14th International Particle Accelerator Conference,Venice, Italy JACoW Publishing
ISBN: 978-3-95450-231-8 ISSN: 2673-5490 doi: 10.18429/JACoW-IPAC2023-THPL099

GNU Radio 4.0 FOR REAL-TIME SIGNAL-PROCESSING AND FEEDBACK APPLICATIONS AT FAIR

Ralph J. Steinhagen, Matthias Kretz, Alexander Krimm, (GSI Helmholtzzentrum, Germany),
Derek Kozel, Josh Morman (GNU Radio Project), Ivan Čukić, Frank Osterfeld (KDAB, Germany)

Content from this work may be used under the terms of the CC BY 4.0 licence (© 2022). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

Abstract

At FAIR, GNU Radio[1] is being used as part of the generic monitoring and first-line diagnostics for accelerator related devices, and to further support equipment experts, operation, and FAIR users in developing basic to advanced top-level measurement and control loops.

GNU Radio is a free and open-source software development toolkit supporting hundreds of low-cost to high-performance industrial digitizers with sampling frequencies ranging from a few MS/s to GS/s. At its core are directed signal flow graphs expressing arbitrary post-processing and feedback control loop logic that are both numerically highly efficient as well as providing an intuitive yet detailed nuts-and-bolts representation. This facilitates to inspect and/or to reconfigure existing systems by accelerator-, control- or other system domain-experts alike with little to no prior required programming experience.

This contribution describes the community-driven improvement and modernisation process leading to GNU Radio 4.0 supporting improved type-safety, improved performance, and new features such as event-driven data processing, nanosecond-level synchronisation using White-Rabbit, and slow feedback loops.

INTRODUCTION

The Facility for Antiproton and Ion Research (FAIR) aims to develop cutting-edge accelerator technology to investigate the fundamental properties of matter. In this context, GNU Radio, an open-source software-defined radio (SDR) framework, has emerged as a powerful tool for generic monitoring and first-line diagnostics for accelerator-related devices and beams [1–5]. The flexibility of its flow-graphs enable equipment experts, operation, and users of FAIR experiments in developing ad-hoc basic to advanced top-level measurement, real-time signal-processing, and feedback control loop applications. GNU Radio in its version 3.7 (GR3) is utilised in a wide range of applications within the FAIR accelerator facility, including monitoring of magnet, RF, fast kicker, and HV power supply systems. GR is also being used for AI-based monitoring of the primary electrical network, addressing non-intrusive load monitoring, transient monitoring, and compliance assurance with network operator regulations.

GRAPH-BASED SIGNAL PROCESSING

GNU Radio lowers the threshold for transferring lab or test bench prototypes into 24/7 operational use by sharing the same and easily reviewable logical structure, thanks to the intuitive graph representation and a more efficient, near-zero overhead API. Directed signal flow graphs form the core of the framework, enabling the expression of arbitrary post-processing and feedback control loop logics as shown in Figures 1 and 2. Flow graphs can be accessed programmatically and graphically without impacting processing performance. This approach allows for efficient numerical computation while offering an intuitive and detailed representation of the underlying system. During commissioning and early operation, the graph-based structure facilitates easy adaptation to accommodate new requirements or changes in hardware or software components, ensuring modularity, clarity, and performance. This is crucial since many accelerator experts do not necessarily possess the required RSE expertise needed for reliable long-term operation of these systems, while software engineers often have only limited or partial knowledge of the required specific accelerator sub-domain.

FAIR CONTRIBUTIONS AND ENHANCEMENTS TO GNU RADIO 4.0

FAIR has contributed its expertise in high-performance computing, research software engineering (RSE), real-time signal-processing, diagnostics, and feedback system applications to the modernisation of GNU Radio, resulting in the upcoming version, GNU Radio version 4.0 (GR4). Enhancements that were contributed to the GR4 codebase include improved type-safety, reduced overhead, and a more intuitive and extensible design. Improved type-safety and following C++ Core Guidelines minimises data corruption and segmentation faults, catching most issues and errors at compile-time before operational deployment. Modern programming techniques like the preference for ’composition’ over ’inheritance’ and the adoption of the latest C++ standard (ISO/IEC 14882:2020) result in better performance and a more efficient, near-zero overhead API. A clearer separation of concerns in class and interface hierarchies and well-defined buffer objects associated with ports lead to a leaner and more extensible design. Notable improvements contributed by FAIR include [2, 3]:

- Type-safe, high-performance lock-free IO buffers, which improved throughput and enhances the run-time performance by at least an order of magnitude as shown in Figure 3 and Table 1.
- Zero run-time overhead (node ’merge-API’) for sub-graphs whose topology is determined at compile-time.
- Out-of-the-box portable and register-width agnostic SIMD support with transparent handling of SIMD loop pro- and epilogue [6–10].

[1] https://www.gnuradio.org/

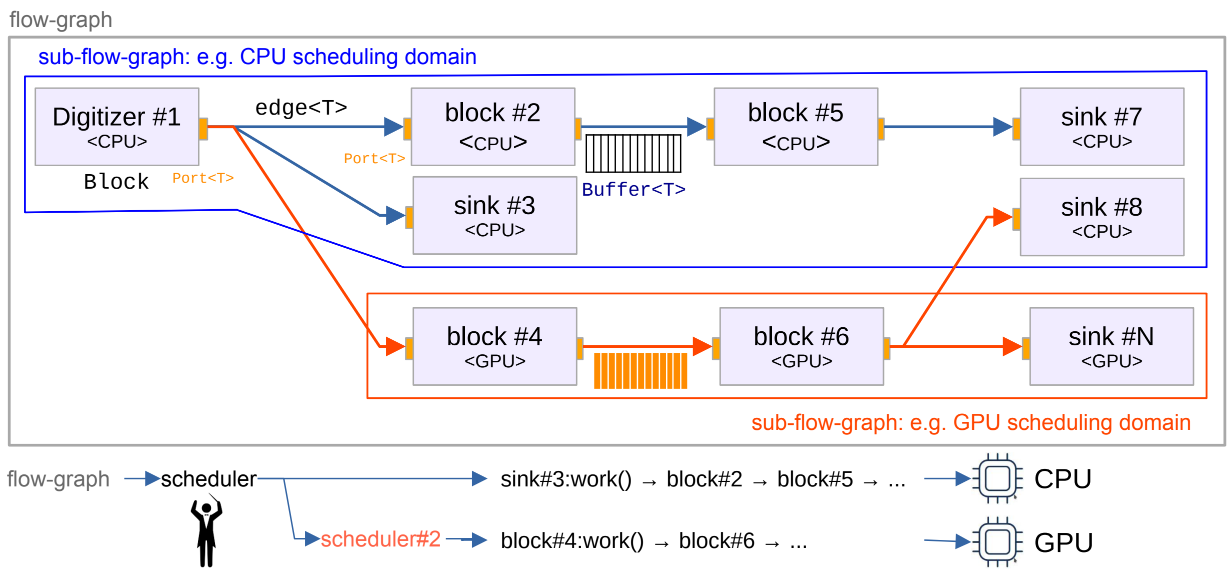


Figure 1: Schematic flow-graph based processing and scheduling concept.

- Tag based timing system integration (e.g., White-Rabbit) allowing nanosecond-level synchronisation and adding timing context information into the sample stream (e.g., using White-Rabbit technology [11]).
- Settings management supporting transactional settings updates and multiplexing of settings for different timing contexts.
- Extension of continuous sample-by-sample based signal processing to support synchronised chunked-data processing found in transient-recording, event-based, or recurring semi-periodic signal applications and slow feedback loops.
- Cross-platform support, including WebAssembly.
- Efficient, user-adaptable and pluggable work scheduler architecture adaptable to different execution domains (e.g., CPU, GPU, NET, FPGA, DSP, ...) and scheduling constraints (throughput, latency, real-time).
- Fostering of continuous improvement and investment in RSE practices and user communities for long-term maintenance, security, and improvements fostering collaboration and knowledge sharing.

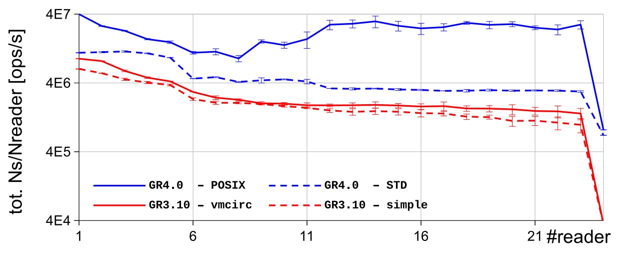
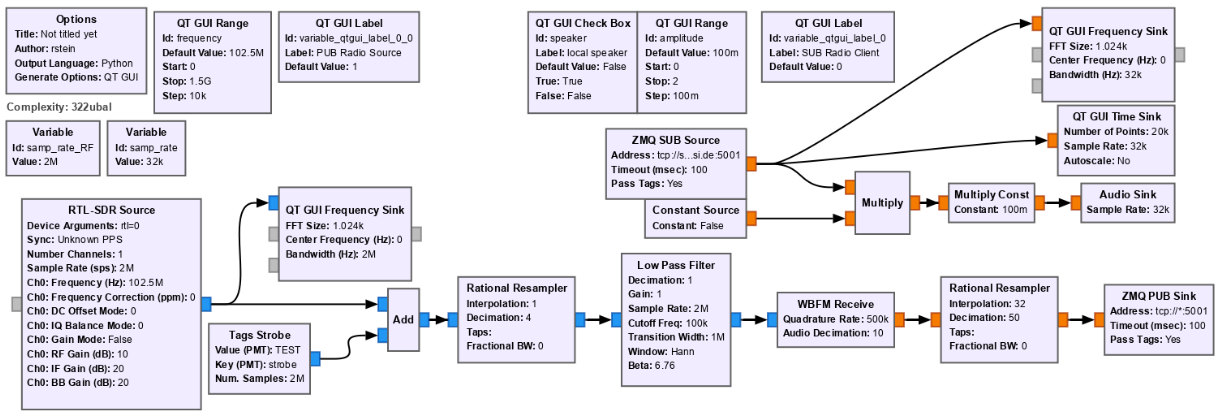


Figure 3: Throughput performance comparison of the GR3 and GR4 lock-free IO buffer implementations. The GR4 implementation for both, POSIX and strict fully std-C++ compliant implementation, an order of magnitude faster compared to the previous GR3 one. Large part of this performance improvement is due to the elimination of vtable-indirections and using a lock-free programming paradigm.



Options
Title: Not titled yet
Author: rstein
Output Language: Python
Generate Options: QT GUI

Complexity: 322ubal

QT GUI Range
Id: frequency
Default Value: 102.5M
Start: 0
Stop: 1.5G
Step: 10k

QT GUI Label
Id: variable_qtgui_label_0_0
Label: PUB Radio Source
Default Value: 1

Variable
Id: samp_rate_RF
Value: 2M

Variable
Id: samp_rate
Value: 32k

RTL-SDR Source
Device Arguments: rtl=0
Sync: Unknown PPS
Number Channels: 1
Sample Rate (sps): 2M
Ch0: Frequency (Hz): 102.5M
Ch0: Frequency Correction (ppm): 0
Ch0: DC Offset Mode: 0
Ch0: IQ Balance Mode: 0
Ch0: Gain Mode: False
Ch0: RF Gain (dB): 10
Ch0: IF Gain (dB): 20
Ch0: BB Gain (dB): 20

QT GUI Frequency Sink
FFT Size: 1.024k
Center Frequency (Hz): 0
Bandwidth (Hz): 2M

Tags Strobe
Value (PMT): TEST
Key (PMT): strobe
Num. Samples: 2M

Add

Rational Resampler
Interpolation: 1
Decimation: 4
Taps:
Fractional BW: 0

QT GUI Check Box
Id: speaker
Label: local speaker
Default Value: False
True: True
False: False

QT GUI Range
Id: amplitude
Default Value: 100m
Start: 0
Stop: 2
Step: 100m

QT GUI Label
Id: variable_qtgui_label_0
Label: SUB Radio Client
Default Value: 0

ZMQ SUB Source
Address: tcp://s...si.de:5001
Timeout (msec): 100
Pass Tags: Yes

Constant Source
Constant: False

QT GUI Frequency Sink
FFT Size: 1.024k
Center Frequency (Hz): 0
Bandwidth (Hz): 32k

QT GUI Time Sink
Number of Points: 20k
Sample Rate: 32k
Autoscale: No

Multiply

Multiply Const
Constant: 100m

Audio Sink
Sample Rate: 32k

Low Pass Filter
Decimation: 1
Gain: 1
Sample Rate: 2M
Cutoff Freq: 100k
Transition Width: 1M
Window: Hann
Beta: 6.76

WBFM Receive
Quadrature Rate: 500k
Audio Decimation: 10

Rational Resampler
Interpolation: 32
Decimation: 50
Taps:
Fractional BW: 0

ZMQ PUB Sink
Address: tcp://*:5001
Timeout (msec): 100
Pass Tags: Yes

Figure 2: GR3 graph example illustrating a basic FM-type receiver with ZeroMQ-based connectivity.

© Content from this work may be used under the terms of the CC BY 4.0 licence (© 2022). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

Table 1: Processing performance for graphs known at compile-time or runtime. The 'constraint' benchmark case limits the min/max number of samples to 'src(N=1024)->b1(N≤128)->b2(N=1024)->b3(N=32...128)->sink'. The merging of blocks forgoes the IO buffers if the connection and topology between processing block is known at runtime. In turn allows the compiler to see and optimise the merged algorithm. The merge-API is used for common topologies but allows nevertheless the introspection and easy (albeit offline) modifications.

Benchmark	**cache misses**	**mean**	**stddev**	**max**	**ops/s**
compile-time src->sink	1.3k / 3k = 46%	626 ns	110 ns	952 ns	16.4G
compile-time src->copy->sink	391 / 971 = 40%	957 ns	106 ns	1 us	10.7G
compile-time src->constraint->sink	398 / 960 = 41%	957 ns	103 ns	1 us	10.7G
compile-time src->mult(2)->divide(2)->add(-1)->sink	401 / 1k = 40%	3 us	108 ns	4 us	3.0G
compile-time src->(mult(2)->div(2)->add(-1))^{10}->sink	470 / 1k = 42%	41 us	189 ns	42 us	248M
runtime src->sink	9k / 174k = 5%	42 us	98 us	336 us	241M
runtime src->constraint->sink	20k / 648k = 3%	125 us	328 us	1 ms	81.7M
runtime src->(mult(2)->div(2)->add(-1))^{10}->sink	56k / 686k = 8%	127 us	28 us	198 us	80.6M
runtime src->mult(2)->... - via process_one(..)	24k / 663k = 4%	105 us	259 us	882 us	97.5M
runtime src->mult(2)->... - via process_bulk(..)	24k / 664k = 4%	152 us	358 us	1 ms	67.3M

```cpp
template<typename T>
requires (std::is_arithmetic<T>())
struct CustomNode : public node<CustomNode<T>> {
  IN<T>   in;
  OUT<T>  out;
  T       scaling_factor = static_cast<T>(1);
  std::string context;     // <-> multiplexing settings

  template<t_or_simd<T> V> // -> intrinsic SIMD support
  [[nodiscard]] constexpr V
  process_one(const V &a) const noexcept {
      return a * scaling_factor;
  }

/* alternate interface:
  process_bulk(std::span<const V> in, std::span<V> out) {
        // [..] user-defined processing logic [..]
  }
*/
};
ENABLE_REFLECTION_FOR_TEMPLATE_FULL((typename T),
  (CustomNode<T>), in, out, scaling_factor, context);
```

Listing 1: Custom GR4 Block Example. The IO-ports as well as block settings are defined as plain public data members. These can be programmatically accessed through compile-time reflection and to automatically generate the required meta information, (optional) string-based key-value mapping, (de-)serialisers etc. without any additional IDL specification or external code-generation tool (N.B. intrinsically done by by the C++ compiler preprocessor). This simplifies writing new processing blocks and reduces a common error source for most new developers.

GNU Radio includes many common signal processing blocks, often enabling full application development without writing any sample processing code. However, the modified API makes it easy to implement custom extensions if a specific function is unavailable or cannot be composed using existing blocks. A new block can be as simple as the following code example 1:

This approach simplifies implementation, unit testing, quality assurance, and reasoning about the system compared to other signal processing and front-end software frameworks. Consequently, transition from prototypes to 24/7 operational use is easier, leading to faster development cycles. Together, the improvements of GNU Radio 4.0 over 3.7 are instrumental in advancing FAIR's research capabilities and may also benefit other research facilities, industries, academia, and private enthusiasts alike.

IMPORTANCE OF SUSTAINABLE AND MAINTAINABLE RSE STANDARDS

The adoption of sustainable and maintainable research software engineering (RSE) standards is crucial for the long-term success and adaptability of a facility like FAIR. Sustainable RSE practices help bridge the gap between software engineers and domain experts, fostering collaboration and knowledge sharing. FAIR adheres to these standards to ensure that its software infrastructure remains robust, auditable, and secure while enabling rapid prototyping and adaptation to accommodate the facility's ever evolving research mission.

GR4 not only benefits FAIR by providing significant advancement to the most broadly used open source signal processing framework, benefitting researchers in industry, academia, hobbyists/enthusiasts, across a wide range of fields such as wireless communications, radio astronomy, and infosec research. The shared value created by GR4 fosters an ecosystem that spawns new ideas and fosters continuous improvement of the solutions offered. FAIR is grateful for the cooperation with the GNU Radio core team and its openness to new ideas, suggestions, and improvements that led to the development of GR4.

CONCLUSION

This paper invites readers to explore, adapt, and contribute to the GR4 project. The development and collaboration between FAIR and the GNU Radio core team have resulted in a robust and modernised solution for real-time signal-processing and feedback applications. With an emphasis on sustainable and maintainable RSE practices, GR4 offers a powerful and adaptable platform for addressing the evolving needs of FAIR and the broader research community.

© Content from this work may be used under the terms of the CC BY 4.0 licence (© 2022). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

ISBN: 978-3-95450-231-8 ISSN: 2673-5490 doi: 10.18429/JACoW-IPAC2023-THPL099

REFERENCES

[1] GNU Radio Development Team, *GNU Radio*, `https://www.gnuradio.org`, Accessed: May 3, 2023, 2001.

[2] GNU Radio Development Team, *GNU Radio GitHub repo*, `https://github.com/gnuradio/gnuradio`, 2001.

[3] FAIR & GNU Radio Development Team, *GNU Radio 4.0 – graph-prototype*, `https://github.com/fair-acc/graph-prototype`, 2022.

[4] R. J. Steinhagen *et al.*, "Generic Digitization of Analog Signals at FAIR – First Prototype Results at GSI", in *Proc. 10th International Particle Accelerator Conference (IPAC'19), Melbourne, Australia, 19-24 May 2019*, Melbourne, Australia, 2019, pp. 2514–2517. `doi:10.18429/JACoW-IPAC2019-WEPGW021`

[5] R. Steinhagen, A. Krimm, D. Ondreka, I. Cukic, and F. Osterfeld, "OpenDigitizer: Digitizer Modernisation using OpenCMW and GNU Radio 4.0 for FAIR", presented at IPAC'23, Venice, Italy, May 2023, paper THPL098, this conference.

[6] M. Kretz, "vir-simd", Tech. Rep., version 0.2.0, 2023. `doi:10.5281/zenodo.7892320`

[7] "Iso/iec 19570:2018, Programming Languages — Technical Specification for C++ Extensions for Parallelism", Standard, 2018. `https://www.iso.org/standard/70588.html`

[8] M. Kretz, "Extending c++ for explicit data-parallel programming via simd vector types", Frankfurt (Main), Univ., Ph.D. dissertation, 2015. `doi:10.13140/RG.2.1.2355.4323`

[9] M. Kretz and V. Lindenstruth, "Vc: A C++ library for explicit vectorization", *Software: Practice and Experience*, 2011. `doi:10.1002/spe.1149`

[10] M. Kretz, "Efficient use of multi- and many-core systems with vectorization and multithreading", Diplomarbeit, University of Heidelberg, 2009. `doi:10.13140/RG.2.2.12005.27360`

[11] P. Moreira, J. Serrano, T. Wlostowski, P. Loschmidt, and G. Gaderer, "White Rabbit: Sub-nanosecond Timing Distribution over Ethernet", in *Proc. of 2009 International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*, 2009, pp. 1–4. `doi:10.1109/ISPCS.2009.5340196`

Content from this work may be used under the terms of the CC BY 4.0 licence (© 2022). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.