# Aries ATU (Arduino Nano 33 IoT)

# 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  eg 00001850000 = 1.85 MHz  (expected to be steps of 10KHz) |
| TX Antenna change | CAT message ZZOCn;  Sent from PC to Aries. | n=1: Ant1; n=2: Ant2; n=3: Ant3 |
| RX Antenna change | CAT message ZZOAn;  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 |

# 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.



1st bench model: capacitance steps in 10pF steps up to total 2800pF; inductance stepped 40nH steps to total 10uH. Tuner could match 8:1 low impedance and 8:1 high impedance resistive loads at 1.9MHz.

Suggested change: change capacitance step to 5PF, inductance step to 20nH to give better resolution on the other bands and accept limitation on Topband.



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 S11on an Argand diagram. It is a useful way to visualise what a tuner needs to do. 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.

Frequency measurement needs a crystal clock; that would rule out the Arduino Nano Every which has an RC oscillator. A prescaler (eg divide by 16) would be needed. An Arduino Nano 33 IoT seems suitable. This has a SAMD21G18A processor.

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

## Timers

Previous projects suggest I should know what timers are already used! The Arduino time functions (eh milis() seem to use the processor “systick” timer. From the variant.cpp file:

TCC0, TCC1, TCC2, TC3, TC4, TC5 can all be used for PWM. TC5 used for PWM on Dig3

I will need a timer tick interrupt. There is an Arduino Zero timer library Github.com/EHbtj/ZeroTimer seems to use TC3 and/or TCC0. The timer period is specified in microseconds.

## Hardware I/O

Relays could be SPI driven using 2x TPIC6B595 shift register.

A divide-by-16 prescaler is needed; we also need a way to hold it in a “not counting” state under processor control. This is roughly what I’ve used before:



Figure 5: Prescaler for frequency count

## Debug connector

|  |  |  |  |
| --- | --- | --- | --- |
| **Pin** | **Signal** | **Pin** | **Signal** |
| 1 | +5V | 2 | GND |
| 3 | SCL | 4 | SDA |
| 5 | ENC1A | 6 | ENC1B |
| 7 | ENC2A | 8 | ENC2B |
| 9 | Encoder pushbutton | 10 | tune pushbutton |

4x20LCD display needed for debug

SCL, SDA connect to the LCD display

The encoder and pushbutton inputs need pullup resistors.

Encoder 1: adjust inductance

Encoder 2: adjust capacitance

Encoder button press: toggle coarse/fine tune (L/C step size)

Encoder button long press: toggle between Low Z and High Z mode

Tune button press: initiate TUNE algorithm manually



## I/O Pin assignment

|  |  |  |
| --- | --- | --- |
| **Function** | **Pin** | **Comment** |
| Serial | TX/DIG0  RX/DIG1 | Allows use of Nextion touchscreen I/O |
| SPI | D13/SCK  D11/MOSI  (D12 unused) | Serial clock and data to TPIC6B595 |
| I2C | A4 / SDA  A5 / SCL | For EEPROM  also LCD for debug display uses I2C |
| LOAD | DIG6 | Rising edge loads serial data from SPI |
| High/Low Z Relay | DIG8 | 1=high Z; 0=low Z |
| PTT | DIG10 | 0=TX; 1= RX. Needs pullup. Interrupt driven. |
| VSWR Bridge inputs | A0, A1 | A0=fwd; A1=rev. We will need to scale the analogue voltage for 3.3V max |
| External TUNE input | DIG9 | Needs pullup & interrupt. 0=TUNE; 1=no tune. |
| Status LED | DIG7 |  |
| Freq Count input | DIG2 |  |
| Count enable output | DIG3 | Must be PWM capable so we can use PWM to give timed “count” period |
| Debug pins: |  |  |
| Encoder 1 | A2, A3 | Can these all go to a 10 pin header, with +5v GND SCL and SDA? That will mean all the debug resources are on one connector. |
| Encoder 2 | A6, A7 |
| Encoder pushbutton | DIG4 |
| “Tune now” pushbutton | DIG5 |

## I2C Device Assignment

|  |  |
| --- | --- |
| 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)

# EEPROM I2C Interface

An external EEPROM will be needed. We 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. The Microchip 1Mbit EEPROM (24FC1026-I/P) will be suitable and an Arduino library is available. FC devices can clock at 1MHz.

The EEPROM needs to be connected to the I2C interface, which is also used in development/debug for an LCD display.

## Race problem

There was a potential race issue with the EEPROM. This can be fully avoided. Simply ensure that the display is not accessed during the TUNE algorithm, and that the new solution data is written back to EEPROM before TUNE complete.

In operational use, the debug display code should be removed anyway.

## EEPROM Data access

Data for an entire antenna will be read into SRAM when the antenna is changed.

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).

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.

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

# SPI Interface

Serial data will be shifted into two TPIC6B595 shift registers. A third will be provisioned for an add-on board to drive a T/R relay and 3 antenna select relays: this would allow standalone operation with older HPSDR radios. Byte 1 is the first shifted, reaching the end of the SR chain.

|  |  |
| --- | --- |
| Byte | Meaning |
| 1 | T/R and relay control word Bit 0: TR relay. 1 = TX  Bits 3:1: antenna select 3-1. 1 = antenna selected. |
| 2 | Capacitance values. Bit 0 = smallest. |
| 3 | Inductance values. Bit 0 = smallest. |

RX and TX relays could be different. In most cases this register will be written whenever there is a T/R change. However any change of RX antenna should be sent to the hardware immediately unless TX is already in progress.

SPI data to be written whenever PTT changes state, when RX antenna changed, or when new TUNE solution sent out during tune. There are 2 potential race conditions

1. If RX antenna data is already being written when PTT asserted, the data will need to be written again when the shift ends. A flag will need to be set.
2. If TX strobe deasserted while tune solution being written out during the tune algorithm

# PTT

When PTT pressed you get several “press” events. This could be because of the presence of RF, moving the ground while the signal passes through the threshold. Suggest instead:

1. Set PTT from interrupt;
2. Set 32 ms “min PTT duration” timer;
3. Poll PTT to see if it should be released
4. When released, set 16ms “min PTT inactive” timer

# 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 = 0.0837N where N = Arduino ADC reading.

(From the Stockton bridge spreadsheet: 3.3V input corresponds to 147W, ie 85.7Vrms

ADC reading N = 1024\*Vin/3.3 Therefore Vrms=85.7N/1024 = 0.0837N)

Power = Vrms2/50

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

# Stored Tune solution Data Structures

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

Solutions are stored every 10KHz, starting at 0; each antenna stores solutions for all frequencies from 0 to 61.49MHz (just over the Nyquist rate for HPSDR radios). Solutions numbered 0-6149; total count 6150

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 |

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

Proposed approach: store solutions for whole HF band in SRAM for one antenna. If the antenna changes, load a different block from EEPROM.

* 1. Storage required ~18Kbyte. 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

|  |  |  |
| --- | --- | --- |
| unsigned int | GTunedFrequency10 | Frequency the ATU is tuned to, in units of 10KHz |
| bool | GATUEnabled |  |
| unsigned int | GQueuedCATFrequency | 0xFFFF: no frequency stored  0-6149: 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 |
| 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  This holds full data for one antenna |



Figure 6: 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  Select closest solution or bypass  Send CAT message with solution available/not available |
| When new TX antenna received | Temporarily set bypass “null solution”  Read new block of data from EEPROM  Check tuning solutions for antenna N around current frequency  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 |

## Original 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);

The current algorithm executes potentially 144 search steps:

* Initial Quick tune:
  + 16 fine steps L
  + 16 fine steps C
* Full tune, if quick fails:
  + 16 coarse C, low impedance
  + 16 coarse L, high impedance
  + 16 coarse L or C depending on best match
  + 16 mid step L
  + 16 mid step C
  + 16 fine step L
  + 16 fine step C
* Note that:
  + For low frequencies we can accept bigger steps
  + For high frequencies we can limit the search space

## New Search Algorithm

With real antennas the algorithm does not always find a match, or the best match. The “narrow down” process seems OK but the initial search isn’t comprehensive.

|  |  |
| --- | --- |
| Current |  |
| Required |  |
| And Potentially |  |
| And for some loads |  |

Suggested approach:

* Keep the 2 stage algorithm
* Have each part table driven, so the search range can be set by frequency
* For the first part, have the number of search steps set by the table
* Potentially that might allow user settable search algorithms to trade speed & performance
* Stage 1a: perform sweeps of one parameter according to table
* Stage 1b: for the minimum found, perform a sweep of the opposite parameter
* Stage 2a: medium sweep 1st parameter
* Stage 2b: medium sweep 2nd parameter
* Stage 2c: fine sweep 1st parameter
* Stage 2b: fine sweep 2nd parameter
* In all cases you swap over the parameter being swept at each stage
  + 1st table:
    - Upper frequency
    - Stage 1a Start Row for this frequency
    - Number of rows for this frequency
    - Max L, Max C
    - Stage 1b min, max, step
    - Stage 2 mid step +/- range, step size
    - Stage 2 fine step +/- range
    - (the entire algorithm can be changed by replacing this table with no code change)
  + 2nd table with Parameters for stage 1a:
    - High/Low Z
    - Sweeping L/C
    - Fixed value
    - Min, max, step value
* To be recorded in 1st stage, to find min
  + C value
  + L value
  + High/Low Z value
  + VSWR value
  + (it doesn’t matter which sweep we find it on)



# Testing

## 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

# Arduino Libraries

## Board support

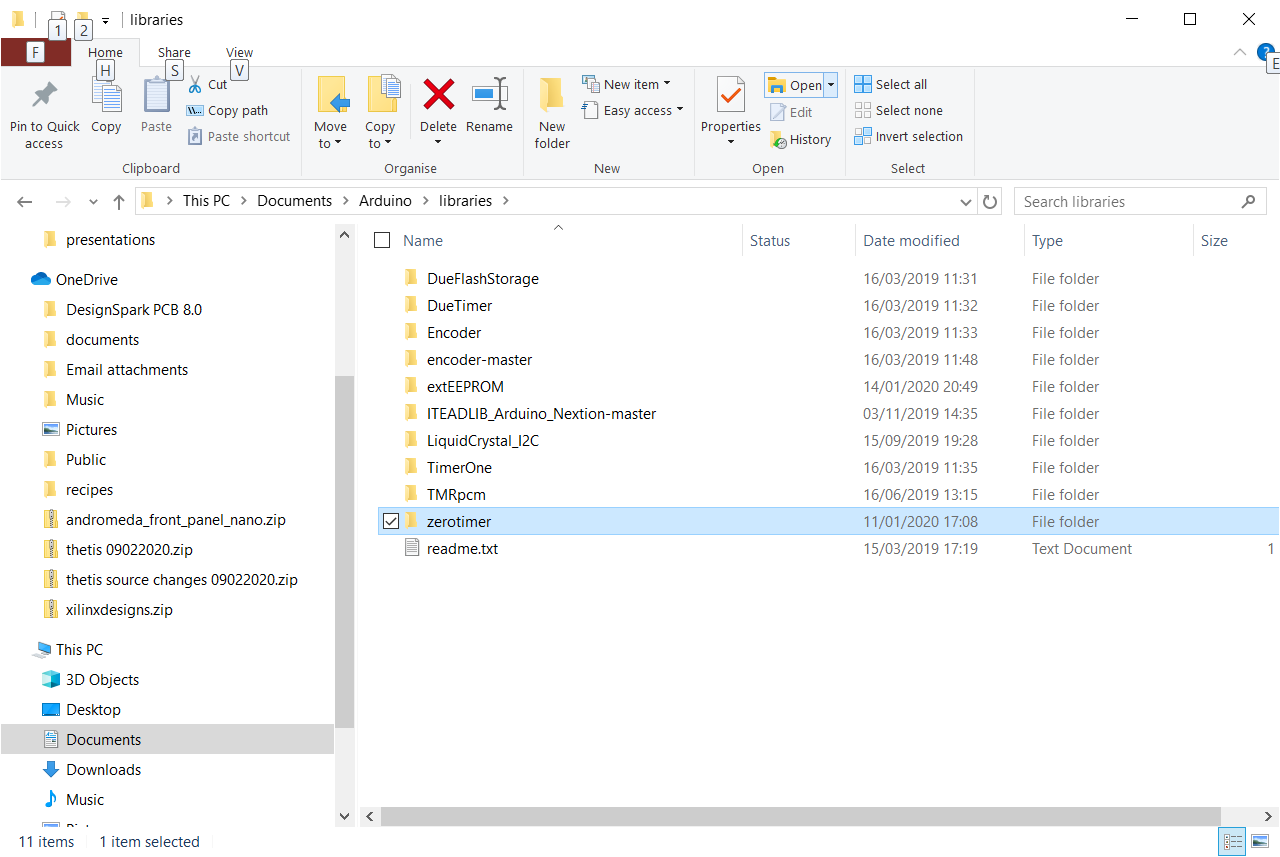
Install “Arduino SAMD boards (32 bit ARM Cortex-M0+) by Arduino”



## ZeroTimer Library

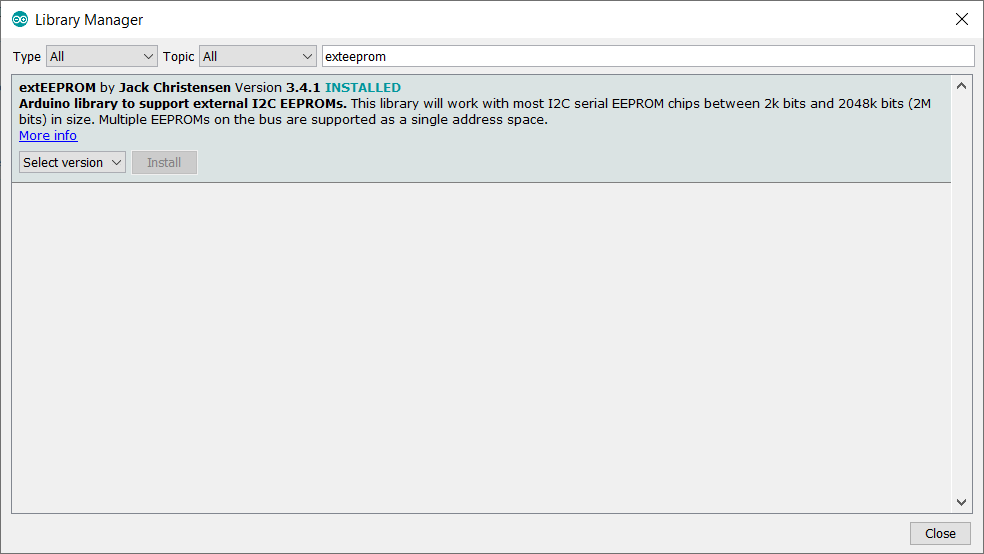
1. Download from <https://github.com/EHbtj/ZeroTimer>
2. Open the zip file and extract all files. You will now have a folder “ZeroTimer-master” which will hold one folder also called “ZeroTimer-master”
3. Rename the second folder “ZeroTimer” (remove the “-master” part)
4. Copy that whole folder to your “documents\arduino\libraries” folder

Your Arduino libraries folder will now include ZeroTimer:



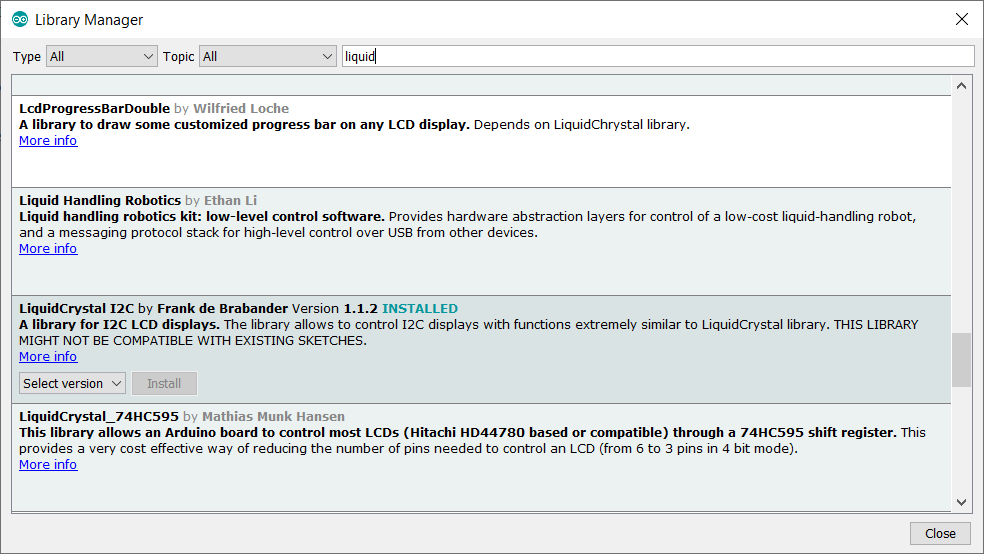
## extEEPROM Library

1. click Tools > Manage Libraries…
2. The “installed libraries” form opens
3. Type “exteeprom” into the bar at the top
4. The library “extEEPROM” should be shown. Click Install
5. The library should show “installed”



## LiquidCrystal\_I2C Library

1. click Tools > Manage Libraries…
2. The “installed libraries” form opens
3. Type “LiquidCrystalI2C” into the bar at the top
4. The library “LiquidCrystalI2C” should be shown. Click Install
5. The library should show “installed”



## Compiling

Select Tools > Board > Arduino NANO 33 IoT

Click the “tick” icon to compile

Click the “right arrow” icon to download