# GANYMEDE Amplifier Protection Module Design notes

# Damage Mechanisms

LDMOS amplifiers seem to be vulnerable to damage from several effects:

1. Input overdrive, leading to damage through excessive power dissipated in the gate region
2. Excessive reverse power caused by incorrect LPF selection – likely to damage the LPF too
3. Excessive reverse power caused by poor antenna match
4. Excessive temperature
5. Excessive supply voltage, leading to exceeding VDSMax
6. Excessive current, exceeding IDmax

Factors 1 and 2 are relevant to add-on amplifiers used with existing transceivers where user error is possible. If the amplifier is part of a single transceiver system, these factors can be eliminated by design.

Factor 3 is a hazard with any amplifier. The consequence could be to exceed VDSMax or IDmax leading to rapid device failure.

Factor 4 is a hazard with any amplifier, but if appropriately heatsinked the temperature change will be relatively slow.

Factor 5 could be the consequence of a power supply failure. It could lead to rapid device destruction, but it only worth detecting quickly if there is an instant way to remove PA supply (eg with zero delay FET switch). If the PSU has failed, the transceiver system has failed anyway.

Factor 6 can be avoided within a single transceiver by design – by avoiding overdriving, and by setting appropriate margins between normal operating current and IDmax

# Protection Required

From those hazards an appropriate set of protections may be:

* Detect PSU voltage, and don’t turn on FET switch if PSU voltage too high.
* Instant amplifier “trip” by removing bias and PTT if reverse power is detected.
* Quick (c. 10us) trip by turning off FET switch in drain supply if excessive PSU voltage detected. (This may mitigate PSU failure, but will not eliminate risk of damage.)
* Quick (c. 10us) trip by removing bias and PTT if over-current is detected.
* Slow (c. 2ms) trip by removing bias and PTT if heatsink temperature exceeds trip threshold
* If fan control needed
  + Fan turned on if heatsink temperature exceeds “fan on” threshold
  + Fan turned off if heatsink temperature below “fan off” threshold



# User Interface

There needs to be indication to the user of what protective action has been taken and why:

* Excessive reverse power is most likely, and requires action to match the antenna
* Excessive Temperature means the TX duty cycle needs to be reduced after it has cooled down. The fault will clear itself, through the fan, given some time.
* Overvoltage conditions probably mean the PSU has failed. This can clear itself once voltage returns to normal, but there is probably an underlying problem.
* Overcurrent could be the result of excessive drive or a failure elsewhere. I

Possibly a RESET button should be provided before operation will resume from over voltage or overcurrent, because it’s likely there is a fault elsewhere and if uncorrected they will happen again immediately.

There could be user indications provided of the current sensor inputs:

* There needs to be user indication of VSWR – but that already exists.
* Indication of heatsink temperature and fan on/off would be useful but not essential
* Voltage and current could be indicated for debugging, but “normal” users don’t need to know them.

# Suggested Design

## Block Diagram



## Functions

1. An Arduino with simple software provides the control for the unit.
2. Each sensor value (temp, current, voltage, reverse power) is scaled using simple op amps
3. Each scaled sensor value is measured using the ADC in the Arduino
4. For current, voltage and reverse power: a comparator detects over-threshold using a processor-set threshold
5. Over current and over voltage thresholds are monitored using Arduino interrupts, with ~10us response time
6. Excessive reverse power operates a flip flop instantly
7. The effect of reverse power, current or voltage trips is to remove the bias supply, power supply to amplifier and drive to PTT relay
8. Temperature is monitored using a slower software process. The fan is operated automatically as required. Thresholds are programmed for fan on, fan off and over-temp trip.
9. A CAT message will indicate to THETIS that the amplifier has tripped, allowing PTT to be removed automatically.
10. User interface will be provided via:
    1. Touchscreen LCD;
    2. Using CAT messages, via a new screen in THETIS.

## Sensing

The sensing arrangements designed are:

Current sensed using a Hall probe.

PSU voltage sensed using a simple divider.

Temperature sensed using a thermistor

Forward and reverse Power can be sensed using a conventional VSWR bridge

## Temperature

Simple op amp circuit with 10K thermistor Digikey 495-2163-ND. Op amp output voltage v temperature characteristic calculated in spreadsheet.



The LM311 comparator input range is 0.5v to 3.5V with 5V VCC; if operated at 4V threshold we do NOT get a normal “output high”.

The Comparator threshold inputs need C/R filters. I have used 2x18K resistors and 2x1uF capacitor on each PWM signal.

# Arduino Issues

An Arduino Nano Every is suitable.

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

Suggested I/O Pin assignment:

|  |  |  |
| --- | --- | --- |
| **Function** | **Pin** | **Comment** |
| Built in LED | D13 | Try to keep free for debug usage |
| Current analogue Input | A0 |  |
| Temp analogue input | A1 |  |
| PSU voltage analogue input | A2 |  |
| Reverse power analogue input | A3 |  |
| Forward power analogue input | A6 | This isn’t strictly needed |
| Current threshold PWM out | D3 | Needs C/R filter before comparator |
| Voltage threshold PWM out | D5 | Needs C/R filter before comparator |
| Rev power threshold PWM out | D9 | Needs C/R filter before comparator |
| Current Comparator input | D11 | CPU pin PE00 |
| Voltage comparator input | D12 | CPU pin PE01 |
| SR flip flop “tripped” input | D8 | CPU pin PE03 |
| Reset out to Flip Flop | D4 |  |
| Amp enable output to AND gates | D7 |  |
| PSU enable output | D2 | Needs an open drain driver to power FET |
| FAN on/off output | D6 | Needs an open drain driver |
| PTT in | D10 |  |
|  |  |  |
| Nextion display | D1 (TX)  D0 (RX) | “Serial1”  D1(TX) to Nextion “RX”; D0 (RX) to Nextion “TX” |

Try to keep A4, A5 free (I2C). A7 free. D13, A6 not strictly needed – could be used if required.

The Arduino should be powered by a 12v feed to the board – not by USB. It needs to protect even if USB unconnected.

# Nextion Display

A 2.8” display will connect to Arduino D0, D1 (Serial1). Baud rate = 115200. 320x240 pixels.

Physical connection via 4 wires, with ends to plug onto a 4 pin 0.1” pitch header.

Red, black wires = +5v, GND. Approx 100mA and the Arduino 5V output is fine.

Blue: TX data from display. To Arduino RX0 pin

Yellow: RX data to display. To Arduino TX1 pin.

Screens suggested:

|  |  |
| --- | --- |
| Splash (Startup) screen 0  Displays for 10 seconds, or until after supply voltage OK AND PTT deasserted.  Then moves to RX screen automatically. |  |
| Normal RX (not tripped)  screen 1  “About” button brings up the “about” screen |  |
| Normal TX (not tripped)  Screen 2  “key down” counter at top right  “Info” button toggles between more/less parameters displayed |  |
|  |  |
| Tripped display  Screen 3  Highlighted red indicates cause. Current values shown  RESET greyed out until PTT removed and temp within limits |  |
| About  Screen 4  (information screen; needs s/w version display) |  |

# CAT

CAT commands can be added to report tripped/not tripped (and cause) and allow RESET. By choice these wouldn’t provide parameter values – just OK yes/no. That’s probably only one or two commands.

|  |  |  |
| --- | --- | --- |
| **Event** | **Message** |  |
| Amplifier trip | To PC: CAT Message ZZZAnn;  From PC: ZZZAnn; | nn=0: no trip; amplifier OK  Trip reports:  nn=1: tripped- excessive reverse power  nn=2: tripped – excessive drain current  nn=4: tripped – PSU voltage out of spec  nn=8: tripped – high heatsink temperature  nn=16: tripped – high forward power  To reset: nn=32: reset trip  Amplifier sends back no response if successful, or it unsuccessful a further trip report. |
| 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 (ZZZA)

# Functionality Required

|  |  |
| --- | --- |
| At start up | Send analogue thresholds to PWM DAC outputs  Clear the SR flip flop  Measure PSU voltage analogue input  If within limits, assert PSU enable  If over voltage, initiate trip  After few seconds, move on from “splash” screen |
| Temperature measurement | 1. Poll the temperature measurement approx. every 10ms 2. If temp above fan “on” threshold, set fan on 3. If temp below fan “off” threshold, set fan off 4. If temp above trip threshold, activate trip 5. If value visible on screen, update value |
| PSU voltage measurement | 1. Poll the voltage measurement approx. every 10ms 2. If value visible on screen, update value   If comparator reports voltage above threshold (interrupt), activate trip |
| PSU current measurement | 1. Poll the current measurement approx. every 10ms 2. If value visible on screen, update value   If comparator reports current above threshold (interrupt), activate trip |
| Reverse power measurement | 1. Poll the rev power measurement approx. every 10ms 2. If value visible on screen, update value   If SR flip flop activated (interrupt), activate trip |
| Forward power measurement | 1. Poll the current measurement approx. every 10ms 2. If value visible on screen, update value |
| When PTT activated  When PTT deactivated | 1. If not tripped, display “TX” on screen (if correct screen) 2. Initiate “key down” timer 3. If trip has occurred, disable reactivation 4. Remove “TX” display on screen 5. Stop key down timer and display 6. If trip has occurred, enable re-activation |
| When trip occurs (immediate, by interrupt code) | 1. Deassert PTT, bias enable 2. Deassert PSU enable 3. Set “tripped” |
| After trip occurred (slower s/w) | 1. Send CAT message with fault state 2. Record current analogue input values 3. Change Nextion screen 4. Approx. every 10ms thereafter: 5. Poll analogue sensor values 6. If now within limits, and PTT has been removed, activate “RESET” button |
| When RESET pressed | 1. Clear SR flip flop 2. Check comparator inputs OK 3. Check temp OK 4. If all OK:    1. re-assert PTT/bias enable and PSU    2. set “not tripped”    3. Change Nextion display    4. Send CAT message with clear state |
| When CAT RESET message received | If values now within limits, and PTT has been removed, perform RESET action above |
|  |  |

# Software Build Instructions

# 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:

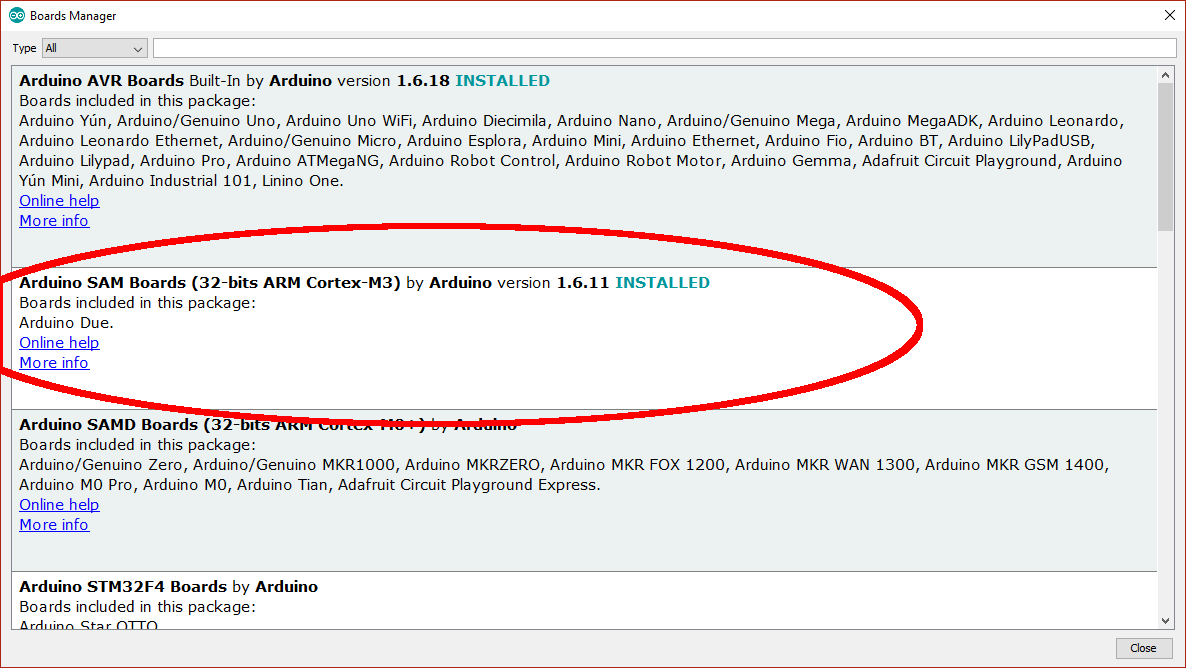


This is showing you a new, blank program. Arduino programs are called “sketches”.

# Add Support for the Arduino Nano Every Board

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 SAM Boards (32-bit ARM Cortex-M3)” and click “install”
4. Your screen should now look something like this:



# Install Libraries into the Arduino IDE

The next step is to install 4 libraries into the Arduino library. This will provide access to the code that we have used as part of the Odin build.

The Arduino system loads libraries into a folder it created on your computer; usually that folder is installed into the “documents” folder called “Arduino\libraries”. On my computer that folder is “C:\Users\loz barker\Documents\Arduino\libraries”. Use windows explorer to find that folder so you know where it is.

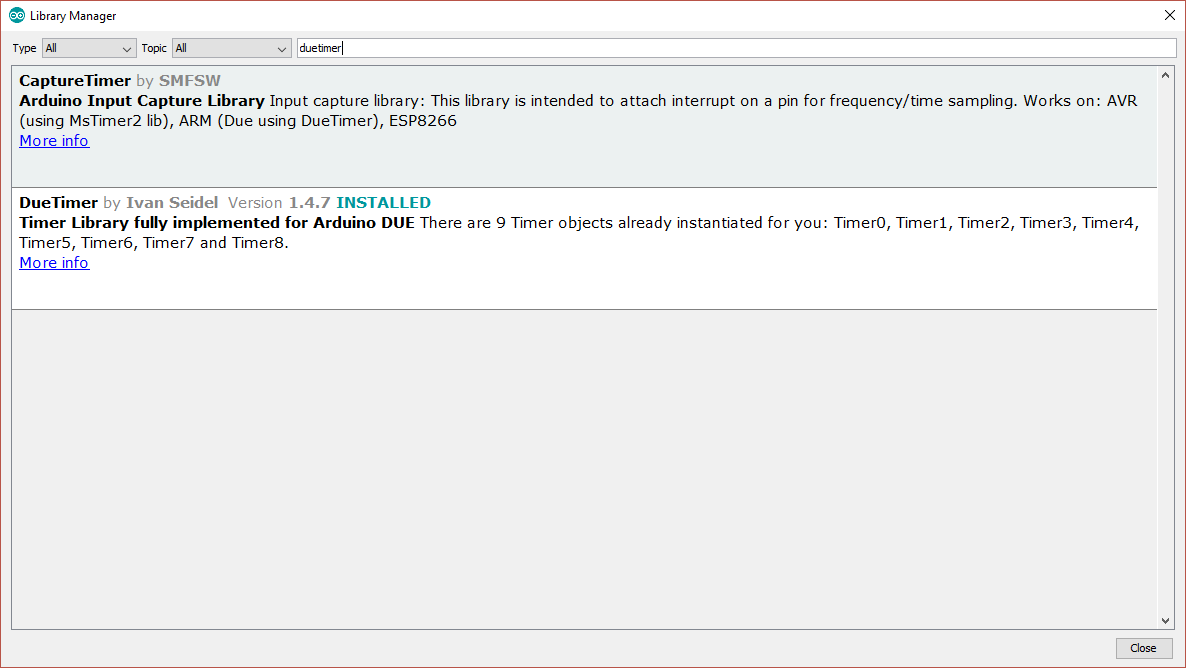
The 4 required libraries are:

|  |  |
| --- | --- |
| DueFlashStorage | Provides persistent storage so settings are kept while power is removed |
| DueTimer | Provides access to the Arduino’s timer, so the code can synchronise to a 1 millisecond time count |
| Encoder | Controls the optical VFO encoder |
| ITEADLIB\_Arduino\_Nextion | Controls the touchscreen display |

The first two are accessed through the the Arduino library manager; the second two have to be installed manually.

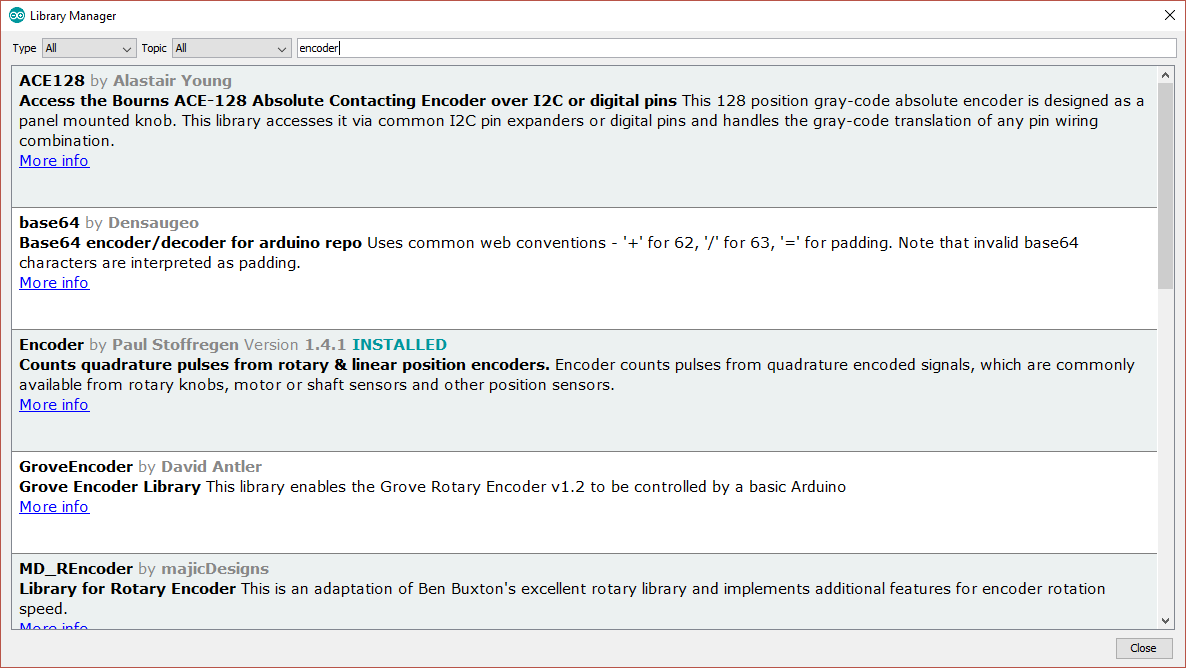
## DueTimer

1. Open the Arduino IDE
2. Click "Sketch | Include Library | Manage libraries..." on the menu
3. In the library manager type "due timer" where it says "filter your search" and hit enter
4. Find "DueTimer" by Ivan Seidel and click "install"
5. You should now have a folder “Documents\Arduino\libraries\DueTimer”
6. Your screen will look something like this:



## Encoder

1. In the library manager type "encoder" where it sayds "filter your search" and hit enter
2. Find "encoder" by Paul Stoffregen and click "install"
3. Your screen will look something like this:



## DueFlashStorage

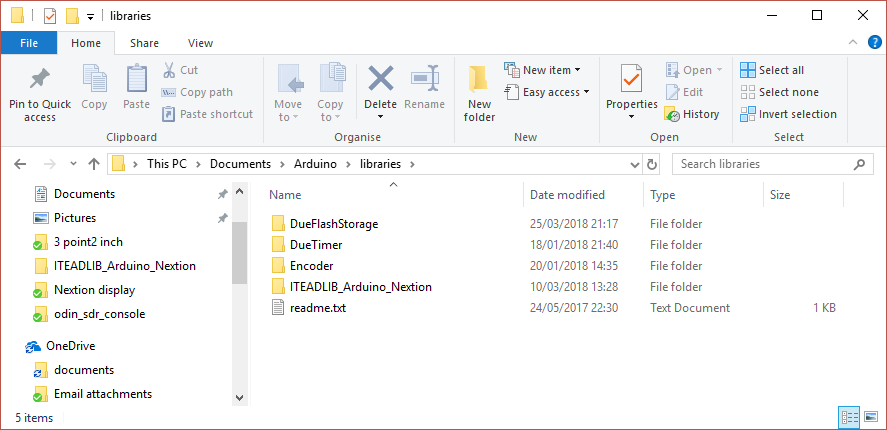
This needs to be installed manually. The procedure is as follows:

1. Visit the repository on github: <https://github.com/sebnil/DueFlashStorage>
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 all files. You will now have a folder “DueFlashStorage-master” which will hold one folder also called “DueFlashStorage-master”
5. Rename the second folder “DueFlashStorage” (remove the “-master” part)
6. Copy that whole folder to your “documents\arduino\libraries” folder

## ITEADLIB\_Arduino\_Nextion

This needs to be installed using a similar process:

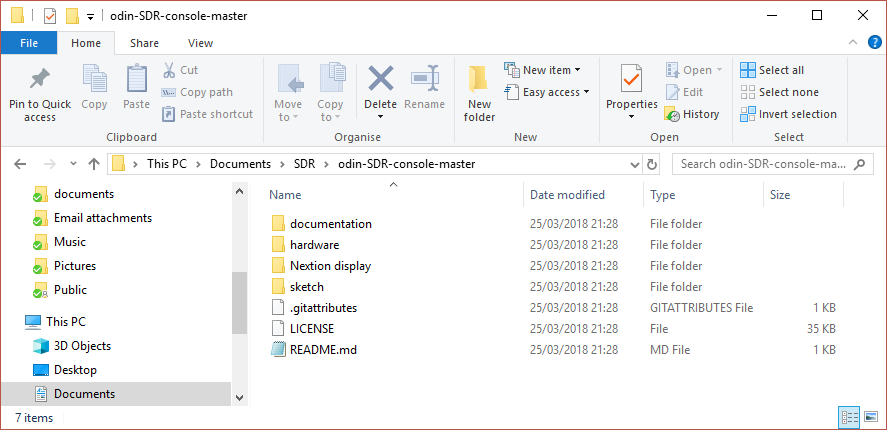
1. Visit the repository on github: <https://github.com/itead/ITEADLIB_Arduino_Nextion>
2. Click “clone or download” then “download zip”
3. Store the zip file on your PC for example in the “downloads” folder
4. Open the zip file and extract all files. You will now have a folder “ITEADLIB\_Arduino\_Nextion-master” which will hold one folder also called “ITEADLIB\_Arduino\_Nextion-master”
5. Rename the second folder “ITEADLIB\_Arduino\_Nextion” (remove the “-master” part)
6. Copy that whole folder to your “documents\arduino\libraries” folder
7. (This is the library published by the display manufacturer. Be aware there is some foul language in the "html" folder - delete the entire "html" folder if you do not want that)
8. Your “documents\arduino\libraries” folder should now look something like:



The ITEADLIB folder needs to be patched in the next phase!

# Download the Odin Software Repository

1. Visit the repository on github: <https://github.com/laurencebarker/odin-SDR-console>
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 “odin-SDR-console-master” in your “SDR” folder and its contents will look something like:



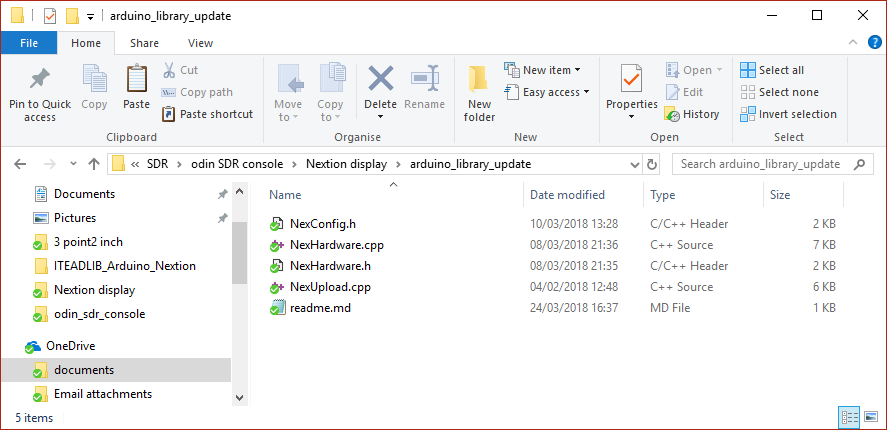
There are several folders:

|  |  |
| --- | --- |
| Documentation | The user guide and this installation guide |
| Hardware | The schematics and PCB layouts for the console PCB |
| Nextion Display | Files for 2 things:   * For the Nextion display itself, setting out the layouts of the screens used * Files to patch the Arduino library for the display |
| Sketch | The Arduino program for the console. |

## Patch the ITEADLIB Library

Four files (plus a readme file) need to be copied from the Odin repository to the ITEADLIB folder in the Arduino libraries.

1. Open the folder “nextion display\arduino\_library\_update”
2. It will have files as follows:

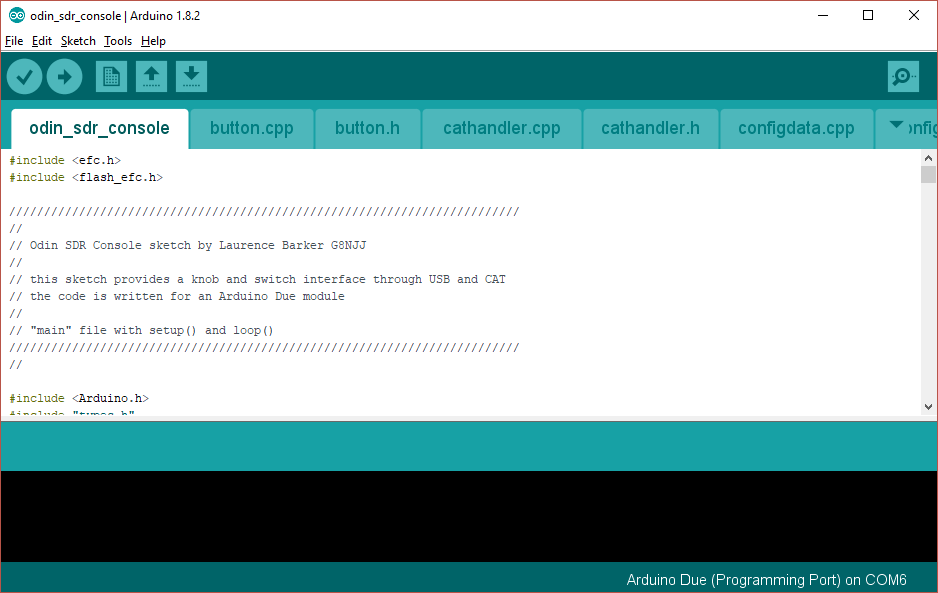


1. Select then copy those files
2. Navigate to your folder "documents\arduino\libraries\ITEADLIB\_Arduino\_Nextion"
3. Paste the 5 files there. 4 existing files will be replaced and the readme file will be added.

## Build the code

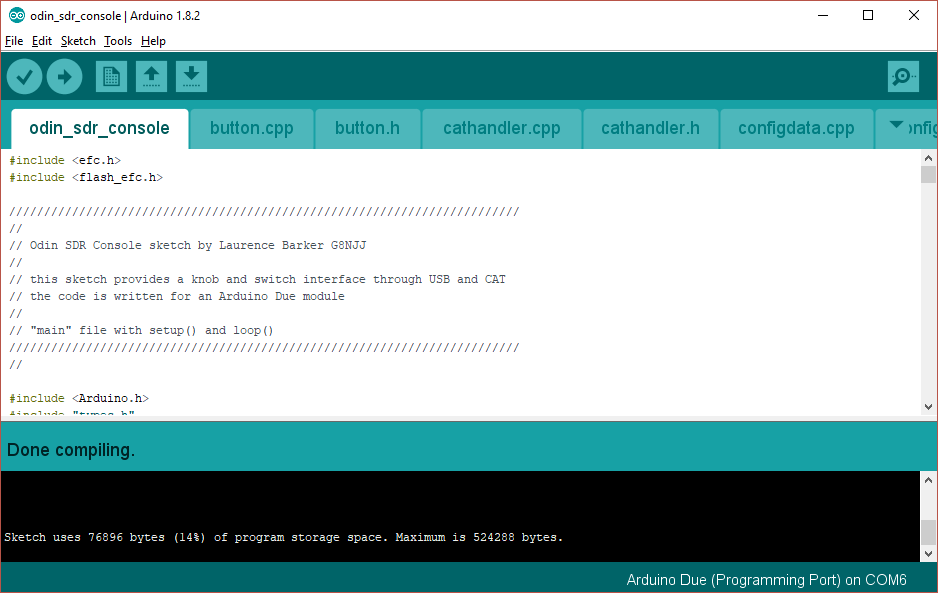
To open the Odin software sketch:

1. Run the Arduino IDE
2. Use the "File|Open..." menu command
3. Navigate to "odin\_sdr\_console.ino" in folder "Documents\SDR\odin-SDR-console-master\sketch\odin\_sdr\_console" and click "open"
4. you should now see the files listed in tabs above the editor window



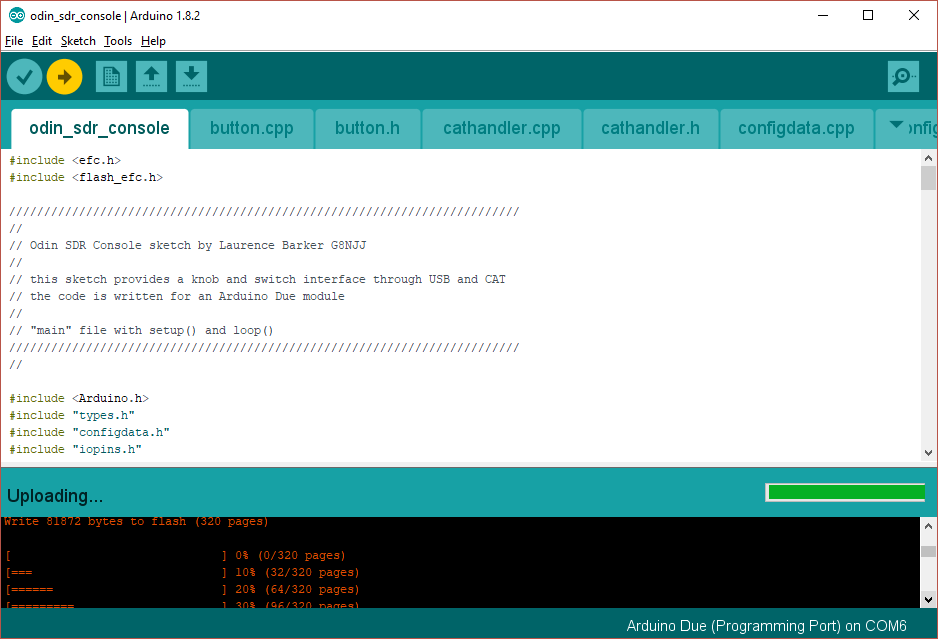
You now need to tell the IDE what kind of board it is compiling for, and which serial port to use to connect to it.

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.
3. Click "board" on the "tools" menu and select "Arduino Due (programming Port) from the list"
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"





Your Arduino should now be executing the Odin code!

## Programming the Nextion Display

The Nextion display needs to be programmed with the file “ganymede display.tft”. The simplest way is as follows:

1. Use windows explorer to copy the file to a micro SD card
2. Turn off the protection board
3. Insert the SD card into the socket on the display
4. Turn on the protection board
5. The display will recognise the SD card and copy the programming data into itself. It will give a message to say when it has finished.
6. When it has finished, remove power and remove the SD card.