gr-iio: Nuances, Advanced Features, and New Stuff

Travis Collins, PhD <travis.collins@analog.com>
Outline

- Intro to IIO, libIIO, and gr-iio
- libIIO isms in GNU Radio
- Accessing Custom IP
- libIIO Performance
- New Features of gr-iio
- gr-iio GREP
Intro to IIO, libIIO, gr-iio
Industrial Input/Output (IIO) Framework

- Intended to provide support for ADCs, DACs, sensors...

- Devices that fall into this category are:
  - ADCs
  - Accelerometers
  - Gyros
  - IMUs
  - Capacitance to Digital Converters (CDCs)
  - Pressure Sensors
  - Color, Light and Proximity Sensors
  - Temperature Sensors
  - Magnetometers/DACs
  - DDS (Direct Digital Synthesis)
  - PLLs (Phase Locked Loops)
  - Variable/Programmable Gain Amplifiers (VGA, PGA)

- Device which don’t fit into Hwmon or the Input Framework

- Many vendor build IIO drivers
  - Analog Devices
  - AMS
  - STMicroelectronics
  - Maxim
  - Murata
  - Qualcomm
  - Bosch
  - Texas Instruments
  - Xilinx
  - …
Access Sensors and Other Devices
IIO Driver To GNU Radio?
ADI ZIF Transceivers

PCB Component frequency wide turning range (70 – 6000 MHz) (Close to datasheet specs)
Radio to Host Interface
Pluto Gain Control

Shell Commands:

```
/sys/bus/iio/iio:device0 # cat name
ad8366-lpc
/sys/bus/iio/iio:device0 # echo 6 > out_voltage1_hardwaregain
/sys/bus/iio/iio:device0 # cat out_voltage1_hardwaregain
5.765000 dB
```
IIO Driver

©2019 Analog Devices, Inc. All rights reserved.
Goal: How can I control the device?

LO Frequency
Sample Rate
Gain Mode
TDD ENSM
RSSI
IIO – libiio

- System library
- Abstracts away low level details of the IIO kernel ABI
  - Kernel ABI is designed to be simple and efficient
  - libiio focuses on ease of use
- Provides high-level C, C++, C# or Python programming interface to IIO (Language bindings)

```python
import iio
ctx = iio.Context('ip:192.168.2.1')
phy = ctx.find_device('ad9361-phy')
```

For more information:

https://github.com/analogdevicesinc/libiio
http://wiki.analog.com/resources/tools-software/linux-software/libiio_internals
http://analogdevicesinc.github.io/libiio/

©2019 Analog Devices, Inc. All rights reserved.
IIO – libiio – Backends

- Support for backends
  - Backend takes care of low-level communication details
  - Provide the same API for applications
  - Transparent from the applications point of view
libIIO and applications

- iio-utils (command line tools)
- IIO-Scope
- MATLAB and Simulink
- GNU Radio
- 3rd part apps
  - SDR Angel
  - GQRX
  - SDR#
  - …

Language support for libIIO
- C/C++
- Python
- Nodejs (3rd party)
- C# (3rd part)
IIO Driver

ADALM-PLUTO, ADRV9361-Z7035 RF SOM, ADRV9364-Z7020 RF SOM, ADRV9009-ZU11EG RF SOM

FMCOMMS2, FMCOMMS3, FMCOMMS4, FMCOMMS5, ADRV9371 ADRV9375 ADRV9009

Xilinx Zed Board, Xilinx ZC702, Xilinx ZC706, Xilinx KC705, Xilinx KCU105, Xilinx ZCU102
Intel Arria 10 SoC, Intel Stratix 10 SoC

©2019 Analog Devices, Inc. All rights reserved.
IIO – libiio – Command line tools

- **iio_info**: Information about all IIO devices, backends and context attributes
  - `iio_info -s`
  - `iio_info -u ip:192.168.2.1`

- **iio_attr**: Read and write IIO attributes
  - `iio_attr -c ad9361-phy altvoltage0 frequency 2450000000`

- **iio_readdev**: Read samples from an IIO device
  - `iio_readdev -u usb:1.100.5 -b 100000 cf-ad9361-lpc | pv > /dev/null`

- **iio_writedev**: Write samples to an IIO device
  - `iio_readdev -b 100000 cf-ad9361-lpc | iio_writedev -b 100000 cf-ad9361-dds-core-lpc`

- **iio_reg**: Read or write SPI or I2C registers in an IIO device (useful to debug drivers)
  - `iio_reg adrv9009-phy 0`

©2019 Analog Devices, Inc. All rights reserved.
IIO-Scope

- Capture and display data
  - Time domain
  - Frequency domain
  - Constellation plot
  - Markers
  - Math operations
- Device configuration
- Plug-in system allow to create device or complex specialized GU
- Should support any IIO device
- Cross platform
IIO GNU Radio Support: gr-iio

- SDR
- Attributes
- Streams
- Math (Scopy)
- SDR
- Math (Scopy)
Hardware Support Through IIO

CUSTOM STREAM DEVICE
Contexts in context

- URIs are formatted as [Type]:[Address]
  - Network: ip:192.168.2.1
  - USB: usb:1.3.5
  - Local: local:

- Special
  - Local Autodiscover: “blank”
  - Network Autodiscover: ip:<hostname>
    - ip:analog, ip:pluto
Separate Contexts

ip:pluto.local

ip:192.168.2.1
Custom IP
Why should I care?

- Offload heaving processing tasks
- Tight timing control over hardware
  - Frequency hopping
  - Radar processing
  - Time division MAC
- Going embedded
- Standard FPGA tools do not provide great software control mechanisms
Example: Can the AD936X frequency hop?

- Paper at 2016 GRCon

- Technique
  - Start hop:
    - `tb.usrp.setcenterfreq(targetfreq)`
  - Check Settling:
    - `tb.usrp.get_sensor("lo_locked")`

- Using this API can be as bad as 100 ms

<table>
<thead>
<tr>
<th>MB</th>
<th>DB</th>
<th>$t_{\text{tune}}$ (µs)</th>
<th>$t_{\text{lock}}$ (µs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>N210</td>
<td>WBX</td>
<td>510</td>
<td>290</td>
</tr>
<tr>
<td>N210</td>
<td>SBX</td>
<td>483</td>
<td>289</td>
</tr>
<tr>
<td>X310</td>
<td>UBX</td>
<td>610</td>
<td>418</td>
</tr>
<tr>
<td>B100</td>
<td>WBX</td>
<td>782</td>
<td>573</td>
</tr>
<tr>
<td>B100</td>
<td>SBX</td>
<td>772</td>
<td>546</td>
</tr>
<tr>
<td>B200</td>
<td>-4</td>
<td>3310</td>
<td>117</td>
</tr>
<tr>
<td>B210</td>
<td>-4</td>
<td>3318</td>
<td>124</td>
</tr>
<tr>
<td>E310</td>
<td>-4</td>
<td>31258</td>
<td>285</td>
</tr>
<tr>
<td>HackRF</td>
<td>-4</td>
<td>215</td>
<td>0</td>
</tr>
</tbody>
</table>
AD936X: Fast Frequency Hopping

- Part capable of ~25us LO changes
  - To go faster requires external LO
- Fast hopping named by using “Fastlock Profiles”
- Profile selection possible through SPI or GPIO
  - Limited to 8 profiles on chip
  - Others can be sideloaded at runtime from BBP

**Default LO Control**

If \(|\text{Freq}_N - \text{Freq}_N+1| > 100\text{MHz}\)
- Change VCO
- Recal VCO

Else
- Change VCO

**Fastlock LO Control**

If \(|\text{Freq}_N - \text{Freq}_N+1| > 100\text{MHz}\)
- Load Fastlock Profile

Else
- Change VCO

<table>
<thead>
<tr>
<th>Selectable Profiles</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIN Control</td>
</tr>
<tr>
<td>SPI</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AD936X</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7</td>
</tr>
</tbody>
</table>

©2018 Analog Devices, Inc. All rights reserved.
Pin Control

Figure 63. AD9361 Interface
Controller for Pins

Documentation: https://wiki.analog.com/resources/eval/user-guides/adrv936x_rfsom/tutorials/frequency_hopping
Driver Options

- **MW AXI-MM IIO**
  - Provides direct register access
  - Patch available for ADI kernel
    - Enabled in MathWorks kernel (E310, FMComms, ADRV)
  - Great for starting out, testing, debug
  - Driver logic lives in application

- **Custom IIO Driver**
  - Complex logic built into driver
  - Better portability
  - Not a great starting point
GNU Radio Example

► See video

► Video uses ADRV9361-Z7035 below
libIIO Performance
Local Buffers: High-Speed mmap interface
Remote Contexts: Synchronous Transfers

**Remote App**
- Wait for request
- Wait for transfer
- Wait for request
- Wait for transfer

**iidod**
- Initialize kernel buffers
- Pull first buffer
- Pull next buffer from ring

Remote Contexts: Synchronous Transfers

Remote App

- Wait for request
- Wait for transfer
- Wait for transfer
- Wait for transfer

iiod

- Initialize kernel buffers
- Pull first buffer
- Pull next buffer from ring
- Pull next buffer from ring

Performance: IP vs Local

IIO Local Context PackRF

Overflow %

Rx Rate (MSPS)

Buffersize=1024
Buffersize=2048
Buffersize=4096
Buffersize=8192
Buffersize=16384

IIO Local Context PackRF

Overflow %

Rx Rate (MSPS)

Buffersize=1024
Buffersize=2048
Buffersize=4096
Buffersize=8192
Buffersize=16384
New Features
More Filtering Options

- New filter tab
  - Off: no filter
  - Auto: Preset filters
  - File: Load from ftr file
  - Design: Create filter on the fly based on:
    - RF bandwidth
    - Sample rate
    - F Stop
    - F Pass
    - From libad9361 library
Underflow/Overflow Indicators

Generating: `/home/tcollins/top_block.py`

Executing: `/usr/bin/python2 -u /home/tcollins/top_block.py`

©2019 Analog Devices, Inc. All rights reserved.
Pluto and ADRV FPGA Filters

Decimation/Interpolation Filter exist in FPGA for Pluto and new ADRV designs

- Automatically configured now from GR
- Allows rates down to 65104 Hz
DDS Controller Block
DDS Controller Block Demo

- See video
- PlutoSDR used for example
GREP: gr-iio
GREP to mainline gr-iio

GREP 0017 -- Add support for IIO devices

- Original Author: Travis Collins travis.collins@analog.com
- Champion: Travis Collins travis.collins@analog.com
- Status: Active

History:
- 22-Feb-2019: Initial Draft
- 28-Feb-2019: Changed Status to Active

Abstract

GNU Radio is an invaluable tool for data processing, especially for data sources that can provide real-time data like sounds devices and software-defined radios. Such support today is provided by in-tree audio, gr-uhd, gr-comedi, and gr-video-sdl. However, a large category of sensors/DAC/ADC among the Industrial Input/Output Framework of the Linux kernel is not supported in-tree today.

Making it easier for users!
IIO GNU Radio Support: gr-iio

SDR
Attributes
Streams
Math (Scopy)

Math (Scopy)
pyadi-iio

- IIO version

```python
# Setup contexts
try:
    ctx = iio.Context('ip:192.168.2.1')
except:
    print("No device found")
    sys.exit(0)

ctrl = ctx.find_device("ad9381-phy")
txdac = ctx.find_device("cf-ad9381-dds-core-lpc")
rxadc = ctx.find_device("cf-ad9381-lpc")

# Configure transceiver settings
rx0 = ctrl.find_channel("altvoltage0", True)
rx0.attrs["frequency"].value = str(int(RXLO))
tx0 = ctrl.find_channel("altvoltage1", True)
tx0.attrs["frequency"].value = str(int(TXLO))

tx = ctrl.find_channel("voltage0")
tx.attrs["rf_bandwidth"].value = str(int(RXBN))
tx.attrs["sampling_frequency"].value = str(int(RXFS))
tx.attrs["hardwaregain"].value = '-30'

rx = ctrl.find_channel("voltage0")
rx.attrs["rf_bandwidth"].value = str(int(TXBN))
rx.attrs["sampling_frequency"].value = str(int(TXFS))
rx.attrs["gain_control_mode"].value = 'slow_attack'
```

- pyadi-iio version

```python
# Create radio
sdr = adi.Pluto()

# Configure properties
sdr.rx_rf_bandwidth = 4000000
sdr.rx_lo = 200000000
sdr.tx_lo = 200000000
sdr.tx_cyclic_buffer = True
sdr.tx_hardwaregain = -30
sdr.gain_control_mode = "slow_attack"
```

pip install pyadi-iio
(Don’t pip install iio)
Thank you!

**Code:** github.com/analogdevicesinc  
**Support:** ez.analog.com  
**Doc:** wiki.analog.com
References

- https://www.ti.com/store/ti/en/p/product/?p=LM335Z/NOPB&utm_source=google&utm_medium=cpc&utm_campaign=ecm-tistore-invf-GMC_US-cpc-store-google-wwe&utm_content=GMC&ds_k=PRODUCT_GROUP&DCM=yes&gclid=CjwKCAjwq4fsBRBnEiwANTahcH1fAXFxufRlK8 Zi4XudpDhnjjTaN6VP2fb5Z7mGvP7TmWu2vERoCvb0QAaD_BwE

- https://www.mouser.com/new/omron-electronics/omron-d6t/

- https://www.digikey.com/product-detail/en/adafruit-industries-llc/163/1528-1120-ND/5353580?WT.srch=1&gclid=CjwKCAjwq4fsBRBnEiwANTahcBaKgAgJWOhSFFioC-Hd5_aZgPyoLWK86AXxRjP_BE2QOL2iwFW7LRoCD0AQAvD_BwE#images