# Log VSWR Bridge

Kjell has designed a high power VSWR bridge which could be connected to an Arduino to implement a VSWR meter with electronic display. This could be an Arduino Nano Every with Nextion display.

## VSWR Bridge

The VSWR bridge is designed to have 50dB loss to the log amp inputs. Each input is routed to an AD8310 log amp. The output voltage is given in the Analog Devices data sheet. ~24mV/dB. It does tail off at very low powers so it will read a non zero power output for 0W in.

Kjell has recommended these component value changes from the 1st schematic:

If the maximum power is set to 2 KW, the total attenuation has to be: 63 dBm minus 13 dBm=50 dB. As we have 23 dB in the coupler, 6 dB for the first attenuator, the second attenuator must be 21 dB. By changing the value of R13 and R15  to 60 Ohm and R16 to 280 Ohm we can use standard values (2x120 Ohm and 2x560 Ohm) I believe that this is near enough.

The power dissipation in the first attenuator must be 20W if we talk about continuous duty cycle but if we say 2 kW PEP the present values and 3W for each resistor, it should work.

C13 and C14 must be removed, the output of the detector should not have capacitive load. I have also set the gain in the last OpAmp to 1,66 as you suggested(R26/R28=15K and R25/R27=10K).

A quad op amp provides buffering and scaling of the analogue power readings. There is a simple op amp voltage follower (U1a, U1d) followed by a variable gain and offset op amp. For the processor version, fixed gain and offset would be better with the values chosen to scale to 0-5V or most of that range. A fixed gain of 1.66 with no offset would be OK. Kjell suggests R26/R28=15K and R25/R27=10K to give a scaling of 1.667.

The PCB will be designed for a Hammond enclosure, Type 1455 PN 1455K1201 (122x78x43mm).

# Arduino Hardware

Two analogue inputs are needed for forward and reverse power. Suggest A0=forward, A1=rev. The analogue inputs need a CR filter with ~5ms time constant.

There is nothing more needed – a simple prototype board is all that’s required.

# Displays

The idea is to have several displays, with a button to step between the displays. The unit will power on to the last display used and the last display scale used.

|  |  |
| --- | --- |
| The simplest display will be a bargraph of linear power and VSWR  **Average** button toggles between average reading and peak reading  **Scale** button changes display full scale 2W/20W/200W/2000W  **Display** button moves to the next display | Forward power (W)  VSWR  Display |
| We may want a display with log power (dBm) for each of forward and revers ports. This shows the full dynamic range.  No “peak” display on this one! | Forward power (dBm)  VSWR  Display |
| A crossed needle display is harder. The difficulty is in drawing the display background. But they do look good! It will suffer a little from flicker as the background has to be erased and redrawn to move the needles. |  |
| An analogue meter with forward power, and bargraph VSWR is available. |  |
| An engineering display with forward and reverse power and input voltages is also available. |  |

For several of the displays there is a choice of immediate or peak reading. And several power scales needed. For linear power, scales of 2W/20W/200W/2KW are used.

For each display: have a bitmap background with Nextion drawn bars. Use draw.io to draw the backgrounds, then export a bitmap of the right size. That seems to work OK.

# Sketch Code

At the moment this works with a 20ms “main” timer tick and a 1ms “fast” tick.

The fast tick records a new ADC reading every tick (ie both forward & reverse readings takes 2ms) so the hardware design will need a CR filter to hold peaks with a time constant of ~5ms.

I’m aware that the display code currently sometimes takes longer that 20ms to execute (when redrawing the crossed needle display) and consequently the 1Hz LED blink slows down. This does not cause any other adverse impact.

At the moment there is debugging code that writes to the serial port. The code runs slowly unless a serial text window in the Arduino editor is opened. Once the code is complete this can be removed.

## Analogue I/O Code

Long term I should read from the ADC much faster, using the ADC interrupt.

10 bit ADC; reads 0-4.99v

ADC input voltage = 5\*reading/1024

Logamp output voltage = (5\*reading/1024)/1.666

Logamp output voltage rises by 23.4mV/dBm and intercept ~-96dBm (from graph)

Therefore input power = -96 + 42.75\*logamp output voltage

input power = -96 + 42.75\*(5\*reading/1024)/1.666

**input power = -96 + 0.1253 \* reading**

Every 1ms the h/w driver reads new voltages. For each of forward and reverse it keeps the largest value found and a sum value. Nominally every 20ms it calculates parameters, then updates display. It calculates both an average power (using the sum) and a peak power (using the largest). Input values are used in two ways:

1. To convert directly to a power reading in dBm using simple linear scaling
2. Using a large lookup table it is converted to a line voltage, from which linear power can easily be calculated. The lookup table will need to be replaced to calibrate, and a spreadsheet is used to calculate the data that can simply be pasted into the sketch.

## Display Code

A Nextion 3.2” 400x240 pixel display is used – NX4024T032\_011. The displays have been drawn using the Nextion editor.

Be aware that the Nextion display takes time to draw bars!

A discovery on this project has been that the Nextion library stalls the processor in the calls to nexloop(). When redrawing the display this stalls for 40ms and apparently when drawing a line stalls for 80ms.

This code follows the usual pattern, with initialisation code then “tick” code to update.

Obscure problem when USB not connected: the 1HZ blink was much slower than normal and using the touch buttons often failed. If turns out when most commands were sent the code ended up trapping the 100ms timeout on the recvRecCommandFinished() function. It was waiting for a display response that never came.

nexInit() sends sendCommand("bkcmd=1"); to say “send a response on successful commands”

when USB connected that worked; when USB disconnected it didn’t.

Solution: when changing from splash page to operational page, add another sendCommand("bkcmd=1");

And we get correct 1Hz tick with or without USB.

# 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 “Nano Every” 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 now look something like this:



# Install Libraries into the Arduino IDE

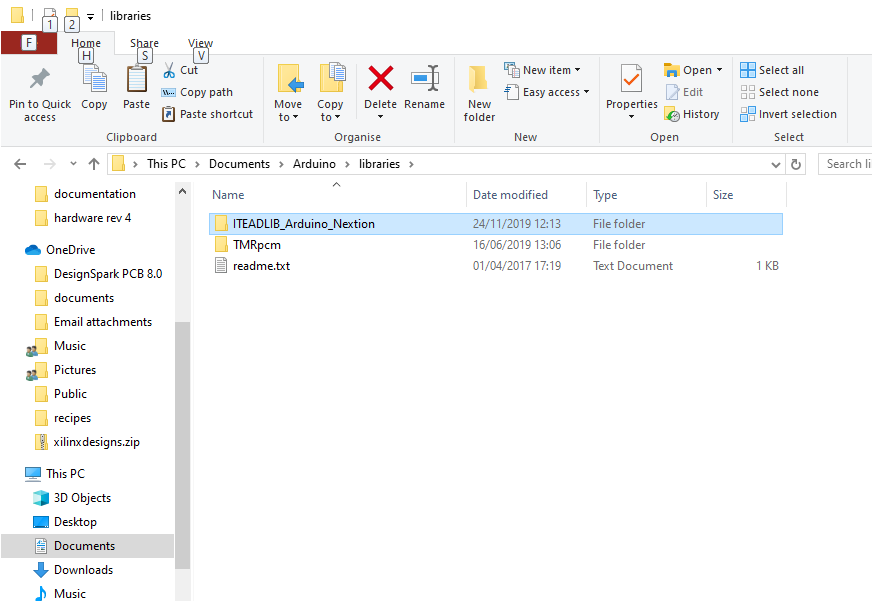
The next step is to install a library into the Arduino library set. This will provide access to the code that we have used as part of the Ganymede build.

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 required library is to control the touchscreen display: “ITEADLIB\_Arduino\_Nextion”. It has to be installed manually.

## ITEADLIB\_Arduino\_Nextion

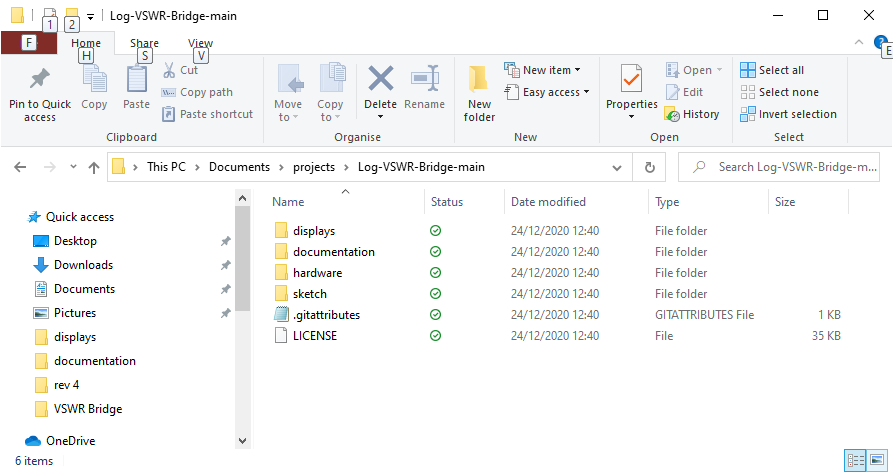
This needs to be installed using a similar process:

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 have that library:
9. 

The ITEADLIB folder needs to be patched (replacing 4 files) in the next phase!

# Download the Log VSWR Bridge Software Repository

1. Visit the repository on github: <https://github.com/laurencebarker/Log-VSWR-Bridge>
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 “projects” in “documents”
5. There will be a folder called “Log-VSWR-Bridge-main” in your “projects” folder and its contents will look something like:



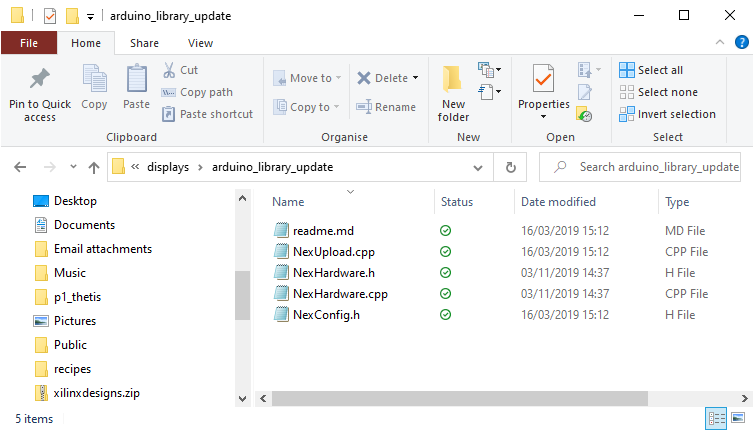
There are several folders:

|  |  |
| --- | --- |
| Displays | Bitmap backgrounds for the displays and the Nextion display editor file  Code to patch the Nextion library |
| Documentation | This installation guide and some calculator spreadsheets |
| Hardware | Schematics for the hardware (incomplete) |
| Sketch | The Arduino program for the controller. |

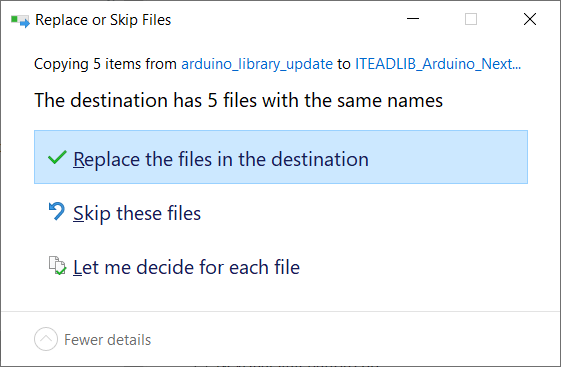
## Patch the ITEADLIB Library

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

1. Open the folder “displays\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. Make sure you select “replace the files in the destination”

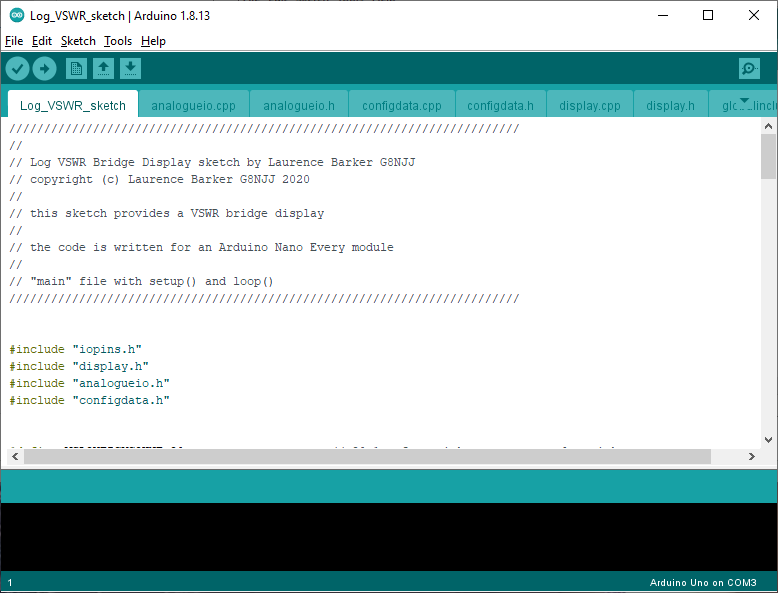


1. 4 existing files will be replaced and the readme file will be added.

## Build the code

To open the log VSWR bridge software sketch:

1. Run the Arduino IDE
2. Use the "File|Open..." menu command
3. Navigate to " Log\_VSWR\_sketch.ino" and click "open". It is in folder: "\documents\projects\Log-VSWR-Bridge-main\sketch\Log\_VSWR\_sketch"
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 Nano Every” from the list
4. Click “registers emulation” on the “tools” menu and select “none (ATMEGA 4809)”
5. Click “port” on the “tools” menu and choose the Arduino COM port listed (mine is COM4)
6. Click "Verify/compile" on the "sketch" menu to compile
7. (A message “compiling sketch…” will appear. This will take around a minute and should result in a message saying the % of program space used)

Graphical user interface, text, application, email

Description automatically generated

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, “uploading”
* When it has successfully finished the last message will be “done uploading”
* (A warning “avrdude: jtagmkII\_initialize(): Cannot locate "flash" and "boot" memories in description” is normal can be ignored)

Graphical user interface, text, email

Description automatically generated

Your Arduino should now be executing the VSWR Bridge code!

## Programming the Nextion Display

The Nextion display needs to be programmed with the file “logbridge\_32 display.tft”. This will be in the “displays” folder. 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.