
On-Device GNU Radio for an SoC-Based Underwater Acoustic Modem

Gabriel Garcia
George Sklivanitis
Dimitris A. Pados

GARCIAG2022@FAU.EDU
GSKLIVANITIS@FAU.EDU
DPADOS@FAU.EDU

Center for Connected Autonomy and AI (CA-AI.fau.edu), Florida Atlantic University, Boca Raton, FL, USA

Abstract

The FAU underwater acoustic modem is a low size, weight, power and cost (SWaP-C) Zynq-7000 System-on-Chip (SoC) platform developed for micro autonomous underwater vehicle (μ AUV) research. Its first generation was controlled exclusively by hand-written user-space C and Python applications that memory-map the AXI DMA engine directly, an arrangement that substantially complicated the evaluation of new signal processing functions. In this work, we present a Yocto/PetaLinux integration that builds a headless GNU Radio and an out-of-tree (OOT) module for the FAU modem, a pair of OOT blocks (FAU Modem Source and FAU Modem Sink) that serve simultaneously as GNU Radio blocks and as the platform's sole DMA driver, and a desktop-side deployment tool that compiles a GNU Radio Companion (GRC) flowgraph, removes host-PC-only peripherals from it, transfers it to the embedded FAU modem over a 115200-baud serial console, initiates its execution, and exposes its QT GUI controls on the host PC in real time. We describe the scatter-gather descriptor-ring management scheme used by the processing blocks to provide deterministic detection of DMA under-run and overrun events on a streaming datapath operating without interrupt-driven DMA signaling. We further define the shutdown and recovery sequence required to safely quiesce outstanding transactions, drain or invalidate in-flight bursts, and prevent residual AXI/DMA activity from leaving the FPGA datapath or host interface in a stalled state. The source code is available at:

<https://github.com/C2A2-at-Florida-Atlantic-University/GNU-Radio-FAU-Modem.git>.

Proceedings of the 16th GNU Radio Conference, Copyright 2026 by the author(s).

1. Introduction

Coordinating fleets of micro autonomous underwater vehicles requires acoustic modems that are low-SWaP and sufficiently open to be reprogrammed by the researchers deploying them. The FAU Center for Connected Autonomy and AI addressed the first two constraints with a custom Zynq-7000 transceiver board whose first generation demonstrated run-time reconfigurability at the microsecond level and data rates up to 2000 bps at ranges up to 50 m (Hermans et al., 2022). Three variants exist: S10 (single-channel receive and transmit, Zynq-7010), M10 (multi-channel receive, single-channel transmit, Zynq-7010), and M20 (the same topology on a Zynq-7020).

The third constraint, openness, has proven the most difficult to satisfy in practice. The first-generation software stack comprises a set of standalone user-space Python and C programs that open `/dev/mem`, `mmap` the AXI DMA control and status registers, and drive the engine in direct register mode. This was a deliberate choice, as it obviated the authorship and maintenance of a kernel driver (Hermans et al., 2022). Consequently, signal-processing functionality and memory-mapped register manipulation are implemented within the same source module, coupling the data-plane processing path with low-level control-plane operations. For example, modifying the modulation waveform requires editing a program that is also responsible for buffer descriptor layout, cache coherence, and engine teardown; furthermore, each such program re-implements these concerns in a slightly different manner.

In this work, we describe our effort to transition to GNU Radio (The GNU Radio Project, 2026) as the platform's signal processing development environment on the processing subsystem of the SoC. GNU Radio is a mature framework, supported by an active community, and is already employed in both academic and commercial applications. It is open source and provides a reliable and robust development environment, a property that benefits human contributors and agentic coding workflows alike.

The rest of this paper is organized as follows. Section 2 describes the FAU modem hardware and the FPGA bitstream

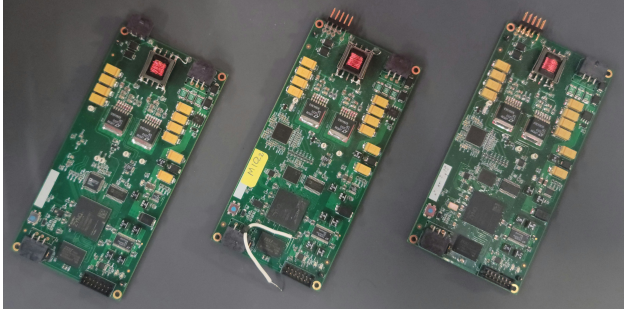


Figure 1. The three FAU modem transceiver boards: S10, M10 and M20, left to right. All share the same analog front end, with the tuned transformer at the top of each board coupling to the hydrophone; they differ in their Zynq part and receive channel count.

properties the OOT blocks were written against. Section 3 covers the Yocto and PetaLinux integration that produces a headless GNU Radio image for a 512 MB ARMv7 board. Section 4 describes the two OOT blocks, which are the platform’s DMA driver as much as they are GNU Radio blocks, and the design decisions that follow from a bitstream that connects no DMA interrupt. Section 5 describes the host-PC-side tooling.

2. FAU Underwater Acoustic Modem Platform

The modem is built around a custom in-house PCB (Figure 1) carrying a Xilinx Zynq-7000 SoC (AMD Xilinx, 2021): two ARM Cortex-A9 cores at 667–886 MHz, 512 MB of DDR3L, removable microSD storage, and two 115200-baud UART ports. The Zynq-7010 parts provide 28k logic cells and 80 DSP slices; the Zynq-7020 provides 85k logic cells and 220 DSP slices. The analog front end follows a Teledyne RESON TC4013 hydrophone through a tuned transformer into a 19 dB preamplifier, a variable gain amplifier, a 1 MHz low-pass filter, and a 14-bit ADC; the transmit chain runs a 14-bit DAC into an I/V converter, a low-pass filter, and a 5 W power amplifier (Hermans et al., 2022). The S10 uses an AD9764 DAC with an AD9244 ADC; M10 and M20 pair the AD9764 with an LTC2171.

Because underwater acoustic channels are low frequency, all processing is done by direct sampling, and the programmable logic (PL) carries CIC interpolation and decimation, a DDS numerically-controlled oscillator (NCO) for the transmit and receive carriers, and an AXI DMA engine bridging the PL to the processing system (PS).

The sample format on the DMA is one 32-bit word per complex sample, little endian, with the in-phase component as a signed Q15 value in bits [15:0] and quadrature in [31:16], scaled by 32767. The CIC compilers in

all four current designs are built with minimum rate 4 and maximum rate 400, which bounds the usable sample rate to [25 kS/s, 2.5 MS/s] in both directions.

3. On-Device GNU Radio

The board runs PetaLinux 2024.2 (AMD Xilinx, 2024), which is a Yocto scarthgap distribution (The Yocto Project, 2024) with AMD’s board support tooling layered on top. There are two independent PetaLinux projects in the repository, one per SoC, and a Yocto layer, `meta-fau-modem`, that is shared between them as a git submodule and contains both the recipes and the OOT module source. Prior work by Philip Balister on `meta-sdr` (Balister, 2025) supports this portion of the integration directly: the layer already builds GNU Radio and its dependencies for an embedded target, and was adapted to our platform through changes to the build configuration. The remaining integration consists of packaging our own module.

The module is packaged as an ordinary Yocto recipe and built into the board image alongside the headless GNU Radio it links against, so a modem boots with both already installed and nothing is ever compiled on the board itself. Since each SoC has its own project, a change to the blocks reaches a board only once that board’s image has been rebuilt and written to it.

4. OOT Block Architecture and DMA Management

FAU Modem Source and FAU Modem Sink are a `gr_complex` source block and a `gr_complex` sink block, respectively, parameterized by sample rate, NCO frequency, and buffer geometry. In the absence of an underlying kernel driver, the blocks open `/dev/mem`, map the DMA registers and a reserved memory window, construct scatter-gather buffer descriptor rings, and poll for completion. The blocks therefore constitute the platform’s DMA driver, and the GNU Radio scheduler is their only thread of control.

4.1. Common Parameters

Both blocks are configured by a sample rate, an NCO frequency, and a buffer geometry expressed as a descriptor size in samples and a descriptor count. Both additionally expose a stall-timeout floor, a verbosity flag, and a switch that permits operation against an unreserved DMA window; the last is intended for bring-up on an image whose physical memory map has not yet been confirmed, and is otherwise left disabled so that the check described in Section 4.4 applies.

Each block additionally publishes counters that a flowgraph

in	FAU Modem Sink (AXI DMA MM2S)
	Sample Rate (Hz): 400k
	NCO Frequency (Hz): 120k
	TX Scale: 1
	Samples per BD: 1.024k
	Ring Depth (BDs): 16
	Prefill (BDs): 4
	Drain on Stop: True
	Allow Unreserved DMA Window: Yes (bring-up only)
	Stall Timeout (s): 20
	Verbose: False

Figure 2. FAU Modem Sink as instantiated in GNU Radio Companion, shown with representative values. The ring of 16 descriptors of 1024 samples provides 82 ms of buffering at 200 kS/s, of which the four prefill descriptors account for 20 ms.

or a bring-up script may read back: descriptors transferred, underruns, overruns, saturation events on transmit and receive, and short packets, separated into acquisition drops and stream drops.

All parameters are fixed at construction. The sole run-time setter is analog receive gain, which affects no part of the DMA path.

4.2. FAU Modem Sink

The sink drives the MM2S direction of the engine and therefore the transmit chain. Beyond the common parameters it exposes a transmit scale, a prefill depth, and a drain-on-stop flag. The transmit scale is the linear factor folded into the Q15 conversion described below, and because that conversion saturates, a scale above unity clips rather than inverts. The prefill depth is the number of descriptors posted before the engine is started, and therefore sets both the initial transmit latency and the margin available before the first opportunity for an underrun. The drain-on-stop flag selects whether descriptors already posted are allowed to retire at teardown or are discarded, which is the difference between a transmission that ends at a sample boundary and one that is truncated.

4.3. FAU Modem Source

The source drives the S2MM direction and therefore the receive chain. It carries none of the transmit-side parameters: there is no scale to apply, nothing to prefill, and nothing outstanding to drain, since on receive the engine is the producer and the flowgraph the consumer. It is otherwise configured identically to the sink, and the two instantiations shown in Figures 2 and 3 differ chiefly in descriptor size. Both geometries are inherited from the standalone bring-up scripts that preceded this work rather than derived from a

out	FAU Modem Source (AXI DMA S2MM)
	Sample Rate (Hz): 400k
	NCO Frequency (Hz): 120k
	Samples per BD: 8.192k
	Ring Depth (BDs): 16
	Allow Unreserved DMA Window: Yes (bring-up only)
	Stall Timeout (s): 20
	Verbose: False

Figure 3. FAU Modem Source as instantiated in GNU Radio Companion, shown with representative values. The ring of 16 descriptors of 8192 samples provides 328 ms of buffering at 400 kS/s. The receive direction has no transmit scale, no prefill, and no drain-on-stop.

rate or latency budget, and no systematic study of descriptor size against sample rate has yet been carried out.

4.4. Ring Discipline

The engines are used in scatter-gather mode (AMD Xilinx, 2022) with a non-cyclic, tail-bumped ring rather than a cyclic ring. Cyclic buffer descriptor mode provides no hardware interlock: on transmit, an underrun silently retransmits stale data, and on receive, an overflow is undetectable. Tail-bumping instead yields an exact test: if the engine reports itself idle at the instant the tail pointer is updated, it has consumed all posted work before software was able to extend it, which constitutes an underrun on transmit and an overrun on receive.

The window occupies the uppermost 16 MB of DRAM, which in this image lies outside kernel-managed memory; no device tree reservation is therefore required in order to claim it. The blocks verify this against the kernel's physical memory map prior to arming, and acquire a cross-process advisory lock so that no two flowgraphs can claim the window concurrently.

4.5. Sample Path

Conversion between GNU Radio's complex float samples and the packed Q15 wire format uses VOLK (The VOLK Project, 2026), but never directly against the device mapping: samples are copied to and from a cached staging buffer, and the vectorized conversion runs only on that copy. A linear transmit scale is folded into the same conversion and therefore incurs no additional cost; because the conversion saturates rather than wraps, a scale factor above unity clips the waveform instead of inverting it. The block reports the frequency of such events.

The transmit block additionally maintains a dedicated pre-zeroed silence buffer that is substituted for valid data when-

ever the flowgraph fails to sustain the required rate. This prevents the DAC from holding a stale sample as a DC offset and concurrently satisfies the startup prefill requirement.

4.6. Failure and Teardown

A block's work function never throws, because an exception escaping it terminates the process outright and leaves the DMA engine running. Fatal conditions instead emit diagnostics and signal to the scheduler that the block has completed; only construction and startup raise exceptions. Polling consists of a bounded spin followed by a sleep, rather than an unbounded immediate return of no work, which would cause the GNU Radio scheduler to spin.

Teardown follows a fixed order: stop the engine, wait for it to report itself halted, and only then reset the fabric. Resetting a running engine orphans an in-flight transaction on the processor's high-performance port, and on the order of a few dozen such leaks render the board unresponsive until it is power cycled. Two watchdogs supplement the error check, as the engine may also stall without reporting an error. A stall watchdog trips when nothing retires within a timeout. The exposed value is a floor. At arm time the block raises it in proportion to the configured buffer geometry, so that a low sample rate cannot produce a false positive. A ring-integrity watchdog re-verifies descriptor linkage and buffer addresses several times a second and is fatal on divergence, since a corrupted buffer address indicates that the engine is about to access memory outside its allocation.

The lower half of Table 1 reports the counters from a 49.5 s transmit run at 400 kS/s, corresponding to a sustained 1.6 MB/s across the DMA and approximately 2417 descriptors retired. No underrun or overrun was recorded and the process exited with status zero, the ring geometry of 16 descriptors of 8192 samples providing 328 ms of buffering at that rate.

4.7. Analog Receive Gain

The receive block also owns the control voltage of the front end's variable gain amplifier ([Analog Devices, Inc., 2020](#)), which it sets through a small SPI DAC ([Texas Instruments, 2015](#)). That SPI DAC is attached to the processing system rather than the fabric, so its setting persists across both a fabric reset and a process restart. The voltage is not a construction parameter of the block; it is written only through a run-time setter, which permits a GRC slider to adjust receive gain while a flowgraph is executing.

Gain is parameterized as that control voltage rather than in decibels, since the amplifier's absolute gain depends on a front-end strap the platform does not read back; the block

clamps a request to the range the amplifier can accept and reports changes as relative. It is the only setting of either block that may change while the block is running: because it affects no part of the DMA path, it is safe to expose to the live control channel of Section 5.

5. Host-PC-Side Deployment

The blocks cannot be instantiated on any host other than the board itself, as their constructors open `/dev/mem`. The modem has no network interface available in the field; its console is a UART. A flowgraph authored in GRC on a desktop must therefore be compiled, transferred to the board over a serial link, and executed there; performing this sequence manually imposes an overhead sufficient to discourage iterative development.

The design objectives for the tooling that eliminates this overhead were as follows: the engineer or developer should not be required to interact with the modem's terminal; as much GRC functionality as possible should be retained; flowgraphs that employ QT GUI sinks and desktop SDR peripherals for debugging should not require editing prior to deployment; and the desktop GNU Radio installation should require little or no modification. These objectives led to a standalone application with a shared core and two front ends: a Tk graphical interface intended to approach the convenience of executing a flowgraph in GRC, and a command-line interface for SSH sessions and agentic workflows.

5.1. Pipeline

The tool accepts a `.grc` file as direct input; Figure 5 presents one such flowgraph together with the signal it produced following deployment. Preflight checks verify that a FAU block is present, infer the target role, and validate the file format version. The headless transform then operates on a deep copy and performs three operations: it removes QT GUI sinks, splicing a null sink of matching type onto any output thereby left unconnected; it removes desktop-only SDR peripherals; and it fixes GUI variable controls to constants, reporting each substitution. An unrecognized GUI block is rejected rather than passed through, since passing it through would generate code that imports Qt on a board on which Qt is not present.

Compilation invokes GRC's own code generator as a subprocess, rooted at the original flowgraph's directory. GRC *evaluates* a flowgraph's import blocks during code generation, so a flowgraph that imports a sibling helper module cannot be compiled from any other location. Payload assembly follows the same rule and ships those helpers alongside the generated flowgraph.

The FAU source and sink also build as pure Python no-op

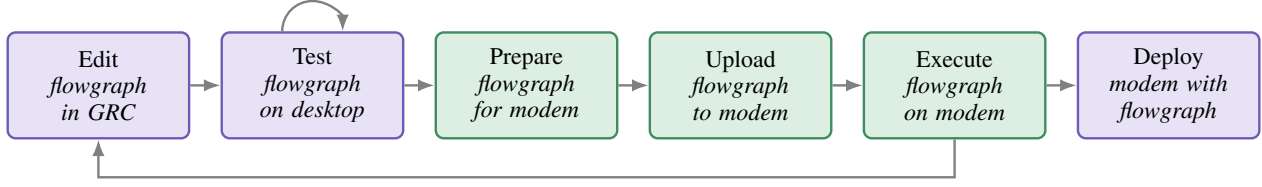


Figure 4. The deployment workflow. Purple steps are performed by the engineer, green steps by the tool. A flowgraph is authored and iterated on entirely in GNU Radio Companion, tested on the desktop against the no-op shim blocks, then pre-processed, uploaded and executed on the modem without leaving the tool; the lower return path is the iteration loop that runs against real hardware, and the final step is deploying the modem with the flowgraph it will carry.

shims for the desktop, where the source emits zeros and the sink discards them. They exist so that a flowgraph containing FAU blocks can be opened, validated, and generated in GRC without requiring that those blocks be disabled, which permits authoring and deployment to proceed as a single continuous operation.

The resulting workflow is illustrated in Figure 4: an engineer authors a flowgraph in GRC in the usual manner, optionally employing QT GUI sinks and a desktop SDR peripheral for debugging, then submits the `.grc` file to the deployment tool, which executes it on the modem. The graphical front end performs the entire sequence from a single control and can monitor the file, re-processing it whenever GRC writes it; deployment is never initiated automatically, since an unattended transfer to a board with a live DMA engine risks interrupting an active flowgraph without the teardown sequence of Section 4. The command-line interface exposes each phase separately for scripted and agentic use.

5.2. Transport

The payload is gzip-compressed, base64-encoded, and framed into 512-character chunks with a per-chunk CRC32 and an end-of-transfer SHA-256. The per-chunk CRC exists so that a corrupted chunk is caught and retransmitted immediately rather than at the end of a transfer over a 115200-baud line. The board-side receiver is implemented in Python and uses `hashlib`, `base64`, and `gzip` from the standard library, which is always included in the FAU modem’s PetaLinux image.

Table 1 reports a representative deployment. The payload of the flowgraph of Figure 5 totals 25,237 B across four files and is carried in 22 chunks in approximately 2.3 s, with no retransmission and no negative acknowledgement, and with the end-of-transfer hash confirmed on the board before the flowgraph is launched. Since the payload is compressed before framing, the transfer conveys source content at a rate above the raw capacity of the line, while occupying roughly half of the available wire time.

The protocol is designed around the fact that it shares a

Table 1. Deployment and run statistics for the flowgraph of Figure 5 on an M20, taken from a different run from the one shown in Figure 6. Quantities marked $\dagger$ are derived from the others rather than measured directly.

<i>Deployment over the 115200-baud console</i>	
Payload	25,237 B
Chunks	22
Elapsed time	≈ 2.3 s
Retransmissions	0
Negative acknowledgements	0
Integrity check	SHA-256
<i>Run</i>	
Sink sample rate	400 kS/s
DMA throughput $\dagger$	1.6 MB/s
Ring geometry	16×8192 samples
Ring depth $\dagger$	328 ms
Prefill	4 descriptors (82 ms $\dagger$)
Run duration	49.5 s
Descriptors retired $\dagger$	≈ 2417
Underruns	0
Overruns	0

physical link with a login prompt, a message of the day (MOTD), kernel `printk` output, and ANSI escape sequences. Sentinel matching is therefore performed by substring search and is never anchored to line boundaries, since a matched sentinel frequently does not constitute an entire line. Traffic that must be distinguished from that background carries a per-run nonce, generated at launch and borne by every control message, every reply, and the marker that reports the flowgraph’s exit status. The nonce is written to a file beside the flowgraph by the launch command itself rather than passed through the environment, since `sudo` does not preserve it.

5.3. Running, Stopping, and Live Control

Optionally, the tool loads a bitstream before running, using `fpgautil` on the board. A board’s transmit or receive role is a property of the bitstream currently loaded rather than of its hostname or part number, and a single-channel board legitimately supports both. A load additionally clears the fabric’s configuration registers and must therefore pre-

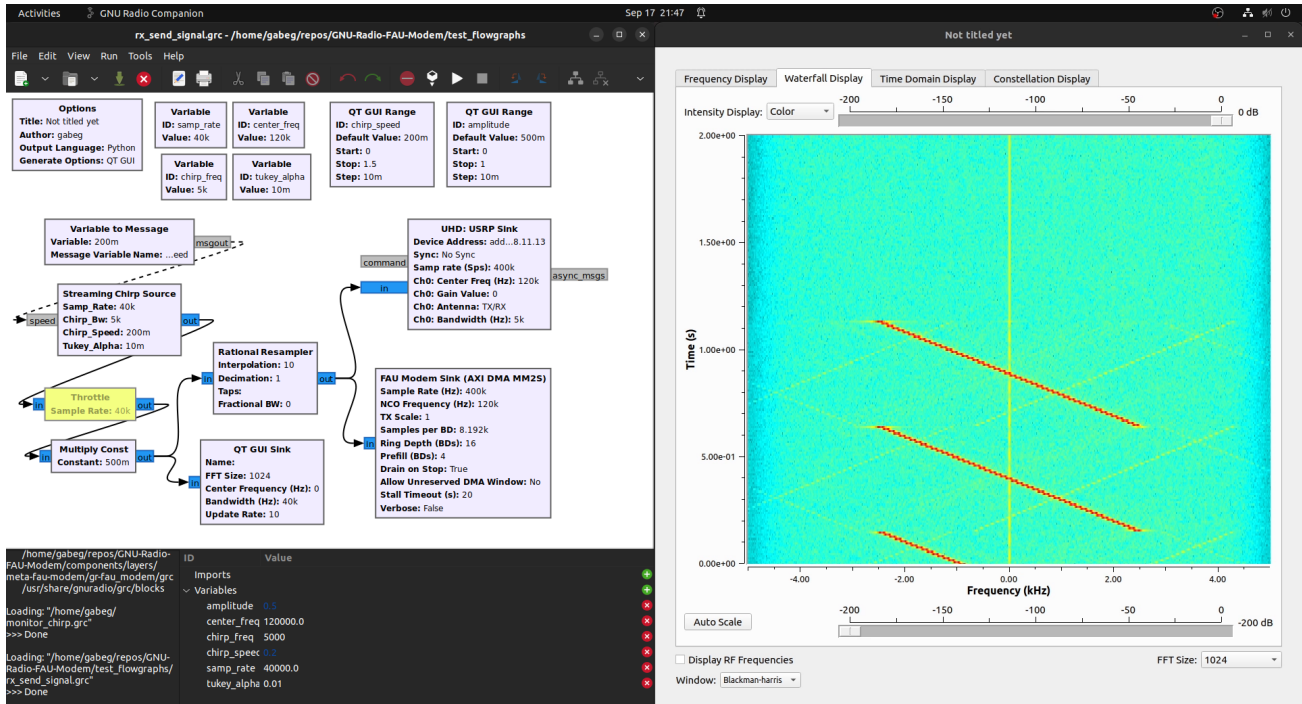


Figure 5. A chirp transmitter authored on the desktop and deployed to the modem. Left: the flowgraph in GNU Radio Companion, driving FAU Modem Sink at 400 kS/s with a 120 kHz NCO, alongside the QT GUI blocks and the USRP peripheral kept for desktop debugging. Right: the resulting transmission, captured from the modem’s output by a USRP N210 and displayed by a separate monitoring flowgraph.

cede arming of the block, never follow it.

The flowgraph executes in the foreground and is terminated by an interrupt issued on the console. Three requirements here are safety-critical. First, that interrupt is the only termination signal the tool will issue: there is no escalation to a kill and no force-stop control, as either would orphan the DMA burst that the teardown discipline of Section 4 is intended to retire; a termination that is not confirmed is reported as a distinct stalled state rather than escalated. Second, the interrupt strictly precedes logout from the session, since a logout issued while a flowgraph remains live signals the process in a manner that bypasses that teardown entirely. Third, completion is confirmed from the shell, by the nonce-tagged exit marker described above, rather than from any output emitted by the flowgraph itself, as only the shell establishes that the process was reaped and its cleanup executed.

Finally, as Figure 6 shows, the tool renders a flowgraph’s QT GUI *input* blocks — Range, Entry, Chooser, Check Box, and Push Button — as desktop widgets and pushes their values to the running flowgraph over the same console. GRC already generates a setter for every variable in a flowgraph, and all five of these blocks compile to an invocation of such a setter, so remote control reuses the stan-

dard code path with a serial link inserted into it. The dispatcher is `fau_ctl.py`, a module that rides in the payload as a sibling of the generated flowgraph rather than being installed into the board image. It is started by a GRC Snippet block in the `main_after_start` section, this being the only hook provided with a reference to the running flowgraph; a snippet body reaches that reference as `self` rather than by the flowgraph variable’s own name, since GRC emits the body as the body of a generated function. The dispatcher reads commands from the flowgraph’s standard input on the shared serial console, and the transform therefore forces GRC’s `run` option so that standard input remains available. A manifest, `ui_spec.json`, is carried in the same payload and enumerates the exported control identifiers as a whitelist: if it is absent or unparseable the whitelist is empty, every write is refused, and the flowgraph continues to run normally. Commands and replies carry the same per-run nonce as the exit marker, which is what separates them from the board’s own console output. A flowgraph containing no QT GUI input blocks receives no snippet, no sidecar files, and an unchanged launch command. QT GUI *sinks* remain permanently out of scope owing to bandwidth constraints.

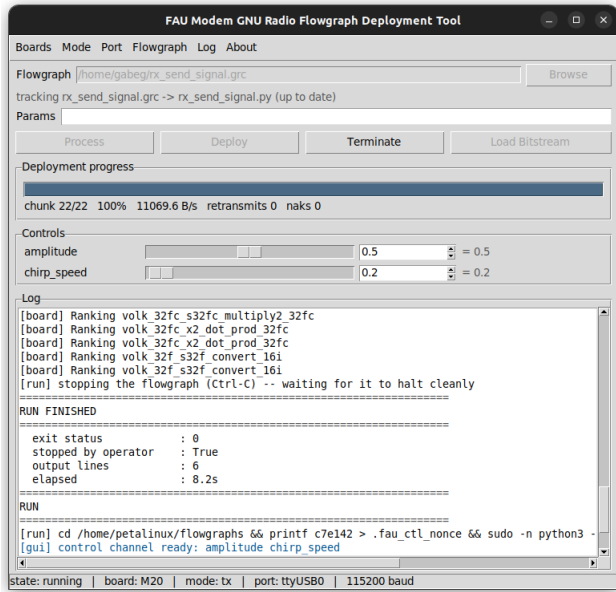


Figure 6. The deployment tool driving an M20 over a 115200-baud console. The transfer has completed with no retransmits or negative acknowledgements, the flowgraph is running on the modem, and the two QT GUI Range blocks of Figure 5 are rendered as live controls that push values to it over the same console. The log carries the board’s own output; the summary above it is the confirmed teardown of the previous run.

6. Limitations and Future Work

All signal processing above the DMA layer belongs in conventional GNU Radio blocks rather than within the source and sink: framing, BPSK modulation and demodulation, preamble search, carrier frequency offset correction, equalization, and bit error rate analysis. Excluding this functionality from the blocks has been an explicit scope boundary. With that boundary now enforced by a functioning DMA layer, a BPSK link has been demonstrated end to end between an S10 transmitting and an M20 receiving, with modulation, symbol synchronization, carrier recovery, and packet deframing implemented entirely as flowgraph blocks. That demonstration is preliminary. The receive chain requires further refinement, and the link has not been characterized under controlled conditions; reporting packet and bit error rates against a known channel is left to future work.

A second limitation concerns the multi-channel variants. Although the M10 and M20 carry several receive channels, the present blocks acquire only the first of them; a block that presents each receive channel as a separate stream remains to be written.

The DMA layer likewise awaits characterization. The run reported in Table 1 operates at a small fraction of the plat-

form’s rate ceiling and with several hundred milliseconds of ring depth. Establishing that the underrun and overrun detection of Section 4 behaves as designed requires operation nearer that ceiling, together with a deliberately induced underrun in which the detector is observed to fire.

We are also investigating a port of the platform to GNU Radio 4, which we expect to undertake once the packaging required to cross-build it for this target becomes available. Both the blocks and the deployment tool will require revision against its block API and its generated code; the DMA discipline of Section 4, however, is a property of the hardware rather than of the framework and is expected to transfer intact.

7. Conclusion

Making GNU Radio native to an SoC platform whose bitstream connects no DMA interrupt requires accepting that the source and sink blocks are themselves the driver, and designing them accordingly: a tail-bumped scatter-gather ring for exact underrun and overrun detection, bounded polling within the blocks’ work functions, a work function that never throws, and a teardown order that never resets a live engine. Rendering the resulting environment *usable* required an equal effort on the desktop side: because the blocks can be instantiated only on the board, and the board is reachable only over a serial console, iterative development is impractical without tooling that automates the compile–transfer–execute cycle. The combination of authoring in GRC, single-action deployment, execution on the modem, and live adjustment of flowgraph controls is what transforms the modem from a platform that requires driver authorship into one that supports research and development.

Acknowledgments

The authors thank Malik Sharif for his documentation of the modem’s programmable logic and for his collaboration in implementing the fabric features on which these blocks depend. Neither the source and sink blocks nor the DMA discipline described in Section 4 would have been possible without it.

References

- AMD Xilinx. *Zynq-7000 SoC Technical Reference Manual (UG585)*. AMD Xilinx, 2021. Revision 1.13. [Online]. Available: <https://docs.amd.com/v/u/en-US/ug585-Zynq-7000-TRM>.
- AMD Xilinx. *AXI DMA v7.1 LogiCORE IP Product Guide (PG021)*. AMD Xilinx, 2022. [Online]. Available: https://docs.amd.com/r/en-US/pg021_axi_

[dma](#).

AMD Xilinx. *PetaLinux Tools Documentation: Reference Guide (UG1144)*. AMD Xilinx, 2024. Release 2024.2.

Analog Devices, Inc. *AD8334/AD8335: Ultralow Noise Dual/Quad Variable Gain Amplifier with Preamplifier*. Analog Devices, Inc., 2020. Data Sheet, Rev. D.

Balister, Philip. meta-sdr: A Yocto layer for software defined radio. <https://github.com/balister/meta-sdr>, 2025.

Hermans, Jared, Sklivanitis, George, and Pados, Dimitris A. A first-of-its-kind low size, weight and power run-time reconfigurable underwater modem. In *Proceedings of the 6th Underwater Communications and Net-*

working Conference (UComms), Lerici, Italy, 2022. doi: 10.1109/UComms56954.2022.9905700.

Texas Instruments. *DAC7512: 12-Bit, Low-Power, Rail-to-Rail Out Digital-to-Analog Converter*. Texas Instruments, 2015. Data Sheet, SBAS276.

The GNU Radio Project. GNU Radio – the free and open source radio ecosystem. <https://www.gnuradio.org>, 2026.

The VOLK Project. VOLK: Vector-optimized library of kernels. <https://www.libvolk.org>, 2026.

The Yocto Project. Yocto Project reference manual, release 5.0 (scarthgap). <https://docs.yoctoproject.org>, 2024.