



**SA:**

a Spectrum Analyzer/Sniffer for PicOS  
Networks Based on  
CC1100 RF Modules

version 0.01

(lame and tentative, but presentable)



June 2012

## 1 Introduction

SA is a simple sniffer/RF-scanner based on standard CC1100 hardware, i.e., the underlying praxis runs on a standard node equipped with a CC1100 module, e.g., the old Warsaw box or the new CC430-based node. Needless to say, SA is quite far from an industrial grade spectrum analyzer, but it does offer a few features that may prove useful in our development work. Its OSS program, having been written in Tcl/Tk, is open for extensions that will materialize quickly, as soon as we find them useful. As of now, the program offers:

1. Flexible hardware settings of the CC1100 options (registers), which can be easily edited and saved in a database for future use. A handy collection of hover tips makes it easy to modify the register contents without having to consult the documentation.
2. Two graphical scan modes: a) *band scan*, whereby a range of frequencies is periodically swept for RSSI readings, and b) *single frequency scan*, whereby a single frequency is monitored for RSSI and (optionally) for packets.
3. A packet reception option available as a special mode of the single frequency scan, where the program can receive packets and report them in a special window.
4. A packet injection option (also available as part of the single frequency scan), where you can predefine collections of packets to be injected into the network manually or automatically at prescribed intervals. Selected fields of those packets can be incremented/decremented with each injection. Similar to register configurations, editable packet layouts can be stored in a database for future reference.

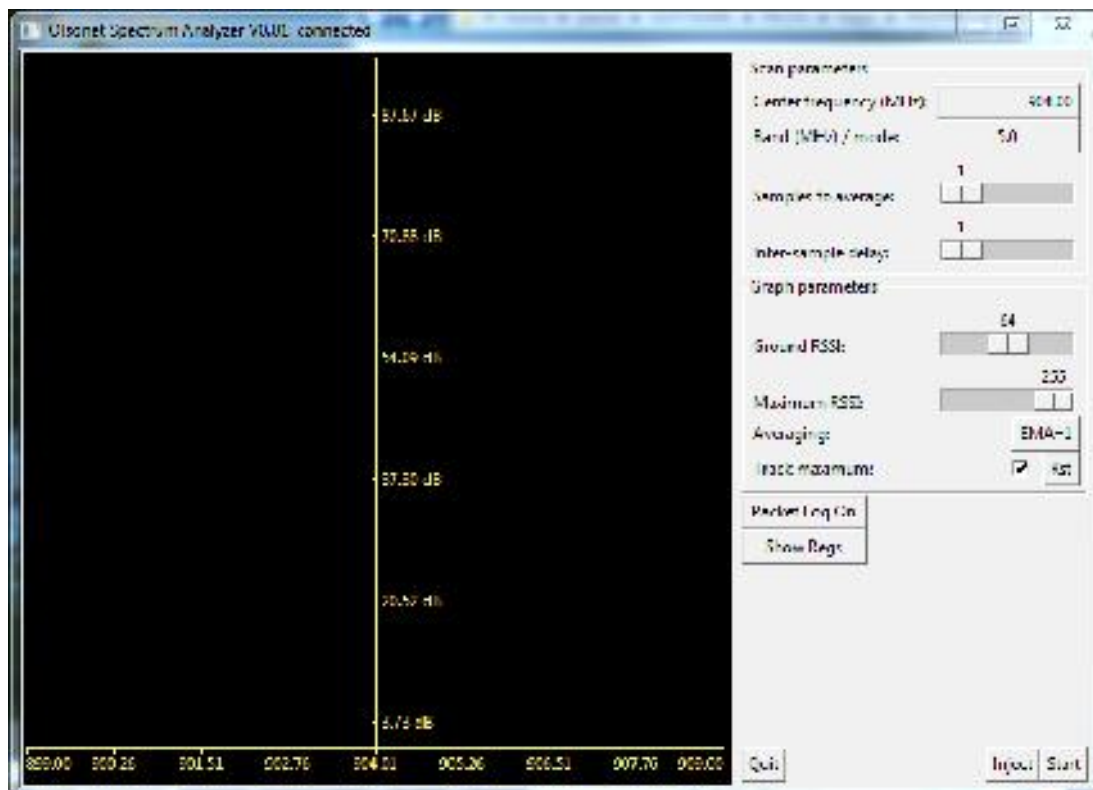


Figure 1: The main window as it shows up immediately after startup.



## 2 Prerequisites

You need a Warsaw node. Compile the SA praxis and load it into the node. Connect the node to your PC via the standard FTDI (UART-USB) dongle.

The OSS program for the praxis is a Tcl/Tk script. The simplest way to make it easily and independently executable is to package it with *freewrap*. (see the *mkexe.sh* script in the praxis directory). The script needs Tcl/Tk 8.5, but it doesn't depend on any exotic features (that wouldn't be found in *freewrap*). Needless to say, you can also execute the script directly using your existing installation of Tcl/Tk (again, it must be at least 8.5).

Run the script (or the pre-wrapped executable). After a (hopefully not too long) while it should *automatically* locate the node and connect to it. There is no connect button, there is no need to look up COM ports, or anything like that! The connection status is indicated in the main window's title. If it says “connected”, it means that the script is talking to the node. The main window (in its virgin startup shape) is shown in Figure 1.

When the program starts for the first time (and it has no *rc* file) its parameters take some default values (some of them are shown in Figure 1 as preselected options). In particular, it uses the default configuration of registers (named *Built In*), which you can see when you click on the “Show Regs” button. Their setting closely matches our default setting, which means that you are immediately ready to sniff within our (typical) networks.

The program stores its *rc* database file (named *.sapicrc*) in the user's home directory (whatever that means). Under UNIX, and also under Cygwin, when SA is run using Cygwin's “native” Tcl/Tk, the situation is clear. When run from the wrapper, or in any other circumstance when the Tcl/Tk does not know about Cygwin, the home directory is typically *C:/Users/username*.

## 3 A band scan

By default, the program is set to the *band scan* mode. The center frequency for the scan is set to 904 MHz, and the band is 5 MHz on each side. When you click the “Band/mode” menu (see Figure 2), you will see a list of options. The top two entries select the other (*single-frequency scan*) mode. Don't choose them for now.

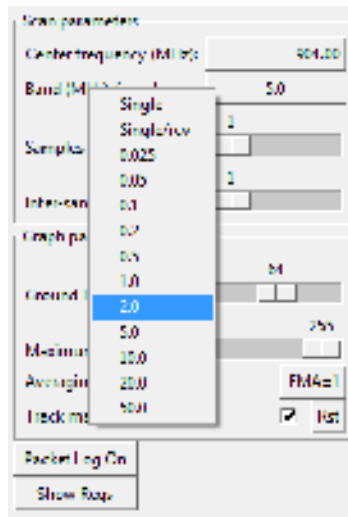


Figure 2: Mode selection.

With the default settings (without touching any controls), click “Start” (in the right bottom corner). You will see some activity in the window (see Figure 3). The dancing green dots



represent the current (momentary) readings of RSSI at the given frequencies; the red curve shows the maximum readings (obtained so far) at the respective locations.

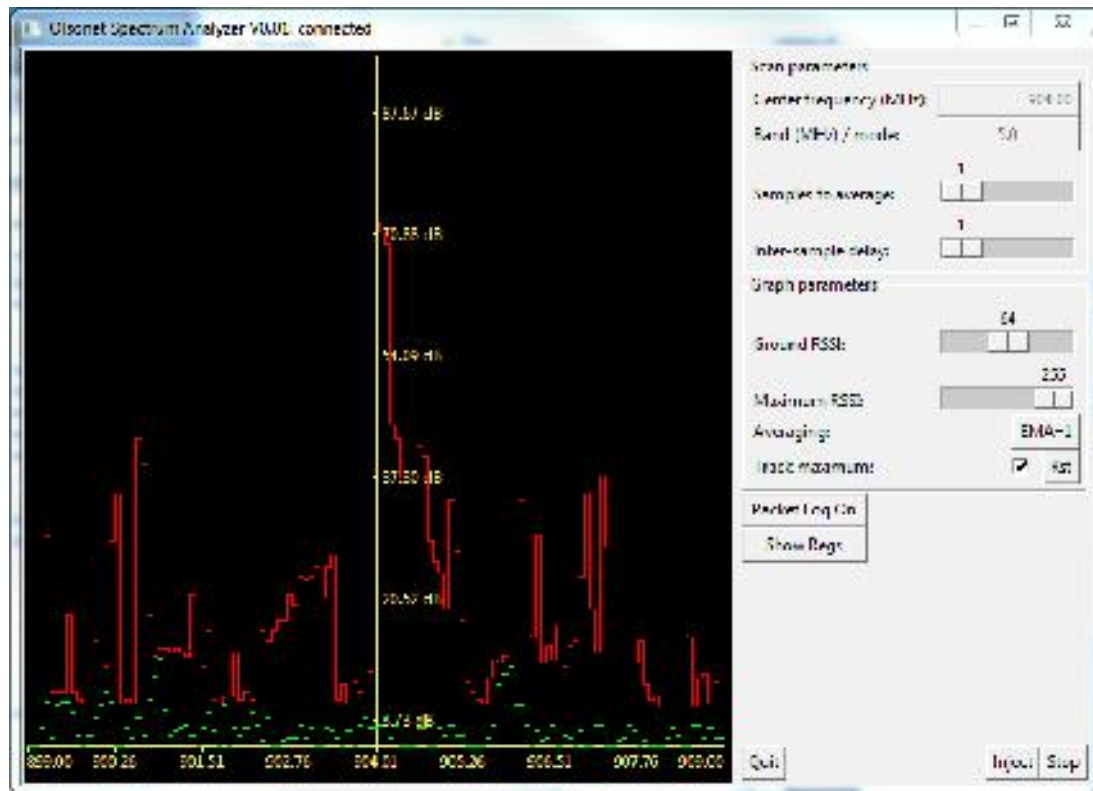


Figure 3: A band scan in progress.

During a scan, some controls (the ones under “Scan parameters”) stay frozen, but you can play with “Graph parameters”. In particular, you can reset the red (maximum) curve by hitting the “Rst” button (at “Track maximum”), so the maximum will be tracked from scratch. You can also switch it off completely by unchecking the box located to the left of the button. Not everything necessarily makes a lot of sense. This is all tentative, and things may change.

Within “Graph parameters”, the sliders labeled “Ground RSSI” and “Maximum RSSI” allow you to redefine the vertical scale, i.e., the bounds of the RSSI range. The “Averaging” button opens yet another slider that you can use to dampen the changes of momentary RSSI samples (the dancing green dots). The default (EMA=1) means no damping, i.e., the green dots in fact represent individual RSSI samples arriving from the node. With EMA=4, for example, every dot is an exponential moving average of the last 4 readings. As you increase EMA, you will see that the dots calm down. At the highest setting of 64 they hardly move at all (unless your environment exhibits long-duration RF activities at the scanned frequencies).

The topmost two menus from “Scan parameters” are not difficult to comprehend. The two sliders, however, do require a bit of explanation. “Samples to average” defines the number of samples (RSSI readings) that will be averaged at the node before being sent to the script as one sample. For example, if you set it to 4, then the node will read 4 times the RSSI at every given frequency and will send you the average of those readings as a single sample. This is slightly different from the other kind of averaging described above (which is done within the script).

To appreciate the difference, you have to realize that the node scans the prescribed frequency in sweeps. The samples averaged by the script (EMA) come from different



sweeps, so they are separated by relatively long intervals (depending on the inter-sample delay, as explained below). On the other hand, the averaging done by the node refers to temporary close samples, ones having a much higher likelihood of belonging to the same activity.

The “Inter-sample delay” is the delay (in PicOS milliseconds) following every frequency change and preceding the first RSSI sample read for that frequency. It should be at least 1: the chip does require some break following a frequency change, but the zero setting is also available for experiments.

Note that the inter-sample delay does not apply to multiple (averaged) samples collected at the same frequency, which are collected very fast (in a CPU loop). Note: the way it is done at present probably makes little sense, because the RSSI update rate is well below the reading frequency.

The full sweep consists of 129 samples (this is the actual resolution of the x-axis). Thus, at the minimum sensible inter-sample delay, one sweep takes about 0.25s. Again, I may look into speeding it up in my spare time (or, possibly, taking advantage of some features of the radio chip to improve the resolution/scanning frequency). Note that lots of experiments are required to arrive at the optimum parameters of a frequency scan. That involves setting some tricky parameters, like channel bandwidth, gain control, and so on. Feel free to do it on your own (by fooling around with the settings of various registers).

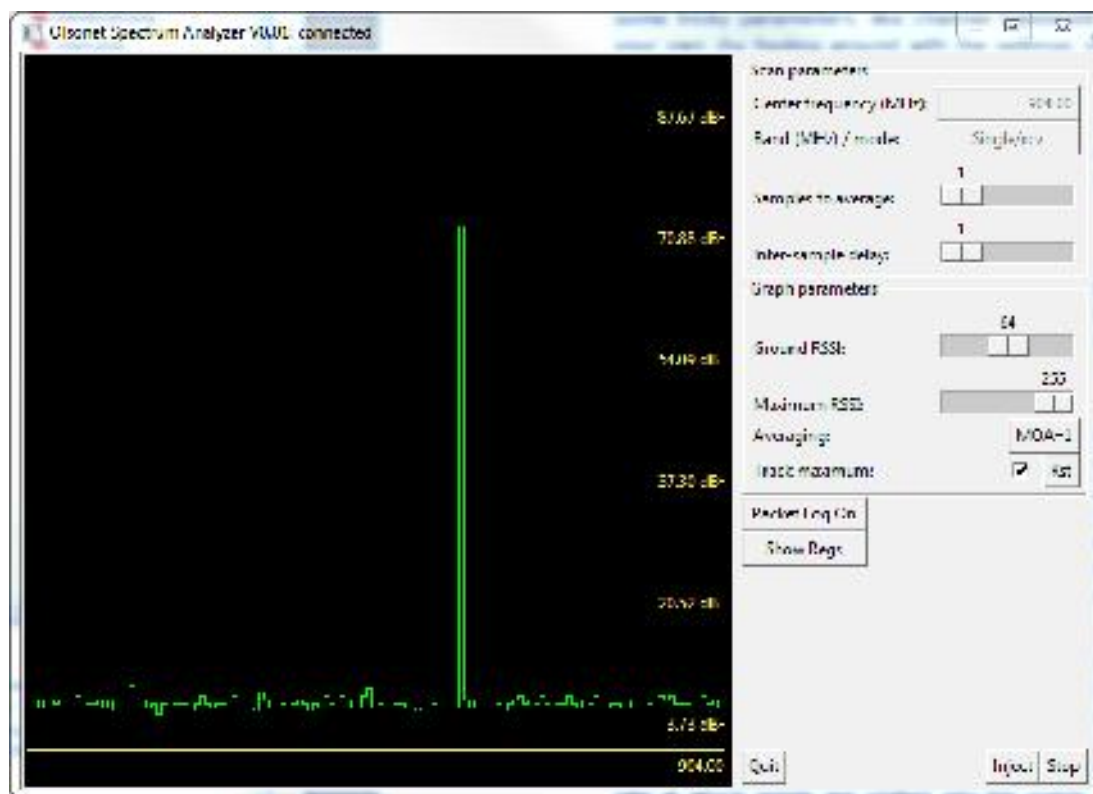


Figure 4: A single-frequency scan.

#### 4 A single-frequency scan

Click “Stop”. The green dots will stop their dance and freeze. After a short while (some skidding is expected), the LEDs on the node will go pure green (which, BTW, means that the



node happens to be idle). Click the band selection menu and choose “Single/Rcv”. The graph layout will change a bit. Then click “Start” again.

You will see a shifting sequence of RSSI samples for the selected “Center frequency” with the maximum (so far) now represented by a straight horizontal red line (see Figure 4). While the “Ground RSSI” and “Maximum RSSI” settings retain their meaning (the vertical scale is the same as for the band scan), “Averaging” now refers to something slightly different (note that the button's title has changed from “EMA” to “MOA”).

By default, with the MOA (Maximum Of Averages) setting of 1, each RSSI point shifted into the graph is the maximum of 128 consecutive samples collected at the specified “Inter-sample delay”. The 128 samples constitute a package of data arriving from the node. With the MOA setting of 4, for example, each point refers to the maximum of averages of 4-tuples of samples, i.e., 4 consecutive samples are averaged, and the maximum of those averages within the 128-sample package is presented as one point. In the extreme case of MOA=128, all 128 samples arriving in the package are averaged into a single (the degenerate maximum of 1) point.

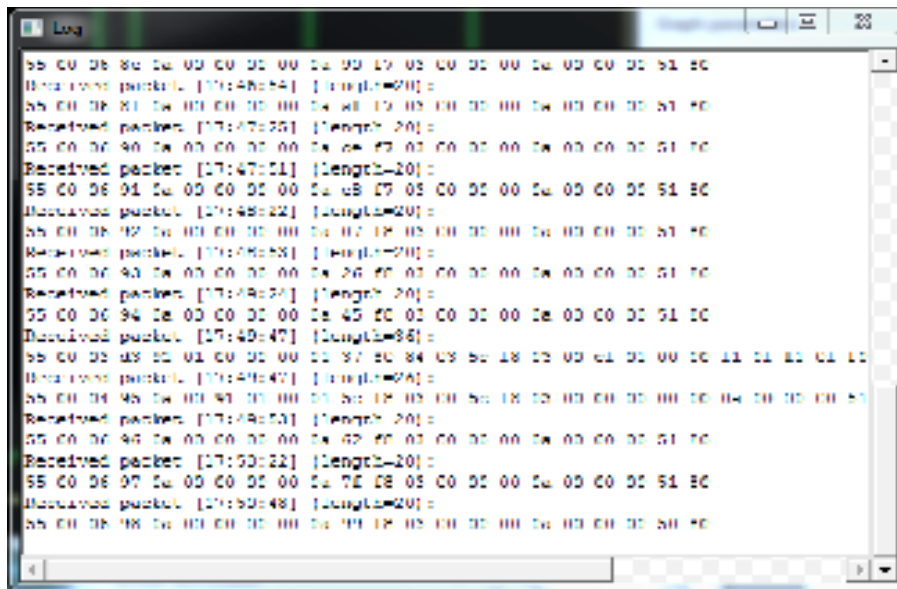


Figure 5: The packet log window.

“Inter-sample delay” from “Scan parameters” applies to the spacing between consecutive samples, but if “Samples to average” is more than 1, then all readings of one averaged sample are collected with no spacing. In contrast to band scanning, it makes sense to set “Inter-sample delay” to 0, in which case the readings will be collected at the full rate at which the CPU can read RSSI from the module's register. Note that still 128 samples (each possibly representing multiple averaged readings) contribute to one package, which translates into a point shifted into the graph. By increasing the inter-sample delay, you can slow down the rate at which points are shifted in. For example, at the spacing of 8 milliseconds, the shift-in rate is about 1 second per point, and you will still won't miss any activities resulting from sane packets sent at 10,000 bps (our standard rate), especially with MOA set to 1. With that setting, the 128 points of the full horizontal span of the graph represent approximately the last 2 minutes of the network's history.

## 5 Packet log

Note that with the “Band/mode” selection of “Single/rcv”, the program will be receiving packets (that can be physically received with the given configuration settings of the RF





module). Such packets are shown in the “Packet Log” window (see Figure 5). Note that the log is maintained (for a considerable number of packets into the past) even when the window is closed.

## 6 Packet injection

When you click the “Inject” button, you will an editor of sorts allowing you to construct the layout of a packet to be injected into the network. While you can build such layouts at any time (and possibly store them in the program's database), actual injection is only possible during a single-frequency scan (regardless of its parameters). Multiple injection windows can be present at any given time. You can use them to inject multiple (different) packets, possibly in an automated repeat fashion.

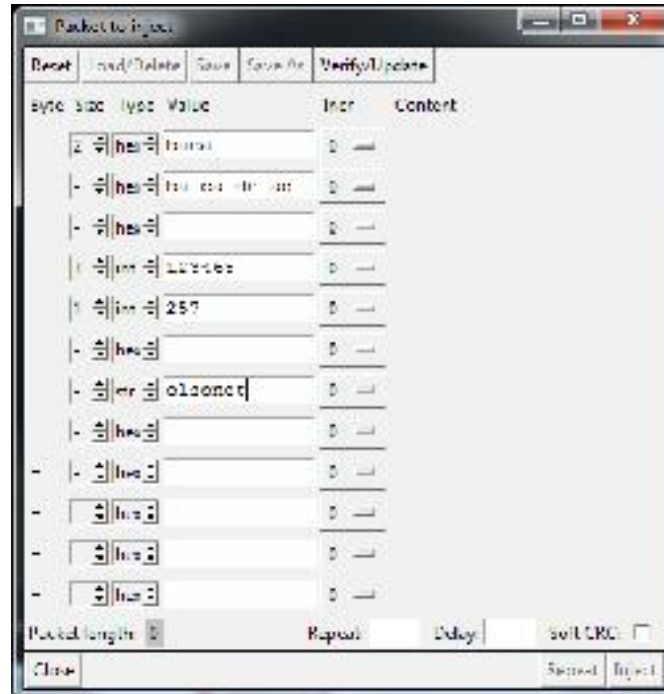


Figure 6: The injection window.

A packet is described in terms of fields. Up to 12 such fields can be specified in the window (see Figure 6). The length of a field (size), understood as the length of the packet portion represented by the field, is expressed in bytes. If the length is left unspecified and the value is non-empty, the program will make a guess (which can be corrected manually later).

All filled-in fields contribute to the packet in order from top to bottom. For example, the layout in Figure 6 consists of 5 fields. Note that you can leave holes (unspecified size and no value); they are ignored and take no space in the packet. If the value is nonempty, but the size is not specified, the field's size will be deduced from the type and the value. If the size is specified, but the value is empty, the value will be set to zero (or to an empty string, if the field's type happens to be “str”).

A multi-byte hexadecimal value written as a single number (the first field in Figure 6) is interpreted as little-endian, i.e., its least significant byte will go first. If a hexadecimal value is written as pairs of (hex) digits separated by spaces (e.g., the second field), then each pair stands for one byte, and the bytes go exactly in the written order (from left to right). To see the actual content of the packet, press “Verify/Update”. This will cause the fields to be



parsed. The unspecified sizes will be calculated and the actual byte content of the packet will be generated and shown (up to 8 bytes of every field), as illustrated in Figure 7.

The screenshot shows a window titled "Packet Inject" with a table of fields. The table has columns: "Byte", "Size", "Type", "Value", "Test", and "Content". The fields are as follows:

Byte	Size	Type	Value	Test	Content
1	2	hex	00 00	0	00 00
2	4	hex	0a 0b 0c 0d	0	0a 0b 0c 0d
-	-	hex		0	
6	3	int	123456	0	00 02 01
4	1	int	257	0	01
-	-	hex		0	
10	8	str	olsonet	0	00 00 00 00 00 00 00 00
-	-	hex		0	
-	-	hex		0	
-	-	hex		0	
-	-	hex		0	
-	-	hex		0	

At the bottom of the window, there are controls for "Packet length: 18", "Repeat", "Delay", "Soft CRC" (checkbox), and buttons for "Close", "Report", and "Inject".

Figure 7: The effect of Verify/Update.

Note that the string "olsonet" takes 8 bytes (and includes the zero sentinel). The total length of the packet amounts to 18 bytes.

A manually specified size overrides any ideas the program might have about the field's size based on the type and value. For example, the value 257 in the fourth field has been squeezed into a single byte (truncated) yielding 01. For strings (as well as multiple hexadecimal values, like the one in field 2), when the explicitly specified size is shorter than the value, the trailing bytes of the value (which notion is always clear) are ignored. If the size is larger than required, the trailing bytes of the field will be filled with zeros. Simple numbers are always interpreted as little endian, so basically the same principle applies to them as well, except that for a negative integer ("int" type) the trailing bytes of an oversized field are filled with the sign extension.

The role the "Incr" selectors is to declare some fields as being automatically incremented or decremented on every injected packet (as a way of differentiating those packets). For example, when you select +1 in the "Incr" widget at the fourth field, the second packet injected from this window will have 02 in its 9'th byte, the third 03 and so on. Only simple numerical values can be incremented/decremented; strings, as well as, multi-part hexadecimal values, cannot.

Whenever you change anything in the configuration of fields, you have to press "Verify/Update" for the change to be reflected in the actual shape of the packet. Any problems with the interpretation of fields will be then diagnosed and reported. This also applies to the two input widgets at the bottom labeled "Repeat" and "Delay", as well as the "Soft CRC" checkbox. When that box is checked, the program will append software CRC (ISO 3309) at the end of the packet before submitting it for transmission. This is only possible if the packet consists of an even number of bytes.





With “Repeat” and “Delay” you can set up a looped transmission of the packet “Repeat” times at “Delay” intervals (expressed in milliseconds). Both those fields must be filled with reasonable integer numbers (and verified-updated) before the “Repeat” button will activate. Generally, the “Inject” button activates when the layout has been verified as correct (there is a packet ready to be injected) and the program is currently executing a single-frequency scan. For the “Repeat” button, the extra condition includes sane contents of the “Repeat” and “Delay” fields.

By hitting “Inject” you inject a single copy of the packet. If the packet log window is open (Figure 5), you will see the packet in the window. For example, for a packet generated according to the layout in Figure 7, you will see a line shown at the bottom of Figure 8.

```
Received packet [20:45:16] (Length:20):
55 00 18 11 0a 00 00 00 00 0a 71 71 04 00 00 00 0a 00 00 00 20 20 20
Received packet [20:45:53] (Length:20):
55 00 18 11 0a 00 00 00 00 0a 47 71 04 00 00 00 0a 00 00 00 20 20 20
Injecting packet [20:46:12] (Length:10):
0a 0a 0a 0a 0a 0a 40 40 01 01 60 60 75 60 6a 65 74 00
```

Figure 8: An injection report.

To see the effect of decrementing/incrementing, modify the layout in the way reflected in Figure 9. Note that the “Repeat” button is now enabled. When you hit it, you will see in the log window ten packets showing up at 2 second intervals (Figure 10). Note how the modified fields differ. Also note that after every injection the inject window reflects the updated values of those fields (see Figure 11).

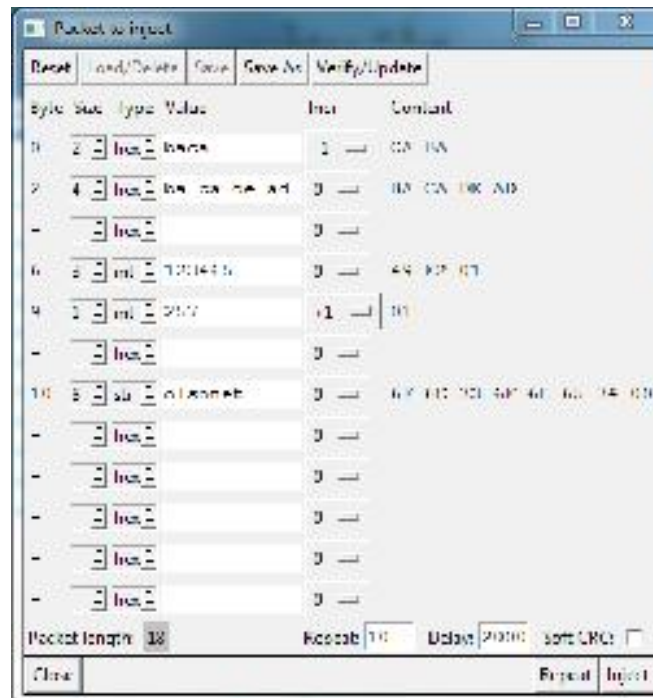


Figure 9: An illustration of incrementing/decrementing.



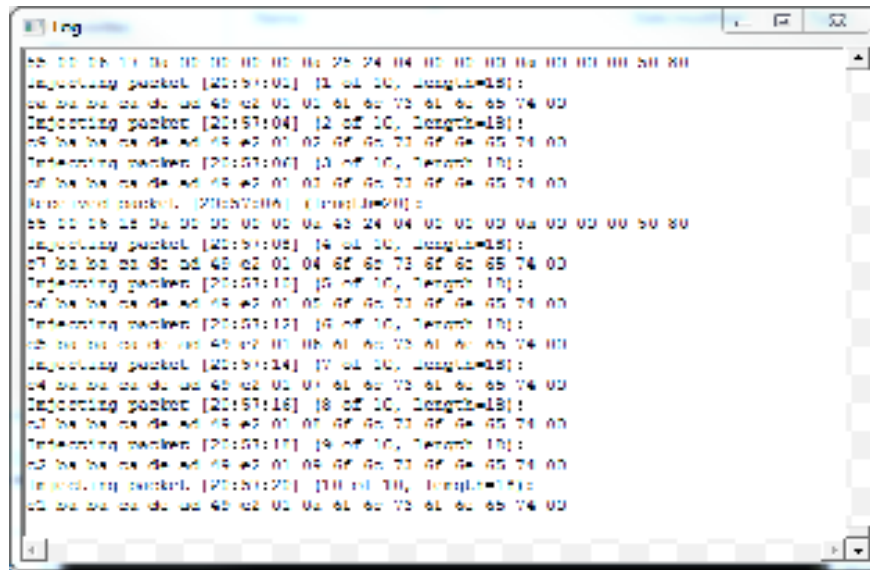


Figure 10: A report from repeat injection.

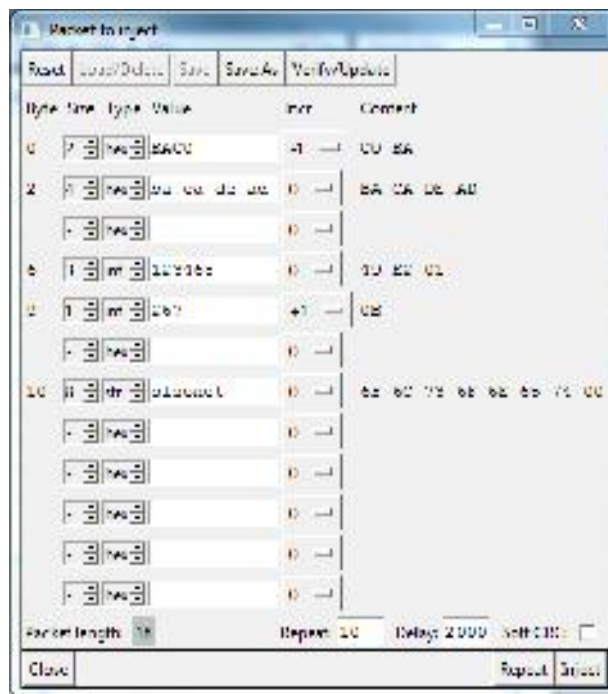


Figure 11: After the repeat injection.

A packet layout can be saved and assigned a name under which it can be later retrieved. The usage of buttons “Load/Delete”, “Save”, “Save As”, as well as “Reset” is obvious.



## 7 Registers

When you hit the “Show Regs” button in the main window, you will see the “current” configuration of registers (see Figure 12). If you haven't defined any “personal” register sets, there is a default set, named “Built In”, which is selected automatically. If you define and save a set named “Default”, that set will be loaded upon startup replacing the “Built In” set. In any case, the “Built In” (immutable) set can always be referenced and loaded explicitly.



Figure 12: The registers window.

The current register set, be it “Built In”, “Default”, or any other set currently in effect, is written to the device before the commencement of any scan operation. It is obvious that by modifying the register contents you can mess up the node's RF module; however, there is always an easy way out: load the “Built In” set and write it to the node. You don't even have to hit the “Write” button in the registers window, because, as we said, the current set is always written to the node before a scan. In fact, to make sure that the node is in an absolutely sane state, it is reset before the registers are written, so it is pointless to try to modify them otherwise before a scan. Consequently, the “Write” button (and maybe “Read” as well) is probably useless (I may eliminate it in a future version).

Individual registers can be read and written during a scan by clicking the “r”/“w” buttons (they are not really buttons but special labels). Note that some registers are read-only. You will see them at the bottom of the pane, if you scroll it all the way down.

Note that those registers that are logically partitioned into fields have their fields listed and their contents presented in binary (in addition to the hexadecimal content of the complete register). When you hover the mouse over a register/field name, you will see a functional description of the respective item. It may be cryptic in some cases; the manual is cryptic in spots, too, and I do not understand everything.

To modify a register, you can point the mouse to the hexadecimal value (of the full register) or to the binary value (of a field) and click the digit you want to change. A left click on a hex digit will advance it forward, while a right click will push it backward. For a binary digit, either click will flip the bit. Note that such a change is not immediately sent to the node: for that you have to click the small “w” button on the right (or wait until the next scan). You can also edit the register values in order to save the set (under some name) for future reference.

