# “Andromeda” Console Implementation Notes

This documentation is for the Andromeda front panel PCB using the Arduino Nano Every processor and 2 MCP23017 ICs.

# Control Layout



## Encoder Functions:

From a software perspective, each dual encoder can have A (upper) and B (lower) encoders plus a “click” function. s/w numbering allows up to 20 numbers making encoders 1-40 and switches 1-20

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Encoder number** | **Function** | **Encoder report #** | **Encoder number** | **Function** | **Encoder report #** |
| ENC1 Top | RX1 AF | 1 | ENC7 Top | Diversity gain | 7 |
| ENC1 bottom | RX1 AGC | 2 | ENC7 bottom | Diversity phase | 8 |
| ENC1 click | RX1 mute |  | ENC7 click | Diversity on/off |  |
| ENC3 Top | RX2 AF | 3 | ENC9 Top | RIT | 9 |
| ENC3 bottom | RX2 AGC | 4 | ENC9 bottom | n/a | 10 |
| ENC3 click | RX2 mute |  | ENC9 click | RIT/XIT clear |  |
| ENC5 Top | Filter high | 5 | ENC11 Top | Multifunction | 11 |
| ENC5 bottom | Filter low | 6 | ENC11 bottom | Drive | 12 |
| ENC5 click | Filter reset |  | ENC11 click | Multi click |  |

**

Figure 1: Encoder numbers

## Pushbutton Functions

Pushbuttons have two numbers: the software scan code from the matrix algorithm; and the number that is reported to THETIS. A lookup table from scan code gives the THETIS number. The scan codes are only used by the Arduino software and have no user meaning.

**

Figure 2: Software Scan Codes

**

Figure 3: Pushbutton Numbers for Thetis

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| s/w scan code | CAT report | function | s/w scan code | CAT report | function |
| 0 | 21 | Softkey 1 | 20 | 39 | User 1 |
| 1 | 22 | Softkey 2 | 21 | 36 | A>B |
| 2 | 23 | Softkey 3 | 22 | 33 | Band - |
| 3 | 24 | Softkey 4 | 23 | 30 | Band + |
| 4 | 25 | Softkey 5 | 24 | 45 | CTUNE |
| 5 | 26 | Softkey 6 | 25 | 44 | VFO lock |
| 6 | 27 | Softkey 7 | 26 | 43 | VFO A/B |
| 7 | 28 | Softkey 8 | 27 | 42 | RIT/XIT |
| 8 | 11 | Encoder 11 | 28 | 41 | User 3 |
| 9 | 3 | Encoder 3 | 29 | 38 | Split |
| 10 | 1 | Encoder 1 | 30 | 35 | Filter - |
| 11 | 50 | Two tone test | 31 | 32 | Filter + |
| 12 | 49 | Puresignal on | 32 | 0 | n/a |
| 13 | 48 | TUNE | 33 | 0 | n/a |
| 14 | 47 | MOX | 34 | 0 | n/a |
| 15 | 46 | SDR on | 35 | 0 | n/a |
| 16 | 40 | User 2 | 36 | 29 | Shift |
| 17 | 37 | B>A | 37 | 9 | Encoder 9 |
| 18 | 34 | Mode - | 38 | 7 | Encoder 7 |
| 19 | 31 | Mode + | 39 | 5 | Encoder 5 |

Encoder buttons start at 1; pushbuttons start at 21; 4x3 keypad starts at 30. They are laid out geographically.

(encoder numbers correspond to the 1st encoder number. So encoder 3,4 has switch scan code 3)

## Indicator Functions

**

Figure 4: Indicator Numbers

(starting at 1)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| THETIS number | s/w number | function | THETIS number | s/w number | function |
| 1 | 0 | MOX | 7 | 6 | CTUNE |
| 2 | 1 | ATU ready | 8 | 7 | RIT |
| 3 | 2 | TUNE active | 9 | 8 | XIT |
| 4 | 3 | Puresignal on | 10 | 9 | VFO A |
| 5 | 4 | Diversity enabled | 11 | 10 | VFO lock |
| 6 | 5 | SHIFT on |  |  |  |

## Available functions for Each Control

### Indicators and Pushbuttons

For pushbuttons and indicators: try to use the same command set as for the menu buttons. See if any of the following are not available as menu buttons, and add them if needed.

|  |  |
| --- | --- |
| Pushbuttons (including encoder “press”)   * A/B VFO select * MOX * TUNE * AF MUTE (explicit RX1/2 needed) * Filter reset * Band + * Band – * Mode + * Mode – * AGC speed * NB step * NR step * SNB on/off * ANF on/off * Off /RIT on / XIT on * 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 (deprecated) * MON on / off * Diversity Enable * VFO Sync * Clear RIT/XIT * Filter + * Filter – * VAC1 on/off * VAC2 on/off * Display Centre * F1-F8 button | Indicators   * MOX * TUNE * ATU Ready * 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 * VFO sync * XIT   Go through and check if these available! |

### Encoders

* RX1 AF gain
* RX2 AF gain
* Sub RX AF Gain
* Master AF gain
* RX1 stereo balance
* RX2 stereo balance
* Sub RX stereo balance
* RX1 AGC
* RX2 AGC
* RX1 step attenuation
* RX2 step attenuation
* RX1 Squelch
* RX2 Squelch
* Selected RX Filter high cut
* Selected RX Filter low cut
* TX Drive
* TX Mic Gain
* VFO A tune
* VFO B tune
* RIT
* VOX gain
* VOX delay
* Compander threshold
* CW sidetone
* CW speed
* Diversity Gain
* Diversity Phase
* Multifunction
* Display Pan
* Display Zoom

# Arduino Software Structure

A simpler cheaper is being used from the first prototypes: an 8 bit Arduino Nano Every.

## Concept for Operation

The front panel controller will exchange simple messages with the SDR application (eg Thetis) so that all the “radio” functionality is in one place, to minimise latency. The controller will debounce buttons and encoders and pass the results for processing.

The serial queue to / from the PC will use normal Arduino library code. Simple TX commands will be generated for each pushbutton or encoder event. RX commands will be processed for indicator setting, to allow s/w version to be queried and potentially to allow encoder rates to be adjusted.

As a temporary measure for debug, the older Odin approach and Odin based code is used.

## Serial Command Set

This should be quite simple. CAT compatible commands proposed.

|  |  |  |
| --- | --- | --- |
| **Control type** | **Command** | **Meaning** |
| VFO encoder | Up: ZZZUnn;  Down: ZZZDnn; | nn = number of clicks (0-99) |
| Other encoder | ZZZEnnm; | nn = encoder number:  0: unused  1-20: encoder 1-20, clockwise  51-70: encoder 1-20, anticlockwise  m=number of clicks (0-9) |
| Pushbutton (including encoder) | ZZZPnnm; | nn=button number (1-99, 0=n/a);  m=0: released; m=1: pressed; m=2: long pressed |
| Indicator | ZZZInnm; | nn=indicator number (1-99, 0=n/a);  m=0: unlit; m=1: lit |
| Query s/w Version | ZZZS;  Response ZZZSppnnmmm; | pp=product id  1: Andromeda 2: Aries 3: Ganymede  nn= hardware version  mmm= s/w version |
| Query/change encoder increments | ZZZX;  Response ZZZXmmn; | mm = VFO encoder increment; typically 01 (120ppr encoder) or 04 (400ppr encoder)  n= other encoder increment  (typically 1,2 or 4) |

# Arduino Hardware

“Arduino Nano Every” has 19 usable I/O: D0-D12, A0-A3, A6-A7 (D13 is LED; A4-5 are I2C; we don’t need to use serial TX1, RX1 so they are available as GPIO) at very low cost.

## I2C ports

I2C (A4, A5) connect to MCP23017 devices. Chip address = 0x20, 0x21. Class: Wire

## Arduino Issues

Arduino Nano Every has an 8 bit ATMEGA4809 processor. I need to change some libraries

* EEPROM.h
* Timer – new code will be needed; no known libraries for ‘4809
* Serial connection to PCB – “Serial” is the normal one.
* You have to install “Arduino Mega AVR Boards” in the Arduino console Boards manager
* Interrupt driven VFO encoder library doesn’t support ‘4809

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

Modified ClickEncoder works well for the other “mechanical” encoders. I’m already using modified encoder code, not in a library. Should be simple to change it again to pass 2 bits of data into it, rather than having I/O pin numbers passed to it.

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

We could have up to 8 MCP23017 on one I2C bus (3 ID pins). MCP23017 are 3mA I/O: so not suitable for driving LEDs in a matrix.

The processor is only 8 bit: so change variables wherever possible to be byte (8 bit unsigned) int8\_t (signed 8 bit) or int (16 bit) rather than long (32 bit)

## I/O Approaches

Add 2xMCP23017 and we have 51 I/O available. Pushbuttons scanned switches in a matrix.

### I2C

I2C is normally clocked at 100KHz, but can be faster; use wire.SetClock() to adjust. Could be 400KHz or 1MHz for example. I2C is quite slow: at 400KHz, ~100us per transaction.

In principle the processor doesn’t need to wait for data to come back – could call “wire.requestfrom” and do something else then call “wire.read” later

## Optical Encoder Scanning

The Broadcom type encoder gives 120 pulses per revolution. By counting every edge, ie 480 edges per revolution, it is good enough for VFO control (4.8KHz per revolution @10Hz step).

The ball bearing encoders are too fast. The 600ppr encoder gives 2400 edges per revolution and when turned at about 1.5 turns per second, the VFO control on Thetis starts to run backwards.

A solution is to have conditional compilation, and for the high resolution optical encoders only interrupt on the rising edge of one input (therefore 600 interrupts per revolution) and use the other input to sense direction.

In globalinclude.h:

To compile for the high resolution encoder:

#define HIRESOPTICALENCODER 1

To compile for the Broadcom type encoder:

//#define HIRESOPTICALENCODER 1

## Keypad scanning

I will need to use a simple sequencer to scan the pushbuttons. Don’t attempt to cope with more than one button press. Assert a new column low every software tick, then read the rows and look for a row with one or more bits at zero. Only one row driven low at a time. A “helper” function reads the row input and assigns a row code: 0: no button pressed; 1-8: row 0-7 pressed; FF: more than one pressed.

Columns driven using pseudo open drain outputs from MCP23017, so if there are shorted columns it doesn’t matter. Outputs only drive a logic 0 level; inactive column outputs are disabled by turning the pins into inputs.

If the same key was pressed for more than 2 seconds, a “long press” is declared in the BUTTON PRESSED state.



Figure 5: Suggested keypad scanning sequencer

# Use of Timeslots

Suggest consider 2ms timeslot as the starting point.

@400KHz, byte read over I2C ~45us. @1MHz, will be ~20us. SPI could be 10x faster. Will need to segregate I2C activity to avoid stalling waiting for it.

In one timeslot we need to:

Do one update of the key matrix (involves I2C read, then I2C write);

Update half of the encoders; (I2C 16 bit read);

Update any LEDs.

Keypad – we could initiate the next column write at the end of the time slot, so we don’t wait for it.

So work to an assumption that we will, in one “tick”:

1. Read switch matrix row input, update sequencer
2. Read encoder 16 bit input
3. Update half the encoders
4. Process serial messages
5. Update any LEDs
6. Update Arduino LED
7. Write switch matrix column

# Device, Pin Allocations

|  |  |  |  |
| --- | --- | --- | --- |
| **MCP23017** | **Number:** | **1** | **A2=0 A1=0 A0=0** |
| **Port A** | **Encoder inputs** | **Port B** | **Encoder inputs** |
| GPA7 | Encoder 1 1A (RX1 AF) | GPB7 | Encoder 5 1A (IF Shift High) |
| GPA6 | Encoder 1 1B | GPB6 | Encoder 5 1B |
| GPA5 | Encoder 1 2A (RX1 RF) | GPB5 | Encoder 5 2A (IF Shift Low) |
| GPA4. | Encoder 1 2B | GPB4 | Encoder 5 2B |
| GPA3 | Encoder 3 1A (RX2 AF) | GPB3 | Encoder 7 1A (Diversity Gain) |
| GPA2 | Encoder 3 1B | GPB2 | Encoder 7 1B |
| GPA1 | Encoder 3 2A (RX2 RF) | GPB1 | Encoder 7 2A (Diversity Phase) |
| GPA0 | Encoder 3 2B | GPB0 | Encoder 7 2B |

|  |  |  |  |
| --- | --- | --- | --- |
| **MCP23017** | **Number:** | **2** | **A2=0 A1=0 A0=1** |
| **Port A** | **Switch matrix column OUTPUT** | **Port B** | **Switch Matrix Row INPUT** |
| GPA7 | LED11 | GPB7 | Row 8 |
| GPA6 | LED10 | GPB6 | Row 7 |
| GPA5 | LED9 | GPB5 | Row 6 |
| GPA4 | Column 5 | GPB4 | Row 5 |
| GPA3 | Column 4 | GPB3 | Row 4 |
| GPA2 | Column 3 | GPB2 | Row 3 |
| GPA1 | Column 2 | GPB1 | Row 2 |
| GPA0 | Column 1 | GPB0 | Row 1 |

## Arduino Pins

|  |  |  |  |
| --- | --- | --- | --- |
| **Arduino:** | **Arduino Nano Every** |  |  |
| DIG0 / TX | VFO encoder A | DIG11 | LED5 () |
| DIG1 / RX | VFO encoder B | DIG12 | LED6 () |
| DIG2 | Encoder 9A (RIT) | DIG13 | Arduino LED |
| DIG3 | Encoder 9B | A0 | LED7 () |
| DIG4 | Encoder 10A | A1 | LED8 () |
| DIG5 | Encoder 10B | A2 | Encoder 11A (Multi) |
| DIG6 | LED1 () | A3 | Encoder 11B |
| DIG7 | LED2 () | A4 / SDA | I2C SDA |
| DIG8 | LED3 () | A5 / SCL | I2C SCL |
| DIG9 | LED4 () | A6 | Encoder 12A |
| DIG10 | PWM for display brightness | A7 | Encoder 12B |

## Switch Matrix

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Column 5**  **2GPA4** | **Column 4**  **2GPA3** | **Column 3**  **2GPA2** | **Column 2**  **2GPA1** | **Column 1**  **2GPA0** |
| **Row 8**  **2GPB7** | CAT:5  Encoder 5 | CAT:32  Filter + | CAT:30  Band + | CAT:46  SDR on | CAT:28  Softkey 8 |
| **Row 7**  **2GPB6** | CAT:7  Encoder 7 | CAT:35  Filter - | CAT:33  Band - | CAT:47  MOX | CAT:27  Softkey 7 |
| **Row 6**  **2GPB5** | CAT:9  Encoder 9 | CAT:38  SPLIT | CAT:36  A>B | CAT:48  Tune | CAT:26  Softkey 6 |
| **Row 5**  **2GPB4** | CAT:29  SHIFT | CAT:41  User3 | CAT:39  User1 | CAT:49  PS on | CAT:25  Softkey 5 |
| **Row 4**  **2GPB3** | N/A | CAT:42  RIT/XIT | CAT:31  Mode + | CAT:50  2 Tone | CAT:24  Softkey 4 |
| **Row 3**  **2GPB2** | N/A | CAT:43  A/B | CAT:34  Mode - | CAT:1  Encoder 1 | CAT:23  Softkey 3 |
| **Row 2**  **2GPB1** | N/A | CAT:44  VFO Lock | CAT:37  B>A | CAT:3  Encoder 3 | CAT:22  Softkey 2 |
| **Row 1**  **2GPB0** | N/A | CAT:45  CTUNE | CAT:40  USER2 | CAT:11  Encoder 11 | CAT:21  Softkey 1 |

5x8 switch matrix – gives us 4 spare switches

Switch scan code is row number + (Column number-1) \*8

## LEDs

Simple lookup from the software number to a CPU pin

|  |  |  |
| --- | --- | --- |
| **Arduino pin** | **s/w number** | **Function** |
| D6 | 1 | MOX |
| D7 | 2 | ATU Tune |
| D8 | 3 | ATU OK |
| D9 | 4 | Diversity enable |
| D11 | 5 | Puresignal enable |
| D12 | 6 | SHIFT |
| A0 | 7 | CTUNE |
| A1 | 8 | RIT |
| MCP23017 #2. GPA5 | 9 | XIT |
| MCP23017 #2. GPA6 | 10 | SYNC |
| MCP23017 #2. GPA7 | 11 | VFO Lock |

## Switch Matrix Wiring



Every row has a pullup resistor. Columns are driven by the MCP23017; one column will be 0, the others will be 1. If no buttons pressed, the Row word reads out all 1s ie 0xFF

# Main LCD Brightness Control

The Andromeda PCB includes a Newhaven 7” touchscreen LCD display, 1024x600 pixel resolution. Part number NHD-7.0-HDMI-HR-RSXP-CTU (Mouser stock number 763-NHD7HDMIHRRU)

There is a need to be able to set the brightness of the main LCD display. Implemented with a simple PWM output, which has ~1KHz PRF. Setting changed if a defined encoder turned while a defined pushbutton pressed; setting stored in EEPROM and retrieved at power up. Setting written back ~5s after it was last updated. Currently this is the RX1 AF gain encoder and button.

The LCD brightness signal drives a TPS61165 LED driver chip in the display. That can use either PWM brightness (with spec min 5KHz PRF) or a one wire digital signal. The 5KHz limitation is because if the signal is low for >200us, the digital signal is selected and PWM no longer works.

Workaround: Kjell discovered that going to full brightness then working down restored the behaviour to PWM signalling. The software starts on full brightness then ramps down to the setting needed.

Two potential long term solutions:

1. Increase the PWM rate to >5KHz
2. Use digital signalling.

## PWM Change

To increase the PWM rate: the D10 pin (PB01) is controlled by timer/counter TCA. I could increase its clock rate (currently CK/64) but its clock operates all the other counter/timers.

Timer TCB0 is used for my timer tick interrupts

Timer TCB3 is used for Arduino functions including millis() and delay()

Experimentally:

Change TCA control A register to 0x07 (CK/8)

Increase timer count for TCB0 by 8

Increase any delays by 8

# Software Combination

We have several PCB layouts and several different software programs. This section explains which are which.

All of the software is downloaded from <https://github.com/laurencebarker/Andromeda_front_panel>

(click “clone or download” then “Download zip”). Save to a folder, and you will need code from one of the subfolders.

|  |  |  |
| --- | --- | --- |
| Front panel |  | Andromeda 6 encoder + VFO 4th prototype |
| Processor | Arduino Nano Every | |
| Sketch | andromeda\_front\_panel\_nano.ino | |
| Sub-folder | front panel\Arduino Nano Every Sketch\andromeda\_front\_panel\_nano | |
| Customisation steps | None currently required | |
| Works with PC code: | Thetis 2.6.8+  Does not work with PowerSDR | |
| Functionality | This is the newest software, with the radio logic in Thetis and a lower cost Arduino. | |
| Status | Not tested with this PCB yet. | |

# Arduino Software Installation

This guide describes how to download, install and load the Arduino software for the Odin console. The guide assumes that you are using the Arduino Integrated Development Environment (IDE) running on a windows platform. For users with different operating systems, different folder locations will probably apply.

# Install the Arduino IDE

The Arduino IDE is downloaded from the Arduino web page. The download links are on this page:

<https://www.arduino.cc/en/Main/Software>

Download and install the IDE. When you run it for the first time, it will look something like:



This is showing you a new, blank program. Arduino programs are called “sketches”.

# Add Support for the Nano Every Boards

As shipped the Arduino IDE can build code for some of the processor types used in the Arduino range, but not for the Arduino “Due” used in this project. A simple download will add the Due:

1. Open the Arduino IDE
2. Click “Tools|Board|Boards manager” on the menu
3. Scroll down to the entry for “Arduino Mega AVR boards by Arduino” and click “install”
4. Your screen should look something like this:



# Download the Andromeda Software Repository

1. Visit the repository on github: https://github.com/laurencebarker/Andromeda\_front\_panel
2. Click “clone or download” then “download zip”
3. Store the zip file on your PC for example in the “downloads” folder
4. Open the zip file and extract to your PC; for example into a folder “SDR” in “documents”
5. There will be a folder called “Andromeda\_front\_panel-master” in your “SDR” folder

There are several folders:

|  |  |
| --- | --- |
| 1st prototype hardware | PCB design for an early prototype, with Arduino Due processor |
| Arduino nano every sketch | The Arduino program for the console. |
| documentation | The user guide and this installation guide |
| hardware |  |
| Hardware rev 4 | Prototype schematics |
| Hardware rev 5 | The schematics and PCB layouts for the console PCB |
| Odin based sketch | The Arduino program for an early prototype, using the Arduino Due and Odin CAT commands |
| sketch | The Arduino program for an early prototype |

## Build the code

To open the appropriate software sketch (the filenames etc are listed in the tables above)

1. Run the Arduino IDE
2. Use the "File|Open..." menu command
3. Open the “Arduino nano every sketch” folder then “andromeda\_front\_panel\_nano” folder
4. Navigate to "andromeda\_front\_panel\_nano.ino" and click "open"
5. you should now see the files listed in tabs above the editor window

You now need to tell the IDE what kind of board it is compiling for, and which serial port to use to connect to it.

1. Connect a USB cable between the Arduino programming port (next to the black power connector) and your PC.
2. It may be necessary to install device drivers at this point – follow any instructions.
   1. Click "board" on the "tools" menu and select “Arduino Nano Every”
3. Select “register emulations” on the “Tools” menu to say “none (ATMEGA4809)”
4. Click “port” on the “tools” menu and choose the Arduino COM port listed (mine is COM6)
5. Click "Verify/compile" on the "sketch" menu to compile
6. (A message “compiling sketch…” will appear. This will take around a minute and should result in a message saying the % of program space used)

Finally you need to upload the code to your Arduino:

* Click "Upload" on the "sketch" menu to upload to the Arduino
* A simple progress bar will show in the bottom window of the IDE, twice - for each of "programming" and "verify"
* When it has successful finished the last message will be "CPU reset"
* (note that an error message **avrdude: jtagmkII\_initialize(): Cannot locate “flash” and “boot” memories in description** is reported for the Arduino Nano Every but this can be ignored)

Your Arduino should now be executing the Andromeda code!

## Testing Your Panel

You can carry out a simple test while the panel is still connected to the Arduino software.

Click the “magnifying glass” symbol at the top right. That will open a new serial monitor screen.

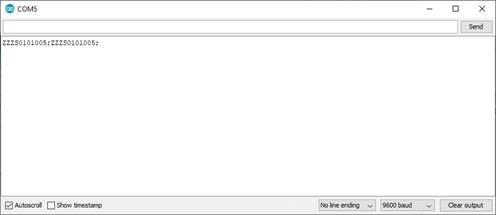
Turn your VFO encoder. You should see a “ZZZU” or “ZZZD” CAT message

Turn a different encoder: you should see a “ZZZE” CAT message

Press a button. You should see a “ZZZP” message (not the top left “power” button).

In the top box you can type a CAT command. Type **ZZZS;** and press Send. You should see a response in the bottom box. This means “I am an Andromeda front panel. My hardware version is 1 and my software version is 5”.

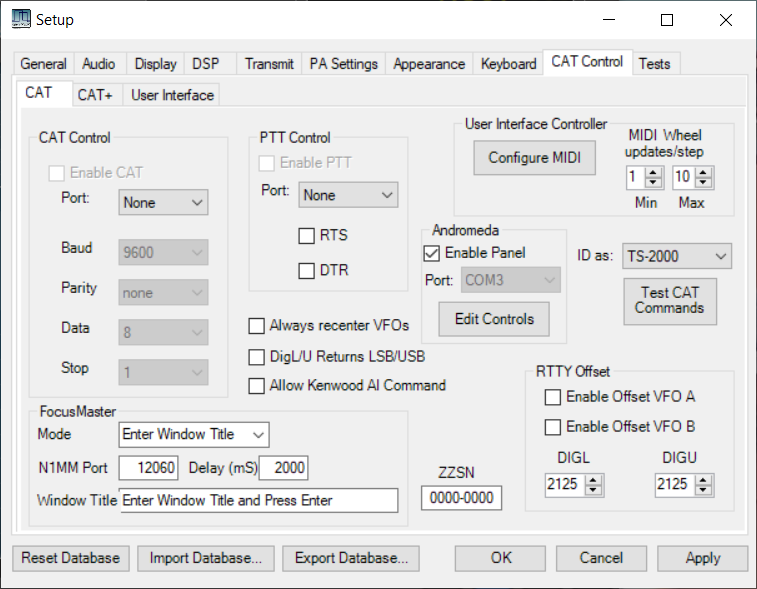
In the top box type **ZZZI011;** you should see an LED indicator turn on. Type **ZZZI010;** and it should go off.



## Connecting to Thetis

To connect the panel to Thetis:

1. Start the Thetis program
2. Open the “Setup” screen and select the “CAT” tab.
3. In the “Andromeda” section choose the COM port and click “Enable Panel”.
4. Close the setup screen
5. Your panel should now operate Thetis.
6. If you use a different “CAT control” box to enter the COM port number, the encoders and buttons will work but the indicators will not be controlled by Thetis.



# Thetis changes still needed

1. Different encoder functions during TX (to allow ATU fine tune)