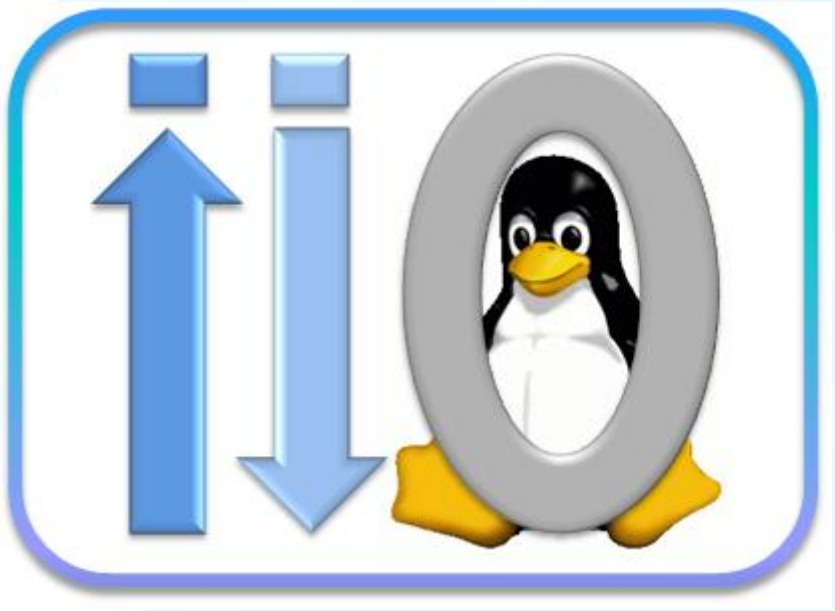
Libiio v1.0

Paul Cercueil <paul.cercueil@analog.com>

AHEAD OF WHAT'S POSSIBLE™

IIO: Industrial I/O framework

- IIO is a subsystem of the Linux kernel
  - drivers/iio/
- For all devices that don’t fit into the « hwmon », « audio » or « input » subsystems
  - ADCs, DACs, VGAs, PGAs, magnetometers, gyrometers, light sensors, pressure sensors, temperature sensors, etc etc…
- With wrappers in other subsystems (input, hwmon)
- Widely supported by many vendors
  - Analog Devices, Texas Instruments, Xilinx, Qualcomm, STMicroelectronics…



Libiio

- https://github.com/analogdevicesinc/libiio.git

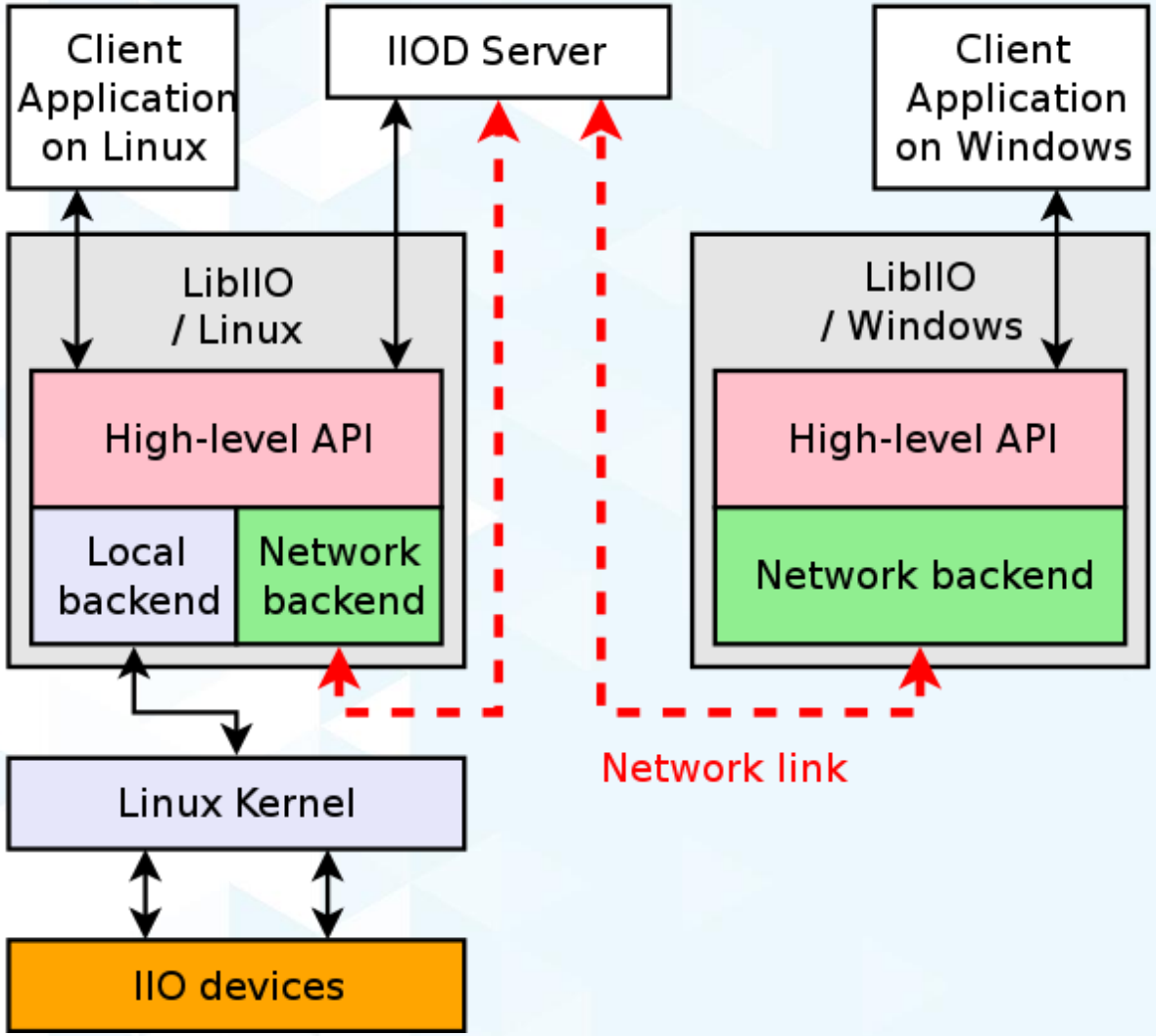
- Developed and maintained by Analog Devices since 2014
  - 56 different contributors as of September 2022
  - Latest version is v0.24, which will (hopefully) be the last 0.x version

- Goals
  - Interface with the IIO subsystem of the Linux kernel
  - Abstract away the low-level details of the IIO framework
  - High-level representation of IIO objects

- Design
  - Provides a simple, easy-to-use API
  - Portable C99, modular, thread-safe, runs on Linux, Windows, OSX, BSDs, ARM MBED…

Libiio – Backends

- Libiio has a clear separation between the high-level API, and backends
  - One application can run locally (on the target board), over the network, over USB, over UART…
  - Transparent to the application
    - Does not need to care about what backend it is running on
    - Same API
    - Doesn't even need to be recompiled!
  - Specific backends can be compiled in/out
- Allows running on non-Linux systems



Libiio – Why the need for a new version?

- In 8 years of existence, Libiio never broke the API or ABI
  - Sources written against Libiio v0.1 still compile with v0.24
  - Binaries compiled against Libiio v0.1 still run with v0.24
- Crust accumulated over the years…
  - Some interfaces needed to be redesigned (e.g. the scan API)
  - Some functions became de-facto obsolete
- Some changes were impossible to implement without breaking ABI
- Hence the need for a v2 (I mean v1)

Libiio v1.x – Design changes

- Libiio v1.x is in development in the “dev” branch
  - No ETA
  - API is pretty much stable, but not fixed in stone

- Major changes
  - Multi-buffer hardware support
  - A lower-level samples buffer handling mechanism
  - Modular backends
  - New asynchronous Libiio / IIOD protocol
  - New experimental userspace / kernel streaming interface

Libiio v1.x – Modular backends

- Libiio v0.24 supports creating a context over Ethernet, USB, UART, or from local IIO devices.
  - Dependencies on libxml2, libusb, libzstd, libserialport…
  - Annoying for distributions
- All backends (except local) can now be dynamically loaded
  - Installed as their own shared library
  - You could simply apt-get install “libiio1-backend-usb”
- Third-party Libiio backends are now possible
  - Libiio has a backends API and helper functions
  - All backends (except local) use exclusively Libiio’s regular and backends APIs

Libiio v1.x – Data streaming

- The old Libiio v0.x API was a bit too high-level
  - Did not give much control on the buffer operations
  - Impossible to support multi-buffer devices
- “iio_buffer” does not stream data anymore
  - “iio_buffer_refill”, “iio_buffer_push” are gone
- “iio_buffer” creation now takes the index of the hardware buffer
  - For use with multi-buffer devices
- Libiio now lets you enable / disable the buffer
  - Allows for synchronized transfers on multi-buffer devices

Libiio v1.x – Low-level iio_block API

- New low-level data streaming API: “iio_block”
  - Application allocates and manages its own iio_blocks
  - Enqueue (give block to Libiio / kernel), dequeue (request application access)
    - The data is transferred between enqueue and dequeue
    - The application must not access the block’s data in the meantime

- Process:
  - Enable desired channels for data streaming (iio_channel_enable)
  - Create a “iio_buffer”, specifying the hardware buffer index (iio_device_create_buffer)
  - Create a pool of blocks with the preferred size (iio_buffer_create_block)
  - Fill them with data if transmitting
  - Enqueue them (iio_block_enqueue)
  - Enable the buffer (iio_buffer_enable)
  - Dequeue one block (iio_block_dequeue), read or write the samples data, enqueue it back
  - Repeat

Libiio v1.x – High-level iio_stream API

- Applications generally don't need such a low-level and complex interface
- For these applications, the "iio_stream" API is better
  - "iio_buffer_create_stream": takes the number of blocks and their size (in samples)
  - "iio_stream_get_next_block": dequeue the next block in the queue, and enqueue the previous one. The returned block can then be read from, or written to
- Actually implemented on top of the low-level "iio_block" interface

Libiio v1.x – Asynchronous protocol

- Libiio v0.x protocol with IIOD was ASCII based
  - More data transferred (more important on slow interfaces e.g. UART)
  - Required a semantic parser
- It was also synchronous
  - You couldn’t send a new command before receiving the response to the previous one
  - Only one transfer direction is busy at any given time (not full-duplex)
  - Impossible to support asynchronous messages (e.g. IIO events)
- Now commands are performed by worker threads
  - One thread handles transmitting, one thread handles receiving
  - IIOD clients (e.g. network backend) get an ID, which identifies commands and responses

Libiio v1.x – New DMABUF interface

- The local backend now supports a new kernel / userspace interface based on DMABUF
  - Only two extra IOCTLs (ALLOC, ENQUEUE)
  - Dequeueing a block using poll()
  - Freeing a block using close()
  - Access the data using mmap()

- In theory…
  - Much faster than the current upstream interface (fileio based)
  - Would allow to dequeue a DMABUF from a IIO device, and enqueue it to a different one
  - Zero-copy!

- In practice:
  - Refused upstream as a simple kernel / userspace interface
  - I need to come up with the full zero-copy mechanism to justify the use of DMABUF
  - Or find another solution…

Libiio v1.x – Compatibility

- Libiio / IIOD communication:
  - Libiio v1.0 can talk to IIOD v0.x (including tinyIIOD) or v1.0
  - Libiio v0.x can talk to IIOD v1.0 as well

- Compatibility layer
  - Separate library, replaces libiio.so.0
  - Run apps built for Libiio v0.x (including IIOD) with Libiio v1.0

Libiio v1.x – Timestamp API

- Planned to be implemented in ADI's axi-dmac IP core
- Allow enqueueing blocks of samples at a very specific point in time
  - E.g. "transfer buffer in exactly +300 clock cycles"
- Allow detection of underruns / overruns

Libiio v1.x – Performance

- “iio_readdev” on a ZedBoard, over 1Gbps Ethernet, 32k samples blocks:
  - Libiio v0.24: 43 MiB/s
  - Libiio v1.0: 64 MiB/s
- Same experience with a Pluto SDR, USB backend:
  - Libiio v0.24: 24 MiB/s
  - Libiio v1.0: … 24 MiB/s
- Huge perf increase only when the link is the bottleneck

Libiio v1.x – Why should you care?

- gr-iio uses Libiio

- gr-iio will have to be updated when Libiio v1.0 is released
  - Support for multi-buffer IIO devices
  - Less overhead thanks to the async protocol
  - … that's about it

- In the meantime… it works fine with the compatibility layer
  - Get the benefits of Libiio v1.x for free
  - No rush to update it

Thank you!

Code: github.com/analogdevicesinc/libiio

Support: ez.analog.com

Doc: wiki.analog.com