Joystick-Controlled Radar Beamformer

8-element phased array · X-band 10.525 GHz · ADI-sponsored · NCSU Senior Design Group 34 · Aug 2024 - May 2025

Topics: *Phased-array beamforming at X-band · ADI ecosystem (CN0566 PHASER / ADAR1000 / PlutoSDR / ADF4159 PLL) · per-element phase & gain control · multi-threaded Python · real-time hardware integration · SDR signal chain & FFT-based peak detection*

> **The one-liner:** A real-time joystick-controlled 8-element phased-array beamformer at X-band (10.525 GHz), built end-to-end on Analog Devices' phased-array dev ecosystem (CN0566 PHASER + ADAR1000 + PlutoSDR) hosted on a Raspberry Pi 4. Built for trade-show demos by Analog Devices.
>
> **Why it's interesting:** Hands-on work with the actual ADI parts (CN0566, ADAR1000, PlutoSDR) that show up across phased-array, RF, SDR, and satellite-comms hardware. The project spans the full RF-to-software stack, from the beamformer ICs to the Python control layer.
>
> **My role:** Led the hardware integration (ADAR1000 interfacing with HB100, real-time phase/gain control per element), all Raspberry Pi 4 system integration, and worked on the application software (GUI app, ADAR1000 control script, LUT generation).

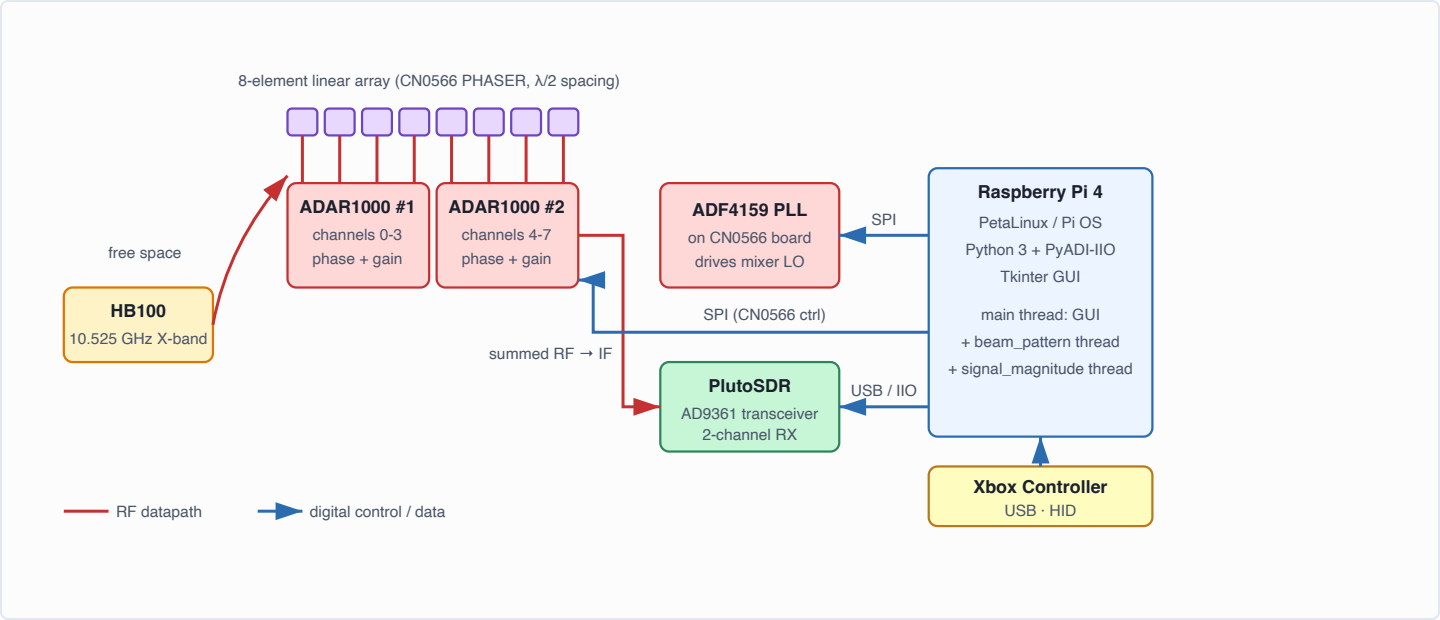
What it does

An Xbox controller's left joystick maps directly to the beam-steering angle of an 8-element X-band phased array. As the user moves the stick, per-element phase shifts are pushed to the ADAR1000 chips in real time, the array's main lobe steers, and a Tkinter GUI shows both:

- The live **polar + cartesian beam pattern** (radiation pattern at the current steering angle)
- The current **signal magnitude** from the PlutoSDR receive path

The right joystick controls displayed signal magnitude (demo placeholder - substitutes for an actual signal-strength readout in the trade-show demo flow). The system runs multi-threaded Python on a Raspberry Pi 4 host.

Hardware stack



End-to-end hardware. HB100 transmits at 10.525 GHz in free space; the 8-element PHASER array (driven by two 4-channel ADAR1000s) applies per-element phase shifts to steer the beam; the summed RF is downconverted (LO from the ADF4159 PLL) and digitized by the PlutoSDR's AD9361. The Pi 4 orchestrates everything over SPI + USB-IIO and drives the GUI.

Component	Role
ADI **CN0566 "PHASER"**	8-element phased-array dev board, the host board for the two ADAR1000s + ADF4159 PLL. The whole RF front-end sits on this board.
ADI **2× ADAR1000**	4-channel X-band analog beamformer ICs. One handles array channels 0-3, the other 4-7. Per-channel phase shift and gain control via SPI.
ADI **ADALM-PlutoSDR**	SDR built around the AD9361 transceiver. RX-side digitization of the down-converted IF.
ADI **ADF4159 PLL**	On the CN0566 board. Drives the down-conversion mixer LO. Programmed to `(SignalFreq + IF − offset) / 4` because feedback comes off the VCO's /4 output.
HB100	Off-the-shelf X-band microwave source (10.525 GHz). Acts as the target signal during demos.
Raspberry Pi 4	Host platform. Runs Python, PyADI-IIO, Tkinter GUI; drives ADAR1000 + PlutoSDR over SPI and IIO USB.
Xbox controller	USB HID input device; left stick → beam-steering angle, right stick → magnitude display.

Software stack

File	What it does
`Joystick_Controlled_Beamformer_App.py`	Main app. Creates the Tkinter GUI, spawns two daemon threads: `beam_pattern` (reads controller input, looks up the precomputed beam pattern for the current θ in the LUT, updates the polar/cartesian plot) and `signal_magnitude` (reads right-stick input, updates the magnitude readout). Wraps θ mod 360 with display remapping for negative angles.
`ADAR1000_test_script.py`	Command-line script that takes a target angle in degrees, configures CN0566 + ADAR1000s + PlutoSDR, calls `my_phaser.set_beam_phase_diff(angle)`, captures I/Q, runs `spec_est` (window + FFT + dB), prints peak frequency, plots the time-domain I/Q + spectrum.
`GUI_classes.py`	Tkinter GUI class (`JCBGUI`) - polar + cartesian plots embedded as matplotlib FigureCanvasTkAgg widgets, current-phase text display, magnitude bar.
`LUT_generation.py` + `LUT_demo.py`	Pre-compute the beam pattern at 1° θ resolution (–180° to +180° → 360 entries), pickle to `LUT_rx_noise.pkl` / `LUT_rx_no_noise.pkl` (~3.2 MB each). Runtime just indexes into the LUT - no live computation cost in the GUI loop.
`beamforming_functions.py`	Simulation classes: `Generate_Source` (far-field plane-wave model with optional Gaussian noise per element, steering-vector `exp(−2jπ · d/λ · n · sin θ)` for n=0..7) and `Scan` (sweeps θ from –π to π in *theta_samples* steps, applies conjugate weights, computes power in dB, generates polar / cartesian plots). Used to *generate* the LUT - not in the runtime GUI loop.

File	What it does
`Controller_Input.py`	Xbox controller polling - wraps controller axes into a `read()` call returning `(left_stick_x, right_stick_y)`.

Threading discipline

The GUI mainloop runs on the main thread (Tkinter's hard requirement). The two driver threads (beam_pattern, signal_magnitude) are daemon threads that poll the controller in tight while-True loops, only fire a GUI update when their input has actually changed (gated on `last_theta` / `lastmag`). The LUT lookup keeps the GUI thread's work bounded.

Signal chain - the actual numbers

Parameter	Value	Set by
HB100 carrier	`10.525 GHz (X-band)`	HB100 hardware
Per-element gain	`64 / 127 (~half scale)`	`set_chan_gain(i, 64)`
PlutoSDR sample rate	`30 MHz`	`my_sdr.sample_rate`
PlutoSDR RX RF bandwidth	`10 MHz`	`my_sdr.rx_rf_bandwidth`
Analog filter	`LTE20 (~20 MHz)`	`my_sdr.filter = "LTE20_MHz.ftr"`
RX local oscillator	`2.2 GHz`	`my_sdr.rx_lo`
PLL frequency (ADF4159)	`(HB100 + RX_LO − 1 MHz) / 4`	set via `my_phaser.frequency`
Resulting IF	`~2.201 GHz`	downconverted from 10.525 GHz
RX buffer size	`1024 samples`	`my_sdr.rx_buffer_size`
Channels enabled	`RX1 + RX2 (voltage0, voltage1)`	`my_sdr.rx_enabled_channels`
RX gain mode	`manual, 0 dB`	both channels
Peak detection	`FFT of (data[0] + data[1]), argmax`	`spec_est()` in PyADI-IIO

The 1 MHz offset on the PLL is deliberate - keeps the downconverted signal slightly off DC so it doesn't land on a DC spike in the spectrum.

Beam-steering math

The 8-element linear array sits at λ/2 element spacing. For a target angle θ, the per-element phase progression that constructively combines plane waves from that direction is:

```
w[n] = exp(−2jπ · (d / λ) · n · sin θ)    for n = 0..7

with d / λ = 0.5  (half-wavelength spacing):

w[n] = exp(−jπ · n · sin θ)
```

In code, the GUI doesn't compute this live - it just calls `my_phaser.set_beam_phase_diff(angle)` from PyADI-IIO, which handles the per-element phase calculation and pushes it down to the ADAR1000s over SPI. The LUT generator (`LUT_generation.py`) precomputes the resulting beam pattern (the radiation pattern of the array under those weights) for visualization.

The `Scan` class in `beamforming_functions.py` uses conjugate-weight delay-and-sum scanning to compute the pattern at 1000 θ points from −π to +π: for each scan angle θ_i, compute weights, conjugate-multiply against the received signal, take the variance in dB. Textbook delay-and-sum - *no original DSP work here*.

What I specifically did

My contributions:

- **Led hardware integration** - got the ADAR1000s talking to the rest of the chain over SPI, configured the CN0566 to play nicely with the HB100 source for real-time phase and gain control on each array element.
- **Managed all Raspberry Pi 4 system integration** - OS setup, dependency stack (PyADI-IIO, IIO drivers, evdev for controller, Tkinter + matplotlib), calibration runs with the ADI utility tools.
- **Worked across the Python application stack** - the main app and threading (`Joystick_Controlled_Beamformer_App.py`), the CLI control + capture + FFT script (`ADAR1000_test_script.py`), the Tkinter UI (`GUI_classes.py`), LUT generation (`LUT_generation.py` / `LUT_demo.py`), and the simulation/scan classes (`beamforming_functions.py`). This was a team project and every file was a group effort.

Design decisions and trade-offs

What it demonstrates:

- Built and worked with a phased-array system end-to-end at the hardware-integration level.
- Comfortable with the ADI ecosystem - CN0566, ADAR1000, PlutoSDR, ADF4159 PLL, PyADI-IIO (`adi.cn0566` , `adi.ad9361` modules).
- Understand how per-element phase shift steers a beam, and have seen it work at 10.525 GHz on real hardware.
- Sponsored by Analog Devices - the demo was built for their trade-show use.

Scope and limitations:

- **No original DSP work.** The beamforming math is textbook delay-and-sum at the simulation layer. PyADI-IIO does the hardware-level heavy lifting.
- **No FPGA gateware on this project.** The Pi 4 is the host, the ADAR1000s do the RF phase shift, the PlutoSDR does the digitization. No HDL was authored.
- **Benchtop X-band demo, not satellite-grade or production-grade.** *Adjacency to production systems, not parity.*
- **Half-wavelength linear array**, not the dense planar arrays that production satcom or radar systems deploy.

Common questions about this project

Question	Short answer
What is the phased-array experience here?	Senior design - 8-element X-band array using ADI ADAR1000 beamformers on the CN0566 PHASER, joystick steering in real time, PlutoSDR RX, Pi 4 host. ADI-sponsored. I led the hardware integration and worked on the application software (GUI, ADAR1000 control, LUT generation).
How does the beam steering work?	Per-element phase progression. λ/2 spacing → each element gets phase `−π · n · sin θ`. PyADI-IIO's `set_beam_phase_diff(θ)` handles the per-channel math and pushes phase/gain to the ADAR1000s over SPI. The GUI looks up the precomputed beam pattern at the current θ from a pickled LUT - no live computation on the display side.
How is the RF down-converted?	ADF4159 PLL on the CN0566 sets the mixer LO to `(10.525 GHz + 2.2 GHz IF − 1 MHz offset) / 4` - the /4 is because PLL feedback comes off the VCO's divided output. Then the PlutoSDR digitizes the ~2.2 GHz IF at 30 MS/s with a 10 MHz RF bandwidth, LTE20 analog filter.
Is there any FPGA work on this project?	No. The Pi 4 was the host, the ADAR1000s did the RF phase shift, the PlutoSDR did the digitization. No HDL was authored on this project.
How much of the beamforming algorithm is original?	The math is conventional delay-and-sum - no novel DSP. The implementation was straightforward; the engineering was in integrating the parts cleanly and making it usable for the demo.
What was the hardest part?	The integration debugging - getting the CN0566 + ADAR1000 + PlutoSDR + ADF4159 to all see each other consistently over the mixed SPI + IIO interfaces. The two-tier connection logic in the test script (try local, fall back to `phaser.local` shared context) is a residue of that.
What would scale differently at production?	Areas worth learning - calibration discipline across many elements, beam-pattern fidelity under thermal drift, polarization control, how the gateware side handles per-element timing alignment. My benchtop demo just assumed all of that worked.
How many elements, and what spacing?	8 elements, λ/2 spacing, linear (1D) array. Educational scale, not production planar arrays.

Question	Short answer
What ADI tooling does it use directly?	PyADI-IIO - specifically `adi.cn0566.CN0566`, `adi.ad9361`, the `set_beam_phase_diff` and `set_chan_gain` APIs, `load_hb100_cal()`, `spec_est()`. Calibration via the ADI utilities. The whole project lives in their ecosystem.
How does an analog beamformer like the ADAR1000 compare to digital beamforming?	Analog beamforming applies the phase shift at RF before the mixer/ADC, so you only need one downconversion chain per beam - cheaper hardware, lower power, but fixed in the analog domain (one beam per RF front-end). Digital beamforming applies weights after the ADC per element, which is more flexible (multiple simultaneous beams, adaptive weights) but needs an ADC per element and a lot more downstream compute. Production satcom often uses hybrid topologies for that reason.
What was the Analog Devices sponsorship like?	ADI was the industry partner for our senior design project - provided the hardware (CN0566 / ADAR1000 / PlutoSDR), set the demo requirement (a controllable phased-array beamformer for trade-show use), and reviewed the project at the demo stage. Light-touch sponsorship; the engineering decisions were ours.