/dspectrum/

by nullwolf

DSpectrumGUI – Rapid Reverse Engineering Guide

Table of Contents

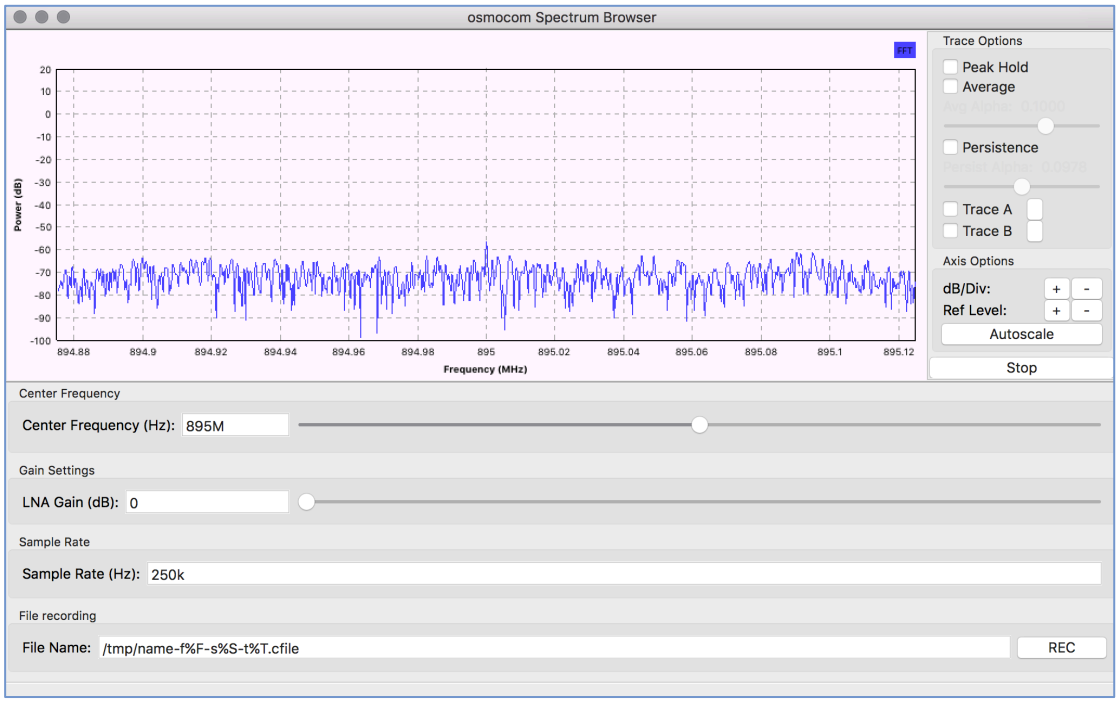
Capturing transmissions

Capture-1 : Connect the RTL-SDR dongle to your computer.
Capture-2 : Open a new terminal window and type:

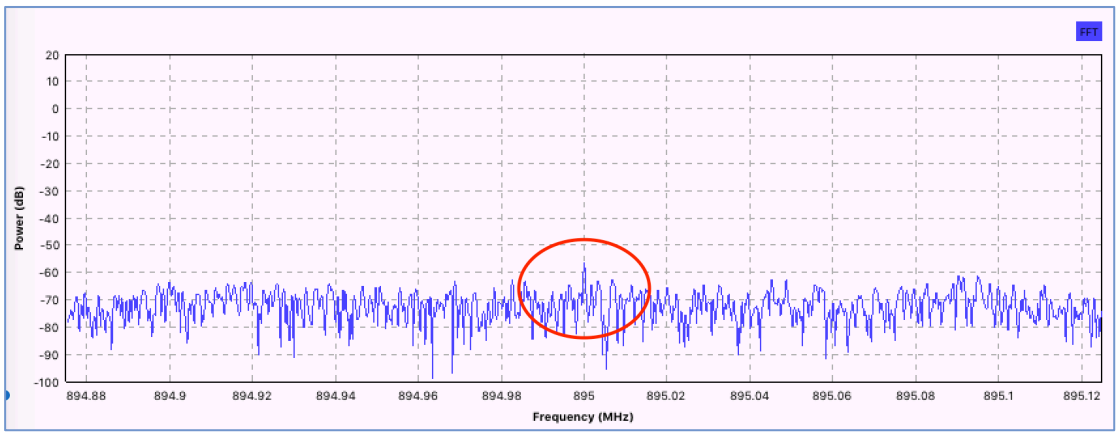
osmocom_fft

Capture-3 : Press enter.

A window similar to the following should have spawned.



Capture-4 : Enter the desired frequency in the Center Frequency field.
Note: the frequency needs to be off-center otherwise the "DC Spike" (shown in red in the screenshot below) will overlap with the signal we are trying to capture.



With a little trial-and-error, we can determine an appropriate offset to use. In this particular example, we ended up with 434.2Mhz being a good frequency to tune into to capture a clean signal on 434.24Mhz. This placed the signal to the right of the DC Spike without overlapping it.

Tip: Make sure you press the Enter button after editing any field in osmocom, otherwise your changes won't be picked up. If any fields have a pale pink background colour, you haven't pressed the enter button and it will ignore your change.

Capture-5 : We also need to modify the file name so that it describes which device we are capturing and what button was pressed to avoid any later confusion. Note that the f%F, s%S, and t%T fields in the name record the frequency, sample rate, and timestamp of the transmission respectively. It's a good idea to leave those parts of the filename intact and just replace the string "name" with a more descriptive string. In this case, our full file name was: "/tmp/remote-blue-arm-f%F-s%S-t%T.cfile"

Capture-6 : With osmocom appropriately configured, press the REC button, wait for your transmission (if you have access to the remote, push the button you trying to capture now). After the transmission, press the REC button again to stop recording and save the transmission. In our case, the file will be saved in the /tmp/ directory as shown in the screenshot above.

Capture-7 : Repeat steps Capture-5 and Capture-6 as appropriate. For example, in this scenario we need to capture at least the ‘arm’ and ‘disarm’ button presses for both remotes.

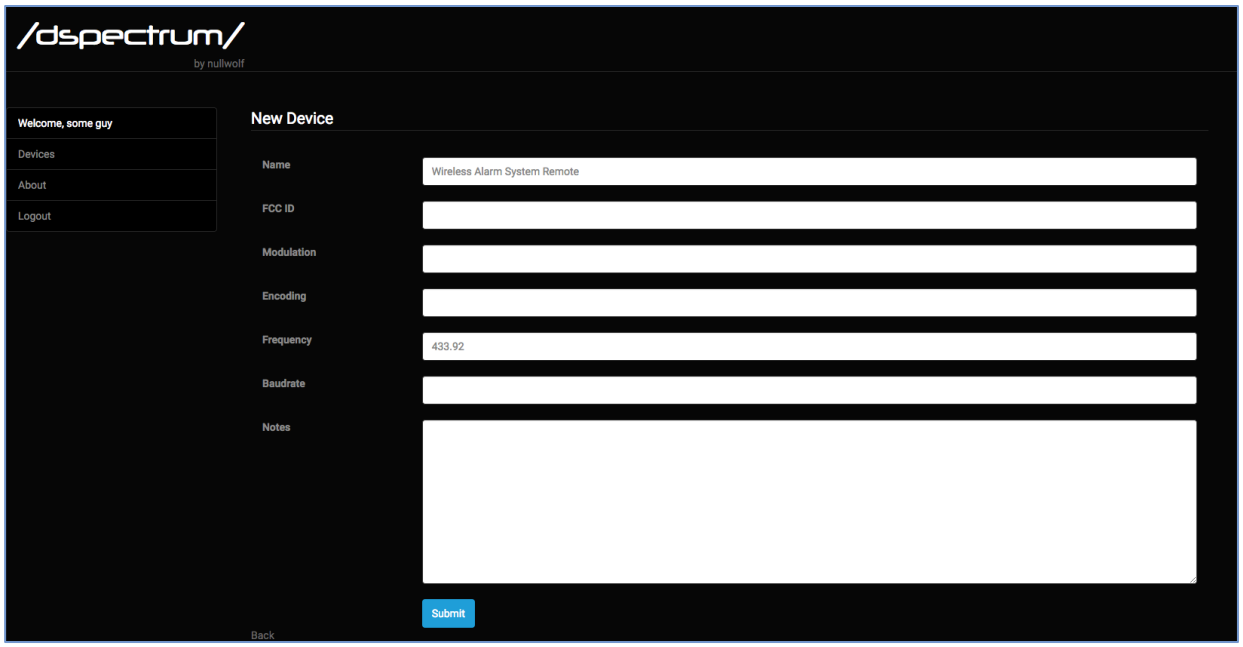
Preparation / Setup

Prep-1 : Navigate to the DSpectrumGUI web application (e.g. http://localhost:3001).

Prep-2 : Authenticate with the default credentials (username: user@example.com, password: password).

Prep-3 : Click “New Device” and fill out the form with the information you have on hand about the device you are reversing. You should have at least the Frequency and a Name at this stage. You may also have an FCC ID.

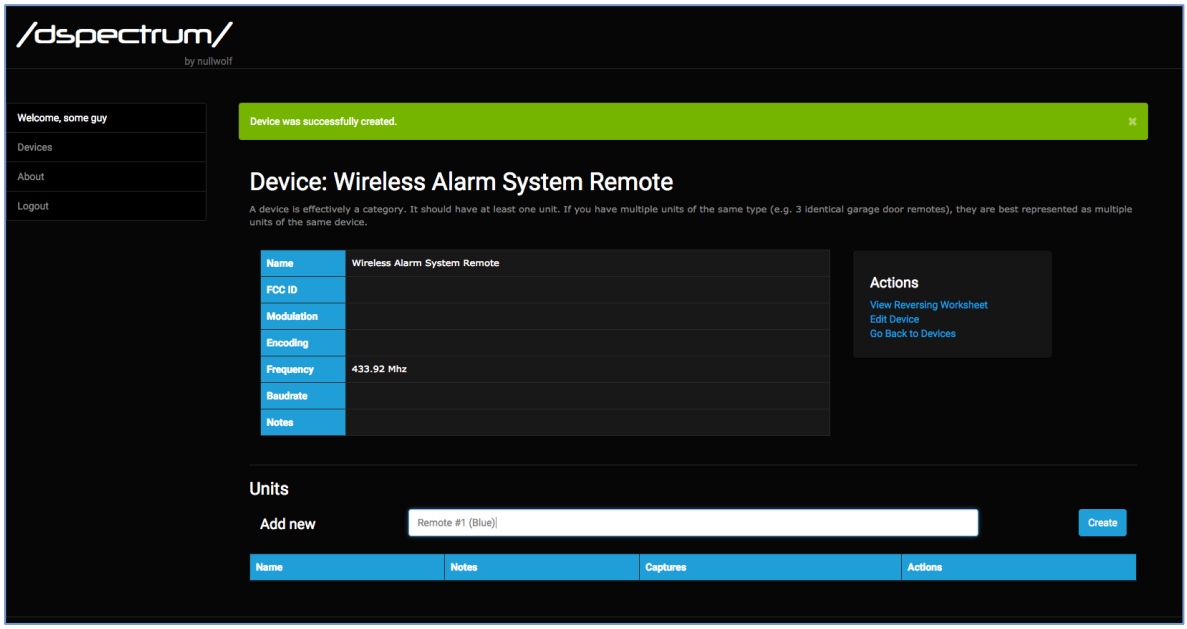
A device should be thought of as a type of device (e.g. Wireless Alarm System).



Prep-4 : Units are individual instances of the device (e.g. 2 remotes came with this alarm system) that you would like to compare the signals of.

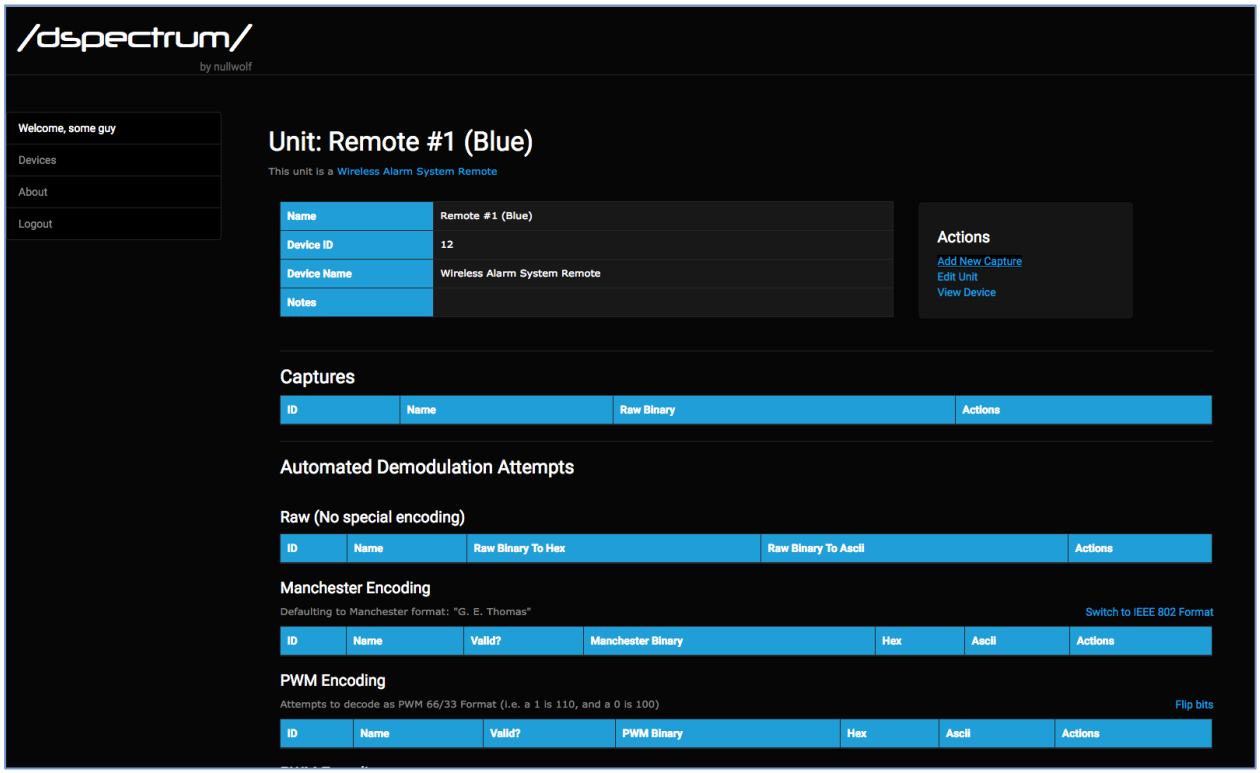
Press the “Add Unit” button.

Prep-5 : Type in a short but descriptive name for your unit. In this example, one of the remotes was blue and the other was silver, so I named the first unit “Remote #1 Blue”, and the other unit “Remote #2 Silver”.

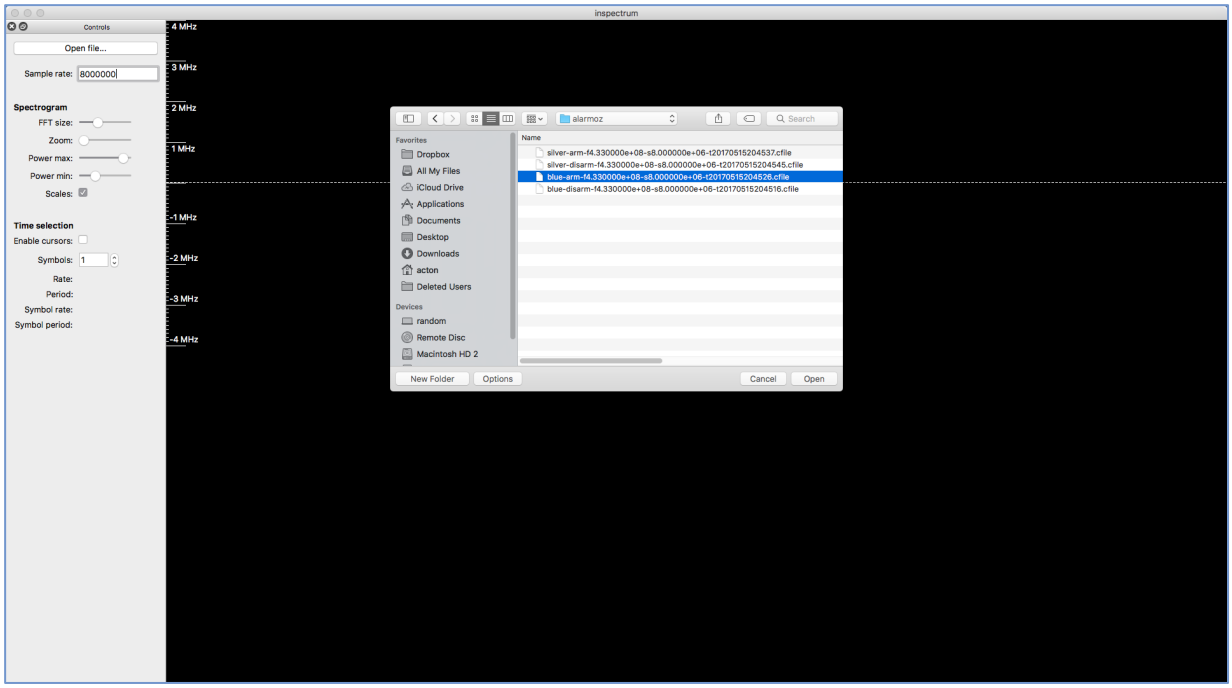


Prep-6 : Click the link to one of the new units you added – it should have appeared just below the “Create” button you pressed.

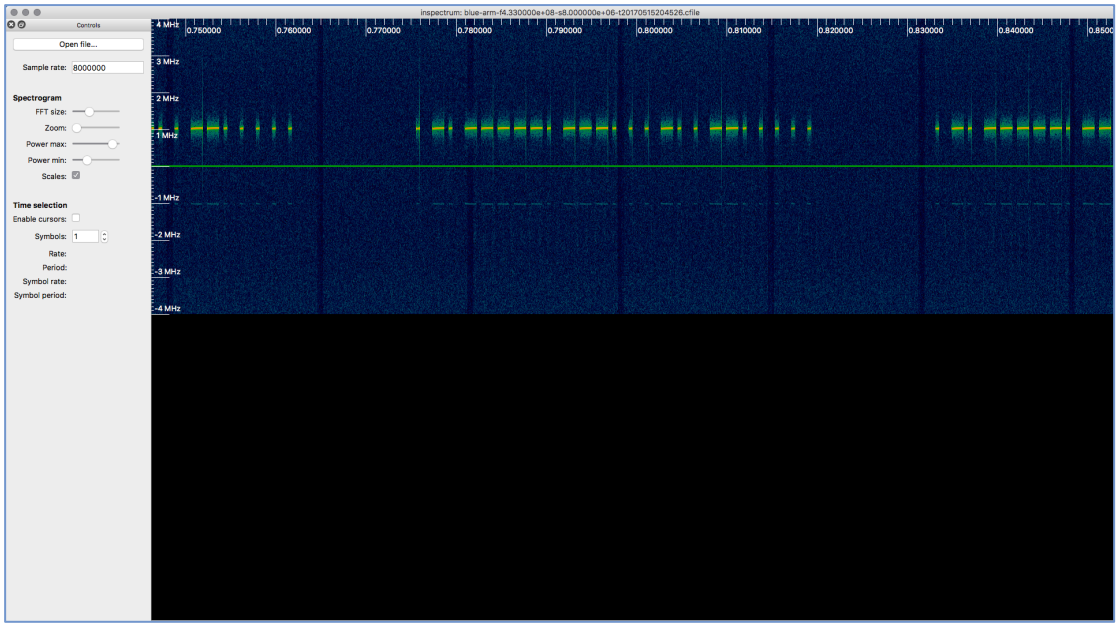
Prep-7 : On the Unit show page, click “Add New Capture” in the “Actions” block on the right.



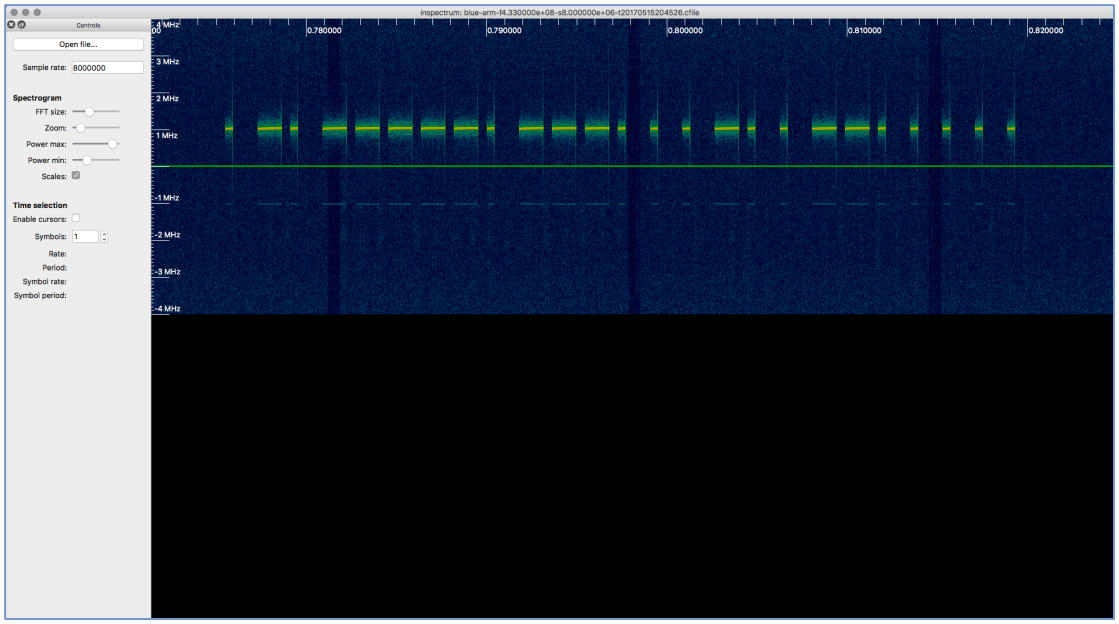
Prep-8 : An instance of Inspectrum will spawn for you. Open your first capture file for this device.



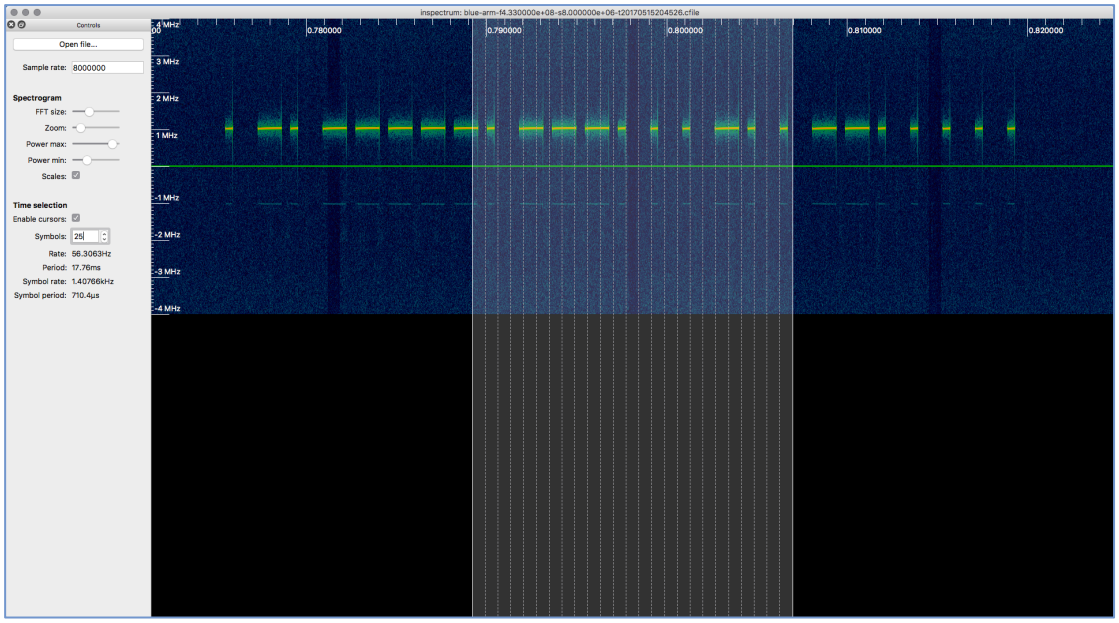
Prep-9 : Scroll to the right until you find your signal. You will likely find that the same 'packet' is repeated numerous times. These should be identical. Center one of packets in the screen.



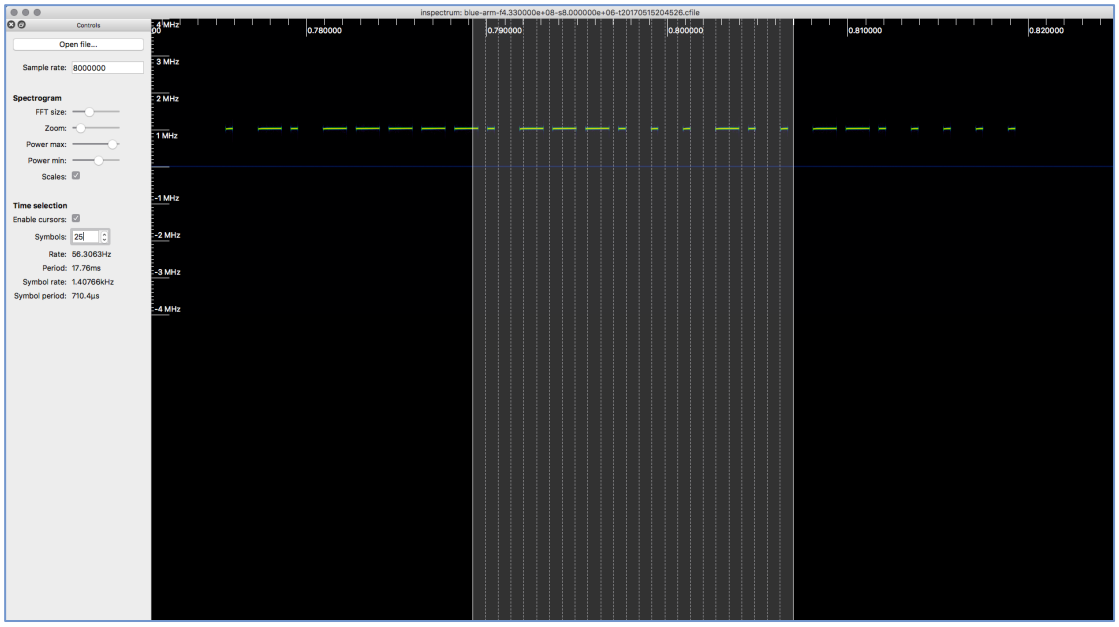
Prep-10 : Drag the "Zoom" slider slowly to increase the size of the packet if necessary.



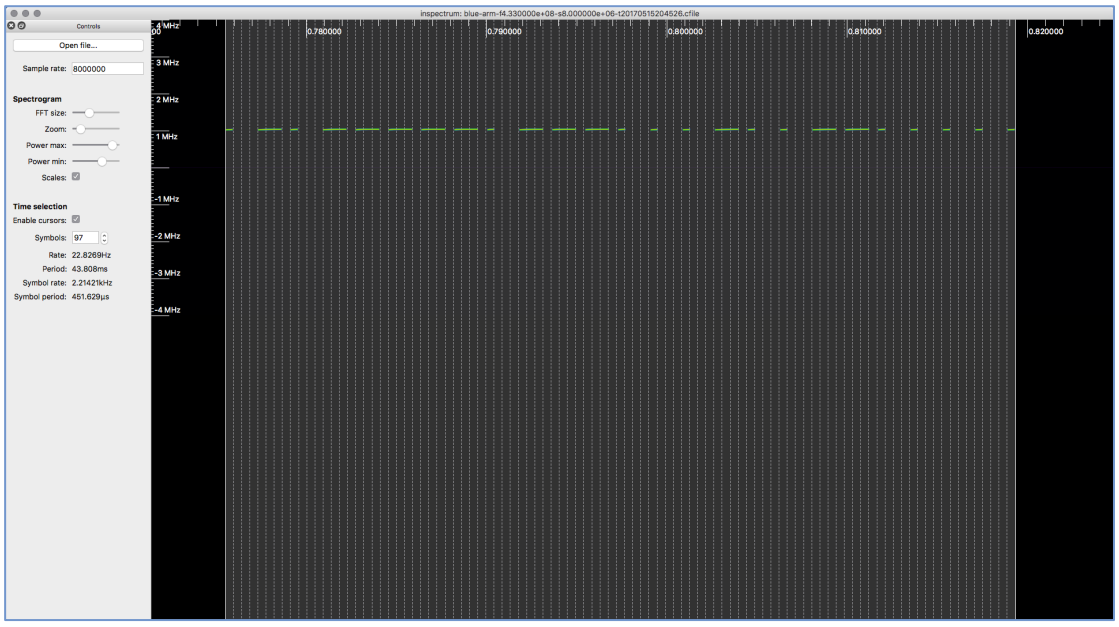
Prep-11 : Type a number into the “Symbols” field on the left. If in doubt, choose a low number like 5 – lower numbers are usually easier to start off with. In the picture below, I started with 25.



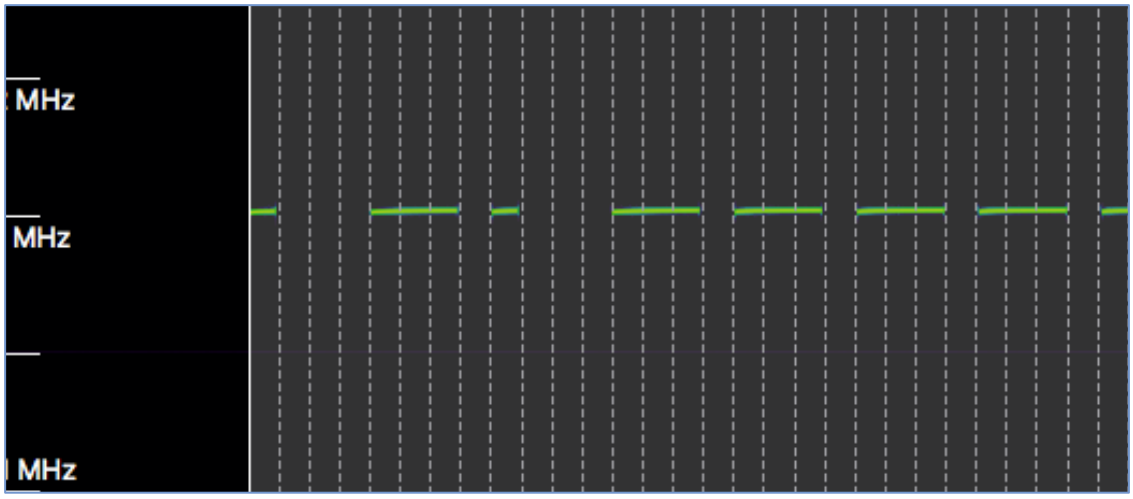
Prep-12 : Drag the “Power min” slider slowly until the background noise fades away and you are left with a crisp representation of the signal you are interested in, as shown in the below picture.



Prep-13 : Line up the start of the grid (drag the unbroken line on the left of the grid overlay) to line up perfectly with the start of the first pulse. Then increase the number of bits (on the left panel) and drag the unbroken white line on the right of the grid overlay, until you have covered the symbols. Line them up so that the shortest possible symbols/spaces are perfectly encapsulated in a single grid. See below.

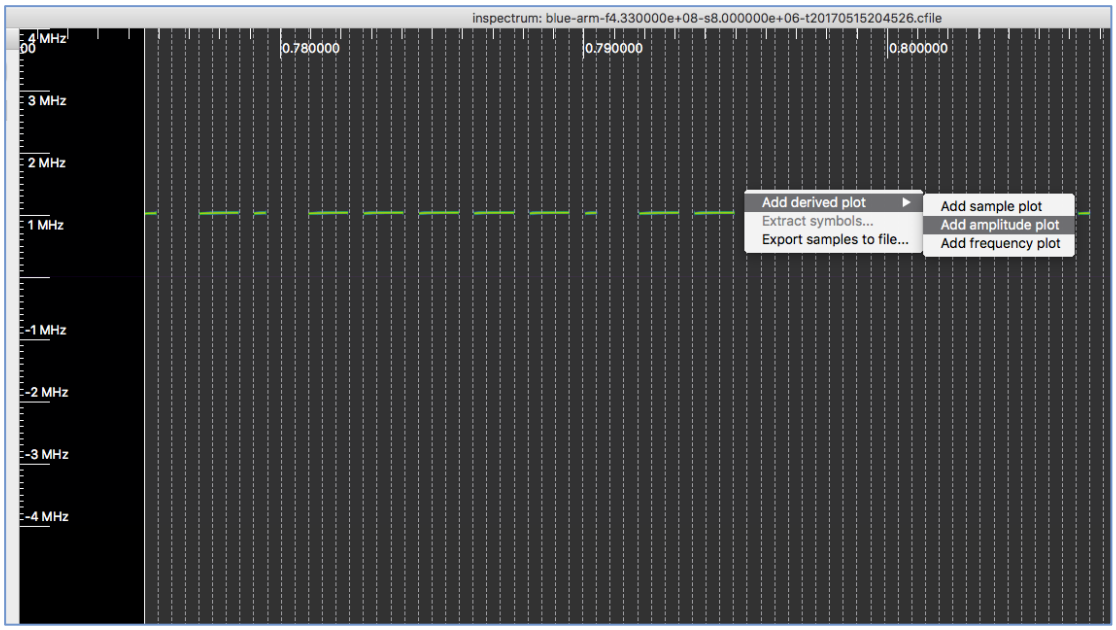


Here is another copy of the above picture, zoomed in so that you can see that the grid overly is aligned as perfectly as possible with the symbols.

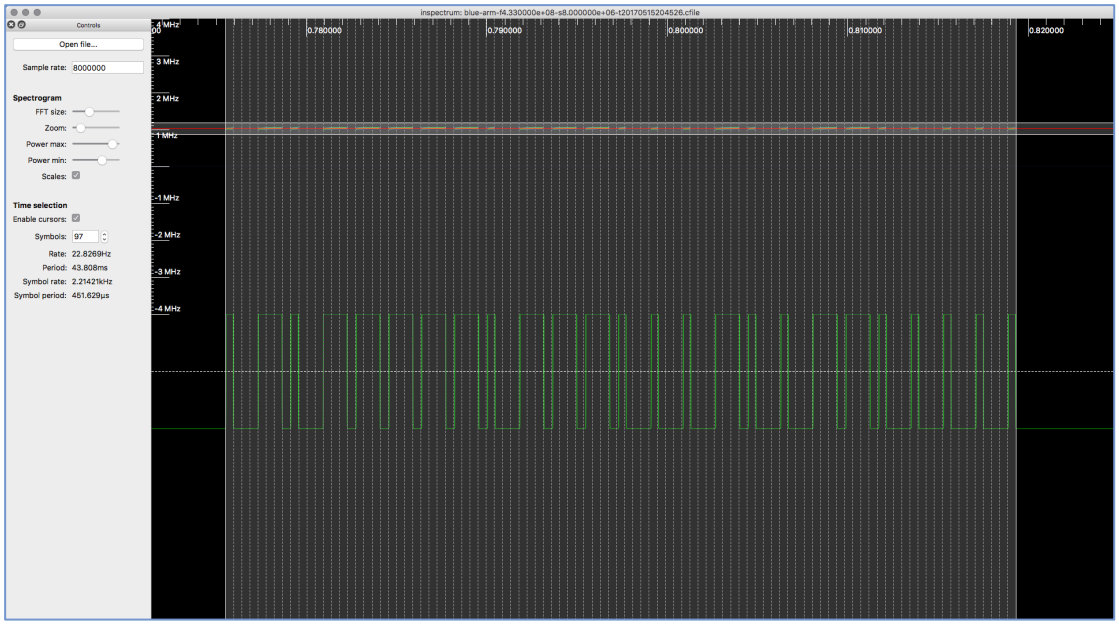


Prep-14 : Once the grid overlay is as perfect as you can make it, make a note of the Symbol rate on the left side panel (e.g. the Symbol Rate here was 2.214).

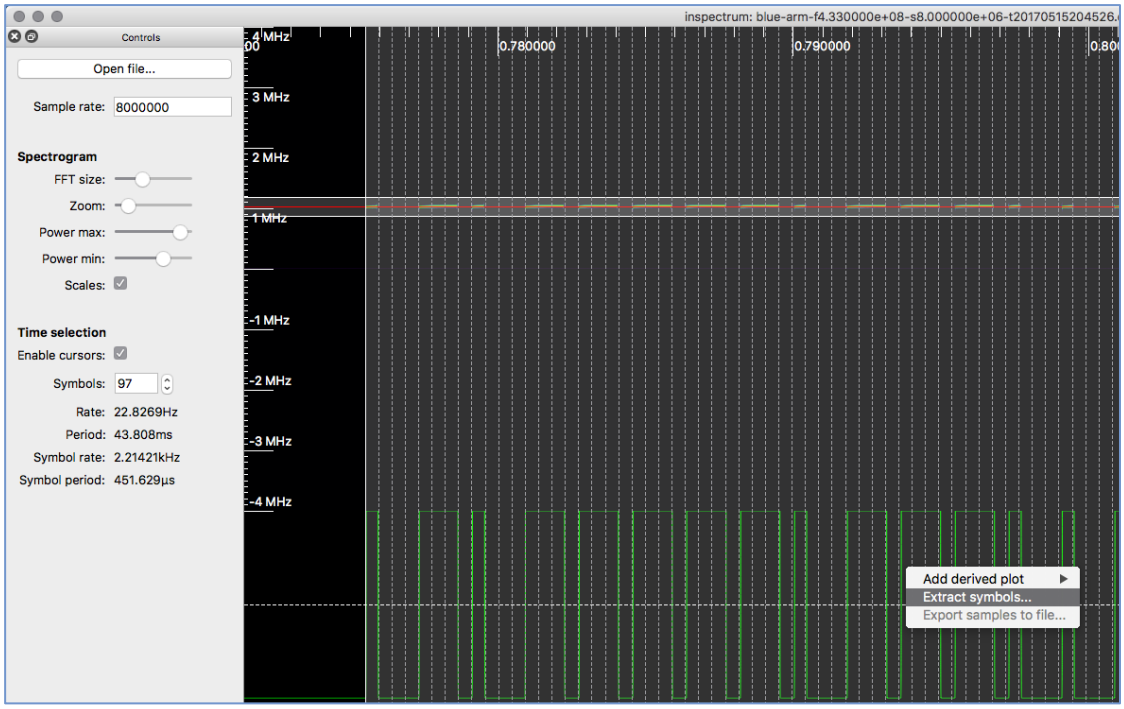
Prep-15 : Right click near the pulses and select “Add derived plot” > “Add amplitude plot”.



Prep-16 : A red line should have appeared towards above the pulses. Drag the red line down so that it’s sitting on top of your pulses.



Prep-17 : A set of green, vertical rectangles that line up with the pulses should have appeared at the bottom. Right click inside this section and then click “Extract symbols…”

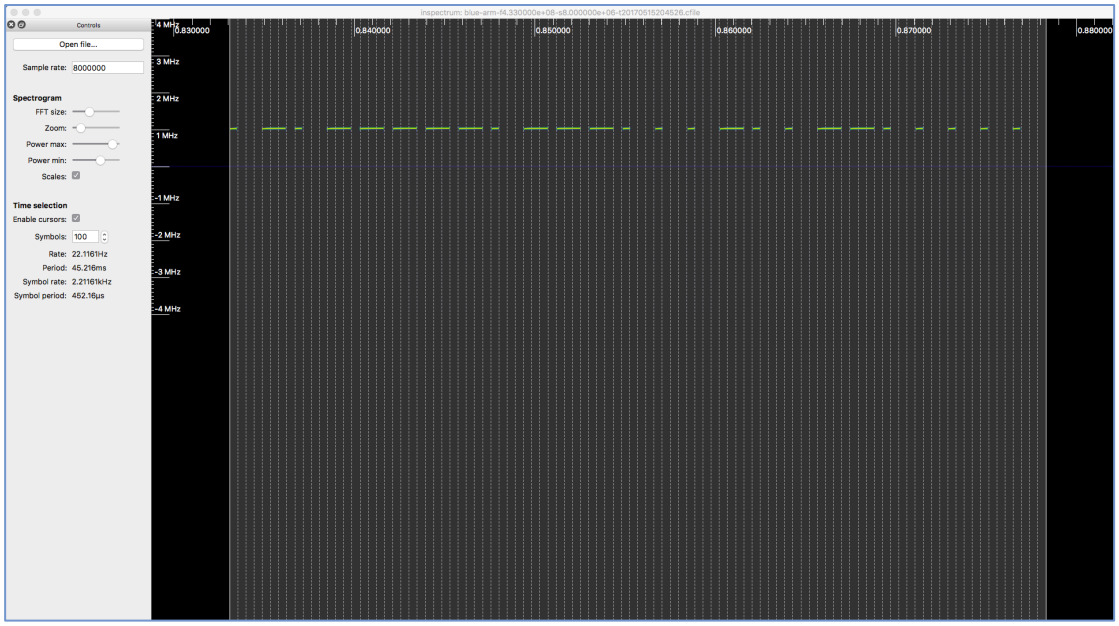


Prep-18 : Inspectrum will close by itself and you will be presented with a form in DSpectrumGUI. Enter a name that describes the context (in this case, the button that was pressed – ‘arm’).

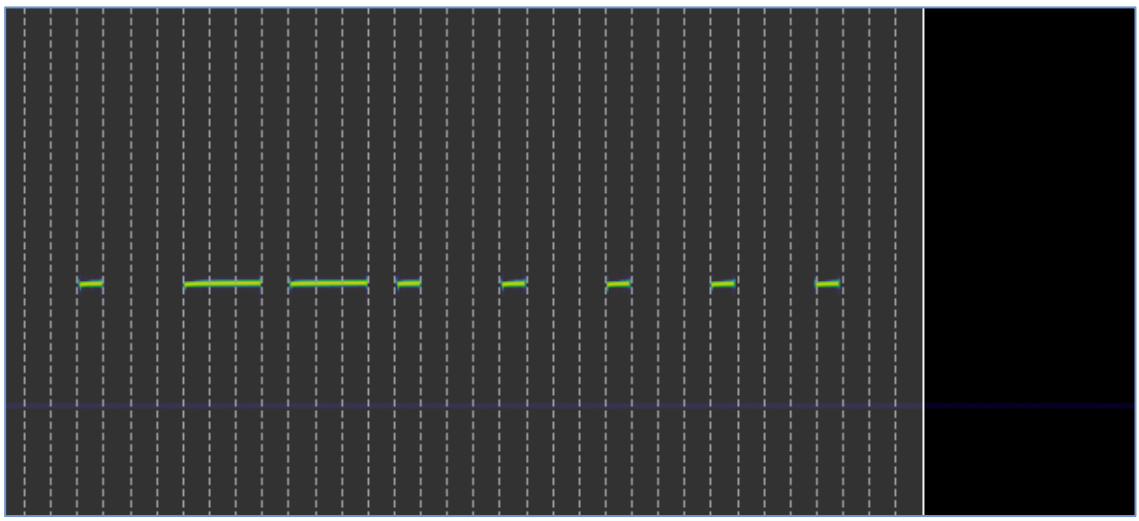
Prep-19 : Scroll down and look at the encoding tables. In our example, DSpectrumGUI flagged that the modulation type is likely to be PWM 77/25, however, it seems we needed to capture some of

the “empty space” at the end in Inspectrum for this to be valid PWM. DSpectrumGUI identified this error for us, and corrected for it by adding the required number of 0s at the end.

A correctly taken capture would have included 3 cells containing whitespace (or 0s) at the end, as demonstrated in the pictures below.

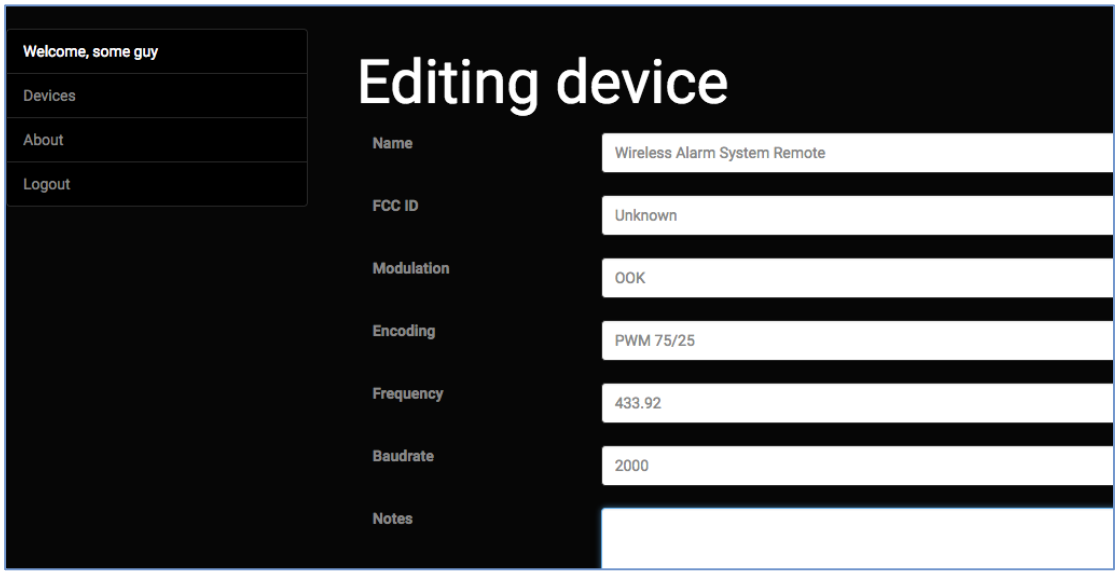


Here is a zoomed in picture showing the last few pulses more clearly.



Prep-20 : DSpectrumGUI told us that the encoding is PWM 75/25, and Inspectrum told us the Symbol Rate is 2.214 once we had the grid perfectly set up. The baudrate is Inspectrum's symbol rate multiplied by 1000. So, the baudrate in this example is 2214.

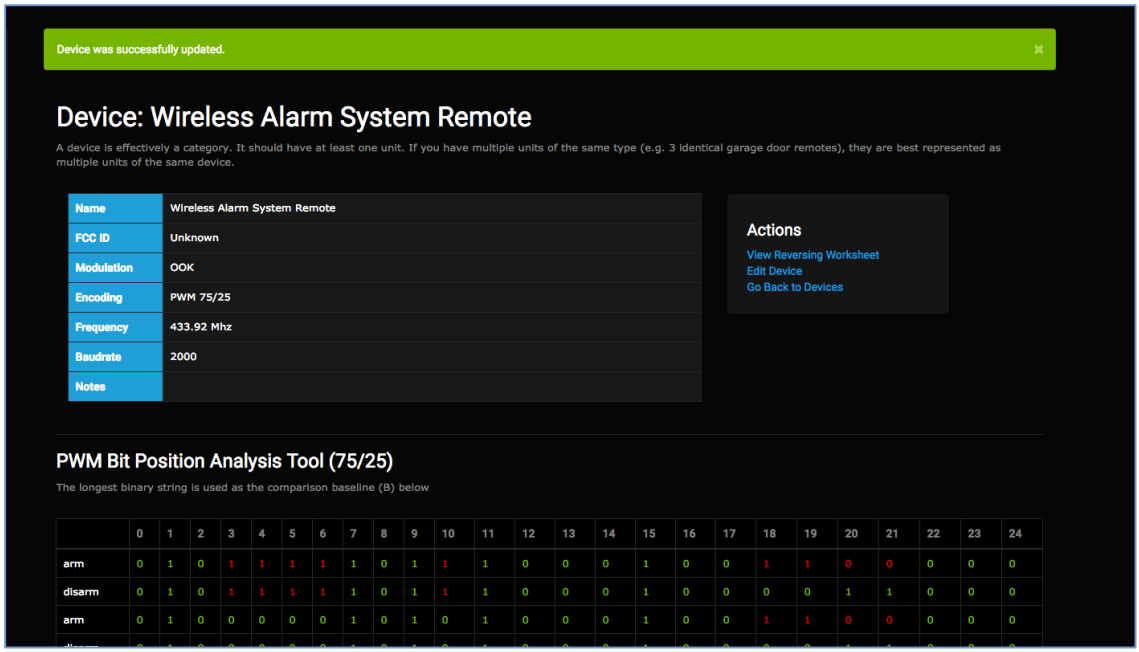
Update the device field in DSpectrumGUI.



Rapid Reverse Engineering

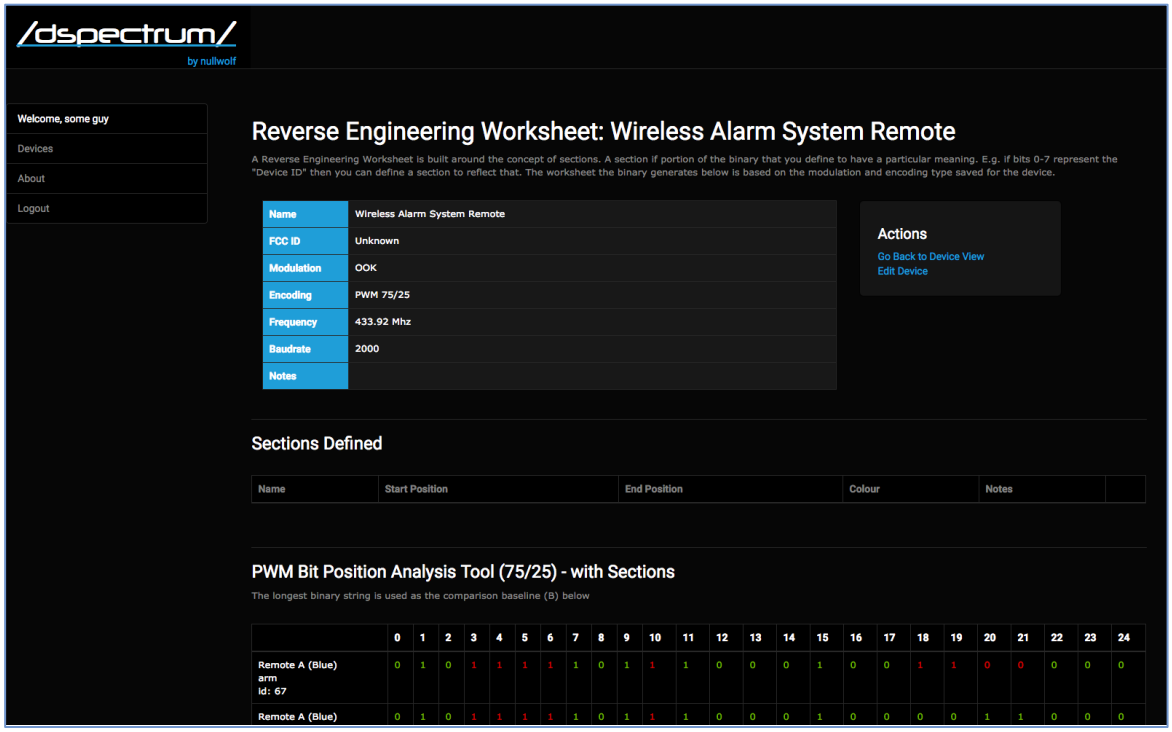
Reversing-1 : Now that we've got all our captures in place, we're ready to begin the fun part – making sense of the data and learning how to create our own valid transmissions!

Click on the "View Reversing Worksheet" link on the right.



The Reverse Engineering Worksheet is similar to the device page, but it only shows us the information we need for manual analysis. It also has some features that allow us to visualize the packet structure and make some notes about our observations.

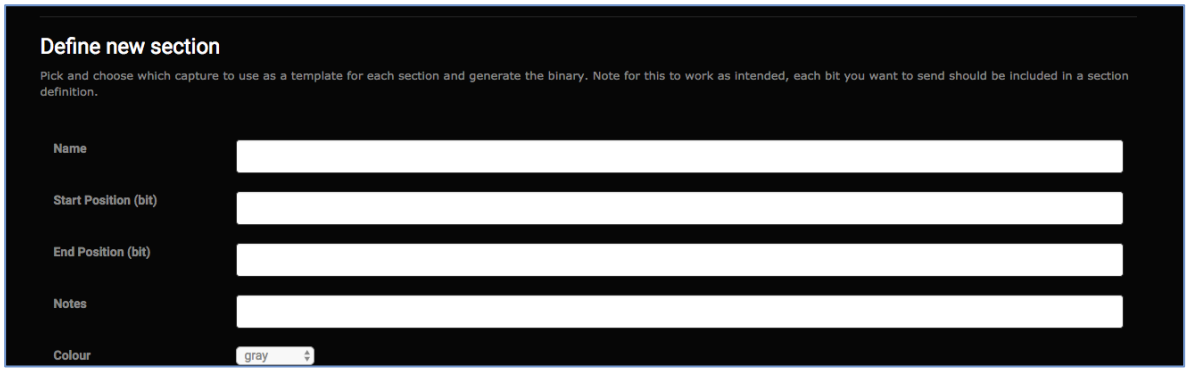
If the device is a reasonably basic one, we should have enough information at the end of this process to learn how to generate our own valid signals to transmit via a tool called RFCat and our Yard-Stick One device.



Reversing-2 : Scroll down and start defining sections. As this is an alarm system and the remote has numerous buttons, we can assume we're expecting to find that each remote transmits its own ID, and also changes the signal to represent which button was pressed.

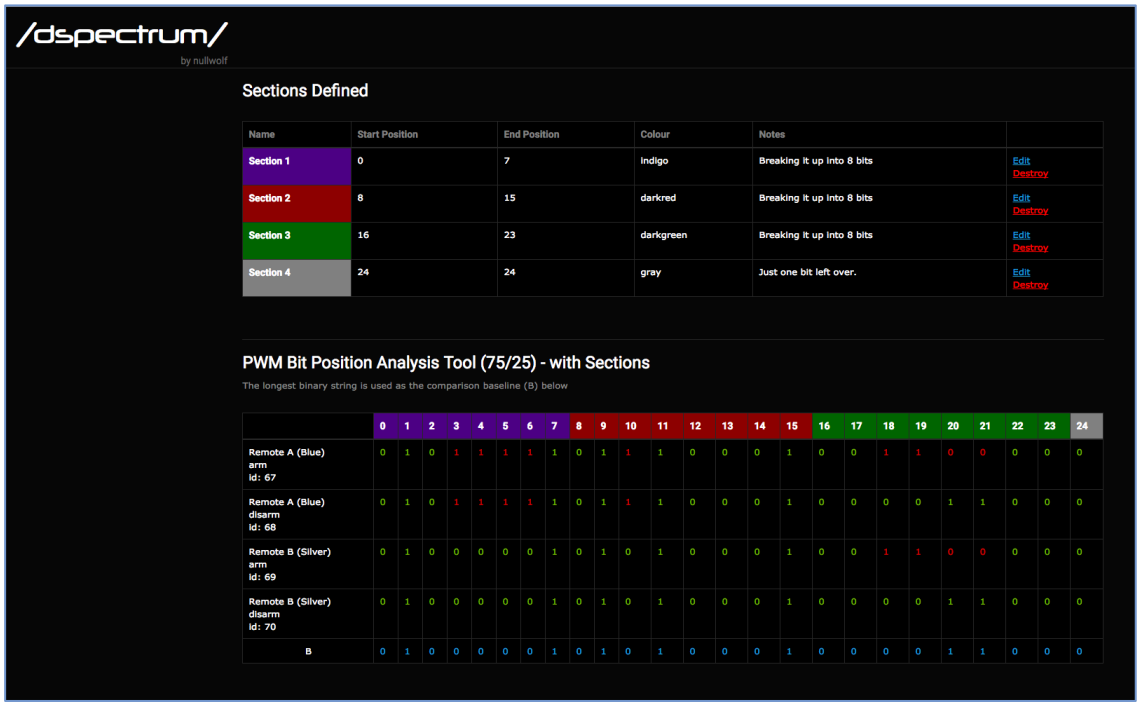
We don't need any of these answers yet though, let's make the table look a little easier to analyse at a glance by defining our sections. When in doubt, it is a good idea to break the packet up into 8bit (1byte) chunks and see if that works out.

In programming languages, counting starts from Zero. In developing this program I have made a decision to do the same. So the first 8 bits starts at bit position '0' and ends at bit position '7'. Fill out the form with this information and pick a colour for each section.



In this example, there was a single bit left over at the end. We define a section for that also, making the start position and the end position have the same value (24).

You should now be left with something that looks a little like this.



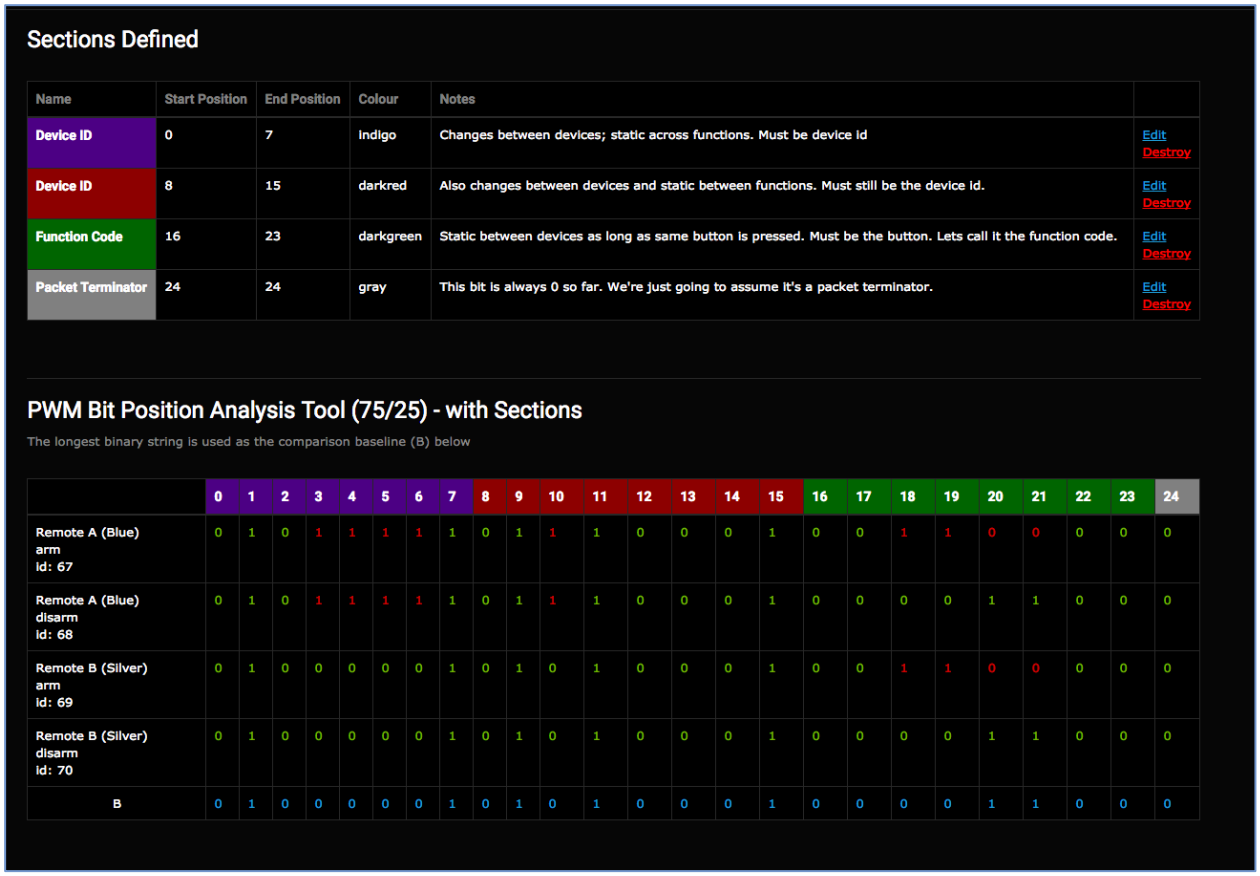
Reversing-3 : We start making observations about the data. As we go, we should click 'edit' for each section and update it with a new name and notes as appropriate.

The first section seems to change when a different remote is used, but it doesn't seem to change if a different button is pressed on the same remote. We can deduce that this is the device ID.

The second section seems to also contain part of the device ID.

The third section seems to change when a different button is pressed (e.g. 'arm') but it doesn't change when a different remote is used with the same kind of button. This must represent the Function ID.

Update each section as appropriate.



Sections Defined

Name	Start Position	End Position	Colour	Notes	
Device ID	0	7	indigo	Changes between devices; static across functions. Must be device id	Edit Destroy
Device ID	8	15	darkred	Also changes between devices and static between functions. Must still be the device id.	Edit Destroy
Function Code	16	23	darkgreen	Static between devices as long as same button is pressed. Must be the button. Lets call it the function code.	Edit Destroy
Packet Terminator	24	24	gray	This bit is always 0 so far. We're just going to assume it's a packet terminator.	Edit Destroy

PWM Bit Position Analysis Tool (75/25) - with Sections

The longest binary string is used as the comparison baseline (B) below

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
Remote A (Blue) arm id: 67	0	1	0	1	1	1	1	1	0	1	1	1	0	0	0	1	0	0	1	1	0	0	0	0	0
Remote A (Blue) disarm id: 68	0	1	0	1	1	1	1	1	0	1	1	1	0	0	0	1	0	0	0	0	1	1	0	0	0
Remote B (Silver) arm id: 69	0	1	0	0	0	0	0	1	0	1	0	1	0	0	0	1	0	0	1	1	0	0	0	0	0
Remote B (Silver) disarm id: 70	0	1	0	0	0	0	0	1	0	1	0	1	0	0	0	1	0	0	0	0	1	1	0	0	0
B	0	1	0	0	0	0	0	1	0	1	0	1	0	0	0	1	0	0	0	0	1	1	0	0	0

Reversing-4 : We have another look at the above, and try to look out for any other useful information. We realize there's little point to separating the first two sections, so we delete one of them. We now edit the remaining Device ID section so that it covers the whole 16bit (2byte) range.

Sections Defined

Name	Start Position	End Position	Colour	Notes	
Device ID	0	15	indigo	Changes between devices; static across functions. Must be device id	Edit Destroy
Function Code	16	23	darkgreen	Static between devices as long as same button is pressed. Must be the button. Lets call it the function code.	Edit Destroy
Packet Terminator	24	24	gray	This bit is always 0 so far. We're just going to assume it's a packet terminator.	Edit Destroy

PWM Bit Position Analysis Tool (75/25) - with Sections

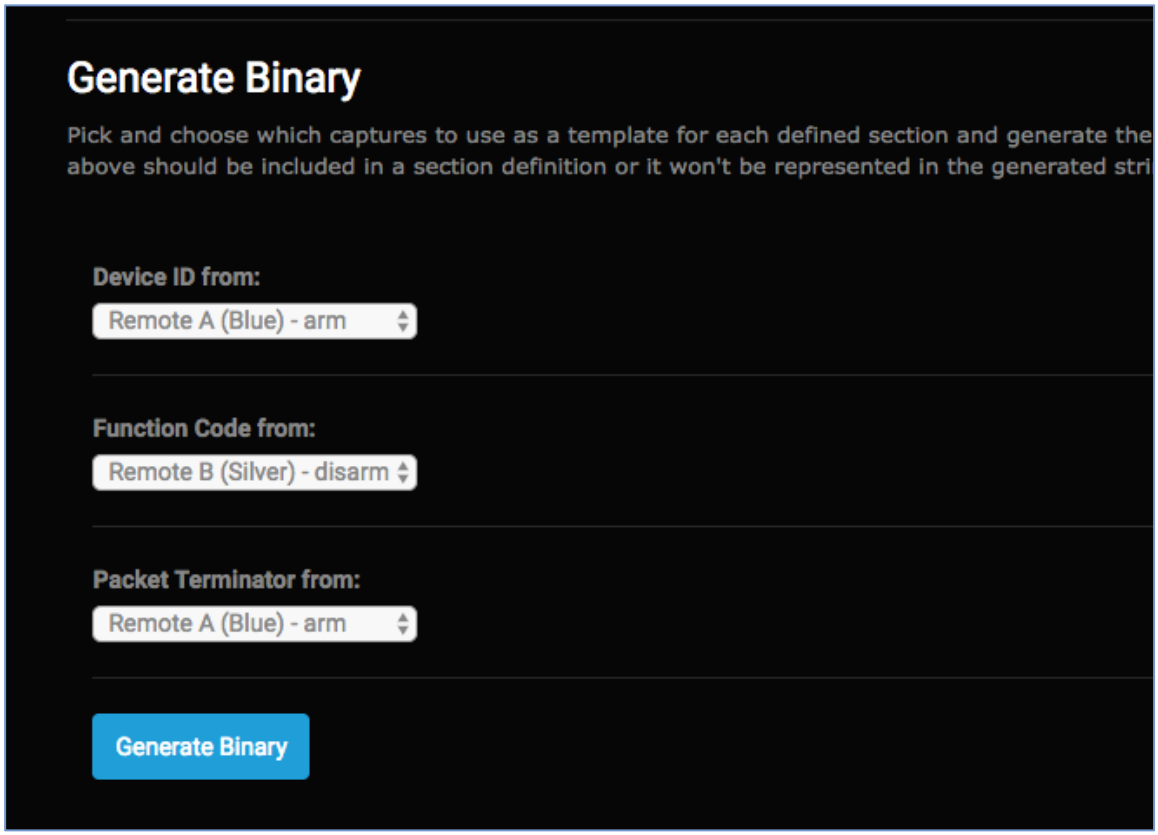
The longest binary string is used as the comparison baseline (B) below

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
Remote A (Blue) arm id: 67	0	1	0	1	1	1	1	1	0	1	1	1	0	0	0	1	0	0	1	1	0	0	0	0	0
Remote A (Blue) disarm id: 68	0	1	0	1	1	1	1	1	0	1	1	1	0	0	0	1	0	0	0	0	1	1	0	0	0
Remote B (Silver) arm id: 69	0	1	0	0	0	0	0	1	0	1	0	1	0	0	0	1	0	0	1	1	0	0	0	0	0
Remote B (Silver) disarm id: 70	0	1	0	0	0	0	0	1	0	1	0	1	0	0	0	1	0	0	0	0	1	1	0	0	0
B	0	1	0	0	0	0	0	1	0	1	0	1	0	0	0	1	0	0	0	0	1	1	0	0	0

Transmitting the signal

Xmit-1 : Scroll down until you find the “Generate Binary” form. This will allow us to generate or own binary by taking “sections” out of specific captures.

This way, if we had a “disarm” message for Remote B, but only an “arm” message for Remote A, and needed to transmit a “disarm” signal impersonating Remote A, we could just cherry pick the components that we need to form our transmission.



Xmit-2 : Select Remote A’s Device ID, and the Function Code from Remote B, and press “Generate Binary”.

We now have the binary, and even the code, that we need to type into RFCat in order to disarm Remote B’s alarm system.

Xmit-3 : Connect the Yard Stick One to your computer.

Xmit-4 : Open a new terminal window and type:

rfcat -r

This should open up an interactive command-line application. DSpectrum has already provided you with the code to use here. You can simply copy and paste it and see if it works.

Go back to DSpectrumGUI's "Generate Binary" result. That page should display the RFCat code required towards the bottom of the page.

Xmit-5 : Copy and paste the code (the lines should start with 'd'), and paste them into RFCat.

Xmit-6 : Hopefully, when you hit enter on the final line (d.RFxmit…), the alarm system was disarmed.

The code we copied and pasted looks similar to the example below.

```
d.setMdmModulation(MOD_ASK_OOK)
d.setFreq(433920000)
d.setMdmSyncMode(0)
d.setMdmDRate(2214)
d.RFxmit('PACKET_AS_HEXCODE_HERE\x00\x00\x00\x00'*5)
```

Here is a very brief explanation of what the above code means.

The string “PACKET_AS_HEXCODE_HERE” is replaced by the hex code you’d like to transmit. The “\x00” is a null byte which effectively represents a gap in our transmission. 4 null bytes is usually a sufficient gap to serve as a packet separator. We are repeating our packet 5 times, because we noted that the packets were repeated by the remote. This happens to increase the likelihood that the transmission will be received by the target device, even if there is momentary interference at that frequency.

The MdmDRate is the baud rate we noted earlier. Inspectrum’s Symbol Rate is represented differently than the baud rate as it is expressed in Inspectrum. Shifting the decimal place 3 places to the right of the Symbol Rate value will give us the correct baud rate to use in RFCat.

Setting MdmSyncMode ensures that RFCat doesn’t prefix a sync-word to the data we’d like to send out. Sync-words are also commonly referred to as a “preamble”. This is used by some devices to let the receiver know to expect a packet. Our example device doesn’t have a preamble / sync-word, so we set this value to ‘0’ to prevent RFCat from sending one, which it does by default.