |
| --- | --- | --- |
| **Event** | **Message** |  |
| TX on/off | Signalled by hardwired signal |  |
| TUNE on/off | CAT message ZZTUn;  Sent from PC to Aries | n=0: no tune; n=1: TUN active |
| Frequency change | CAT message ZZTVmmmmmmmmmmm;  Sent from PC to Aries | mmmmmmmmmmm: 11 digit frequency (Hz)  eg 00014320000 = 14.32 MHz (expected to be steps of 10KHz) |
| Antenna change | CAT message ZZOCn;  Sent from PC to Aries. | n=1: Ant1; n=2: Ant2; n=3: Ant3 |
| Erase Solution | CAT message ZZOZn;  Sent from PC to Aries.  Response: ZZOZn;  Sent by Aries to PC | n=1: erase solutions for Ant1; n=2: erase for Ant2; n=3: erase for Ant3 |
| Fine tune L/C | CAT message: ZZZEnnm;  Sent from PC to Aries. | nn= encoder number and direction. m= number of steps  Allowed values: 01=L c/w; 02=C c/w; 51=L ac/w; 52=C ac/w |
| ATU success/fail | CAT message: ZZOXn;  Sent by Aries to PC | n= 0: no ATU solution found; n=1: suitable tuning solution found. |
| ATU Enable | CAT message: ZZOVn;  Sent from PC to Aries | n=0: ATU inactive; n=1: ATU active, and will tune on demand |
| Query s/w Version | ZZZS;  Response ZZZSppnnmmm; | pp=product id  1: Andromeda 2: Aries 3: Ganymede  nn= hardware version  mmm= s/w version |

Of these only 1 is a new message to be recognised by Thetis (ZZOX)

# Matching network

The matching network is provided by 8 inductors and 8 capacitors in a classic L match arrangement. If the capacitor is connected to the input end, it matches low impedance loads; if to the output, it matches high impedance loads.



For now we will use the principles established by the early auto ATU tuners. Capacitance is stepped in 5pF steps from 0 to 1280pF. Inductance is stepped in 80nH steps from 0 to 20uH.



In both cases: the component is selected in-circuit if the data bit is set to 1.

(A question that could be considered: is it good enough to use capacitor values 4.7-10-22-47-100 etc instead of the exact 2x steps?)

The relay to switch between low and high Z will need to be driven by a processor pin directly.

# ATU and the Smith Chart

The Smith Chart is a plot of the complex reflection coefficient on an Argand diagram. It is a useful way to visualise what a tuner needs to do

* It needs to provide a good match to the Amplifier
* For a range of antenna impedances

The Smith Chart plots the complex reflection coefficient S11. On the one diagram you see:

* Centre dot – perfect match; load impedance = Zo
* Left hand dot – short circuit
* Right hand dot – open circuit

|  |  |
| --- | --- |
|  |  |
| Complex impedance plot | Complex admittance plot (horizontal mirror image) |

It is then possible to plot how inductance and capacitance values move the impedance of a load. A series inductance will move clockwise along a line of constant resistance. A shunt capacitance will move clockwise along a line of constant admittance.

|  |  |
| --- | --- |
|  |  |

Figure 3: Series Inductance and Shunt Capacitance effects

By adding L and C appropriately, the L match network can match any impedance on the diagram. But the series L, shunt C can only achieve one solution, and whether the capacitor is at the input or output depends on where the load impedance is placed. You can then plot regions where the L-match ATU can match – the grey regions cannot be matched:

|  |  |
| --- | --- |
|  |  |
|  |  |

Figure 4: Matching Regions for the 2 Forms of the L-Match ATU

# Processor Issues

The processor does not need to be powerful; it will be idle most of the time waiting for relays to settle.

I need a user interface during development; can be ditched afterwards. The lab model used an I2C LCD display but a Nextion touchscreen might be a better long term bet. An encoder for rapid tuning during debugging is appropriate.

## Processor pin estimates

|  |  |  |
| --- | --- | --- |
| **Function** | **CAT based** | **Non CAT based** |
| dual encoder with fast/slow | 5 | 5 |
| Push sw for high/low Z | 1 | 1 |
| I2C | 2 | 2 |
| VSWR measurement | 2 analogue in | 2 analogue in |
| SPI drives to relay array | 3 optional | 3 optional |
| Serial port | 2 | 2 |
| PTT | 1 | 1 |
| Frequency measure | - | 1 |
| RF detect | - | 1 |
| Tune/done drives to radio | - | 2 |
| Relay drive for high/low Z | 1 | 1 |

Relays could be SPI driven using 3x TPIC6B595 shift register. Alternatively could be I2C, with a different arrangement for the open drain drivers. Assume I2C for now.

Frequency measurement needs a crystal clock; that would rule out the Arduino Nano Every which has an RC oscillator. A plain Nano would be OK. In the Arduino timers 1 and 2 are not used by default; the T1 clock pin is PD5 used as Arduino D5. A prescaler (eg divide by 16) would be needed.

|  |  |
| --- | --- |
|  |  |

## EEPROM

An external EEPROM will be needed. Need 3 bytes per frequency to store tuning solutions. If we store a solution per 10KHz, we need 100 settings per MHz ie approx. 6000 settings for the HF band ie 18 KByte. If we have 3 antennas and separate solutions for each, that’s 54Kbyte ie near 500Kbit. 2Mbit+ EEPROMs are readily available with I2C interface.

@1MHz, I2C read is ~40us, but with sequential read can then get each new byte in ~10us. Might consider reading a few 100KHz worth when a new frequency is detected; if no stored solution for the first one, look outwards and give up at +/-50KHz.

## I/O Pin assignment

|  |  |  |
| --- | --- | --- |
| **Function** | **Pin** | **Comment** |
| Serial | DIG0, DIG1 | Allows use of Nano |
| Built in LED | DIG13 |  |
| Encoder 1 (lower) | DIG2, DIG3 |  |
| Encoder 2 (upper) | DIG4, DIG6 |  |
| Encoder 1 pushbutton | DIG7 |  |
| “Tune now” pushbutton | DIG8 |  |
| High/Low Z Relay | DIG9 | 1=high Z; 0=low Z |
| PTT | DIG10 | 0=TX; 1= RX. Needs pullup. |
| VSWR Bridge inputs | A0, A1 | A0=fwd; A1=rev |
| I2C SDA | A4 / SDA |  |
| I2C SCL | A5 / SCL |  |
| Freq Count | DIG5 | Timer 1 clock for Nano |

## I2C Device Assignment

|  |  |
| --- | --- |
| MCP23017 for relays | 0x20, 0x21 |
| LCD display | 0x27 (confirmed by scan) |
| EEPROM | 0x50, 0x51 (includes A16 as bottom bit) |

The LCD display has a PCF8574T interface. It needs LiquidCrystal\_I2C library. (Confirmed operation with simple sketch)

# I2C Race Issue

## Race problem

We have considered having the radio PTT line hardwired to Aries so that it can get an immediate “TX now” indication. This is present several ms before RF, and could be used to drive the relays to the required solution before there is any current flowing. It could achieve this by interrupting the processor. However if the I2C interface was already in use, the interrupt handler would not get use of it.

I2C is used by 3 interfaces:

1. Relay drivers. These are driven at the start of TX and during tune; there is no other trigger so no race condition.
2. LCD for user interface – this is debug only, so no operational hazard.
3. EEPROM, to retrieve and store L/C solutions. Access to these is required while tuning, and a race hazard could exist. Probably a marginal one, but if PTT was pressed while or immediately after tuning, a solution could be being retrieved.

Proposed solution:

1. Retrieve a block of solutions to SRAM and access from there. Only read from EEPROM when those in SRAM are inadequate.
2. Disable TX interrupt when retrieving a new set.
3. Write a new solution before TUNE state is released; so no hazard exists.

## Options for RAM storage

1. Store solutions for the whole HF band, 3 antennas in memory.
   1. Storage required ~ 48Kbyte. Impractical with 8 bit AVR; would be OK with Arduino Nano 33 (256KB RAM) @ 18 Euro.
   2. Readout time ~500ms; but only read out once at power up.
2. Store solutions for whole HF band but only one antenna
   1. Storage required ~16Kbyte. Impractical with 8 bit AVR, still needs Nano 33
   2. Readout a new batch whenever the antenna changed
   3. Readout time ~165ms; means there is a significant period when TX not possible
3. Store solutions for just the HF bands, one antenna
   1. Storage required ~1800 bytes (OK for AVR)
   2. Readout a new batch whenever the antenna changed
   3. Readout time ~20ms; still a significant period when TX wouldn’t be possible.
4. Store solutions for current frequency +- 300KHz
   1. Enough to be able to find “nearby” solutions
   2. Readout a new batch whenever you tune close to the edge of data held in memory
   3. Storage required ~180 bytes (easy)
   4. Readout time 1.8mS (acceptable deadtime?)

The viable choices are the first or last; the intermediate one don’t offer enough of a benefit. The latter should be doable; the former allows simple lazy software.

## EEPROM Chip Interface

To write back individual solutions it may make sense to use individual byte writes. To erase an antenna worth of settings, use block write (128 bytes take the same 5ms). To read data, use sequential read with one address transaction (~40us) then individual reads transferring one new byte only (10us).

The Microchip 1Mbit EEPROM (24FC1026-I/P) can only read out half of the device sequentially, but this is sufficient. FC devices can clock at 1MHz.

For convenience the “no solution” state should be the shipped condition –solution = 0xFFFFFF.

Each block of data for each antenna should begin at a page boundary (128 bytes) and be sized so that a fixed number of page writes can be made to erase it. That suggests it should be a little bigger than needed. Can we arrange that a solution stays within a page, so we can do a page write?

There is an Arduino library (extEEPROM) that supports the 24FC1026 device.

# VSWR Bridge

Currently using a Stockton Bridge with two ferrite toroids. This requires no “balance” adjustment and gives a calibrated power measurement.

Calculated (see notebook) that RMS line voltage = 0.1074N where N = Arduino ADC reading.

Power = Vrms2/50

VSWR = (Vf+Vr)/(Vf-Vr)

# EEPROM Data structures

Proposed approach: 3 blocks of memory - one for each antenna.

Solutions are stored every 10KHz, starting at 0; each antenna stores solutions for all frequencies from 0 to 61.5MHz (just over the Nyquist rate for HPSDR radios).

There are 3 bytes per solution stored:

|  |  |
| --- | --- |
| **byte** | **meaning** |
| 0 | bit0=1: no data; bit0=0: data OK  bit7=1: high Z; bit7=0: low Z |
| 1 | Inductance word |
| 2 | capacitance word |
| 3 | 1=high Z; 0=low Z |

Address of solution in EEPROM, with 1st antenna = antenna 1:

(Antenna-1)\*32768 + 3\*Int(Freq\_in\_KHz/10KHz)

Hence for antenna 2 at 61.471MHz, the solution will be at address 57356 decimal

We need an EEPROM storing at least 128Kbyte.

Local data / data structures

|  |  |  |
| --- | --- | --- |
| unsigned int | GTunedFrequency10 | Frequency the ATU is tuned to, in units of 10KHz |
| bool | GATUEnabled |  |
| unsigned int | GQueuedCATFrequency | 0xFFFF: no frequency stored  0-6250: ATU frequency passed by Thetis during TX |
| byte | GAntenna | Antenna number (1-3)  =0 if not set |
| bool | GFrequencySet | True if a frequency has been set by THETIS |
| bool | GPCTuneActive | If true, a TUNE is happening |
| bool | GPTTPressed | If true, PTT is pressed |
| unsigned int | GSolutionStartFreq | Frequency (10KHz units) of 1st solution held in internal memory |
| bool | GTXAllowed | True if ATU solution should be sent when TX asserted |
| bool | GATUIsTuned | True if valid solution already set |
| byte\* | GSolutionBuffer | Block of tuning solutions buffered in internal memory  #define GSOLUTIONSIZE 3  #define VSOLUTIONCOUNT 20  No. of solutions held (buffer = 3\*20)  (the logic being: do 2 x 32 byte reads) |



Figure 5: EEPROM Memory map

# Software Algorithms

## Event Response

|  |  |
| --- | --- |
| **Event** | **Action Required** |
| When TX asserted | If existing solution already driven: no action  If new solution ready to go: drive solution to relays  If “Tune in progress” set: commence tune  Set “TX in progress” flag |
| When TX deasserted | Clear “TX in progress” flag  Clear “tune in progress” flag  If Tune algorithm in progress, terminate the algorithm |
| When new frequency received | If already in TX – wait until TX completed  If new frequency within 100KHz of edges of data available:  Read out batch of tuning solutions centred at current frequency  Disable TX interrupt while reading  Select closest solution or bypass  Send CAT message with solution available/not available |
| When new antenna received | Read out tuning solutions for antenna N around current frequency  Disable TX interrupt while reading  Select closest solution or bypass  Send CAT message with solution available/not available |
| When ATU enable received | Treat as a new antenna |
| When ATU Disable received | Drive “bypass” setting to ATU  Set flag |
| When “TUNE start” message received | Set “tune in progress” flag  If TX already asserted, begin algorithm |
| When “TUNE end” message received | Ignore – this is signalled by removal of PTT |
| When TUNE algorithm complete | Store solution to EEPROM  Report success after store complete |
| If L/C fine tune CAT message received | Adjust L/C setting  Do not re-store |
| When erase received for ant N | Erase the block of data for that antenna  If same antenna currently selected:  set solution to use = bypass  clear RAM copy of tuning solutions  Send response message |

## Suggested search algorithm

Do the following while GTuneActive is true:

Quick tune:

1. Try “fine step L and C” around current setting
2. If that fails select full tune

Full Tune:

1. On the basis of frequency, select min/max L and C and coarse, mid step size
2. Select “Low Z”
3. Step through C values at coarse step, find step with min VSWR & achieved VSWR
4. Select “High Z”
5. Step through L values at coarse step, find step with min VSWR & achieved VSWR
6. If best result is low Z:
   1. Re-select Low Z
   2. Select best C value
   3. Step through L values at coarse step, find step with min VSWR & achieved VSWR
   4. Select best L value
   5. Step through C values +/- 2 coarse steps at mid step for min VSWR
   6. Select best C value
   7. Step through L values +/- 2 coarse steps at mid step for min VSWR
   8. Select best L value
   9. Step through C values +/- 2 mid steps at step=1 for min VSWR
   10. Select best C value
   11. Step through L values +/- 2 mid steps at step=1 for min VSWR
   12. Select best L value
7. Else if best result was high Z
   1. Select best L value
   2. Step through C values at coarse step, find step with min VSWR & achieved VSWR
   3. Select best C value
   4. Step through L values +/- 2 coarse steps at mid step for min VSWR
   5. Select best L value
   6. Step through C values +/- 2 coarse steps at mid step for min VSWR
   7. Select best C value
   8. Step through L values +/- 2 mid steps at step=1 for min VSWR
   9. Select best L value
   10. Step through C values +/- 2 mid steps at step=1 for min VSWR
   11. Select best C value
8. Store result to EEPROM if final VSWR < 1.5
9. Report success/fail and end



At each tick:

1. Check if “terminate” signal
   1. Exit if so with no result saved
2. Measure VSWR
3. If VSWR < min already achieved
   1. Record step and VSWR
4. If next step > end:
   1. move to next state
5. Else
   1. set next step
   2. drive solution

To find next candidate solution:

If (setting == end)

Signal end;

Else

New setting = constrain (setting+step, min, max);

# Testing

## New ATU Demonstrator Wiring

26 way ribbon cable; the relay network made from ebay relay boards.

|  |  |  |  |
| --- | --- | --- | --- |
| Pin | Connection | Pin | Connection |
| 1 | GND | 2 | L0 |
| 3 | L1 | 4 | L2 |
| 5 | L3 | 6 | L4 |
| 7 | L5 | 8 | L6 |
| 9 | L7 | 10 | +5V |
| 11 | GND | 12 | C0 |
| 13 | C1 | 14 | C2 |
| 15 | C3 | 16 | C4 |
| 17 | C5 | 18 | C6 |
| 19 | C7 | 20 | +5V |
| 21 | +12V | 22 | +12V |
| 23 | NC | 24 | +12V |
| 25 | GND | 26 | High/Low Z |

All relays are active low drive. The L/C relays are opto isolated, and 5V common. The High/Low Z relay is driven by a PNP transistor to +12V, so needs an open collector drive from a discrete transistor.

## Test Loads

|  |  |  |  |
| --- | --- | --- | --- |
| Switch position | VSWR | Resistance | Construction |
| 1 | 8:1 Low | 6.25R | 16x100R parallel |
| 2 | 5:1 Low | 10R | 10x100R parallel |
| 3 | 4:1 Low | 12.5R | 8x100R parallel |
| 4 | 3:1 Low | 16.6R | 6x100R parallel |
| 5 | 2:1 Low | 25R | 4x100R parallel |
| 6 | 1:1 | 50R | 4x100R parallel, in series with 4 more 100R parallel |
| 7 | 2:1 High | 100R | 2x100R parallel, in series with 2 more 100R parallel |
| 8 | 3:1 High | 150R | 3x100R series, parallel with another 3x100R series |
| 9 | 4:1 High | 200R | 2x100R series  (4x100R in series, parallel with 4x100R series) |
| 10 | 5:1 High | 250R | 2x100R series, +2x100R parallel |
| 11 | 8:1 High | 400R | 4x100R series |

Need to be good for around 5W

# Tuning Solutions

For the “ebay relay” L/C module:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Frequency: 1.9 MHz | High/Low Z sw | Antenna analyser | | | algorithm | | |
| L | C | VSWR | L | C | VSWR |
| 8:1 Low | L | 33 | 255 | 1.5 |  |  |  |
| 5:1 | L | 39 | 219 | 1.4 |  |  |  |
| 4:1 | L | 0 | 90 | 1.8 |  |  |  |
| 3:1 | L | 0 | 70 | 1.6 |  |  |  |
| 2:1 | L | 0 | 46 | 1.3 |  |  |  |
| 1:1 | L | 0 | 8 | 1.0 |  |  |  |
| 2:1 High | H | 71 | 53 | 1.2 |  |  |  |
| 3:1 | H | 117 | 48 | 1.4 |  |  |  |
| 4:1 | H | 146 | 45 | 1.4 |  |  |  |
| 5:1 | H | 182 | 42 | 1.4 |  |  |  |
| 8:1 | H | 234 | 32 | 1.4 |  |  |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Frequency: 3.65MHz | High/Low Z sw | Antenna analyser | | | algorithm | | |
| L | C | VSWR | L | C | VSWR |
| 8:1 Low | L | 2 | 163 | 1.3 |  |  |  |
| 5:1 | L | 7 | 125 | 1.2 |  |  |  |
| 4:1 | L | 10 | 111 | 1.2 |  |  |  |
| 3:1 | L | 12 | 94 | 1.2 |  |  |  |
| 2:1 | L | 17 | 77 | 1.2 |  |  |  |
| 1:1 | L | 0 | 5 | 1.0 |  |  |  |
| 2:1 High | H | 31 | 24 | 1.2 |  |  |  |
| 3:1 | H | 61 | 20 | 1.2 |  |  |  |
| 4:1 | H | 71 | 18 | 1.2 |  |  |  |
| 5:1 | H | 97 | 14 | 1.2 |  |  |  |
| 8:1 | H | 131 | 10 | 1.2 |  |  |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Frequency: 7.1MHz | High/Low Z sw | Antenna analyser | | | algorithm | | |
| L | C | VSWR | L | C | VSWR |
| 8:1 Low | L | 0 | 40 | 2 |  |  |  |
| 5:1 | L | 0 | 35 | 1.5 |  |  |  |
| 4:1 | L | 0 | 35 | 1.3 |  |  |  |
| 3:1 | L | 1 | 31 | 1.3 |  |  |  |
| 2:1 | L | 1 | 25 | 1.2 |  |  |  |
| 1:1 | L | 0 | 3 | 1.0 |  |  |  |
| 2:1 High | H | 17 | 6 | 1.1 |  |  |  |
| 3:1 | H | 25 | 2 | 1.1 |  |  |  |
| 4:1 | H | 31 | 0 | 1.1 |  |  |  |
| 5:1 | H | 49 | 0 | 1.1 |  |  |  |
| 8:1 | H | 61 | 0 | 1.6 |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Frequency: 14.2 | High/Low Z sw | L | C | VSWR |
| 8:1 Low | H |  |  |  |
| 5:1 | H |  |  |  |
| 4:1 | H |  |  |  |
| 3:1 | H |  |  |  |
| 2:1 | H |  |  |  |
| 1:1 | H |  |  |  |
| 2:1 High | L |  |  |  |
| 3:1 | L |  |  |  |
| 4:1 | L |  |  |  |
| 5:1 | L |  |  |  |
| 8:1 | L |  |  |  |

Need to search both L and C on High Z

# To Do List

## Aries

Apart from “just write the code”…

1. Buy eeprom
2. Document the VSWR bridge, signal levels
3. Check algorithm, CAT commands, description are all consistent

## Thetis

1. Add CAT command serial interface (2x .cs files) for Aries (best to add one for Ganymede at the same time)
2. setup.cs
   1. Add an Aries screen to setup
   2. Add event handlers for “ATU enabled” buttons
   3. Add code to send CAT message when Erase pressed, and set status to “erasing”
   4. Add setup.cs CAT command handler for erase response; clear status when done.
3. console.cs
   1. Add function to console for “ATU enabled” buttons in setup to be passed to the console
   2. Add initialise code to set initial Aries state
   3. Add ATU button to collapsed display able to show ATU off, no solution, and solution set
   4. (there is already a CAT message handler for tune success)
   5. Add code so when TUNE is pressed a message is sent
   6. Add code so when TX frequency changes by more than 10KHz a message is sent
   7. Add code to pick up antenna changes (either by setup change or band change)