Software Defined Radio
using the
Linux Industrial IO framework
- A Hardware Abstraction Layer -

Lars-Peter Clausen, Analog Devices
What is IIO?

- Industrial Input/Output framework
  - Not really just for Industrial IO
  - All non-HID IO
    - ADC, DAC, light, accelerometer, gyro, magnetometer, humidity, temperature, rotation, angular momentum, ...
- In the upstream Linux kernel since v2.6.32 (2009)
- Moved out of staging/ in v3.5 (2012)
- ~200 IIO device drivers (v3.19)
  - Many drivers support multiple devices
Why use IIO for SDR?

- Provides hardware abstraction layer
  - Allows sharing of infrastructure
  - Allows developers to focus on the solution
  - Allows application re-use
Why use IIO for SDR?

- Kernel drivers have low-level access to hardware
  - MMIO
  - Interrupts
- IIO provides fast and efficient data transport
  - From device to application
  - From application to device
  - From device to network/storage (soon)
IIOO Framework
IIO – Devices
IIO – Devices

- Main structure
- Typically corresponds to a physical hardware unit
- Represented as directories in sysfs
IIO – Attributes

Diagram showing a Device connected to an Attribute with a 1:n relationship.
IIO – Attributes

- Describe hardware capabilities
- Allow to configure hardware configuration
- Represented as files in sysfs
IIO – Channels
IIO – Channels

• Representation of a data channel
• Has direction, type, index and modifier
• Attributes provide additional information
  – scale, offset
  – Calibration data
  – Filters settings, hysteresis
  – ...

IIO – Buffers
IIO – Buffers

• Used for continuous data capture/transmit
• Channels can be enabled/disabled
• Channels specify their data layout
• /dev/iio:deviceX allows read()/write() access
• Configuration using sysfs files
• Support for different buffer implementations
  – Software FIFO
  – DMA Buffer
  – Device specific buffer
DMA is used to copy data from device to memory

mmap() is used to make data available in application

Allows low overhead high-speed data capture

Data is grouped into chunks (called DMA blocks) to manage ownership
  - Either application or driver/hardware owns a block
IIO – DMA buffer

Userspace - Kernelspace

Application

DMA controller

Incoming queue

Outgoing queue
Example – AD-FMCOMMS2-EBZ

- Software Defined Radio platform
- AD9361 Agile integrated transceiver
- 200 kHz - 56 MHz sample rate
- Tunable from 70MHz to 6GHz
- Full-duplex
- MIMO, 2x RX and TX
  - Each channel a set of 12-bit I and Q data
root@analog:/sys/bus/iio/devices# ls
iio:device0  iio:device1  iio:device2
iio:device3  iio:device4

root@analog:/sys/bus/iio/devices# cat */name
ad7291
ad9361-phy
xadcc
cf-ad9361-dds-core-lpc
cf-ad9361-lpc
# ls iio\:device1/
in_voltage_filter_fir_en
in_voltage_gain_control_mode_available
in_voltage_rf_bandwidth
in_voltage_rf_dc_offset_tracking_en
in_voltage0_gain_control_mode
in_voltage0_hardwaregain
in_voltage0_rssi
...
out_voltage_filter_fir_en
out_voltage0_hardwaregain
out_voltage0_rssi
...
filter_fir_config
...
in_temp0_input2
# ls iio\:device4/
buffer
in_voltage0_calibbias
in_voltage0_calibscale
in_voltage1_calibphase
in_voltage_sampling_frequency
in_voltage0_calibphase
in_voltage1_calibbiias
in_voltage1_calibscale
name
scan_elements
# ls iio\:device4/buffer/
enable
length

# ls iio\:device4/scan_elements/
in_voltage0\_en
in_voltage0\_index
in_voltage0\_type
in_voltage1\_en
in_voltage1\_index
in_voltage1\_type
Plumbing Layer
libiio

- High level C interface to IIO
- Abstracts away low level details of IIO kernel ABI
- Transpareently handles Low-Speed and High-Speed devices
  - Uses high speed interface when available
libiio

- Multiple backends
  - Local, directly using the IIO ABI
  - Network, uses network protocol to talk (remote) server (iiiod)
  - Debug, fake devices for testing
- Bindings for python, C#, (C++)
- Cross platform (Linux, Windows)
• Multiplexing between multiple readers/writers
• Support for remote clients (via TCP/IP)
• Applications do not need system level privileges
• Transparent from the applications point of view
iiod & libiio

Diagram showing the interaction between a client application on Linux or Windows and the iiOD server, with interfaces through LibiIO and High-level API.
IIO Scope

- Capture and display data
  - Time domain, frequency domain, constellation, cross-correlation
  - Markers
- Plug-in system for easy configuration GUIs
- Custom math operations (experimental)
IIO Scope – Capture Window
GNU Radio Plugin

• Two base classes
  – IIO Sink, Transmit data to a IIO device
  – IIO Source, Receive data from a IIO device
• Can select device and inputs/outputs
• Built-in support for Interpolation/Decimation
GNU Radio Plugin

- Possible to subclass IIOSink/IIOSource
  - e.g. to implement device specific specialization
    - GUI
    - Setting attributes
  - Current examples:
    - FMCOMMS2 Sink
    - FMCOMMS2 Source
GNU Radio Plugin

**IIo Device Source**
- **Hostname**: localhost
- **Device Name/ID**: iio:device0
- **Buffer size**: 32.768k
- **Decimation**: 1

**IIo Device Sink**
- **Hostname**: localhost
- **Device Name/ID**: iio:device0
- **Buffer size**: 32.768k
- **Interpolation**: 1

**FMComms2 Source**
- **Hostname**: localhost
- **LO Frequency**: 2.4G
- **Sample rate**: 1.536M
- **RF bandwidth**: 20M
- **Buffer size**: 32.768k
- **Decimation**: 1
- **Quadrature**: True
- **RF DC**: True
- **BB DC**: True
- **Gain Mode (RX1)**: Manual
- **Manual Gain (RX1)(dB)**: 64
- **Gain Mode (RX2)**: Manual
- **Manual Gain (RX2)(dB)**: 64
- **RF Port Select**: A_BALANCED

**FMComms2 Sink**
- **Hostname**: localhost
- **LO Frequency**: 2.4G
- **Sample rate**: 1.536M
- **RF bandwidth**: 20M
- **Buffer size**: 32.768k
- **Interpolation**: 1
- **Cyclic**: False
- **RF Port Select**: A
- **Attenuation TX1 (dB)**: 10
- **Attenuation TX2 (dB)**: 10
• FM radio receiver
Demo Setup

- AD-FMCOMMS3-EBZ (AD9361)
- Zed Board (ZYNQ FPGA) running Linux with AD9361 IIO driver and IIOD
- Laptop running GNU Radio with the IIO plugin
- Laptop connected to ZED board via Ethernet
Demo Flow

- Capture data in the FM radio spectrum
- Stream it from the ZED board to the Laptop
- Decode the FM radio stream in GNU radio
- Playback FM radio on the Laptops speaker
Demo GNU Radio Canvas

Options
ID: top_block
Generate Options: WX GUI
Realtime Scheduling: On

FMComms2 Source
Hostname: 127.0.0.1
LO Frequency: 93.3M
Sample rate: 1.536M
RF bandwidth: 20M
Buffer size: 131.072k
Decimation: 1
Quadrature: True
RF DC: True
BB DC: True
Gain Mode (RX1): Slow
Manual Gain (RX1)(dB): 64
Gain Mode (RX2): Slow
Manual Gain (RX2)(dB): 64
RF Port Select: A_BALANCED

Variable
ID: sample_rate
Value: 1.536M

WX GUI Text Box
ID: fm_station
Label: FM station
Default Value: 93.3M

Parameter
ID: hostname
Label: Hostname
Value: 127.0.0.1
Type: String

Parameter
ID: audio_device
Label: Audio device
Value: dmix:CAR...ntor,DEV=0
Type: String

Parameter
ID: decimation
Label: Decimation
Value: 1
Type: Int

Low Pass Filter
Decimation: 4
Gain: 1
Sample Rate: 1.536M
Cutoff Freq: 44.1k
Transition Width: 44.1k
Window: Hamming
Beta: 6.76

Audio Sink
Sample Rate: 48KHz
Device Name: audio_device

WBFM Receive
Quadrature Rate: 384k
Audio Decimation: 8
Live Demo
Further information

- https://github.com/orgs/analogdevicesinc
  - https://github.com/analogdevicesinc/libiio
  - https://github.com/analogdevicesinc/iio-oscilloscope
  - https://github.com/analogdevicesinc/linux
  - https://github.com/analogdevicesinc/gnuradio
- http://analogdevicesinc.github.io/libiio/
Q/A
IIO – Device Graph (Future)

• Use media controller framework to expose device topology
  – Allows userspace to auto-discover processing pipeline
  – Better support for standard components
• Use vmsplice and friends to provide zero copy
  – High-speed network streaming without CPU interaction
  – High-speed disk writes/reads without CPU interaction