HackRF

Great Scott Gadgets

Jun 03, 2026

USER DOCUMENTATION

CHAPTER

ONE

GETTING HELP

Before asking for help with HackRF, check to see if your question is answered in this documentation, listed in the *Troubleshooting* page, or addressed in the HackRF GitHub repository issues.

For assistance with HackRF general use or development, please look at the issues on the GitHub project. This is the preferred place to ask questions so that others may locate the answer to your question in the future.

We invite you to join our community discussions on Discord. Note that while technical support requests are welcome here, we do not have support staff on duty at all times. Be sure to also submit an issue on GitHub if you've found a bug or if you want to ensure that your request will be tracked and not overlooked.

If you wish to see past discussions and questions about HackRF, you may also view the mailing list archives.

CHAPTER

TWO

TROUBLESHOOTING

2.1 HackRF not detected / "No HackRF boards found."

If the software you're using is unable to detect the HackRF hardware and/or *hackrf_info* returns "No HackRF boards found.", this can be caused by a number of different software or hardware issues.

2.1.1 Solution

1. If you are using a PortaPack addon, make sure to select "HackRF" mode from the main menu.
2. If you are using a virtual machine or Windows Subsystem for Linux (WSL), make sure that it is configured to pass through the USB device.
3. Check whether the device appears in `lsusb` (Linux), Device Manager (Windows), or System Report (macOS). If it doesn't not appear, it could either be a firmware issue, an issue with the cable, or another hardware issue.
4. Try booting the HackRF in DFU mode by holding the "DFU" button when plugging in the device. It should now appear as *NXP Semiconductors LPC4330FET180 [ARM Cortex M4 + M0] (device firmware upgrade mode)* in the locations listed above. If it does appear, then it was likely a firmware issue and you can follow the instructions to *recover the SPI flash firmware*.
5. If the device does not appear in DFU mode then it is likely to be an issue with the USB cable. Charge-only cables (which do not include the data lines) have become very common and will cause this symptom. Ideally, test the cable you're using with another device that does some sort of data transfer to be sure that it works, or just try other cables.
6. If the device still does not appear, it may be a less common issue or possibly a fault with the hardware. See *Getting Help* for information on where to ask for more support, and please include as much detail as you can about what you've already tried.

2.2 There is a big spike in the center of the received spectrum

If you see a large spike in the center of your FFT display regardless of the frequenecy you are tuned to, you are seeing a DC offset (or component or bias). The term "DC" comes from "Direct Current" in electronics. It is the unchanging aspect of a signal as opposed to the "alternating" part of the signal (AC) that changes over time.

Take, for example, the signal represented by the digital sequence:

```
-2, -1, 1, 6, 8, 9, 8, 6, 1, -1, -2, -1, 1, 6, 8, 9, 8, 6, 1, -1, -2, -1, 1, 6, 8, 9, 8,␣
↪6, 1, -1
```

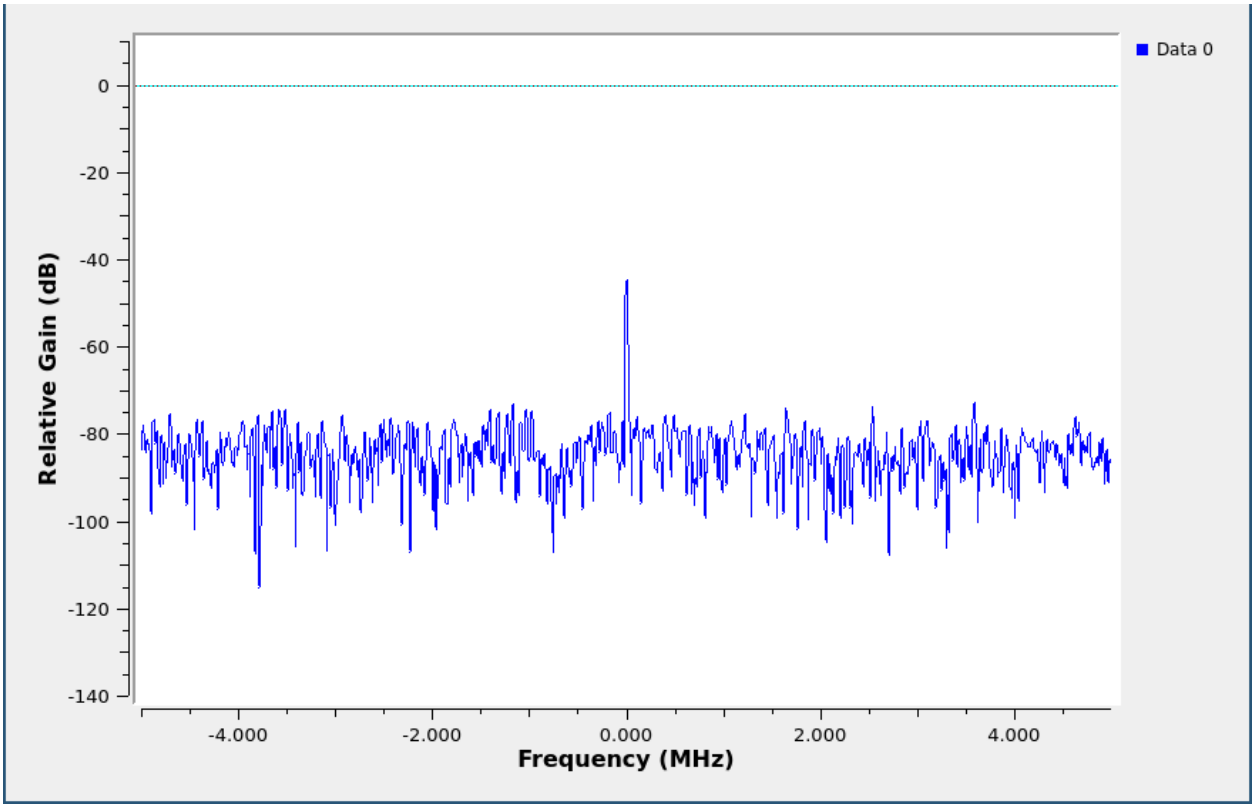


Fig. 1: DC spike

This periodic signal contains a strong sinusoidal component spanning from -2 to 9. If we plot the spectrum of this signal, you can see one spike at the frequency of this sinusoid and a second spike at 0 Hz (DC).

If the signal spanned from values -2 to 2 (centered around zero), there would be no DC offset. Since it is centered around 3.5 (the number midway between -2 and 9), there is a DC component.

Samples produced by HackRF are measurements of radio waveforms, but the measurement method is prone to a DC bias introduced by HackRF. It's an artifact of the measurement system, not an indication of a received radio signal. DC offset is not unique to HackRF; it is common to all quadrature sampling systems.

A high DC offset is also one of a few symptoms that can be caused by a software version mismatch. A common problem is that people run an old version of gr-osmosdr with newer firmware.

2.2.1 Solution

There are a few options:

1. Ignore it. For many applications it isn't a problem. You'll learn to ignore it.
2. Avoid it. The best way to handle DC offset for most applications is to use offset tuning; instead of tuning to your exact frequency of interest, tune to a nearby frequency so that the entire signal you are interested in is shifted away from 0 Hz but still within the received bandwidth. If your algorithm works best with your signal centered at 0 Hz (many do), you can shift the frequency in the digital domain, moving your signal of interest to 0 Hz and your DC offset away from 0 Hz. HackRF's high maximum sampling rate can be a big help as it allows you to use offset tuning even for relatively wideband signals.
3. Correct it. There are various ways of removing the DC offset in software. However, these techniques may degrade

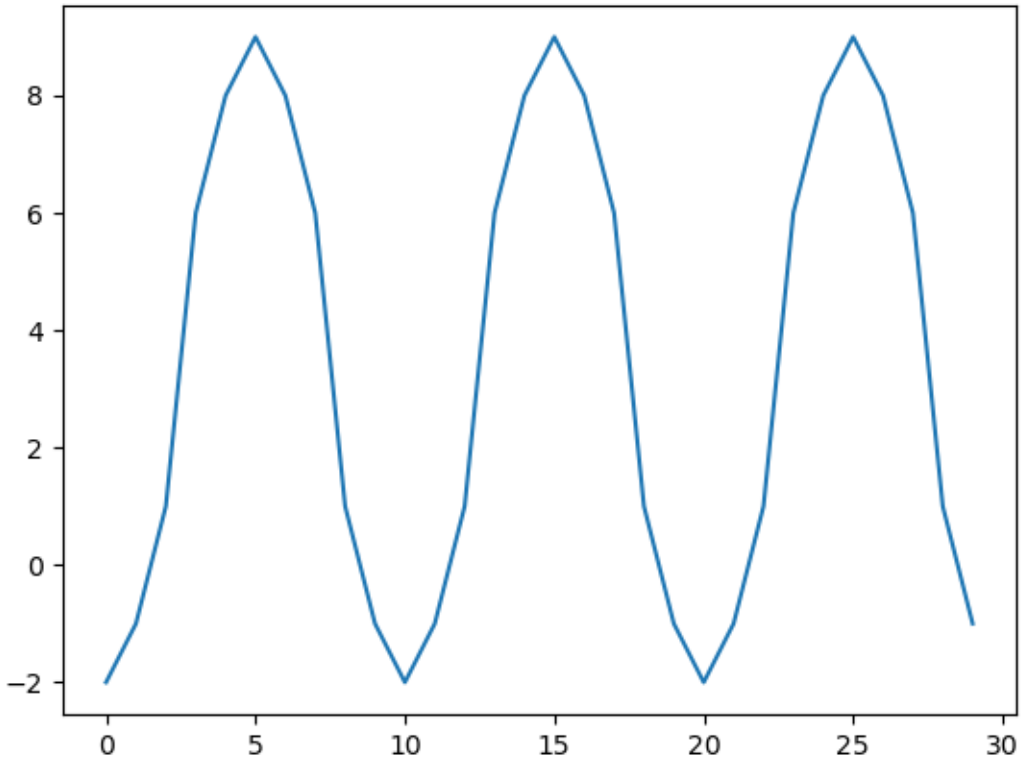


Fig. 2: Example signal

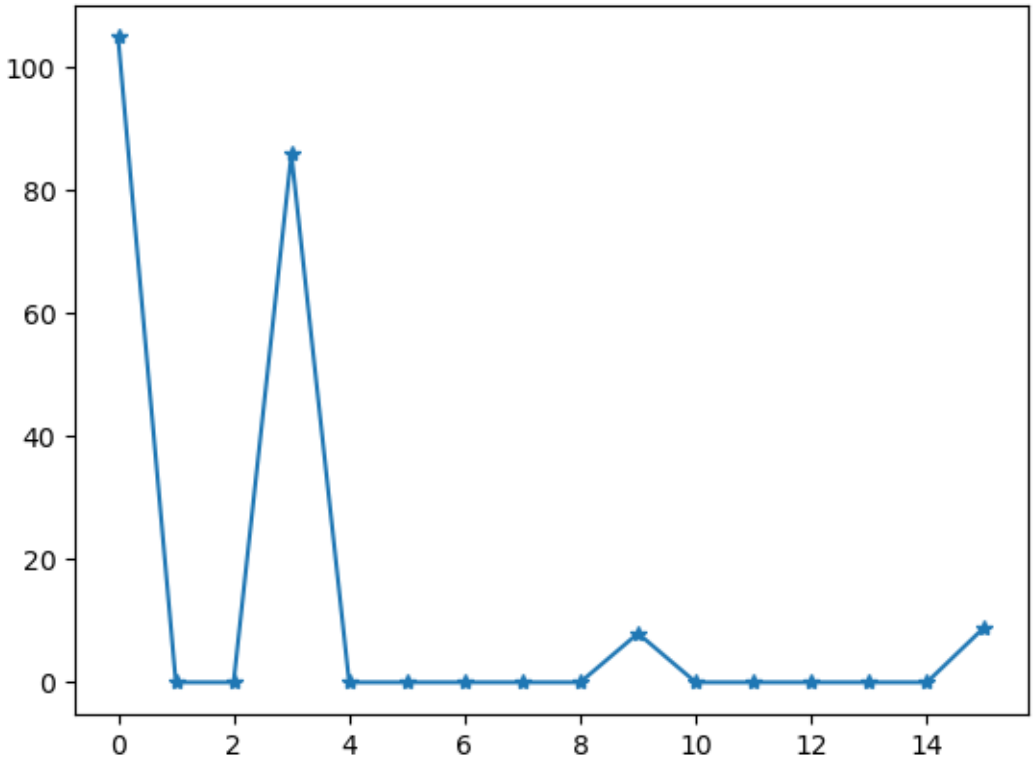


Fig. 3: Spectrum of example signal

parts of the signal that are close to 0 Hz. It may look better, but that doesn't necessarily mean that it is better from the standpoint of a demodulator algorithm, for example. Still, correcting the DC offset is often a good choice.

CHAPTER

THREE

SYNCHRONIZATION CHECKLIST

There are many scenarios where you may want to use multiple HackRF One devices synchronized with each other. For instance, multiple devices can be used with a phased antenna array to implement direction finding or beamforming.

If you're having trouble achieving fully synchronized operation, the following checklist may be useful for troubleshooting:

- **Are you running the latest firmware and host software versions?**

  There have been many bug fixes, so please use the latest releases.

- **Are you applying settings to the right target devices?**

  With more than one HackRF device connected, you need to take care to specify which device should be used where. When using our command line tools, use the `-d` option with a serial number on the command line to specify which device each command should target.

- **Are all HackRFs sharing a clock?**

  If the HackRFs are not sharing a clock, frequencies and sample rates will not match exactly. See the *external clock interface* section for how to connect the clock signals.

- **Is the clock input being detected?**

  Use `hackrf_clock` with the `-i` option to check for clock input. Use the `-d` option to specify the serial number.

  This requires `2022.09.1` or later host software.

  The older way of checking using `hackrf_debug` will not work correctly on some hardware revisions.

- **If the clock source is CLKOUT of another HackRF, has it been enabled?**

  Use `hackrf_clock` with the `-o` option to enable the clock output. Use the `-d` option to specify the serial number.

- **Is the CLKIN waveform correct?**
  - It should be a 10 MHz square wave between 0 V and 3.0 to 3.3 V.
  - A sine wave is OK, but may give greater phase noise.
  - An unbuffered TCXO output may not have sufficient voltage.
  - Some hardware revisions are less tolerant of out-of-spec clock input than others.

- **Is your hardware faulty?**

  Some HackRF clones were sold with a non-functional CLKIN port.

- **Are all HackRFs being started together using hardware triggering?**

If hardware triggering is not used, start times will not match exactly. See the *hardware triggering* section for how to connect the necessary trigger signals.

Use `hackrf_transfer` with the `-H` option to wait for trigger input.

If launching multiple `hackrf_transfer` instances, make sure that those running with `-H` have had time to reach the `Waiting for trigger...` state before the trigger signal is sent.

Note: There is currently no support for hardware triggering via the Osmocom or Soapy blocks in GNU Radio. The Osmocom block may look like it supports this but the options have no effect.

- **Are any samples being lost due to USB throughput problems?**

  If RX samples are dropped or TX samples delayed, signals will become out of sync.

  Use `hackrf_debug -S` on each HackRF after running your software to check if there were throughput problems. If the shortfall count reported is non-zero, some samples were dropped on RX or late for TX, and signals will be out of sync as a result.

  This requires `2022.09.1` or later host software.

  To force shortfalls to stop the device at runtime, use `hackrf_debug -T 0 -R 0` to set zero tolerance for shortfalls.

- **Are your HackRFs sharing a USB bus?**

  A single HackRF One at 20 MHz sample rate uses practically all the bandwidth of a single USB 2.0 bus. Unless using very low sample rates, each HackRF should be connected to its own bus.

CHAPTER

FOUR

HACKRF COMMUNITY PROJECTS AND MENTIONS

Have you done something cool with HackRF or mentioned HackRF in one of your presentations? Let us know and we might post a link here!

- HackRF vs. Tesla Model S (Sam Edwards)
- Jawbreaker/VFD spectrum analyzer (Jared Boone)
- LEGO car (Michael Ossmann)
- wireless microphones (Jared Boone)
- Tesla Charging Port Opener (Radoslav Gerganov)
- Hacking my smart tooth brush (Cyrill Künzi)

4.1 Retired Projects

- Automotive Remote Keyless Entry Systems (Mike Kershaw)
- Decoding Pocsag Pagers With The HackRF (BinaryRF)
- Sniffing GSM with HackRF (BinaryRF)

CHAPTER

FIVE

HACKRF PRO

HackRF Pro is the current hardware platform for the HackRF project. It is a Software Defined Radio peripheral capable of transmission or reception of radio signals from 100 kHz to 6 GHz. HackRF Pro is designed to be backwards compatible with software and hardware developed for use with *HackRF One*, whilst introducing many new features and improvements.

Product page
Where to buy

5.1 Features

- 100 kHz to 6 GHz operating frequency
- Tunable from 0 Hz to 7.1 GHz
- Half-duplex transceiver
- Up to 20 million samples per second
- 8-bit quadrature samples (8-bit I and 8-bit Q)
- Compatible with GNU Radio, SDR#, and more
- Software-configurable RX and TX gain and baseband filter
- Software-controlled RF port power (50 mA at 3.3 V)
- SMA RF connector
- SMA clock input and output for synchronization and triggering
- Convenient buttons for programming
- Internal pin headers for expansion
- High-Speed USB 2.0 with Type-C connector
- USB-powered
- Open source hardware

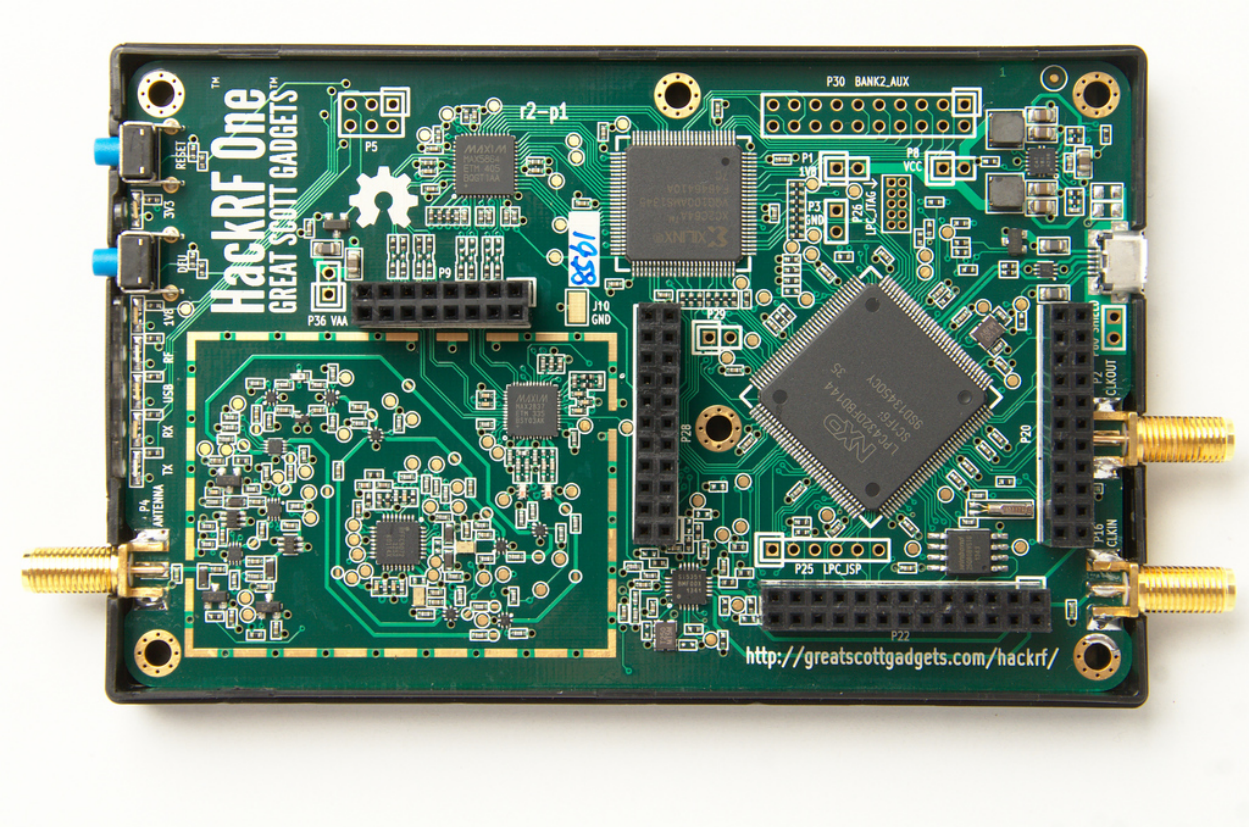
Compared to HackRF One, HackRF Pro introduces a host of new and updated features, including:

- Wider operating frequency range
- Improved RF performance with flatter frequency response
- Modern USB Type-C connector
- Built-in TCXO crystal oscillator for superior timing stability
- Logic upgrade from a CPLD to a power-efficient FPGA
- Elimination of the DC spike
- Extended-precision mode with 16-bit samples for low sample rates (typical ENOB: 9-11)
- Half-precision mode with 4-bit samples at up to 40 Msps
- More RAM and flash memory for custom firmware
- Installed shielding around the radio section
- Trigger input and output accessible through clock connectors
- Cutout in the PCB provides space for future add-ons
- Improved power management
- Enhanced RF port protection
- Facility to hardware-disable transmit mode

CHAPTER

SIX

HACKRF ONE



HackRF One was the first production hardware platform for the HackRF project. It is a Software Defined Radio peripheral capable of transmission or reception of radio signals from 1 MHz to 6 GHz. Designed to enable test and development of modern and next generation radio technologies, HackRF One is an open source hardware platform that can be used as a USB peripheral or programmed for stand-alone operation.

Product page
Where to buy

6.1 Features

- half-duplex transceiver
- operating freq: 1 MHz to 6 GHz
- supported sample rates: 2 Msps to 20 Msps (quadrature)
- resolution: 8 bits
- interface: High Speed USB (with USB Micro-B connector)
- power supply: USB bus power
- software-controlled antenna port power (max 50 mA at 3.0 to 3.3 V)
- SMA female antenna connector (50 ohms)
- SMA female clock input and output for synchronization
- convenient buttons for programming
- pin headers for expansion
- portable
- open source

6.2 Maximum input power

The maximum input power of HackRF One is -5 dBm. Exceeding -5 dBm can result in permanent damage!

In theory, HackRF One can safely accept up to 10 dBm with the front-end RX amplifier disabled. However, a simple software or user error could enable the amplifier, resulting in permanent damage. It is better to use an external attenuator than to risk damage.

6.3 Minimum detectable input power

This isn't a question that can be answered for a general purpose SDR platform such as HackRF. Any answer would be very specific to a particular application. For example, an answerable question might be: What is the minimum power level in dBm of modulation M at frequency F that can be detected by HackRF One with software S under configuration C at a bit error rate of no more than E%? Changing any of those variables (M, F, S, C, or E) would change the answer to the question. Even a seemingly minor software update might result in a significantly different answer. To learn the exact answer for a specific application, you would have to measure it yourself.

HackRF's concrete specifications include operating frequency range, maximum sample rate, and dynamic range in bits. These specifications can be used to roughly determine the suitability of HackRF for a given application. Testing is required to finely measure performance in an application. Performance can typically be enhanced significantly by selecting an appropriate antenna, external amplifier, and/or external filter for the application.

6.4 Typical maximum transmit power

HackRF One's maximum TX power varies by operating frequency:

- 1 MHz to 10 MHz: 5 dBm to 15 dBm, generally increasing as frequency increases (see this blog post)
- 10 MHz to 2170 MHz: 5 dBm to 15 dBm, generally decreasing as frequency increases
- 2170 MHz to 2740 MHz: 13 dBm to 15 dBm
- 2740 MHz to 4000 MHz: 0 dBm to 5 dBm, decreasing as frequency increases
- 4000 MHz to 6000 MHz: -10 dBm to 0 dBm, generally decreasing as frequency increases

Through most of the frequency range up to 4 GHz, the maximum TX power is between 0 and 10 dBm. The frequency range with best performance is 2170 MHz to 2740 MHz.

Overall, the output power is enough to perform over-the-air experiments at close range or to drive an external amplifier. If you connect an external amplifier, you should also use an external bandpass filter for your operating frequency.

Before you transmit, know your laws. HackRF One has not been tested for compliance with regulations governing transmission of radio signals. You are responsible for using your HackRF One legally.

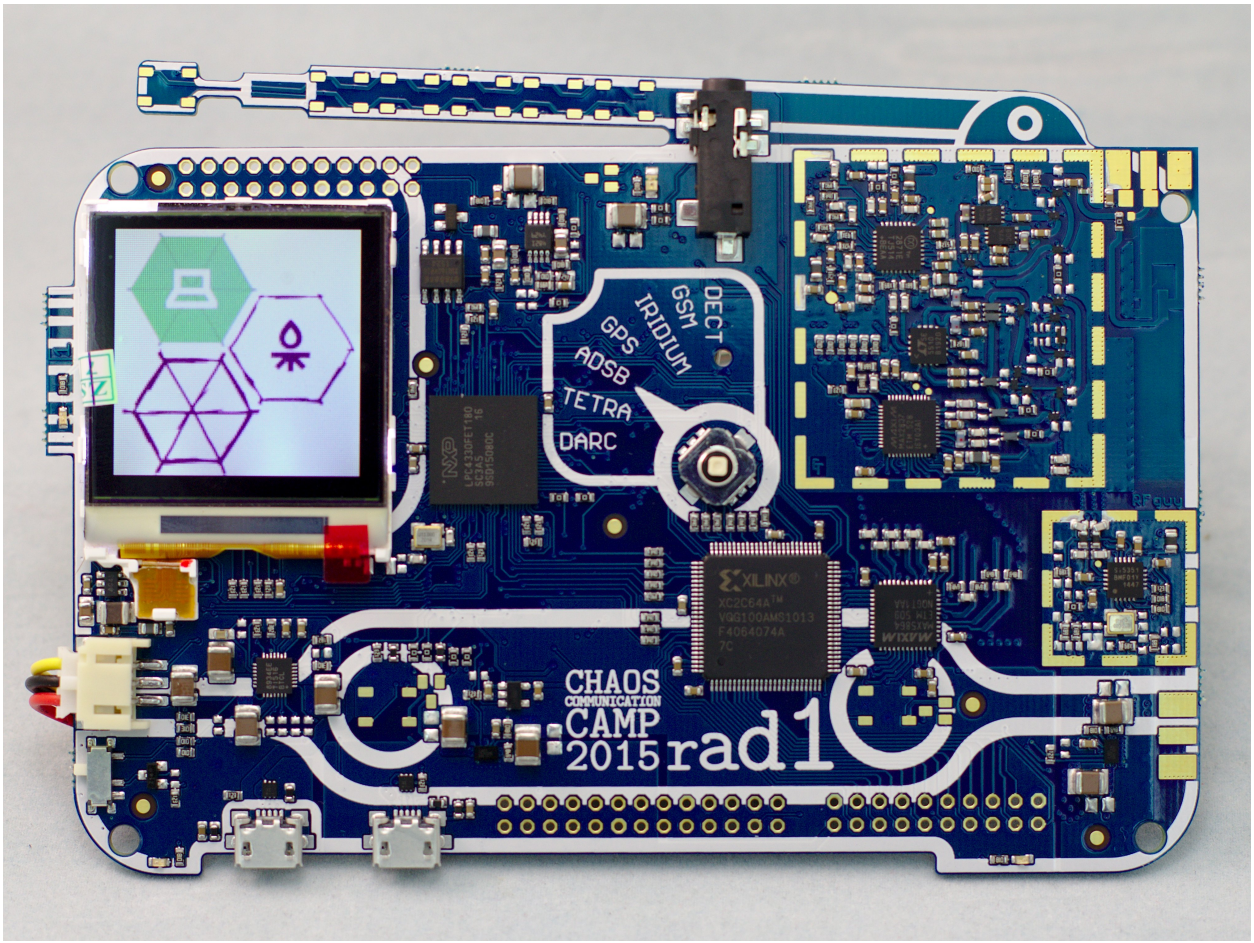
CHAPTER

SEVEN

RAD1O

rad1o is the badge from the 2015 Chaos Communication Camp.

The rad1o badge contains a full-featured SDR (software defined radio) half-duplex transceiver, operating in a frequency range of about 50 MHz - 4000 MHz, and is software compatible to the HackRF.



(rad1o picture provided by Christoph Krichenbauer with Creative Commons License CC-BY-NC_SA)

More information can be found at the rad1o badge wiki

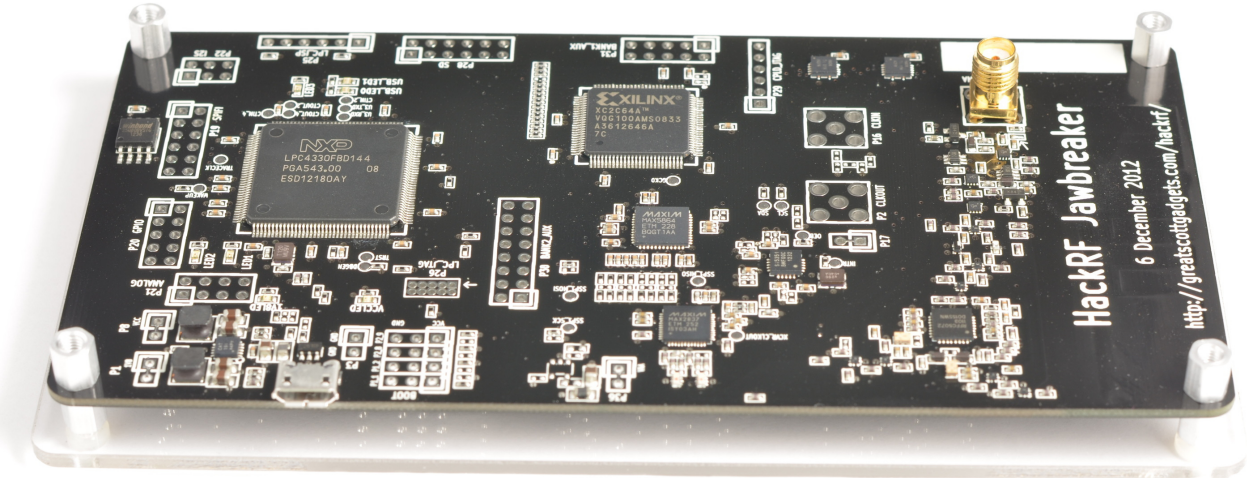
Compared to HackRF One, the rad1o badge uses a different mixer (MAX2871) with a reduced frequency range.

CHAPTER

EIGHT

JAWBREAKER

HackRF Jawbreaker is the beta test hardware platform for the HackRF project.



(Jawbreaker picture provided by fd0 with Creative Commons License CC BY 3.0.)

8.1 Features

- half-duplex transceiver
- operating freq: 30 MHz to 6 GHz
- supported sample rates: 8 Msps to 20 Msps (quadrature)
- resolution: 8 bits
- interface: High Speed USB (with USB Micro-B connector)
- power supply: USB bus power
- portable
- open source

8.2 Hardware Documentation

Schematic diagram, assembly diagram, and bill of materials can be found at https://github.com/greatscottgadgets/hackrf/tree/master/hardware

8.3 Transmit Power

The maximum TX power for Jawbreaker varies by operating frequency:

- 30 MHz to 100 MHz: 5 dBm to 15 dBm, increasing as frequency decreases
- 100 MHz to 2300 MHz: 0 dBm to 10 dBm, increasing as frequency decreases
- 2170 MHz to 2740 MHz: 10 dBm to 15 dBm
- 2700 MHz to 4000 MHz: -5 dBm to 5 dBm, increasing as frequency decreases
- 4000 MHz to 6000 MHz: -15 dBm to 0 dBm, increasing as frequency decreases

Overall, the output power is enough to perform over-the-air experiments at close range or to drive an external amplifier. If you connect an external amplifier, you should also use an external bandpass filter for your operating frequency.

Before you transmit, know the laws for the region you are transmitting in. Jawbreaker has not been tested for compliance with regulations governing transmission of radio signals. You are responsible for using your Jawbreaker legally.

8.4 SMA, not RP-SMA

The connectors on Jawbreaker are SMA, not RP-SMA. SMA connectors and RP-SMA connectors look extremely similar, the difference is that SMA connectors have a center pin. RP-SMA connectors are common on 2.4 GHz antennas and are popular on Wi-Fi equipment. If you connect an RP-SMA antenna to Jawbreaker, it will seem to connect snugly but won't function at all because neither the male nor female side has a center pin.

8.5 Recommended PCB and Antenna Changes

Jawbreaker has an SMA antenna connector and it also includes a built-in PCB antenna intended for operation near 900 MHz. The built-in PCB antenna isn't a very good antenna. A paperclip stuck into the SMA connector of the Jawbreaker is likely to be better. We recommend that you free your Jawbreaker to operate with better antennas by cutting the PCB trace to the PCB antenna with a knife. This enables the SMA connector to be used without interference from the PCB antenna.

The trace to be cut is between the two solder pads inside a box labeled R44. There is an arrow printed on the board that points to the R44 box. A video that demonstrates the antenna modification is on YouTube: HackRF Antenna Modification.

Due to a manufacturing error, there is solder on the pads in box R44 that you should try to remove before you cut the trace. R44 may appear as a single solder blob. If you have a soldering iron and solder wick/braid, use a soldering iron and fine solder wick to remove as much solder as you can from the two R44 pads. Then, use a pen knife to gently cut away the area between the two R44 pads. Make multiple, gentle cuts, instead of one or two forceful cuts. As you cut, you'll break through the black solder mask, then the copper trace between the pads, and stop when you reach fiberglass. Remove the copper trace completely, so just the two R44 pads remain. Use a multimeter or continuity tester to verify that the two R44 pads are no longer connected. If you don't have a soldering iron, you can cut through the copper trace and the solder blob all at once, but it requires a bit more effort. The only reason not to cut the PCB trace is if you want to try Jawbreaker but don't have any antenna with an SMA connector (or adapter).

If you want to restore the PCB antenna for some reason, you can install a 10 nF capacitor or a 0 ohm resistor on the R44 pads or you may be able to simply create a solder bridge.

8.6 Expansion Interface

8.6.1 LPC

Boot config

Default boot configuration is SPIFI. Install headers and jumpers (and optionally resistors) to reconfigure.

Pin	P43	P32	P42	P27
1	VCC	VCC	VCC	VCC
2	P2_9	P2_8	P1_2	P1_1
3	GND	GND	GND	GND

The table below shows which pins to short per header for a given selection.

Selection	P43	P32	P42	P27
USART0	2-3	2-3	2-3	2-3
SPIFI	2-3	2-3	2-3	1-2
USB0	2-3	1-2	2-3	1-2
USSP0	2-3	1-2	1-2	1-2
USART3	1-2	2-3	2-3	2-3

P19 SPIFI Intercept header

Traces may be cut to install header and jumpers or use off-board SPI flash.

Pin	Function
1	Flash DO
2	SPIFI_MISO
3	Flash DI
4	SPIFI_MOSI
5	Flash CLK
6	SPIFI_SCK
7	Flash CS
8	SPIFI_CS
9	Flash Hold
10	SPIFI_SIO3
11	Flash WP
12	SPIFI_SIO2

P20 GPIO

Pin	Function
1	GPIO3_8
2	GPIO3_9
3	GPIO3_10
4	GPIO3_11
5	GPIO3_12
6	GPIO3_13
7	GPIO3_14
8	GPIO3_15
9	GND
10	GND

P21 Analog

Pin	Function
1	GND
2	ADC0_6
3	GND
4	ADC0_2
5	GND
6	ADC0_5
7	GND
8	ADC0_0

P22 I2S

Pin	Function
1	VCC
2	I2S0_TX_SDA
3	I2S0_TX_WS
4	I2S0_TX_SCK
5	I2S0_TX_MCLK
6	GND

P25 LPC_ISP

Pin	Function
1	GND
2	ISP
3	NC
4	U0_RXD
5	U0_TXD
6	RESET

P26 LPC_JTAG

Pin	Function
1	VCC
2	TMS
3	GND
4	TCK
5	GND
6	TDO
7	NC
8	TDI
9	GND
10	RESET

P28 SD

Pin	Function
1	GND
2	VCC
3	SD_CD
4	SD_DAT3
5	SD_DAT2
6	SD_DAT1
7	SD_DAT0
8	SD_VOLT0
9	SD_CMD
10	SD_POW
11	SD_CLK
12	NC

8.6.2 CPLD

P29 CPLD_JTAG

Pin	Function
1	CPLD_TMS
2	CPLD_TDI
3	CPLD_TDO
4	CPLD_TCK
5	GND
6	NCC

P30 BANK2_AUX

Pin	Function
1	B2AUX1
2	B2AUX2
3	B2AUX3
4	B2AUX4
5	B2AUX5
6	B2AUX6
7	B2AUX7
8	B2AUX8
9	B2AUX9
10	B2AUX10
11	B2AUX11
12	B2AUX12
13	B2AUX13
14	B2AUX14
15	B2AUX15
16	B2AUX16

P31 BANK1_AUX

Pin	Function
1	B1AUX9
2	B1AUX10
3	B1AUX11
4	B1AUX12
5	B1AUX13
6	B1AUX14
7	B1AUX15
8	B1AUX16
9	GND
10	GND

8.6.3 External clock

P2 CLKOUT

Install C165 and R92 as necessary to match output. For CMOS output, install 0 ohm resistor in place of C165; do not install R92.

Pin	Function
1	CLKOUT
2	GND
3	GND
4	GND
5	GND

P16 CLKIN

Install C118, C164, R45, R84 and R85 as necessary to match input.

For CMOS input, install 0 ohm resistors in place of C118 and C164; do not install R45, R84, or R85.

Pin	Function
1	CLKIN
2	GND
3	GND
4	GND
5	GND

P17 CLKIN_JMP

Cut P17 short (trace) to enable external clock input. If short is cut, a jumper should be used on P17 at all times when an external clock is not connected to P16.

Pin	Function
1	GND
2	CLKIN

8.6.4 More

Additional headers are available. See the board files for additional details.

8.7 Differences between Jawbreaker and HackRF One

Jawbreaker was the beta platform that preceded HackRF One. HackRF One incorporates the following changes and enhancements (at minimum):

- Antenna port: No modification is necessary to use the SMA antenna port on HackRF One.
- PCB antenna: Removed.
- Size: HackRF One is smaller at 120 mm x 75 mm (PCB size).
- Enclosure: The commercial version of HackRF One from Great Scott Gadgets ships with an injection molded plastic enclosure. HackRF One is also designed to fit other enclosure options.
- Buttons: HackRF One has a RESET button and a DFU button for easy programming.
- Clock input and output: Installed and functional without modification.
- USB connector: HackRF One features a new USB connector and improved USB layout.
- Expansion interface: More pins are available for expansion, and pin headers are installed on HackRF One.
- Real-Time Clock: An RTC is installed on HackRF One.
- LPC4320 microcontroller: Jawbreaker had an LPC4330.
- RF shield footprint: An optional shield may be installed over HackRF One's RF section.

- Antenna port power: HackRF One can supply up to 50 mA at 3.0 to 3.3 V DC on the antenna port for compatibility with powered antennas and other low power amplifiers.
- Enhanced frequency range: The RF performance of HackRF One is better than Jawbreaker, particularly at the high and low ends of the operating frequency range. HackRF One can operate at 1 MHz or even lower.

CHAPTER

NINE

MINIMUM HOST SYSTEM REQUIREMENTS FOR HACKRF

HackRF requires you to supply 500 mA at 5 V DC to your HackRF via the USB port. If your host computer has difficulty meeting this requirement, you may need to use a powered USB hub.

There is no specific minimum CPU requirement for the host computer when using a HackRF, but SDR is generally a CPU-intensive application. If you have a slower CPU, you may be unable to run certain SDR software or you may only be able to operate at lower sample rates.

Most users will want to stream data to or from the HackRF at high speeds. This requires that the host computer supports Hi-Speed USB. Some Hi-Speed USB hosts are better than others, and you may have multiple host controllers on your computer. If you have difficulty operating your HackRF at high sample rates (10 Msps to 20 Msps), try using a different USB port on your computer. If possible, arrange things so that the HackRF is the only device on the bus.

CHAPTER

TEN

HARDWARE REVISIONS

Hardware revisions exist mainly to deal with changes in component availability. Each revision of a product meets the same performance specifications that are measured in the factory.

10.1 HackRF Pro

The initial production revision of HackRF Pro is r1.2.1.

10.2 HackRF One

10.2.1 HackRF One r1–r4

The first revision of HackRF One shipped by Great Scott Gadgets starting in 2014 was labeled r1. Subsequent manufacturing runs incremented the revision number up to r4 without modification to the hardware design. Manufacturing years: 2014–2020

10.2.2 HackRF One r5

This experimental revision has not been manufactured.

10.2.3 HackRF One r6

SKY13350 RF switches were replaced by SKY13453. Although the SKY13453 uses simplified control logic, it did not require a firmware modification. Hardware revision detection pin straps were added. Manufacturing year: 2020

10.2.4 HackRF One r7

SKY13453 RF switches were reverted to SKY13350. USB VBUS detection resistor values were updated. Manufacturing year: 2021

10.2.5 HackRF One r8

SKY13350 RF switches were replaced by SKY13453. Manufacturing years: 2021–2022

10.2.6 HackRF One r9

MAX2837 was replaced by MAX2839. Si5351C was replaced by Si5351A with additional clock distribution. A series diode was added to the antenna port power supply. Manufacturing year: 2023

10.2.7 HackRF One r10

This revision is based on r8, reverting most of the changes made in r9. A series diode was added to the antenna port power supply. Manufacturing year: 2024

10.3 Hardware Revision Identification

HackRF Ones manufactured by Great Scott Gadgets have the revision number printed on the PCB top silkscreen layer near the MAX5864 (U18).

Starting with HackRF One r6, hardware revisions are detected by firmware and reported by `hackrf_info`.

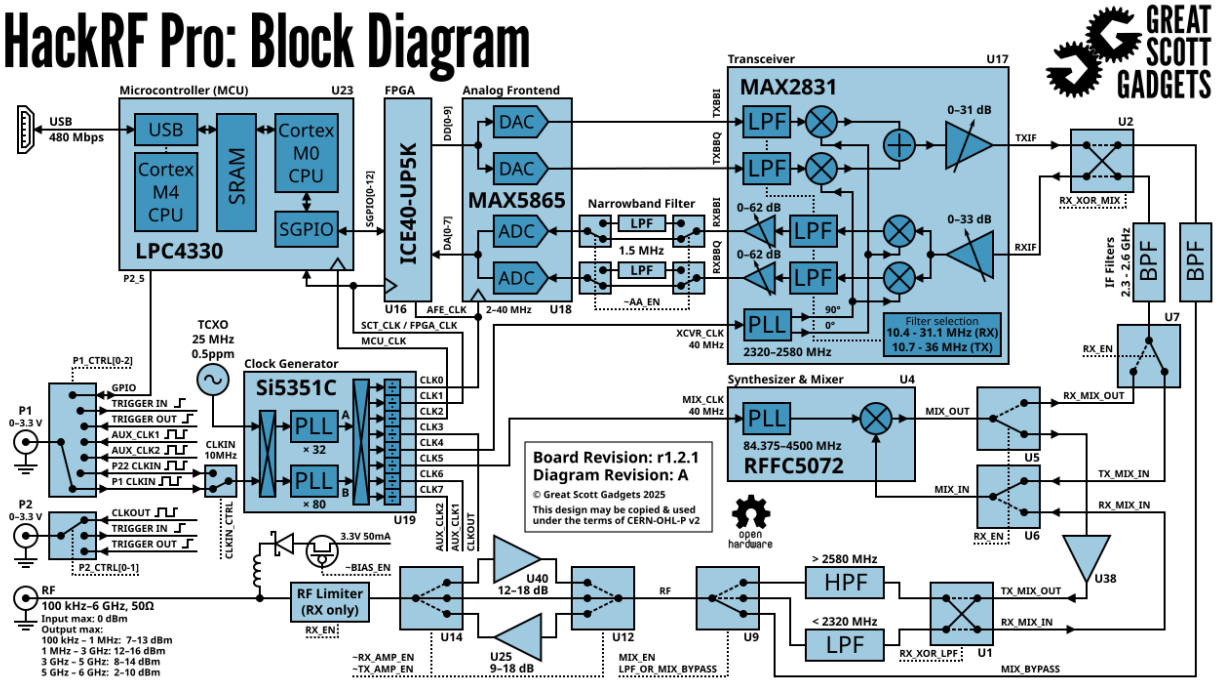
CHAPTER

ELEVEN

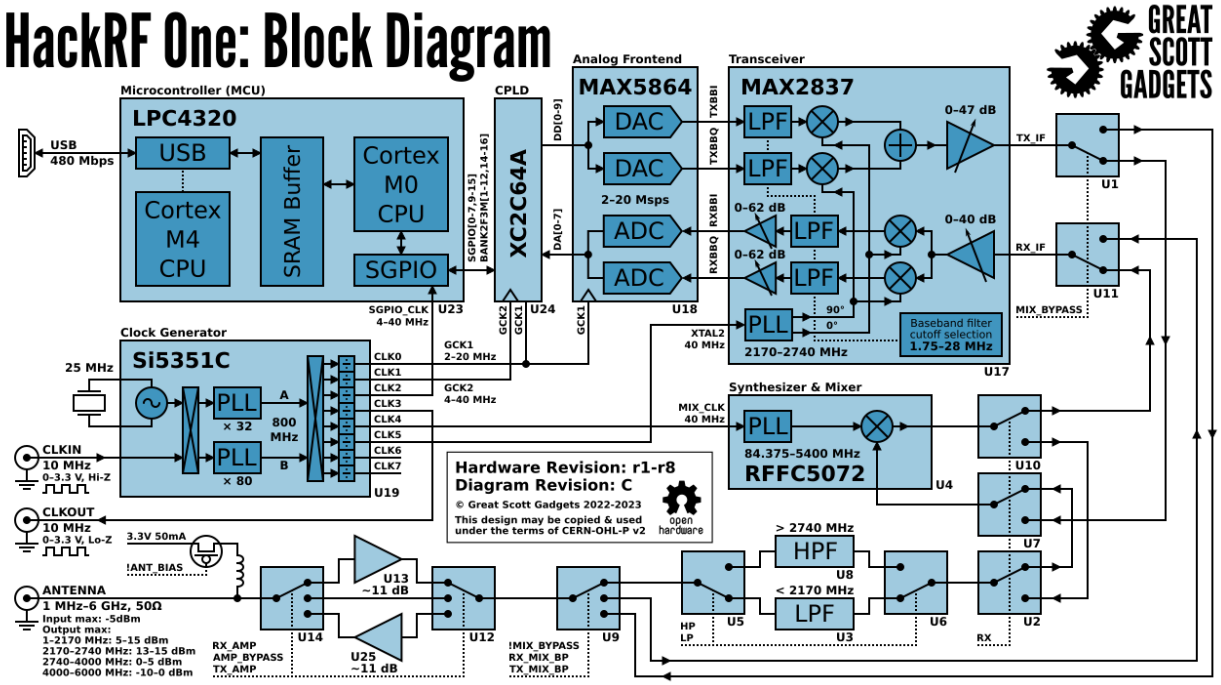
HARDWARE COMPONENTS

11.1 Block Diagrams

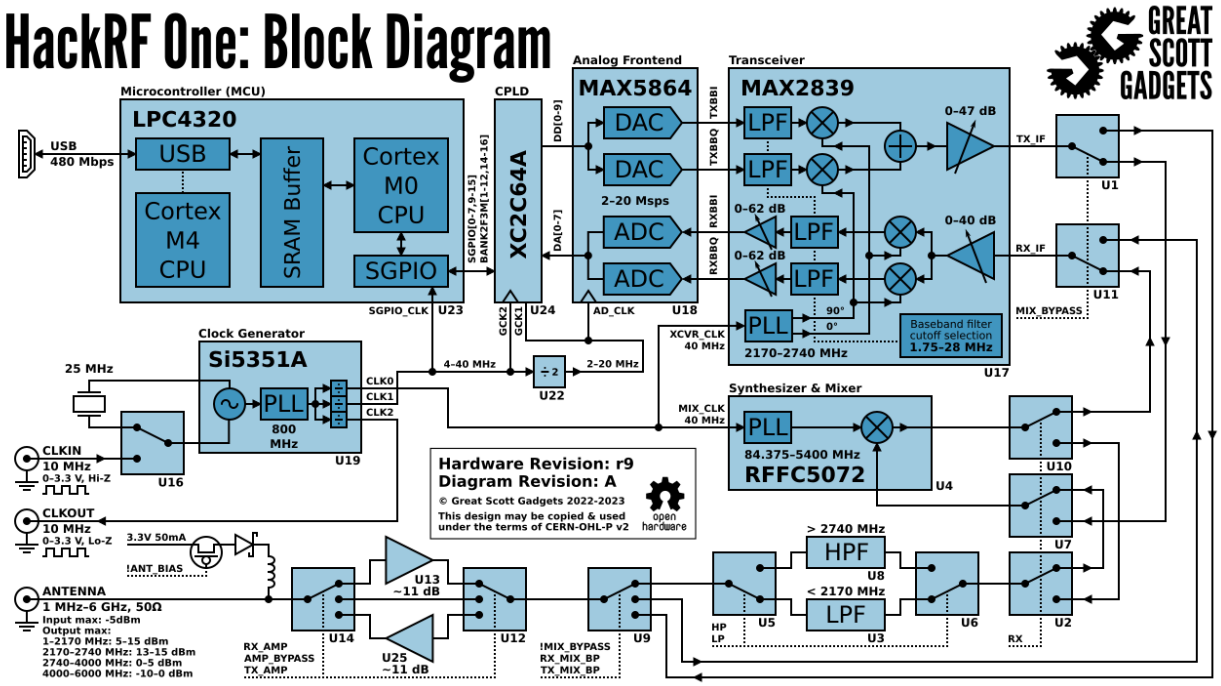
11.1.1 HackRF Pro Block Diagram



11.1.2 HackRF One r1-r8 Block Diagram



11.1.3 HackRF One r9 Block Diagram



11.2 Key Components

Major parts used in HackRF:

- **MAX2831 2.3 to 2.6 GHz transceiver**
  - Used on HackRF Pro.
  - Datasheet
- **MAX2837 2.3 to 2.7 GHz transceiver**
  - Used on HackRF One (except revision r9), Jawbreaker and rad1o.
  - Datasheet
- **MAX2839 2.3 to 2.7 GHz transceiver**
  - Substitution for MAX2837, used on HackRF One revision r9.
  - Datasheet
- **MAX5864 ADC/DAC**
  - Datasheet
- **Si5351 clock generator**
  - AN619: Manually Generating an Si5351 Register Map
  - Datasheet - see AN619 for the complete register map.
  - Other Documentation - includes application notes, user guides, and white papers.
- ice40 UltraPlus FPGA (HackRF Pro)
- CoolRunner-II CPLD (all other platforms)
- **LPC43xx ARM Cortex-M4 microcontroller**
  - User Manual
  - Datasheet
  - Other Documentation (LPC4330FBD144) - includes errata and application notes.
  - ARM-standard JTAG/SWD connector pinout
  - BSDL file for the LPC43xx (For boundary scan)
- **RFFC5072 mixer/synthesizer**
  - Datasheet
  - Other Documentation ; click “Technical Documents” - includes programming guides and application notes.
- W25Q32 32M-bit Flash (HackRF Pro)
- W25Q80BV 8M-bit Flash (all other platforms)

CHAPTER

TWELVE

LEDS

12.1 HackRF Pro

When HackRF Pro is plugged in to a USB host, four LEDs should turn on: MCU, FPGA, RF, and USB. The MCU LED indicates that the primary internal power supply is working properly and that firmware is running. The FPGA and RF LEDs indicate that firmware has switched on additional internal power supplies. The USB LED indicates that the HackRF Pro is communicating with the host over USB.

12.2 HackRF One

When HackRF One is plugged in to a USB host, four LEDs should turn on: 3V3, 1V8, RF, and USB. The 3V3 LED indicates that the primary internal power supply is working properly. The 1V8 and RF LEDs indicate that firmware is running and has switched on additional internal power supplies. The USB LED indicates that the HackRF One is communicating with the host over USB.

12.3 Both versions

The RX and TX LEDs indicate that a receive or transmit operation is currently in progress.

Each LED is a single color. There are no multi-colored LEDs on either HackRF One or HackRF Pro. Adjacent LEDs are different colors in order to make them easier to distinguish from one another. The colors do not mean anything.

CHAPTER

THIRTEEN

BUTTONS

This information is applicable to both HackRF Pro and HackRF One.

The **RESET button** resets the microcontroller. This is a reboot that should result in a USB re-enumeration.

The **DFU button** invokes a USB DFU bootloader located in the microcontroller's ROM. This bootloader makes it possible to unbrick a HackRF with damaged firmware because the ROM cannot be overwritten.

The DFU button only invokes the bootloader during reset. This means that it can be used for other functions by custom firmware.

To invoke DFU mode: Press and hold the DFU button. While holding the DFU button, reset the HackRF either by pressing and releasing the RESET button or by powering on the HackRF. Release the DFU button.

CHAPTER

FOURTEEN

CONNECTORS

The connectors on both HackRF Pro and HackRF One are SMA.

Note: SMA connectors and RP-SMA connectors are visually very similar. If you connect an RP-SMA antenna to a HackRF, it will seem to connect snugly but won't function at all because neither the male nor female side has a center pin. RP-SMA connectors are most common on 2.4 GHz antennas and are popular on Wi-Fi equipment. Adapters are available.

CHAPTER

FIFTEEN

EXTERNAL CLOCK INTERFACE

15.1 HackRF Pro

HackRF Pro has two configurable SMA ports, P1 and P2. By default, P1 is configured as CLKIN and P2 as CLKOUT. The default behaviour of these signals is as described for HackRF One below.

A second CLKIN signal is available on header P22 pin 2. Unlike HackRF One, HackRF Pro's P22_CLKIN is a separate signal from P1_CLKIN. To enable P22_CLKIN instead of P1_CLKIN use `hackrf_clock -c p22`.

Various internal signals can be connected to P1 or P2 instead of the default CLKIN and CLKOUT signals. Use `hackrf_clock -1` or `hackrf_clock -2` to select a different signal.

15.2 HackRF One

HackRF One produces a 10 MHz clock signal on the CLKOUT SMA port. The signal is a 3.3 V, 10 MHz square wave intended for a high impedance load.

The CLKIN SMA port on HackRF One is a high impedance input that expects 3.3 V square wave at 10 MHz. Do not exceed 3.3 V or drop below 0 V on this input. Do not connect a clock signal at a frequency other than 10 MHz (unless you modify the firmware to support this). You may directly connect the CLKOUT port of one HackRF One to the CLKIN port of another HackRF.

The CLKIN signal is also connected to header P22 pin 2. Unlike HackRF Pro, HackRF One has only one CLKIN signal shared between P22 pin 2 and the CLKIN port. Do not connect input signals to both CLKIN and P22 pin 2 simultaneously.

HackRF One uses CLKIN instead of the internal crystal when a clock signal is detected on CLKIN. The switch to or from CLKIN only happens when a transmit or receive operation begins.

To verify that a signal has been detected on CLKIN, use `hackrf_clock -i`. The expected output with a clock detected is *CLKIN status: clock signal detected*. The expected output with no clock detected is *CLKIN status: no clock signal detected*.

To activate CLKOUT, use `hackrf_clock -o 1`. To switch it off, use `hackrf_clock -o 0`.

CHAPTER

SIXTEEN

EXPANSION INTERFACE

The common HackRF expansion interface consists of headers P20, P22, and P28. These headers are present on both HackRF Pro and HackRF One, and support hardware add-ons including PortaPack and Opera Cake.

16.1 P20 GPIO

Providing access to GPIO, ADC, RTC, and power.

Pin	Function
1	VBAT
2	RTC_ALARM (One) / PB_5 (Pro)
3	VCC (One) / 3V3AUX (Pro)
4	WAKEUP
5	GPIO3_8
6	GPIO3_0 (One) / GPIO3_9 (Pro)
7	GPIO3_10
8	GPIO3_11
9	GPIO3_12
10	GPIO3_13
11	GPIO3_14
12	GPIO3_15
13	GND
14	ADC0_6
15	GND
16	ADC0_2
17	VBUSCTRL
18	ADC0_5
19	GND
20	ADC0_0
21	VBUS
22	VIN

16.2 P22 I2S

I2S, SPI, I2C, UART, GPIO, and clocks.

Pin	Function
1	CLKOUT (One) / P2 (Pro)
2	CLKIN
3	RESET
4	GND
5	I2C1_SCL
6	I2C1_SDA
7	SPIFI_MISO (One) / PB_1 (Pro)
8	SPIFI_SCK (One) / PB_3 (Pro)
9	SPIFI_MOSI (One) / PA_4 (Pro)
10	GND
11	VCC (One) / 3V3AUX (Pro)
12	I2S0_RX_SCK (One) / PA_3 (Pro)
13	I2S0_RX_SDA (One) / I2S0_TX_SDA (Pro)
14	I2S0_RX_MCLK (One) / PB_0 (Pro)
15	I2S0_RX_WS
16	I2S0_TX_SCK
17	I2S0_TX_MCLK
18	GND
19	U0_RXD
20	U0_TXD
21	P2_9
22	P2_13
23	P2_8
24	SDA
25	CLK6 (One) / AUX_CLK2 (Pro)
26	SCL

16.3 P28 SD

SDIO, GPIO, clocks, and CPLD.

Pin	Function
1	VCC (One) / 3V3AUX (Pro)
2	GND
3	SD_CD
4	SD_DAT3
5	SD_DAT2
6	SD_DAT1
7	SD_DAT0
8	SD_VOLT0
9	SD_CMD
10	SD_POW
11	SD_CLK
12	GND
13	GCK2 (One) / P5_6 (Pro)
14	GCK1 (One) / P5_7 (Pro)
15	Trigger out: B1AUX14 (One) / TRIGGER.OUT (Pro)
16	Trigger in: B1AUX13 (One) / TRIGGER.IN (Pro)
17	CPLD_TCK
18	BANK2F3M2 (One) / PE_0 (Pro)
19	CPLD_TDI (One) / I2S0_RX_SDA (Pro)
20	BANK2F3M6 (One) / P9_1 (Pro)
21	BANK2F3M12 (One) / P5_3 (Pro)
22	BANK2F3M4 (One) / P1_7 (Pro)

16.4 P9 Baseband (HackRF One)

A direct analog interface to the high speed dual ADC and dual DAC.

Pin	Function
1	GND
2	GND
3	GND
4	RXBBQ-
5	RXBBI-
6	RXBBQ+
7	RXBBI+
8	GND
9	GND
10	TXBBI-
11	TXBBQ+
12	TXBBI+
13	TXBBQ-
14	GND
15	GND
16	GND

Additional unpopulated headers and test points are available for test and development, but they may be incompatible with some enclosure or expansion options.

Refer to the schematics and component documentation for more information.

CHAPTER

SEVENTEEN

HARDWARE TRIGGERING

HackRF transmit and receive operations can be synchronized with another HackRF or with other external equipment by using the trigger input and output. Triggering provides time synchronization with error of less than one sample period.

HackRF Pro has two configurable SMA ports, P1 and P2, which can be set up to provide both clock synchronization and triggering.

HackRF One has CLKIN and CLKOUT ports for clock synchronization, but hardware triggering requires opening the case to access the P28 header.

17.1 Clock Synchronization

When triggering one HackRF from another, it is often desirable to first ensure that the two devices share a common frequency reference. This has an added benefit of grounding the HackRFs to each other, eliminating one of the wires required for triggering. See *External Clock Interface* for instructions.

Either HackRF may serve as the clock source for the other regardless of which is providing the trigger output.

17.2 Usage

Use `hackrf_info` to discover the serial numbers of both HackRFs. Using the serial number of the HackRF to be triggered, use `hackrf_transfer -H` to set up a triggered operation. For example:

- `hackrf_transfer -H -d <serial number> -a 0 -l 32 -g 32 -r rx1.cs8`

The command will print "Waiting for trigger…" until a trigger signal is detected on the device's trigger input.

In another terminal, use the serial number of the triggering HackRF One to initiate an operation to take place at the same time as the triggered operation. For example:

- `hackrf_transfer -d <serial number> -a 0 -l 32 -g 32 -r rx2.cs8`

Note that no special argument is required to activate the trigger output.

Both `hackrf_transfer` commands will start sampling RF signals at the same time, accurate to less than one sample period.

17.3 Additional Devices

Multiple HackRFs may be triggered by a single HackRF. Ensure that all the devices share a common ground and then connect one device's trigger output to the trigger inputs of the other devices (with jumpers connected via a breadboard, for example).

Equipment other than a HackRF may be connected to a HackRF's trigger input or output. The trigger signal is a 3.3 V pulse that triggers on the rising edge.

17.4 HackRF One Triggering Requirements

To connect two HackRF Ones for triggering you will need:

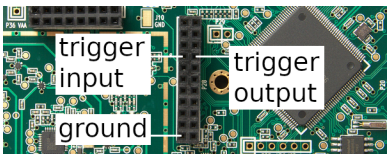
- a male-to-male jumper wire for 0.1" pin headers
- an SMA cable for clock synchronization or a second jumper wire

17.5 Open Your HackRF One

If your HackRF Ones are not bare boards, you will need to open up their cases to access the pin headers on the HackRF Ones. Each HackRF One case has small plastic clips holding it together. These clips may be damaged when the case is opened, but typically the case can still be used after such damage. Please follow the instructions in this video by Jared Boone to open your HackRF One cases.

17.6 Identify the Trigger Pins

HackRF One has four normally-populated pin headers, three of which are arranged in a 'C' shape. On the circuit board these are marked P28, P22, and P20. P28 is the header nearest to the center of the board. Locate pins 15 (trigger output) and 16 (trigger input) on header P28.



17.7 Connect the Trigger Output to the Trigger Input

First ensure that the two devices share a common ground. This may be accomplished by connecting one's CLKIN to the other's CLKOUT as recommended above. Alternatively, connect a jumper wire from P28 pin 2 on one HackRF One to P28 pin 2 on the other HackRF One.

Next use a jumper wire to connect P28 pin 15 (trigger output) on one HackRF One to P28 pin 16 (trigger input) on the other HackRF One.

17.8 References

HackRF's trigger mechanism was contributed by the authors of Synchronisation of Low-Cost Open Source SDRs for Navigation Applications which provides details about the implementation and background.

CHAPTER

EIGHTEEN

ENCLOSURE OPTIONS

Commercial versions of both HackRF Pro and HackRF One from Great Scott Gadgets ship with an injection molded plastic enclosure but are also designed to fit two optional enclosures:

- Hammond 1455J1201: Both HackRF Pro and HackRF One fit this extruded aluminum enclosure and other similar models from Hammond Manufacturing. In order to use the enclosure's end plates, you will have to drill them. An end plate template can be found in the HackRF One KiCad layout.
- Acrylic sandwich: You can also use a laser cut acrylic enclosure with either HackRF Pro or HackRF One. This is a good option for access to the expansion headers. A design can be found in the HackRF hardware directory. Use any laser cutting service or purchase from a reseller.

CHAPTER

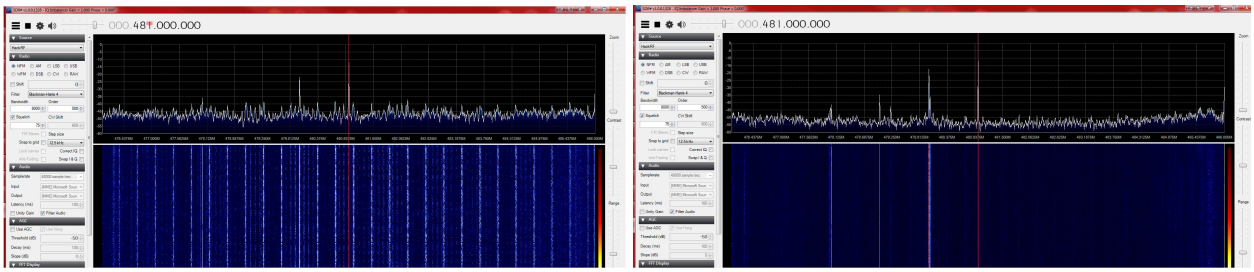
NINETEEN

USB CABLES

The USB cable you choose can make a big difference in what you see when using your HackRF and especially when using it around between 120 and 480 MHz where USB is doing all its work.

1. Use a shielded USB cable. The best way to guarantee RF interference from USB is to use an unshielded cable. You can test that your cable is shielded by using a continuity tester to verify that the shield on one connector has continuity to the shield on the connector at the other end of the cable.
2. Use a short USB cable. Trying anything larger than a 6ft cable may yield poor results. The longer the cable, the more loss you can expect and when making this post a 15ft cable was tried and the result was the HackRF would only power up half way.
3. For best results, select a cable with a ferrite core. These cables are usually advertised to be noise reducing and are recognizable from the plastic block towards one end.

Screenshot before and after changing to a noise reducing cable (view full size image):



A shielded cable with ferrite core was used in the right-hand image.

The before and after images were both taken with the preamp on and the LNA and VGA both set to 24db.

19.1 Why isn't my HackRF detectable after I plug it into my computer?

If your HackRF isn't immediately detectable it is very possible that your USB cable is not meeting HackRF's requirements. HackRF requires quite a bit of supply current and solid USB 2.0 high speed communications to operate. It is common for HackRF to reveal cables with deficiencies such as carrying power but not data, carrying data but not enough power, etc. Please try multiple cables to resolve this issue. More than once people have gotten their HackRF to work after trying their fifth cable.

CHAPTER

TWENTY

RF SHIELD INSTALLATION INSTRUCTIONS

HackRF Pro ships with an RF shield as standard.

Official Great Scott Gadgets HackRF Ones do not come from the factory with an RF shield installed around the radio section of the PCB. They do, however, have pads in place so that one may be installed if a user has a reason and an inclination to do so. The reason that they do not come preinstalled is that early testing revealed that the RF shield did little to improve the performance of the HackRF One. The recommended RF shield is the BMI-S-230-F-R (frame) with the BMI-S-230-C (shield). A two part RF shield is recommended because the shield section can be removed to allow access to the RF section of the HackRF One. This can be important if it becomes necessary to probe any part of the RF section, or to replace any parts of the RF section. However, even with a two part RF shield, it can be difficult to access the RF section of the HackRF One in certain situations. The following steps are a basic set of instructions for installing a RF shield on a HackRF One.

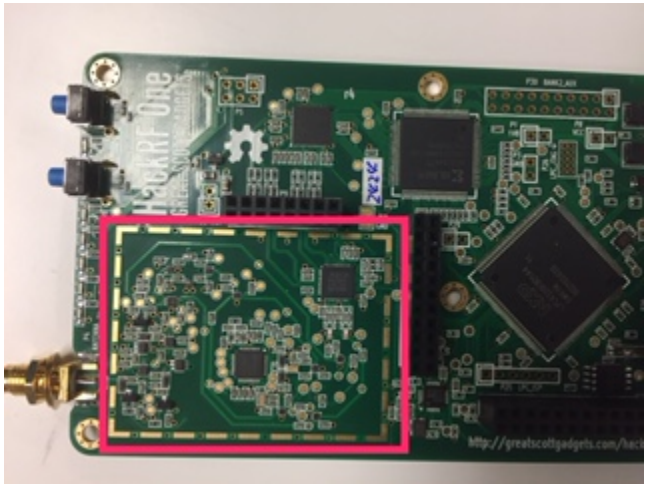
CAUTION: Soldering a RF shield onto a HackRF One comes with a certain amount of risk. Beyond the inherent risks of soldering itself, this process may damage the HackRF One and no warranty is available to cover damage incurred from this process. If you do choose to install a RF shield on your HackRF One please proceed with caution.

1. Remove the HackRF One from the injection molded plastic case.

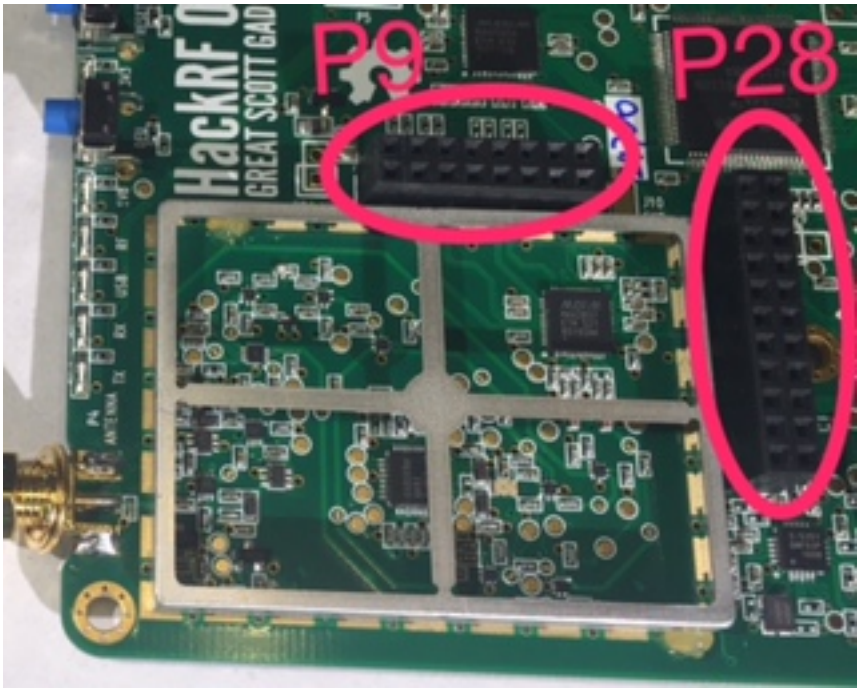
   BE WARNED: Opening the plastic case of your HackRF One will most likely destroy the tabs that hold it together.

   Instructions for removing a HackRF One from its case can be found *here*.

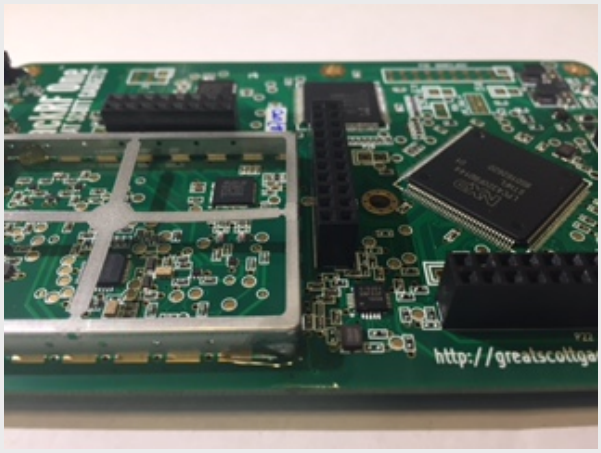
2. Prepare the HackRF One PCB for soldering on the shield frame, by adding flux to the RF shield pads around the radio section on the PCB.



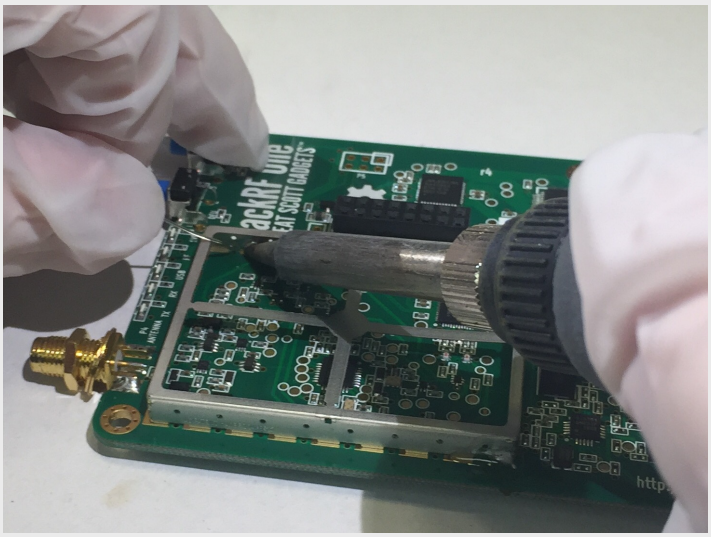
3. Place the RF shield frame on the HackRF One, aligning it so that it makes contact with all of the pads around the RF section of the board.

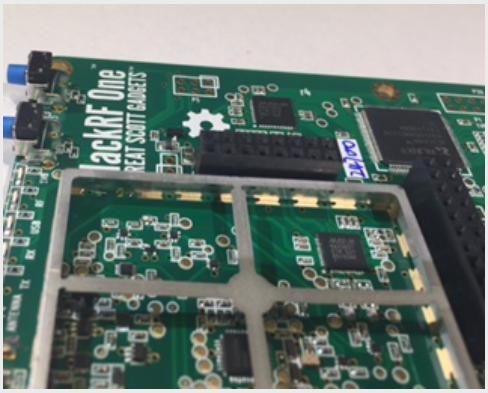


4. Solder the shield to one pad to anchor it to the H1. Visually inspect the frame to assure that it is still aligned properly.



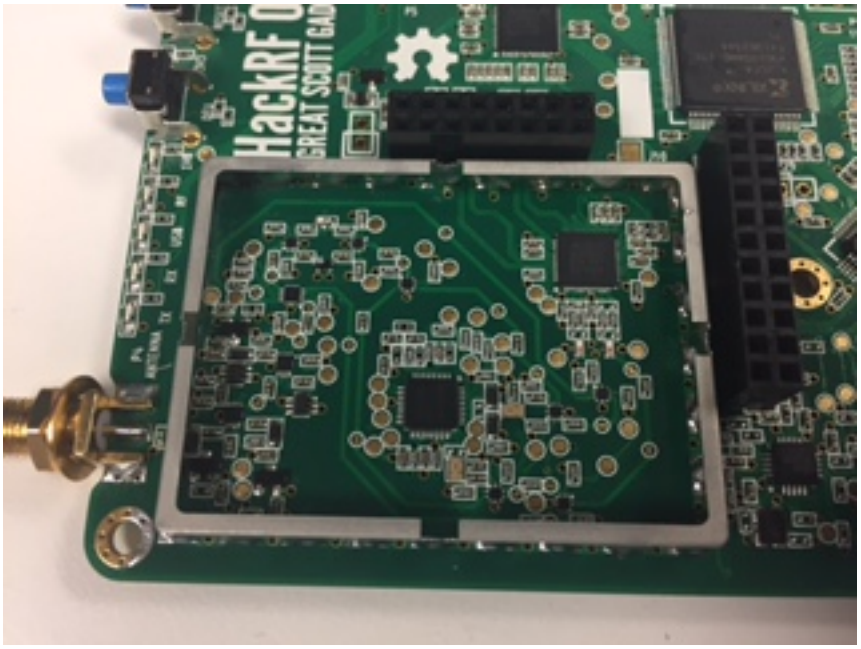
5. Connect the frame to another pad on the opposite side from the first connection. Again, check that the frame is still aligned properly.



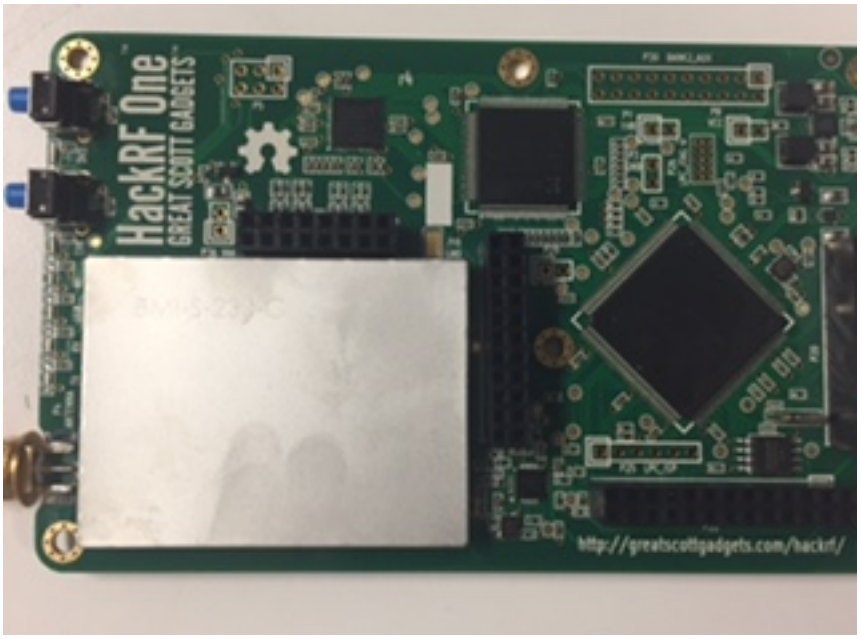


6. Connect at least one more pad, and then remove the pick and place bridge from the frame. *Removing the pick and place bridge is optional, but it is recommended.*

7. Continue soldering the rest of the pads to the frame.



8. Clean the flux and soldering residue with the appropriate solvent for the type of flux used. Be sure to let the HackRF One dry completely before plugging it in to a power source.
9. Place the RF shield onto the frame and snap it into place.



10. If desired, put the HackRF One back into the plastic case, if it is still able to click together.

CHAPTER

TWENTYONE

UPDATING FIRMWARE

HackRF devices ship with firmware on the SPI flash memory. The firmware can be updated with a USB cable and host computer.

These instructions allow you to upgrade the firmware in order to take advantage of new features or bug fixes.

21.1 Updating the SPI Flash Firmware

To update the firmware on HackRF Pro, use the hackrf_spiflash program:

```
hackrf_spiflash -w hackrf_pro_usb.bin
```

You can find the firmware binary (hackrf_pro_usb.bin) in the firmware-bin directory of the latest release package or you can compile your own from the source. For HackRF One or other platforms, use the ".bin" file with the appropriate name for your platform such as hackrf_one_usb.bin. If you compile from source, the file will be called hackrf_usb.bin.

The hackrf_spiflash program is part of hackrf-tools.

When writing a firmware image to SPI flash, be sure to select firmware with a filename ending in ".bin".

After writing the firmware to SPI flash, you may need to reset the HackRF device by pressing the RESET button or by unplugging it and plugging it back in.

If you get an error that mentions HACKRF_ERROR_NOT_FOUND, it is often a permissions problem on your OS.

21.2 Only if Necessary: Recovering the SPI Flash Firmware

If the firmware installed in SPI flash has been damaged or if you are programming a home-made HackRF for the first time, you will not be able to immediately use the hackrf_spiflash program as listed in the above procedure. Follow these steps instead:

1. Follow the DFU Boot instructions to start the HackRF in DFU boot mode.
2. Type `dfu-util --device 1fc9:000c --alt 0 --download hackrf_pro_usb.dfu` to load firmware from a release package into RAM. For HackRF One or other platforms, use the ".dfu" file with the appropriate name for your platform such as hackrf_one_usb.dfu.
3. Follow the SPI flash firmware update procedure above to write the ".bin" firmware image to SPI flash.

21.3 Only if Necessary: DFU Boot

DFU boot mode is normally only needed if the firmware is not working properly or has never been installed.

The LPC43xx microcontroller on HackRF is capable of booting from several different code sources. By default, HackRF boots from SPI flash memory (SPIFI). It can also boot HackRF in DFU (USB) boot mode. In DFU boot mode, HackRF will enumerate over USB, wait for code to be delivered using the DFU (Device Firmware Update) standard over USB, and then execute that code from RAM. The SPIFI is normally unused and unaltered in DFU mode.

To start up HackRF Pro or HackRF One in DFU mode, hold down the DFU button while powering it on or while pressing and releasing the RESET button. Then release the DFU button. On HackRF One, the 3V3 LED should illuminate. On HackRF Pro, all LEDs should remain off. At this point the HackRF is ready to receive firmware over USB.

To start up Jawbreaker in DFU mode, short two pins on one of the "BOOT" headers while power is first supplied. The pins that must be shorted are pins 1 and 2 of header P32 on Jawbreaker. Header P32 is labeled "P2_8" on most Jawbreakers but may be labeled "2" on prototype units. Pin 1 is labeled "VCC". Pin 2 is the center pin. After DFU boot, you should see VCCLED illuminate and note that 1V8LED does not illuminate. At this point Jawbreaker is ready to receive firmware over USB.

You should only use a firmware image with a filename ending in ".dfu" over DFU, not firmware ending in ".bin".

21.4 Obtaining DFU-Util

On fresh installs of your OS, you may need obtain a copy of DFU-Util. For most Linux distributions it should be available as a package, for example on Debian/Ubuntu

```
sudo apt-get install dfu-util
```

If you are using a platform without a dfu-util package, build instruction can be found here on the dfu-util source forge build page.

```
cd ~
sudo apt-get build-dep dfu-util
sudo apt-get install libusb-1.0-0-dev
git clone git://git.code.sf.net/p/dfu-util/dfu-util
cd dfu-util
./autogen.sh
./configure
make
sudo make install
```

Now you will have the current version of DFU Util installed on your system.

21.5 Updating the CPLD

Older versions of HackRF firmware (prior to release 2021.03.1) require an additional step to program a bitstream into the CPLD.

To update the CPLD image, first update the SPI flash firmware, libhackrf, and hackrf-tools to the version you are installing. Then:

```
hackrf_cpldjtag -x firmware/cpld/sgpio_if/default.xsvf
```

After a few seconds, three LEDs should start blinking. This indicates that the CPLD has been programmed successfully. Reset the HackRF device by pressing the RESET button or by unplugging it and plugging it back in.

CHAPTER

TWENTYTWO

FIRMWARE DEVELOPMENT SETUP

Firmware build instructions are included in the repository under firmware/README:

https://github.com/greatscottgadgets/hackrf/blob/master/firmware/README

CHAPTER

TWENTYTHREE

LPC43XX DEBUGGING

Various debugger options for the LPC43xx exist.

23.1 Black Magic Probe

https://github.com/blacksphere/blackmagic

An example of using gdb with the Black Magic Probe:

```
arm-none-eabi-gdb -n blinky.elf
target extended-remote /dev/ttyACM0
monitor swdp_scan
attach 1
set {int}0x40043100 = 0x10000000
load
cont
```

It is possible to attach to the M0 instead of the M4 if you use jtag_scan instead of swdp_scan, but the Black Magic Probe had some bugs when trying to work with the M0 the last time I tried it.

23.2 LPC-Link

(included with LPCXpresso boards)

TitanMKD has had some success. See the tutorial in hackrf/doc/LPCXPresso_Flash_Debug_Tutorial.pdf or .odt (PDF and OpenOffice document) Doc Link [https://github.com/mossmann/hackrf/tree/master/doc)

23.3 ST-LINK/V2

23.3.1 Hardware Configuration

Start with an STM32F4-Discovery board. Remove the jumpers from CN3. Connect the target's SWD interface to CN2 "SWD" connector.

23.3.2 Software Configuration

I'm using libusb-1.0.9.

Install OpenOCD-0.6.0 dev

```
# Cloned at hash a21affa42906f55311ec047782a427fcbcb98994
git clone git://openocd.git.sourceforge.net/gitroot/openocd/openocd
cd openocd
./bootstrap
./configure --enable-stlink --enable-buspirate --enable-jlink --enable-maintainer-mode
make
sudo make install
```

OpenOCD configuration files

openocd.cfg

```
#debug_level 3
source [find interface/stlink-v2.cfg]
source ./lpc4350.cfg
```

lpc4350.cfg

```
set _CHIPNAME lpc4350
set _M0_CPUTAPID 0x4ba00477
set _M4_SWDTAPID 0x2ba01477
set _M0_TAPID 0x0BA01477
set _TRANSPORT stlink_swd

transport select $_TRANSPORT

stlink newtap $_CHIPNAME m4 -expected-id $_M4_SWDTAPID
stlink newtap $_CHIPNAME m0 -expected-id $_M0_TAPID

target create $_CHIPNAME.m4 stm32_stlink -chain-position $_CHIPNAME.m4
#target create $_CHIPNAME.m0 stm32_stlink -chain-position $_CHIPNAME.m0
```

target.xml, nabbed from an OpenOCD mailing list thread, to fix a communication problem between GDB and newer OpenOCD builds.

```
<?xml version="1.0"?>
<!DOCTYPE target SYSTEM "gdb-target.dtd">
<target>
  <feature name="org.gnu.gdb.arm.core">
    <reg name="r0" bitsize="32" type="uint32"/>
    <reg name="r1" bitsize="32" type="uint32"/>
    <reg name="r2" bitsize="32" type="uint32"/>
    <reg name="r3" bitsize="32" type="uint32"/>
    <reg name="r4" bitsize="32" type="uint32"/>
    <reg name="r5" bitsize="32" type="uint32"/>
    <reg name="r6" bitsize="32" type="uint32"/>
```

(continues on next page)

(continued from previous page)

```
    <reg name="r7" bitsize="32" type="uint32"/>
    <reg name="r8" bitsize="32" type="uint32"/>
    <reg name="r9" bitsize="32" type="uint32"/>
    <reg name="r10" bitsize="32" type="uint32"/>
    <reg name="r11" bitsize="32" type="uint32"/>
    <reg name="r12" bitsize="32" type="uint32"/>
    <reg name="sp" bitsize="32" type="data_ptr"/>
    <reg name="lr" bitsize="32"/>
    <reg name="pc" bitsize="32" type="code_ptr"/>
    <reg name="cpsr" bitsize="32" regnum="25"/>
  </feature>
  <feature name="org.gnu.gdb.arm.fpa">
    <reg name="f0" bitsize="96" type="arm_fpa_ext" regnum="16"/>
    <reg name="f1" bitsize="96" type="arm_fpa_ext"/>
    <reg name="f2" bitsize="96" type="arm_fpa_ext"/>
    <reg name="f3" bitsize="96" type="arm_fpa_ext"/>
    <reg name="f4" bitsize="96" type="arm_fpa_ext"/>
    <reg name="f5" bitsize="96" type="arm_fpa_ext"/>
    <reg name="f6" bitsize="96" type="arm_fpa_ext"/>
    <reg name="f7" bitsize="96" type="arm_fpa_ext"/>
    <reg name="fps" bitsize="32"/>
  </feature>
</target>
```

23.4 Run ARM GDB

Soon, I should dump this stuff into a .gdbinit file.

```
arm-none-eabi-gdb -n
target extended-remote localhost:3333
set tdesc filename target.xml
monitor reset init
monitor mww 0x40043100 0x10000000
monitor mdw 0x40043100   # Verify 0x0 shadow register is set properly.
file lpc4350-test.axf    # This is an ELF file.
load                     # Place image into RAM.
monitor reset init
break main               # Set a breakpoint.
continue                 # Run to breakpoint.
continue                 # To continue from the breakpoint.
step                     # Step-execute the next source line.
stepi                    # Step-execute the next processor instruction.
info reg                 # Show processor registers.
```

More GDB tips for the GDB-unfamiliar:

```
# Write the variable "buffer" (an array) to file "buffer.u8".
dump binary value buffer.u8 buffer

# Display the first 32 values in buffer whenever you halt
```

(continues on next page)

(continued from previous page)

```
# execution.
display/32xh buffer

# Print the contents of a range of registers (in this case the
# CGU peripheral, starting at 0x40050014, for 46 words):
x/46 0x40050014
```

And still more, for debugging ARM Cortex-M4 Hard Faults:

```
# Assuming you have a hard-fault handler wired in:
display/8xw args

# Examine fault-related registers:

# Configurable Fault Status Register (CFSR) contains:
# CFSR[15:8]: BusFault Status Register (BFSR)
#   "Shows the status of bus errors resulting from instruction
#   prefetches and data accesses."
#   BFSR[7]: BFARVALID: BFSR contents valid.
#   BFSR[5]: LSPERR: fault during FP lazy state preservation.
#   BFSR[4]: STKERR: derived bus fault on exception entry.
#   BFSR[3]: UNSTKERR: derived bus fault on exception return.
#   BFSR[2]: IMPRECISERR: imprecise data access error.
#   BFSR[1]: PRECISERR: precise data access error, faulting
#            address in BFAR.
#   BFSR[0]: IBUSERR: bus fault on instruction prefetch. Occurs
#            only if instruction is issued.
display/xw 0xE000ED28

# BusFault Address Register (BFAR)
# "Shows the address associated with a precise data access fault."
# "This is the location addressed by an attempted data access that
# was faulted. The BFSR shows the reason for the fault and whether
# BFAR_ADDRESS is valid..."
# "For unaligned access faults, the value returned is the address
# requested by the instruction. This might not be the address that
# faulted."
display/xw 0xE000ED38
```

CHAPTER

TWENTYFOUR

LPC43XX SGPIO CONFIGURATION

The LPC43xx SGPIO peripheral is used to move samples between USB and the ADC/DAC chip (MAX5864). The SGPIO is a peripheral that has a bunch of 32-bit shift registers. These shift registers can be configured to act as a parallel interface of different widths. For HackRF, we configure the SGPIO to transfer eight bits at a time. The SGPIO interface can also accept an external clock, which we use to synchronize transfers with the sample clock.

In the current HackRF design, there is a CPLD which manages the interface between the MAX5864 and the SGPIO interface. There are four SGPIO signals that control the SGPIO data transfer:

- Clock: Determines when a value on the SGPIO data bus is transferred.
- Direction: Determines whether the MAX5864 DA (ADC) data is driven onto the SGPIO lines, or if the SGPIO lines drive the data bus with data for the MAX5864 DD (DAC) signals.
- Data Valid: Indicates a sample on the SGPIO data bus is valid data.
- Transfer Enable: Allows SGPIO to synchronize with the I/Q data stream. The MAX5864 produces/consumes two values (quadrature/complex value) per sample period – an I value and a Q value. These two values are multiplexed on the SGPIO lines. This signal suspends data valid until the I value should be transferred.

24.1 Why not use GPDMA to transfer samples through SGPIO?

It would be great if we could, as that would free up lots of processor time. Unfortunately, the GPDMA scheme in the LPC43xx does not seem to support peripheral-to-memory and memory-to-peripheral transfers with the SGPIO peripheral.

You might observe that the SGPIO peripheral can generate requests from SGPIO14 and SGPIO15, using an arbitrary bit pattern in the slice shift register. The pattern in the slice determines the request interval. That's a good start. However, how do you specify which SGPIO shadow registers are read/written at each request, and in which order those registers are transferred with memory? It turns out you can't. In fact, it appears that an SGPIO request doesn't cause any transfer at all, if your source or destination is "peripheral". Instead, the SGPIO request is intended to perform a memory-to-memory transfer synchronized with SGPIO. But you're on your own as far as getting data to/from the SGPIO shadow registers. I believe this is why the SGPIO camera example in the user manual describes an SGPIO interrupt doing the SGPIO shadow register transfer, and the GPDMA doing moves from one block of RAM to another.

Perhaps if we transfer only one SGPIO shadow register, using memory-to-memory? Then we don't have to worry about the order of SGPIO registers, or which ones need to be transferred. It turns out that when you switch over to memory-to-memory transfers, you lose peripheral request generation. So the GPDMA will transfer as fast as possible – far faster than words are produced/consumed by SGPIO.

I'd really love to be wrong about all this, but all my testing has indicated there's no workable solution to using GPDMA that's any better than using SGPIO interrupts to transfer samples. If you want some sample GPDMA code to experiment with, please contact Jared (sharebrained on #hackrf in Discord or IRC).

CHAPTER

TWENTYFIVE

INSTALLING HACKRF SOFTWARE

HackRF software includes HackRF Tools and libhackrf. HackRF Tools are the commandline utilities that let you interact with your HackRF. libhackrf is a low level library that enables software on your computer to operate with HackRF.

25.1 Install Using Package Managers

Unless developing or testing new features for HackRF, we highly recommend that most users use build systems or package managers provided for their operating system. **Our suggested operating system for use with HackRF is Ubuntu**.

25.1.1 FreeBSD

You can use the binary package: `# pkg install hackrf`

You can also build and install from ports:

```
# cd /usr/ports/comms/hackrf
# make install
```

25.1.2 Linux: Arch

```
pacman -S hackrf
```

25.1.3 Linux: Fedora / Red Hat

```
sudo dnf install hackrf -y
```

25.1.4 Linux: Gentoo

```
emerge -a net-wireless/hackrf-tools
```

25.1.5 Linux: Ubuntu / Debian

```
sudo apt-get install hackrf
```

25.1.6 OS X (10.5+): Homebrew

```
brew install hackrf
```

25.1.7 OS X (10.5+): MacPorts

```
sudo port install hackrf
```

25.1.8 Windows: Binaries

Windows users can use radioconda to get the required binaries installed.

Alternatively, binaries are available as build artifacts under the 'Actions'-tab on github here (GitHub Login needed).

25.2 Installing From Source

25.2.1 Linux / OS X / *BSD: Building HackRF Software From Source

Acquire the source for the HackRF tools from either a release archive or git: `git clone https://github.com/greatscottgadgets/hackrf.git`

Once you have the source downloaded, the host tools can be built as follows:

```
cd hackrf/host
mkdir build
cd build
cmake ..
make
sudo make install
sudo ldconfig
```

If you have HackRF hardware, you may need to *update the firmware* to match the host tools versions.

25.2.2 Windows: Building HackRF Software From Source

Install Visual Studio Community (2015 or later) and CMake (at least version 3.21.4).

Install library dependencies using vcpkg:

```
git clone https://github.com/microsoft/vcpkg
cd vcpkg
bootstrap-vcpkg.bat
vcpkg install libusb fftw3 pthreads pkgconf
```

Open the Visual Studio Developer Command Prompt, and change to the directory where you unpacked the HackRF source.

The following steps assume you installed vcpkg in `C:\vcpkg`.

Configure CMake and build the code:

```
set PKG_CONFIG=C:\vcpkg\installed\x64-windows\tools\pkgconf\pkgconf.exe
set PKG_CONFIG_PATH=C:\vcpkg\installed\x64-windows\lib\pkgconfig
set CMAKE_TOOLCHAIN_FILE=C:\vcpkg\scripts\buildsystems\vcpkg.cmake
cmake -B host\build host
cmake --build host\build
```

CMake will generate a `HackRF.sln` project file which you can open in Visual Studio for editing and development.

CHAPTER

TWENTYSIX

HACKRF TOOLS

Great Scott Gadgets provides some commandline tools for interacting with HackRF.

- **hackrf_info** Read device information from HackRF such as serial number and firmware version.
- **hackrf_transfer** Send and receive signals using HackRF. Input/output files are 8-bit signed quadrature samples.
- **hackrf_sweep**, a command-line spectrum analyzer.
- **hackrf_clock** Read and write clock input and output configuration.
- **hackrf_operacake** Configure Opera Cake antenna switch connected to HackRF.
- **hackrf_spiflash** A tool to write new firmware to HackRF. See: *Updating Firmware*.
- **hackrf_debug** Read and write registers and other low-level configuration for debugging.

26.1 hackrf_sweep

26.1.1 Usage

```
[-h] # this help
[-d serial_number] # Serial number of desired HackRF
[-a amp_enable] # RX RF amplifier 1=Enable, 0=Disable
[-f freq_min:freq_max] # minimum and maximum frequencies in MHz
[-p antenna_enable] # Antenna port power, 1=Enable, 0=Disable
[-l gain_db] # RX LNA (IF) gain, 0-40dB, 8dB steps
[-g gain_db] # RX VGA (baseband) gain, 0-62dB, 2dB steps
[-w bin_width] # FFT bin width (frequency resolution) in Hz, 2445-5000000
[-1] # one shot mode
[-N num_sweeps] # Number of sweeps to perform
[-B] # binary output
[-I] # binary inverse FFT output
-r filename # output file
```

26.1.2 Output fields

`date, time, hz_low, hz_high, hz_bin_width, num_samples, dB, dB, ...`

Running `hackrf_sweep -f 2400:2490` gives the following example results:

Date	Time	Hz Low	Hz High	Hz bin width	Num Samples	dB	dB	dB	dB	dB
2019-01-03	11:57:34.	2400000(	2405000(	1000000.	20	-64.72	-63.36	-60.91	-61.74	-58.58
2019-01-03	11:57:34.	2410000(	2415000(	1000000.	20	-69.22	-60.67	-59.50	-61.81	-58.16
2019-01-03	11:57:34.	2405000(	2410000(	1000000.	20	-61.19	-70.14	-60.10	-57.91	-61.97
2019-01-03	11:57:34.	2415000(	2420000(	1000000.	20	-72.93	-79.14	-68.79	-70.71	-82.78
2019-01-03	11:57:34.	2420000(	2425000(	1000000.	20	-67.57	-61.61	-57.29	-61.90	-70.19
2019-01-03	11:57:34.	2430000(	2435000(	1000000.	20	-56.04	-59.58	-66.24	-66.02	-62.12

Each sweep across the entire specified frequency range is given a single time stamp.

The fifth column tells you the width in Hz (1 MHz in this case) of each frequency bin, which you can set with `-w`. The sixth column is the number of samples analyzed to produce that row of data.

Each of the remaining columns shows the power detected in each of several frequency bins. In this case there are five bins, the first from 2400 to 2401 MHz, the second from 2401 to 2402 MHz, and so forth.

CHAPTER

TWENTYSEVEN

THIRD-PARTY SOFTWARE COMPATIBLE WITH HACKRF

27.1 Software That Has Direct Support For HackRF

- GQRX
  - http://gqrx.dk/
- GNU Radio
  - https://www.gnuradio.org/
- GNU Radio Mode-S/ADS-B
  - https://github.com/bistromath/gr-air-modes
- QSpectrumAnalyzer
  - https://github.com/xmikos/qspectrumanalyzer
- SDR#
  - https://airspy.com/download/
  - Windows OS only
  - Only nightly builds currently support HackRF One
- SDR Console
  - https://www.sdr-radio.com/Console
- Spectrum Analyzer GUI for hackrf_sweep for Windows
  - https://github.com/pavsa/hackrf-spectrum-analyzer
- Universal Radio Hacker (Windows/Linux)
  - https://github.com/jopohl/urh
- Web-based APRS tracker
  - https://xakcop.com/aprs-sdr
- SigDigger (Windows/Linux/macOS)
  - https://github.com/BatchDrake/SigDigger
  - Supports HackRF through SoapySDR / SoapyHackRF

27.2 Software That Can Use Data From HackRF

- Baudline
  - http://www.baudline.com/
  - Can view/process HackRF data, e.g. hackrf_transfer
- Inspectrum
  - https://github.com/miek/inspectrum
  - Capture analysis tool with advanced features
- Matlab

```
fid = open('samples.bin',  'r');
len = 1000; % 1000 samples
y = fread(fid, 2*len, 'int8');
y = y(1:2:end) + 1j*y(2:2:end);
fclose(fid)
```

27.3 Troubleshooting Recommendations

Many of these tools require libhackrf and at times HackRF Tools. It may help you to have updated libhackrf and HackRF Tools when troubleshooting these applications.

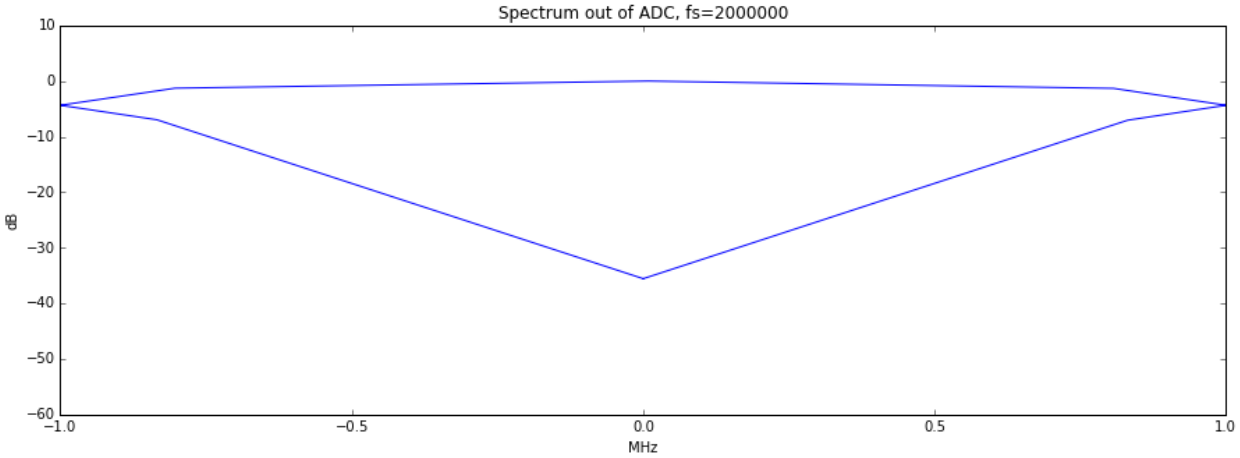
It is also strongly suggested, and usually required, that your HackRF Tools and HackRF firmware match.

CHAPTER

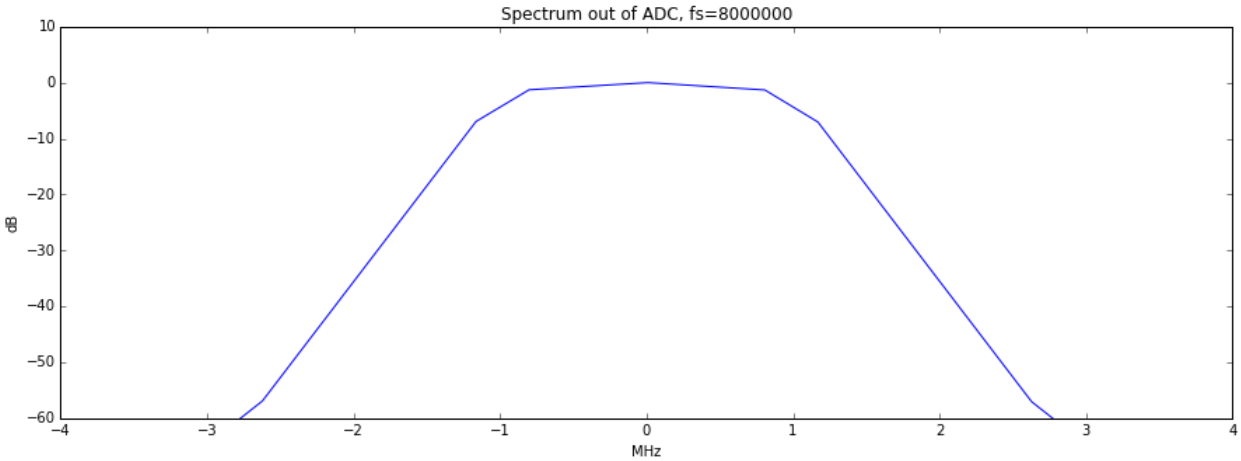
TWENTYEIGHT

SAMPLING RATE AND BASEBAND FILTERS

Using a sampling rate of less than 8MHz is not recommended. Partly, this is because the MAX5864 (ADC/DAC chip) isn't specified to operate at less than 8MHz, and therefore, no promises are made by Maxim about how it performs. But more importantly, the baseband filter in the MAX2837 has a minimum bandwidth of 1.75MHz. It can't provide enough filtering at 2MHz sampling rate to remove substantial signal energy in adjacent spectrum (more than +/-1MHz from the tuned frequency). The MAX2837 datasheet suggests that at +/-1MHz, the filter provides only 4dB attenuation, and at +/-2MHz (where a signal would alias right into the center of your 2MHz spectrum), it attenuates about 33dB. That's significant. Here's a picture:



At 8MHz sampling rate, and using the minimum 1.75MHz bandwidth filter, this is the response:



You can see that the attenuation is more than 60dB at +/-2.8MHz, which is more than sufficient to remove significant

adjacent spectrum interference before the ADC digitizes the baseband. If using this configuration to get a 2MHz sampling rate, use a GNU Radio block after the 8MHz source that performs a 4:1 decimation with a decently sharp low pass filter (complex filter with a cut-off of <1MHz).

CHAPTER

TWENTYNINE

SETTING GAIN CONTROLS FOR RX

29.1 Gain controls

HackRF One provides three different analog gain controls on RX and two on TX.

The three RX gain controls are at these stages:

- RF ("amp", 0 or ~11 dB)
- IF ("lna", 0 to 40 dB in 8 dB steps)
- baseband ("vga", 0 to 62 dB in 2 dB steps)

The two TX gain controls are at these stages:

- RF (0 or ~11 dB)
- IF (0 to 47 dB in 1 dB steps)

Note: in some documents, the RF gain was erroneously quoted to be 14 dB. The confusion was based on the fact that the MGA-81563 amplifier is advertised as a "14 dBm" amplifier, but that specifies its output power, not its amplification. See Martin Ling's comment on issue #1059 for some details!

The TX and RX RF amplifiers have two settings: on or off. In the off state, the amps are completely bypassed. They nominally provide around 11 dB of gain when on, but the actual amount of gain varies by frequency. In general, expect less gain at higher frequencies. For fine control of gain, use the IF and/or baseband gain options.

A good default setting to start with is RF=0 (off), IF=16, baseband=16. Increase or decrease the IF and baseband gain controls roughly equally to find the best settings for your situation. Turn on the RF amp if you need help picking up weak signals. If your gain settings are too low, your signal may be buried in the noise. If one or more of your gain settings is too high, you may see distortion (look for unexpected frequencies that pop up when you increase the gain) or the noise floor may be amplified more than your signal is.

CHAPTER

THIRTY

VIRTUAL MACHINES

HackRF requires the ability to stream data at very high rates over USB. Unfortunately VM software typically has problems with USB passthrough; especially continuous high speed USB transfers. It is recommended to not use a HackRF with a VM.

CHAPTER

THIRTYONE

GATEWARE

One of the significant hardware changes in *HackRF Pro* is the replacement of the CPLD with a FPGA. While the older CPLD primarily provided glue logic between the MCU and RF front end, the FPGA in HackRF Pro introduces more logic and DSP capability. This enables offloading digital signal processing tasks from the MCU.

FPGAs are highly flexible devices whose behavior is defined by *gateware*: hardware descriptions that configure the internal logic fabric. HackRF Pro gateware is written in Amaranth HDL, a Python-based hardware description language.

The specific FPGA device used in HackRF Pro is the Lattice iCE40UP5K, which features 5280 LUT4s and 8 dedicated DSP (multiply-accumulate) blocks. We rely on the open-source iCE40 FPGA toolchain to build the required bitstreams that are bundled in the firmware.

All gateware source code lives under *firmware/fpga/* in the HackRF repository. Top-level designs reside in *firmware/fpga/top/* and are the primary entry points for different operational modes.

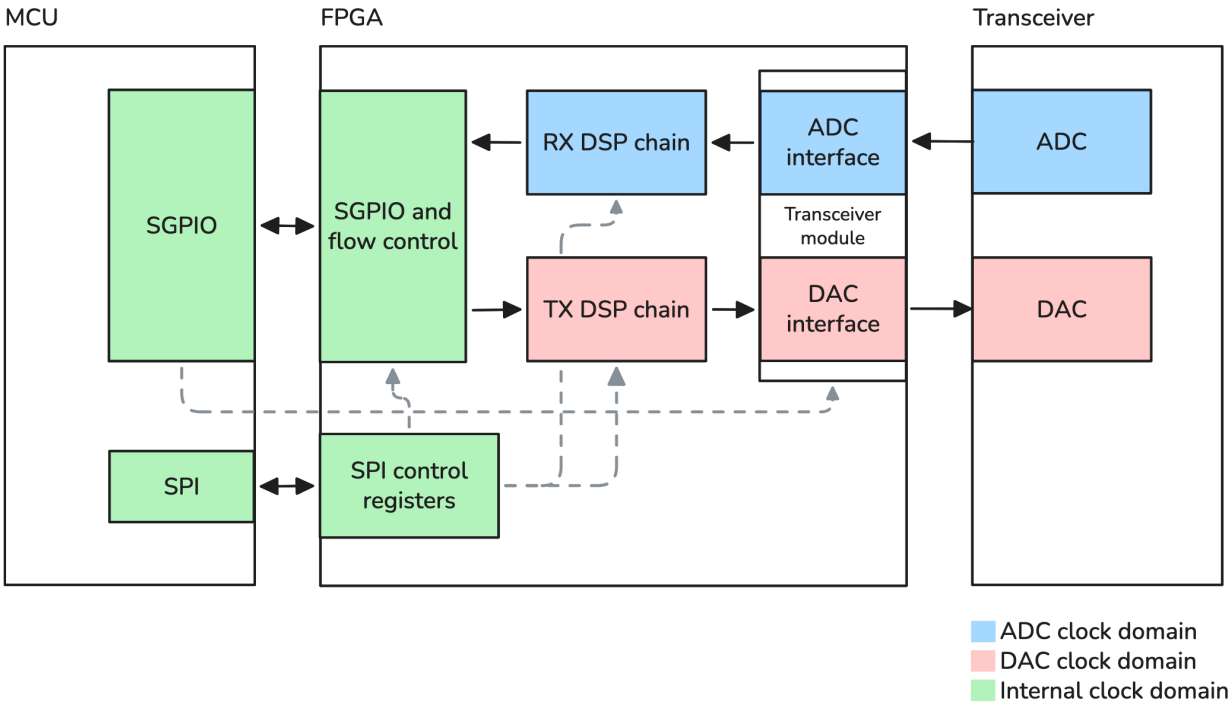
By default, a standard gateware configuration is loaded at boot. However, the firmware can dynamically reconfigure the FPGA at runtime to switch between different gateware variants.

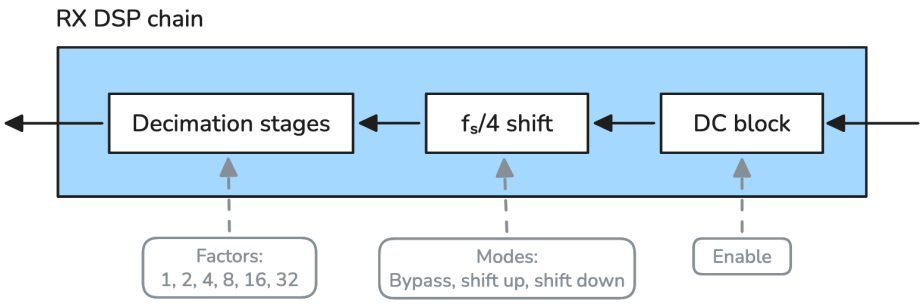
31.1 Standard gateware

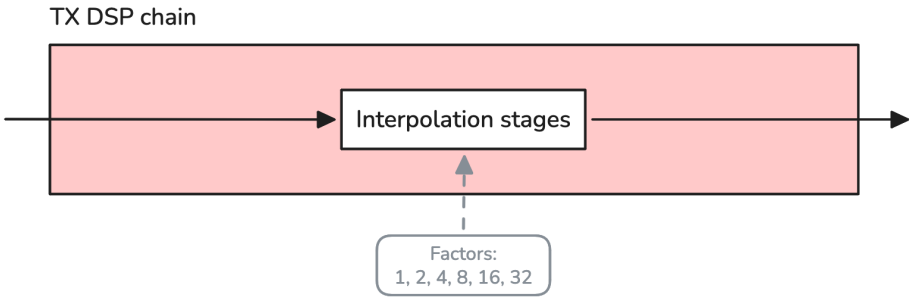
The standard gateware is used by default when the firmware has not requested an alternative bitstream.

The standard gateware provides a balanced configuration optimized for general-purpose operation. It implements configurable digital signal processing paths for the reception and transmission paths, capable of (limited) frequency translation and supporting a wide range of sample rates.

31.1.1 Block diagram







31.1.2 Features

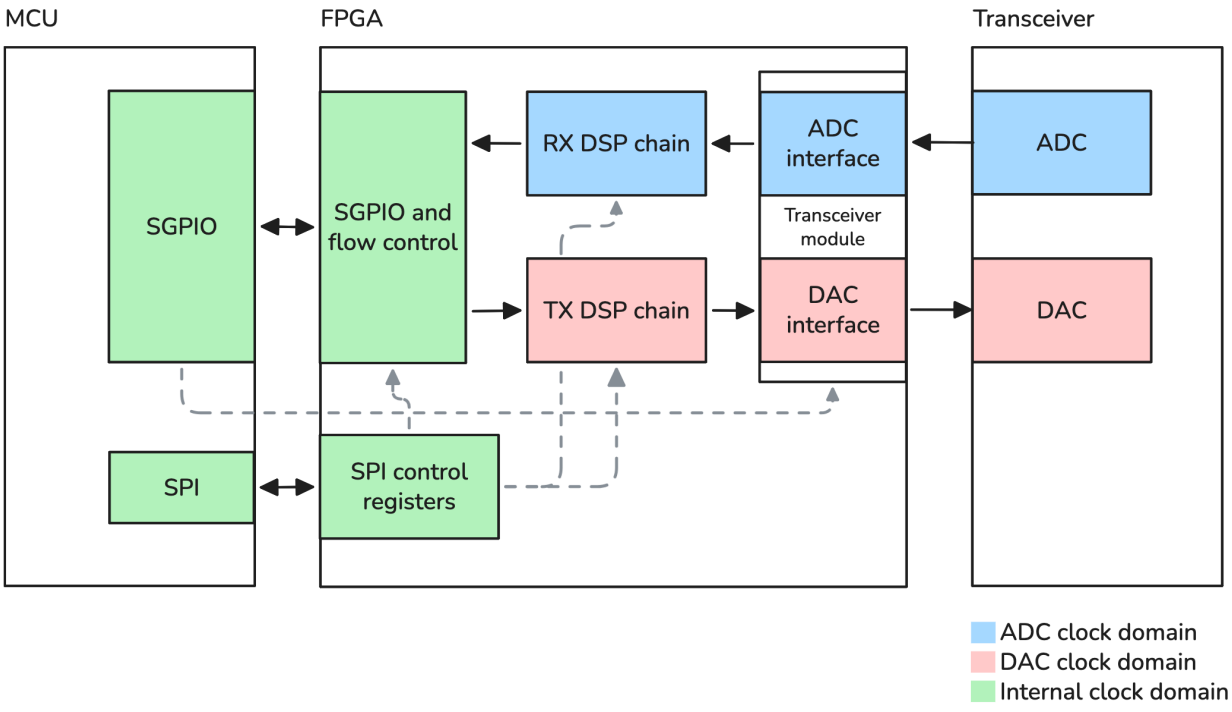
- 8-bit I, 8-bit Q data format
- **Receiver signal chain:**
  - Optional DC offset removal (DC blocker)
  - Configurable fs/4 shifter (quarter sample rate): bypass, shift up or shift down
  - Configurable decimation rates: 1x, 2x, 4x, 8x, 16x, 32x
- **Transmitter signal chain:**
  - Configurable interpolation rates: 1x, 2x, 4x, 8x, 16x, 32x
- SPI control interface for register configuration
- Double data rate (DDR) interface to RF transceiver
- Interface to MCU (SGPIO)

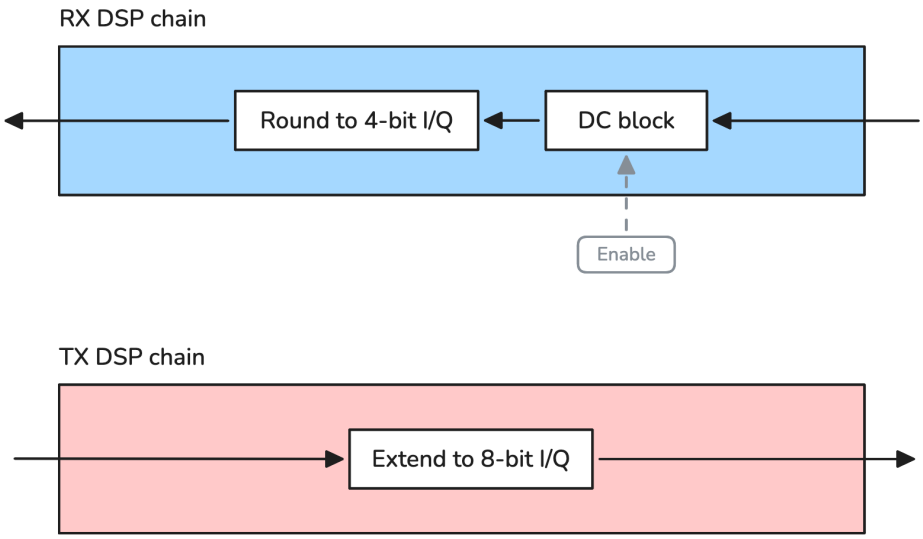
31.2 Half-precision gateware

The half-precision gateware reduces sample width to 4 bits per I/Q component, enabling higher throughput within the constraints of the USB interface (up to 40 Msps).

This configuration is intended for applications where bandwidth is more critical than dynamic range, such as wideband spectrum monitoring.

31.2.1 Block diagram





31.2.2 Features

- 4-bit I, 4-bit Q data format
- **Receiver signal chain:**
  - Optional DC offset removal (DC blocker)
  - Round to 4-bit I/Q
- **Transmitter signal chain:**
  - Extend width to 8-bit I/Q
- SPI control interface for register configuration
- Double data rate (DDR) interface to RF transceiver
- Interface to MCU (SGPIO)

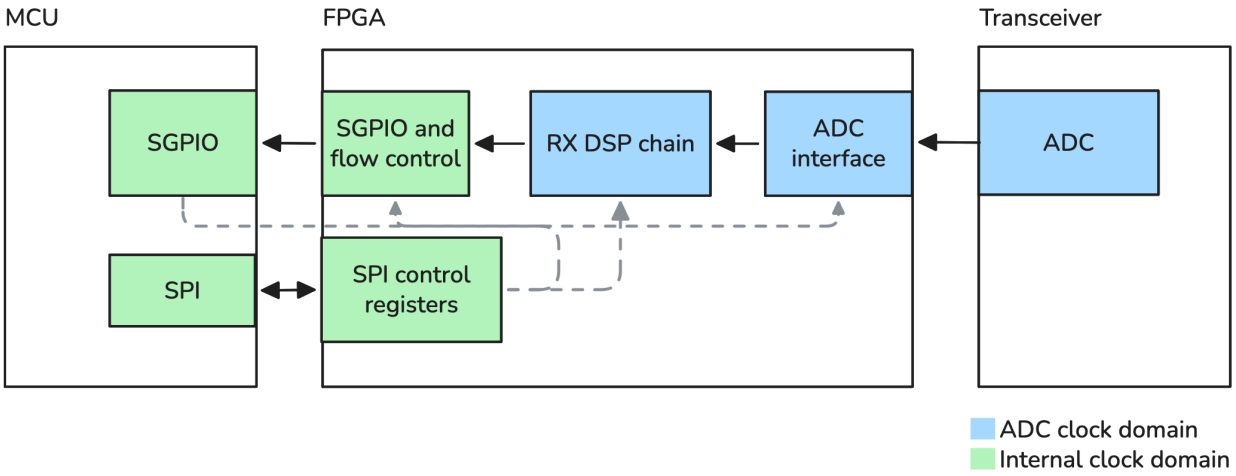
31.3 Extended-precision gateware (RX and TX)

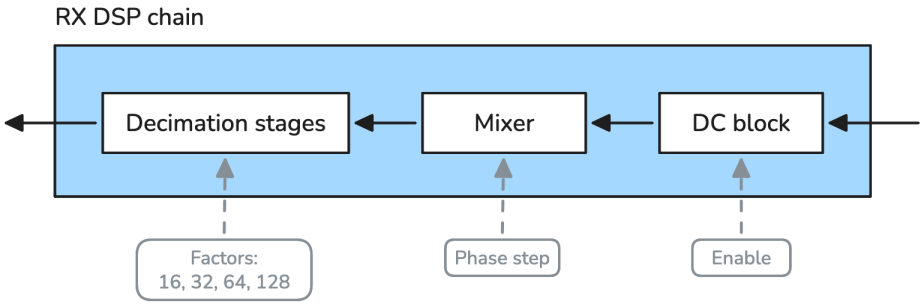
The extended-precision gateware increases internal signal processing precision and output sample width to improve signal quality. The main drawback is that the minimum decimation or interpolation factor is 16x. Due to increased logic requirements, this gateware is split in two top-level designs (RX and TX).

Samples are 16-bit I/Q, while the effective number of bits (ENOB) depends on the selected configuration and typically ranges between 9 and 11 bits.

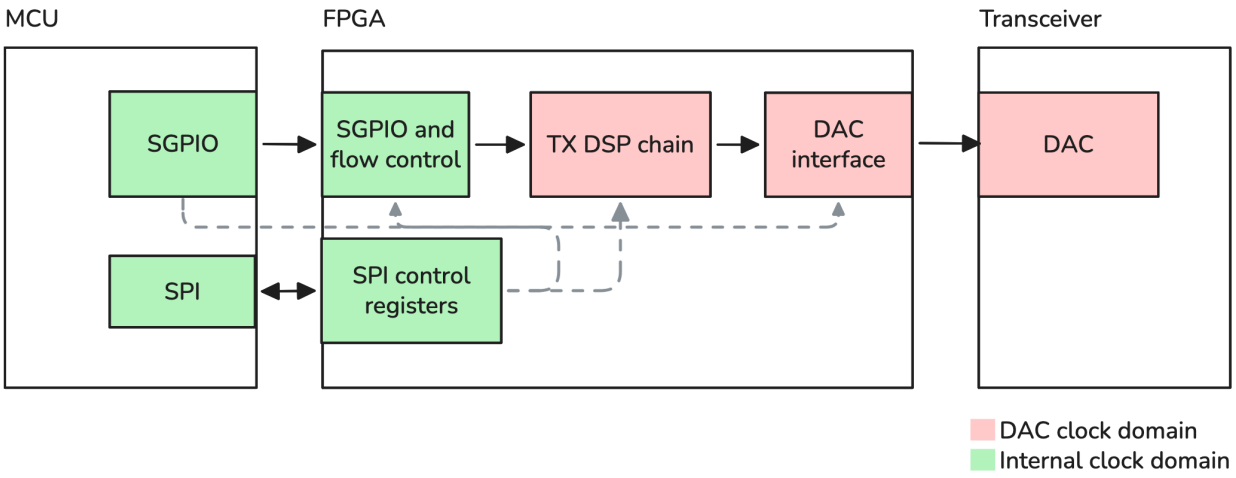
The increased dynamic range of the output makes it particularly useful for weak and/or narrowband signals.

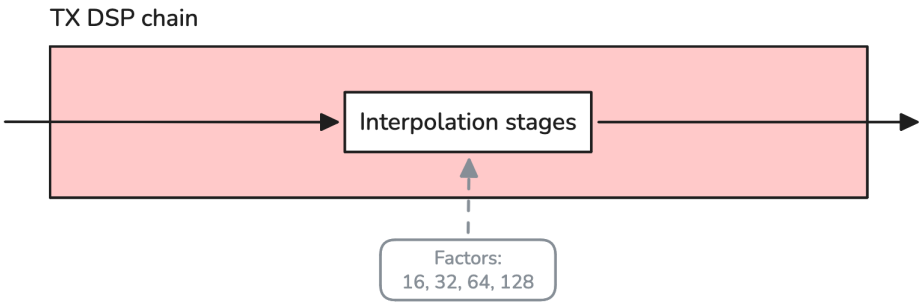
31.3.1 Block diagram (RX)





31.3.2 Block diagram (TX)





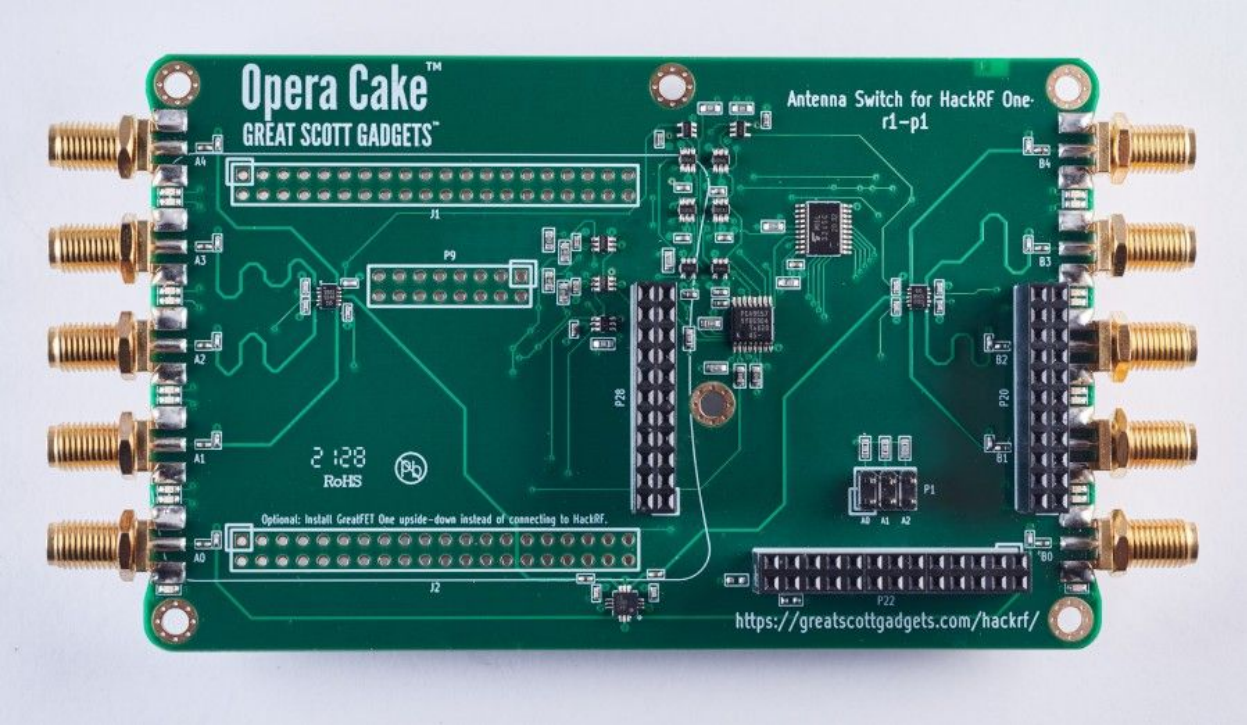
31.3.3 Features

- 16-bit I, 16-bit Q data format
- **Receiver signal chain (RX extended-precision gateware):**
  - Optional DC offset removal (DC blocker)
  - Configurable mixer (in fs/128 steps)
  - Configurable decimation rates: 16x, 32x, 64x, 128x
- **Transmitter signal chain (TX extended-precision gateware):**
  - Configurable interpolation rates: 16x, 32x, 64x, 128x
- SPI control interface for register configuration
- Double data rate (DDR) interface to RF transceiver
- Interface to MCU (SGPIO)

CHAPTER

THIRTYTWO

OPERA CAKE



Opera Cake is an antenna switching add-on board for HackRF One. Consisting of two 1x4 switches, Opera Cake also has a cross-over switch that permits operation as a 1x8 switch. Up to eight Opera Cakes may be stacked onto a single HackRF One provided that each Opera Cake is configured with a different board address.

Opera Cake can be used as a 1x8 switch to connect your HackRF One to a variety of antennas at once, such as a long wire antenna for HF bands, a discone for VHF and UHF, a dipole for 2.4 GHz, and a dish for a satellite band. Once connected to your Opera Cake you can switch between all of your antennas in software instead of making physical hardware swaps.

When set up as a pair of 1x4 switches you could use Opera Cake as a switched filter bank. To do this, connect port A1 to B1, A2 to B2, A3 to B3, and A4 to B4 through physical SMA filters and cables of your choosing. This setup allows you to change your transmit or receive to be through the filter of your choosing without having to reconnect hardware every time you would like to use a different filter.

Opera Cake is configured with the `hackrf_operacake` command-line tool.

CHAPTER

THIRTYTHREE

FREQUENTLY ASKED QUESTIONS

33.1 Why the name 'Opera Cake'?

Internally at Great Scott Gadgets, HackRF related boards are code named with a candy or confection name. The Opera Cake code name was fun enough that we didn't change from it when we released the project.

33.2 When was Opera Cake first for sale?

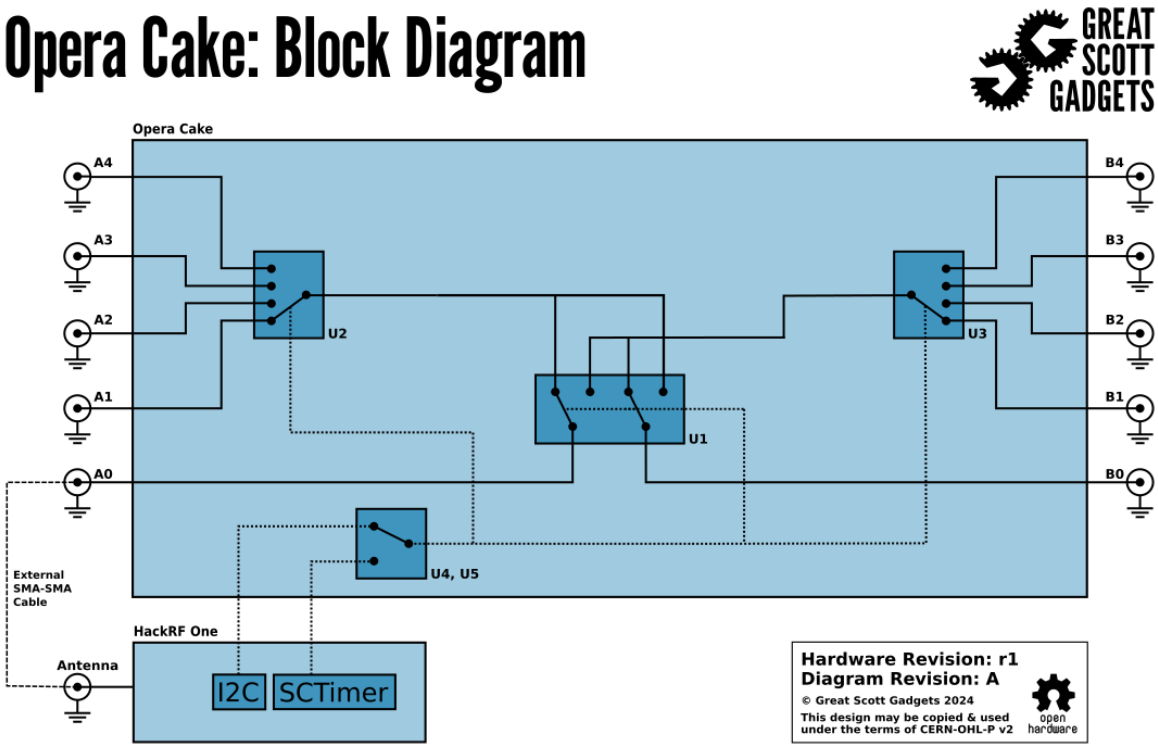
Great Scott Gadgets first released Opera Cake for sale in 2022. The open source project for Opera Cake has been available since 2016 from Great Scott Gadgets.

CHAPTER

THIRTYFOUR

HARDWARE

34.1 Block Diagram



34.2 Banks

The ports on Opera Cake are grouped in two banks (or "sides"), one on each end of the board. Bank A consists of ports A0 through A4 while bank B consists of ports B0 through B4.

34.3 Ports

Opera Cake has two primary ports, A0 and B0, each of which can be switched to any of eight secondary ports, A1-A4 and B1-B4. Each primary port is always connected to one secondary port. By default, A0 is connected to A1, and B0 is connected to B1. It is not possible to connect both primary ports to secondary ports in the same bank at the same time.

34.4 LEDs

Port selections are indicated by LEDs next to each port's connector. Port A0 and the secondary port connected to A0 are indicated with a green LED. Port B0 and the secondary port connected to B0 are indicated with a yellow LED.

CHAPTER

THIRTYFIVE

BOARD ADDRESSING

Each Opera Cake has a numeric address set by optional jumpers installed on header P1. The default address (without jumpers) is 0. The `--list` or `-l` option can be used to list the address(es) of one or more Opera Cakes installed on a HackRF One:

```
hackrf_operacake -l
```

The address may be set to any number from 0 to 7 by installing jumpers across the A0, A1, and/or A2 pins of header P1.

Address	A2 Jumper	A1 Jumper	A0 Jumper
0	No	No	No
1	No	No	Yes
2	No	Yes	No
3	No	Yes	Yes
4	Yes	No	No
5	Yes	No	Yes
6	Yes	Yes	No
7	Yes	Yes	Yes

When configuring an Opera Cake, the address may be specified with the `--address` or `-o` option:

```
hackrf_operacake -o 1 -a A1 -b B2
```

If the address is unspecified, 0 is assumed. It is only necessary to specify the address if the address has been changed with the addition of jumpers, typically required only if multiple Opera Cakes are stacked onto a single HackRF One.

CHAPTER

THIRTYSIX

PORT CONFIGURATIONS

Port connections may be configured manually. For example, to connect A0 to A2 and B0 to B3:

```
hackrf_operacake -a A2 -b B3
```

To connect A0 to B2 and B0 to A4:

```
hackrf_operacake -a B2 -b A4
```

If only one primary port is configured, the other primary port will be connected to the first secondary port on the opposite side. For example, after the next two commands B0 will be connected to A1:

```
hackrf_operacake -a A2 -b B3
hackrf_operacake -a B2
```

CHAPTER

THIRTYSEVEN

MODES OF OPERATION

Opera Cake supports three modes of operation: `manual`, `frequency`, and `time`. The operating mode can be set with the `--mode` or `-m` option, and the active operating mode is displayed with the `--list` or `-l` option.

37.1 Manual Mode

The default mode of operation is `manual`. In manual mode, fixed port connections are configured with the `-a` and `-b` options as in the *port configuration examples*. If the operating mode has been changed, it can be changed back to manual mode with:

```
hackrf_operacake -m manual
```

37.2 Frequency Mode

In frequency mode, the A0 port connection switches automatically whenever the HackRF is tuned to a different frequency. This is useful when antennas for different frequency bands are connected to various ports.

The bands are specified in priority order. The final band specified will be used for frequencies not covered by the other bands specified.

To assign frequency bands to ports you must use the `-f <port:min:max>` option for each band, with the minimum and maximum frequencies specified in MHz. For example, to use port A1 for 100 MHz to 600 MHz, A3 for 600 MHz to 1200 MHz, and B2 for 0 MHz to 4 GHz:

```
hackrf_operacake -m frequency -f A1:100:600 -f A3:600:1200 -f B2:0:4000
```

If tuning to precisely 600 MHz, A1 will be used as it is listed first. Tuning to any frequency over 4 GHz will use B2 as it is the last listed and therefore the default port.

Only the A0 port connection is specified in frequency mode. Whenever the A0 connection is switched, the B0 connection is also switched to the secondary port mirroring A0's secondary port. For example, when A0 switches to B2, B0 is switched to A2.

Once configured, an Opera Cake will remain in frequency mode until the mode is reconfigured or until the HackRF One is reset. You can pre-configure the Opera Cake in frequency mode, and the automatic switching will continue to work while using other software.

Although multiple Opera Cakes on a single HackRF One may be set to frequency mode at the same time, they share a single switching plan. This can be useful, for example, for a filter bank consisting of eight filters.

37.3 Time Mode

In time mode, the A0 port connection switches automatically over time, counted in units of the sample period. This is useful for experimentation with pseudo-doppler direction finding.

To cycle through four ports, one port every 1000 samples:

```
hackrf_operacake -m time -t A1:1000 -t A2:1000 -t A3:1000 -t A4:1000
```

When the duration on multiple ports is the same, the `-w` option can be used to set the default dwell time:

```
hackrf_operacake --mode time -w 1000 -t A1 -t A2 -t A3 -t A4
```

Only the A0 port connection is specified in time mode. Whenever the A0 connection is switched, the B0 connection is switched to the secondary port mirroring A0's secondary port. For example, when A0 switches to B2, B0 is switched to A2.

Once configured, an Opera Cake will remain in time mode until the mode is reconfigured or until the HackRF One is reset. You can pre-configure the Opera Cake in time mode, and the automatic switching will continue to work while using other software.

Although multiple Opera Cakes on a single HackRF One may be set to time mode at the same time, they share a single switching plan.