A 405-line television transmitter in GNU Radio

Project405, 18 October 2026, updated 29 October 2026. A teaching version of the HackRF transmitter: every stage visible and adjustable. For watching films and DVDs, use hacktv (see the HackRF report); this version is for seeing how a 405-line signal is built.

1. What it is

hacktv already turns a Raspberry Pi and a HackRF into a 405-line transmitter, but it's a closed box: picture in, radio out. This GNU Radio flowgraph does the same job **in plain view**. It builds a BBC **System A** signal out of a handful of standard blocks, so each step can be seen on screen and changed while it runs:

- **positive amplitude modulation** of the vision carrier (sync = no carrier, white = full carrier);
- the **vestigial sideband** filter that kept 405-line channels narrow;
- a separate **AM sound carrier** 3.5 MHz below vision, carrying a 1 kHz tone;
- the result sent on **channel B1** (vision 45.00 MHz, sound 41.50 MHz) by a HackRF;
- a **path** that can add a ghost (a reflection) and noise (a weak signal), and a simple **receiver** that shows the video and the picture a set would recover.

It sends the Project405 test card, or any other still picture.

2. Files

All in `Pi_Signal_Generator/hackrf/gnuradio/`:

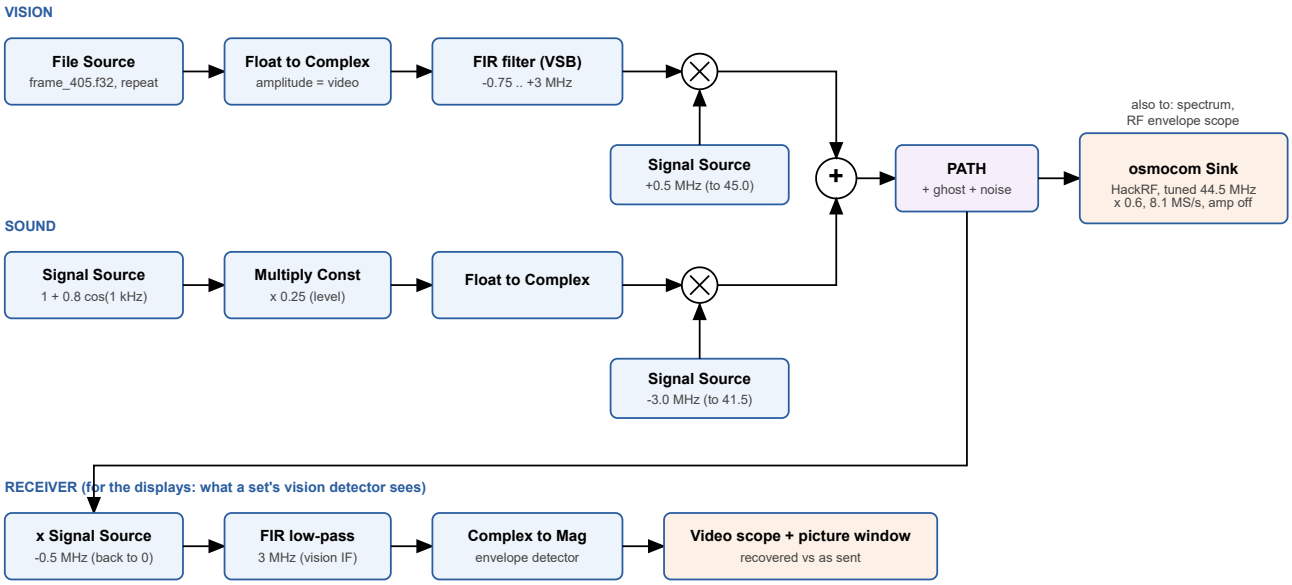
File	What it is
`run_on_screen.bat`	**Double-click to run it on screen**, no HackRF needed
`run_with_hackrf.bat`	Double-click to transmit with the HackRF, with all the displays (needs a faster PC than an i5-4690: see section 6)
`run_tx_only.bat`	**Double-click to transmit the test card with the HackRF**: sliders only, no displays
`run_clip.bat`	**Double-click to transmit a video clip with its sound**: sliders only, no displays (make the clip first)
`open_in_grc.bat`	Double-click to open the flowgraphs in GNU Radio Companion
`tx405_system_a.grc`	**The flowgraph** (transmits with the HackRF)
`tx405_system_a_sim.grc`	The same, on screen only: Throttle on, osmocom Sink off
`tx405_system_a_tx.grc`	Transmit only: the receiver, scopes, spectrum and picture window switched off
`tx405_system_a_clip.grc`	Transmit only, playing a video clip and its sound in place of the test card and tone
`make_405_clip.py`	Turns a stretch of any video into 405-line frames and a sound file for the clip flowgraph
`make_405_frame.py`	Makes `frame_405.f32`, one 405-line frame of the picture, for the flowgraph to play
`frame_405.f32`	The Project405 test card as one frame (324,000 samples, 1.3 MB)
`make_grc.py`	Writes the `.grc` file (only needed to change the flowgraph's layout in code)
`test_chain.py`	Off-air test: runs the transmitter chain without a HackRF and checks the signal
`test_rx.py`	Off-air test of the path and receiver: ghosts, noise, and the recovered video's alignment

© 2026 Project405 · project405.co.uk · CC BY-NC-SA 4.0: free for non-commercial use with credit; selling boards or kits by permission only

`grc_check.py`, `gr_env.ps1`	Checks a `.grc` file with GNU Radio's own engine, and lists any faults
`picture_display_block.py`	The picture window's code (an Embedded Python Block, built into the flowgraph)
`make_figs.py`	Draws the figures in this guide

3. How it works

Figure 1: the 405-line System A transmitter flowgraph (tx405_system_a.grc)



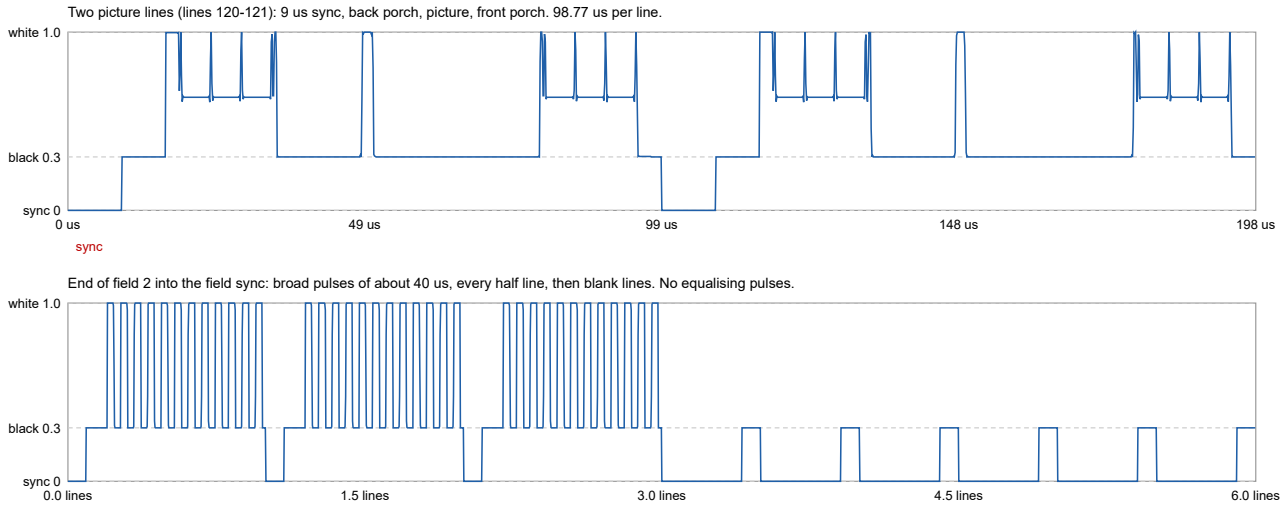
Sliders: vestige (upper), picture sideband (lower), sound level, tone depth, ghost strength and delay, noise, TX gain.

The picture. A 405-line frame is the same every time for a still picture, so `make_405_frame.py` works it out once, and a **File Source** with *Repeat* on plays it round and round. The frame uses exactly the same timing as the Project405 Pico pattern generator, which is proven on real sets:

Sample rate	8.1 MS/s = exactly 800 samples per line
Line	98.77 µs (10,125 lines a second)
Line sync	9.0 µs, then a 7.25 µs back porch; 1.75 µs front porch
Field sync	8 broad pulses of about 40 µs, every half line; no equalising pulses
Interlace	field 2 starts half a line later; 189 picture lines per field
Levels (carrier amplitude)	sync 0, black 0.30, white 1.00

© 2026 Project405 · project405.co.uk · CC BY-NC-SA 4.0: free for non-commercial use with credit; selling boards or kits by permission only

Figure 2: the video, as vision-carrier amplitude (System A: positive modulation)



Positive modulation. In GNU Radio a radio signal is a stream of complex numbers, and the carrier's amplitude is simply their size. So the video sample *is* the carrier amplitude: **Float to Complex** turns the 0-to-1 video into a carrier that's off at the sync tips and full at peak white. That's System A's positive modulation, the opposite of the 625-line systems.

The vestigial sideband. Plain AM would be 6 MHz wide (3 MHz each side). System A sent the full 3 MHz **below** the vision carrier but only a **vestige of 0.75 MHz above** it, the opposite way round to every later system; that is why channel 1's sound sits at 41.5 MHz, just under the picture sideband. The **FIR filter** does that, with taps from `firdes.complex_band_pass(-upper, +vestige)` (the variable `upper` is the full, lower sideband; the name is left over from the first version, which had the sidebands the wrong way round, corrected 20 November 2026). A complex filter can be lopsided like this; a real one can't.

Moving it up the band. The HackRF is tuned to **44.5 MHz**, in the middle of the channel. A **Signal Source** at +0.5 MHz, multiplied in, moves the vision carrier up to 45.0 MHz. Another, at −3.0 MHz, moves the sound carrier to 41.5 MHz.

Sound. A **Signal Source** making 1 + 0.8 cos(1 kHz) is AM with its carrier: a carrier with the tone at 80 % depth. It is scaled to a quarter of the vision peak and added to the vision.

Output. The total is scaled by 0.6 so that its peak (vision plus sound, plus a little overshoot from the filter) stays below the HackRF's full scale of 1. The **osmocom Sink** sends it to the HackRF, with the RF amplifier **off** and the TX gain on a slider.

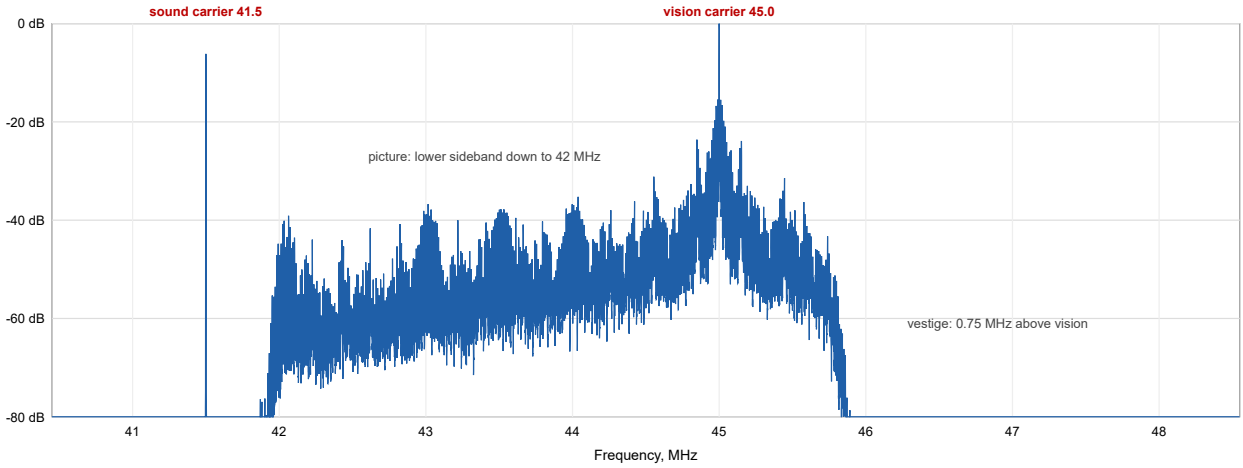
The path. Between the transmitter and the HackRF there's a model of the journey to the set. A **ghost** is a reflection, from a gasholder, a block of flats or a hill: a weaker copy of the signal that arrives a little later. Every microsecond of delay is 300 m of extra path, and on a 405-line screen about 1/80 of the picture width. A **Delay** block and a **Multiply Const** make the copy, and an **Add** puts it back with the direct signal. **Noise** is a **Noise Source** added in: a weak signal is simply one where the noise is large compared with the carrier. Both start at zero. They go in the transmitted signal too, so a real set shows the same ghosts and snow.

The receiver. To show what a set would make of it, the flowgraph also receives its own signal: one more **Signal Source** moves the vision carrier back to 0 Hz, a 3 MHz **low-pass filter** plays the part of the set's vision IF (and removes the sound carrier), and **Complex to Mag** takes the envelope, as the set's vision detector diode does. The **video scope** draws the recovered video in blue over the video as sent in red. A **Delay** block holds the red trace back by the two filters' delay (58 samples), so they line up exactly. The **picture window** (an Embedded Python Block, `picture_display_block.py`) rebuilds the picture: it collects

© 2026 Project405 · project405.co.uk · CC BY-NC-SA 4.0: free for non-commercial use with credit; selling boards or kits by permission only

one frame in four, weaves the two fields back together as a set does, keeps just the picture area, and shows it at 4:3 in its own window, "Recovered picture: what a set would show", with any snow or ghosts in it. (GNU Radio's own Time Raster display was tried first, but it can't keep up with a TV signal: it drops lines, so the picture comes out jumbled.)

Figure 3: the transmitted spectrum (8.1 MHz span, HackRF tuned to 44.5 MHz)



4. Running it

The easy way: double-click `run_on_screen.bat` to try it with no HackRF, or `run_with_hackrf.bat` to transmit. Each one sets up GNU Radio itself, so no GNU Radio prompt or GRC is needed. To look at or change the blocks, double-click `open_in_grc.bat`. Step by step:

1. Plug the HackRF into the PC. (It's normally on the Pi 3: move it across, or install GNU Radio on the Pi.)
2. **Connect the HackRF by cable, through an attenuator**, to the receiver or the set's aerial socket (for an AC/DC set, through an isolating connection: see the HackRF report). Start with the TX gain at **0**.
3. Start **GNU Radio Companion** (Start menu, or `run_GRC.bat` in `C:\Program Files\GNURadio-3.8\bin`), open `tx405_system_a.grc`, and press **Run** (the ▶ button, or F6). Note that `run_gr.bat`, next to it, is different: it only opens a command prompt set up for GNU Radio, and waits for you to type. From that prompt, `gnuradio-companion.py tx405_system_a.grc` opens GRC, and `python tx405_system_a.py` runs the transmitter without GRC.
4. The window shows the sliders, the video scope (recovered and as sent, two lines, triggered on sync), the spectrum, and the RF envelope going to the HackRF. The recovered picture opens in a second window.
5. Tune the set to channel 1, or the SDRplay to 43.25 MHz with an 8 MHz span.

No HackRF? Use `run_on_screen.bat`, or open `tx405_system_a_sim.grc`: it's the same flowgraph with the **Throttle** enabled and the **osmocom Sink** disabled, so it runs on screen only, at the right speed. (In GRC, right-click a block to *Enable*, *Disable* or *Bypass* it.)

A different picture. `python make_405_frame.py my_picture.png` makes a new `frame_405.f32` from any picture (it is scaled to 4:3 and shown in grey). Run the flowgraph again.

A moving picture, with sound. GNU Radio 3.8 can't decode a video file itself, so the clip is prepared first:

```
python make_405_clip.py <video> <start seconds> <length seconds>
```

This writes `clip_405.u8` (the picture as 405-line frames, 8 bits a sample, about 486 MB a minute) and `clip_405_sound.f32` (the sound at 50 kHz) into `yt_work\hackrf_video` in your user folder (give a fourth argument to put them somewhere else). They use exactly the timing of `make_405_frame.py`, so the sync is

© 2026 Project405 · project405.co.uk · CC BY-NC-SA 4.0: free for non-commercial use with credit; selling boards or kits by permission only

the same as the test card's. In the clip flowgraph the picture file replaces the test card frame, and the sound file replaces the 1 kHz tone: each sound sample is held for 162 samples (50 kHz to 8.1 MS/s), and the *Sound: modulation depth* slider sets how loud it is. Both files are the same length, so they stay in step as they repeat. A minute or two is practical; for a whole film, use hacktv. Then double-click `run_clip.bat`. The first clip made was a minute of the Spare Room tour, from 6:42 (the GEC 405-line TV).

File not found? The **File Source** block looks for `frame_405.f32` in the folder GNU Radio was started from, which is this folder when you use the `.bat` launchers. Started some other way, double-click the block and use the **...** button to find the file.

5. Things to try

Slider	What to watch
Upper sideband (vestige)	Narrow it and watch the spectrum above 45 MHz shrink. On a set, too narrow a vestige smears large areas of the picture.
Lower sideband (picture detail)	At 1 MHz the test card's 1.5, 2 and 3 MHz gratings disappear on the set: this is the picture's sharpness.
Sound carrier level	System A had the sound about a quarter of the vision. Too high, and sound buzz appears on the picture.
1 kHz tone depth	The sound carrier's sidebands (zoom in on 41.5 MHz) grow, and the tone gets louder.
Ghost strength and delay	A second, fainter test card appears to the right of the first, in the picture window and on the set. Short delays (under 1 µs) just smear edges; 5–10 µs gives a clear double image. On the scope, a copy of each sync pulse follows the real one. A strong ghost can upset the set's line hold.
Noise	Snow, in the picture window and on the set. On the scope, the lines turn fuzzy; at about 0.2 the picture is still there but the sync is starting to suffer.
TX gain	Output level. With a cable, keep it low.

6. Tested

`test_chain.py` runs the same blocks with no hardware for 0.2 s and checks the output (18 October 2026):

Check	Result
Peak level into the HackRF	0.91 (must stay below 1)
Vision carrier 45.0 MHz / sound carrier 41.5 MHz	0 dB / −6.7 dB
Picture sideband, 1–3 MHz below vision	present, about −68 dB per bin
Beyond the vestige (1.2–2 MHz above vision)	suppressed by about 128 dB
Sound	1000 Hz tone, 82 % depth
Picture demodulated back	matches the frame (correlation 0.985), sync 0.03 / black 0.32 / white 1.00

`test_rx.py` does the same for the path and the receiver:

Path	Recovered video
Clean	lines up with the video as sent exactly (offset 0, filter delays 39 + 19 = 58 samples); match 0.986 (the rest is the plain detector's distortion of a vestigial sideband, which real sets corrected with the slope of their IF response)

© 2026 Project405 · project405.co.uk · CC BY-NC-SA 4.0: free for non-commercial use with credit; selling boards or kits by permission only

Ghost 0.3 at 2 µs	ghost found 16 samples (2 µs) after the picture
Ghost 0.5 at 10 µs	ghost found 81 samples (10 µs) after the picture
Noise 0.05	picture signal-to-noise about 22 dB: light snow
Noise 0.2	about 15 dB: heavy snow

The `.grc` file passes GRC's own checks and compiles.

With the HackRF, 29 October 2026 (after the firmware update to 2026.01.3; cable, switched attenuator, TX gain about 20):

Check	Result
osmocom Sink with the HackRF	Accepted it: *Using HackRF One with firmware 2026.01.3*
Full flowgraph (with receiver and displays)	**Continuous underruns**: all four cores of the i5-4690 flat out. The receiver, scopes, spectrum and picture window are too much at 8.1 MS/s
Transmit only (`run_tx_only.bat`)	About 2.4 of the 4 cores: keeps up
Video clip (`run_clip.bat`)	About 2.6 of the 4 cores: picture on the set
Sound	**None at first**, picture fine. The osmocom Sink's bandwidth was left at 0 (automatic), which at 8.1 MS/s gives the HackRF a 6 MHz filter, ±3 MHz around 44.5 MHz; the sound carrier is at −3.0 MHz, on the filter's edge. The filter is now set to **8 MHz** in all the HackRF versions. Result of the fix: still to be confirmed

With a slower PC, the displays and the HackRF are best kept apart: watch the signal on screen with `run_on_screen.bat`, and transmit with the transmit-only versions.

7. Notes

- **HackRF firmware.** This guide was written when the HackRF still had its 2014 firmware, which the current `hackrf_transfer` refused. On 28 October 2026 the firmware was updated to 2026.01.3, and the osmocom Sink now accepts it (section 6).
- **Stopping it.** Close the slider window. The flowgraph ignores Ctrl+C in its console window.
- **Speed.** The chain is all built-in C++ blocks and runs comfortably on a PC. A Pi 3 would struggle at 8.1 MS/s, as it did with hacktv; a Pi 4 should cope.
- **Licence.** Transmitting on Band I needs a licence in the UK. Work by cable, at low level, with the RF amplifier off.
- **Other radios.** The osmocom Sink can be swapped for a PlutoSDR Sink or a UHD sink later. The Pluto's *Cyclic* option could play a frame from its own memory.

© 2026 Project405 (project405.co.uk). Licensed under CC BY-NC-SA 4.0: free to copy, build and adapt for non-commercial use, with credit to Project405, sharing changes on the same terms. Selling boards, kits or built units is by permission only (workshop@project405.co.uk). Material from others, credited where it appears, remains theirs.