Kuiper Linux Distribution - simplify hardware prototyping with GNU Radio

Michael Hennerich
Jon Kraft
(Software Defined) Radio Systems

- Are much more than Digitizers!
- Lots of frontend **active** RF components which require configuration and runtime controls
  - Programmable Amplifiers and Attenuators (VGA/PGA/DSA)
  - Synthesizers, PLLs, UP/DN Converters
  - RF Switches (TRX, Filters)
  - Tunable Filters, Beamformers
  - PA control
  - Bias/Offset control
  - System Monitoring (V, I, T)
  - etc.
Typical prototyping challenges

- **Observation**
  - Lots of interconnected evaluation/protype circuit boards
  - Lots of different supply voltages needed
  - No homogeneous way to control this mess

- **Implication**
  - Loss of RF performance
  - Mechanically difficult to test this in the field
  - Control nightmare
    - Every active component has its own Windows EVAL software
    - Sometimes missing software and drivers
    - No way to remote control
    - No easy way to integrate command and control into a GRC flowgraph

- **Possible Solutions**
  - Custom PCB design & System controller + Firmware
    - High cost and risk, typically comes 2nd after prototyping phase
  - Modular RF and Microwave building blocks
  - **Commercially of the shelf control hardware and FOSS software**
  - Industry standard soft- and hardware components
  - Single and powerful Embedded Controller which can control all components at a very high level
Prototyping a FMCW Radar (in less than a day)
FMCW Radar

- **FMCW**: Frequency Modulated Continuous Wave
- Different than Pulsed Radars
- **Beat frequency** proportional to time delay -> FFT for range estimation

**Basic Architecture**

\[ R = \frac{c T_s}{2B} (f_b) \]

- \( R \) = distance to target
- \( c \) = speed of light (3E8 m/s)
- \( T_s \) = transmit frequency ramp rate
- \( B \) = bandwidth of the frequency ramp
- \( f_b \) = transmit to receive frequency difference
Build your own 10 GHz FMCW radar

![Diagram of 10 GHz FMCW radar setup](image-url)
Build your own 10 GHz FMCW radar
Hello World FMCW!

Digitizer Control (Pluto, USRP, etc.)

RF/MW Frontend Control

Options
Title: RAD-Pluto-ADF4159
Output Language: Python
Generate Options: QT GUI

QT GUI Range
IDs: LO
Label: LO Frequency
Default Value: 2.2G
Start: 2.2G
Step: 2.2G
Step: 1k

Variable
ID: samp_rate
Value: 600k

Variable
ID: rt_size
Value: 1.024k

PlutoSDR Source
I/O context URI: "local"
LO Frequency: 2.2G
Sample Rate: 600k
Buffer size: 123,576k
Quadrature True
RF DC Correction: True
BB DC Correction: True
Gain Mode (RX): Slow Attack
Filter Configurations: Auto
RF Bandwidth (Hz): 20M

Signal Source
Sample Rate: 600k
Waveform: Cosine
Frequency: 100k
Amplitude: 1
Offset: 0
Initial Phase (Radians): 0

PlutoSDR Sink
I/O context URI: "local"
LO Frequency: 2.2G
Sample Rate: 600k
Buffer size: 1,048,576k
Center Frequency (Hz): 0
Bandwidth (Hz): 600k
Update Rate: 10

Summary
Variable to Message
Variable: 12.2k
Message Variable Name:...ncy

Variable to Message
Variable: 0
Message Variable Name:...own

Variable to Message
Variable: 125M
Message Variable Name:...mp

Variable to Message
Variable: 13k
Message Variable Name:...ima

I/O Attribute Sink
I/O context URI: "local"
Device Name/ID: adf4159
Attribute Type: Channel
Channel Name: statesage

I/O Attribute Chooser
ID: ramp_mode
Label: Ramp Mode
Num Options: 5
Default options: disabled
Option 0: disabled
Label 0: disabled
Option 1: continuous_sawtooth
Label 1: continuous_sawtooth
Option 2: continuous_triangle
Label 2: continuous_triangle
Option 3: single_end burst
Label 3: single_end burst
Option 4: single_ramp burst
Label 4: single_ramp burst

I/O Attribute Topple Button
ID: powerdown
Label: Powerdown
Default Value: 0
Initial State: Released
Pressed: 1
Released: 0

I/O Attribute Range
ID: freq,deviation_stop
Label: Frequency deviation stop
Default Value: 125k
Start: 0
Step: 1M
Step: 1k

I/O Attribute Range
ID: freq,deviation_range
Label: Frequency deviation range
Default Value: 125M
Start: 1G
Step: 1G
Step: 10k

I/O Attribute Range
ID: freq,deviation_time
Label: Frequency deviation time
Default Value: 1k
Start: 100
Step: 100k
Step: 10
Example #1: RPi as control Bridge

Your PC with: GRC, libiio
Network connection to remote HW

Any network connection
• Direct Ethernet cable
• USB-Ethernet
• Wireless

RPi, device drivers, IIOD (Server)

Controlled RF HW

USB connection
• Virtual network
• Direct USB
• Serial
• Mass Storage
• OTG

RF Digitizer/Transceiver
Example #2: Arm-Based Computing Platforms

Your SBC Raspberry Pi running Graphical Desktop Environment and GRC

USB connection
- Virtual network
- Direct USB
- Serial
- Mass Storage
- OTG

HDMI

Controlled RF HW

RF Digitizer/Transceiver

SPI/I2C/GPIO
Example #3: All digital back to the same SoC/FPGA

- All digital back to the same SoC
  - TRX data
  - TRX control
  - Clock control
  - Microwave/RF Control

- Enables:
  - Single / Cohesive environment to interface with digitizers and RF/Microwave
  - System level verification
  - Easier Power Supplies
  - Synchronization between digitizer and MW
  - Sync between data and control
  - Easier device driver verification and development
  - Control MW from GnuRadio, MATLAB or Python
  - Simplifies path to production (all SW together, all HW together)

- FPGA Carrier
- ADI ADC Card
- FMC X-Microwave Bridge [1]
Single cohesive software solution
A distribution based on Raspbian but NOT only for Raspberry Pi [2].

- It incorporates **Linux device drivers for ADI products [6]** and is created with **ease of use** in mind.
- Supports multiple ADI Circuit Note and Eval boards with FMC, RPi HEAD, Arduino and PMOD connectors.

**Minimize the barriers** to integrating hardware devices into a Linux-based system.

ADI Kuiper Linux solves this problem, and includes a host of additional applications, software libraries, and utilities.
Kuiper Imager Utility [4]

- Windows/Linux/Mac
- Downloads Image file from the internet
- Writes image to removable media storage device
- Eases customization for a specific Base Platform + Add-On or Attach
- Released in the coming weeks

$unzip image_2022-08-04-ADI-Kuiper-full.zip
$sudo dd if=2022-08-04-ADI-Kuiper-full.img of=/dev/sdX bs=4M
Customizing?

- While Raspbian targets Raspberry Pi platform boards, ADI Kuiper Linux supports several other platforms in addition:
  - Arduino form factor ARM based FPGA platforms such as
    - Intel/TerASIC DE10-Nano
    - Xilinx/Digilent Cora Z7
  - Most popular FMC FPGA carriers from Xilinx/AMD and Intel with ARM/ARM64 support
    - Zynq7000 (Zed board, ZC706, ZC702)
    - MPSoC (ZCU102)
    - SoC FPGA (A10Soc, A5Soc)

- Configure the Kuiper SD card to be used with a specific Hardware:
  - Base Platform + Add-On or Attach
  - Move boot files from folders to the root of the FAT partition so that they can be picked up by the first stage bootloader (Intel or Xilinx/AMD FPGAs)
    - FPGA Bitstreams
    - Linux kernel blobs
    - Devicetree blobs
  - Instruct early-stage bootloader to load a certain overlay (Rpi: Edit config.txt)
Devicetree (overlay)

- Tree data structure with nodes that describe the physical devices in a system.
- Describes device information in a system that cannot be dynamically detected.
- Organized in nodes and properties
- Overlays enables addition of extra nodes to the live device tree of an embedded Linux system.

- **dts, dtsi**: devicetree source
  - Human-readable
  - Textual
  - Used when editing devicetree files

- **dtb, dtbo**: devicetree blob
  - Machine-readable
  - Binary
  - Flattened
  - The Device tree compiler (dtc) generates dtb, dtbo

Example file: rpi-adf4159-overlay.dts [3][5]

```
/dts-v1/;
/plugin/;

&spi0 {
  #address-cells = <1>;
  #size-cells = <0>;
  status = "okay";

  adf4159:adf4159@0 {
    compatible = "adi,adf4159";
    reg = <0x0>;
    spi-max-frequency = <12500000>;
    
    /* Clocks */
    clocks = <&clkin>;
    clock-names = "clkin";
    clock-output-names = "rf_out";
    #clock-cells = <0>;
    adi,power-up-frequency-hz = /bits/ 64 <6000000000>;
    
    /* --- snip ---- */
  }
};
```

```
$dtc -I dts -O dtb rpi-adf4159-overlay.dts > rpi-adf4159.dtbo
```
The Analog Devices kernel tree contains a number of such Device Tree Overlays in the `arch/arm/boot/dts/overlays` folder.

Each of those overlays, stored in `.dts` file gets compiled into a `.dtbo` files using `dtc`.

On the SD card those `.dtbo` files reside in `BOOT\overlays` or from within the rootfs at `/boot/overlays`

Those `.dtbo` can be loaded and applied to the main Device Tree by adding the following statement to the Rpi config.txt file:

```
Load at boot:
- config.txt file is read by the early-stage boot firmware
- dtoverlay=overlay-name,overlay-arguments
  EXAMPLE:
  dtoverlay=rpi-ADF4159

Load from command line:
- dtoverlay <overlay> [param=<val>...]
  EXAMPLE:
  root@analog:~# dtoverlay rpi-ADF4159
  root@analog:~# dtoverlay -l
  Overlays (in load order):
  0: rpi-ADF4159
  root@analog:~# systemctl restart iod"
What is IIO?

- **Linux kernel Industrial Input / Output framework**
  - Not really just for Industrial IO
  - All non-HID IO
  - ADC, DAC, TRX, light, accelerometer, gyro, magnetometer, humidity, temperature, pressure, rotation, angular momentum, chemical, health, proximity, counters, amplifiers, synthesizers, RF Up/Down converters, tunable filters, etc.

- In the upstream Linux kernel for more than 10 years.

- Mailing list:
  - linux-iio@vger.kernel.org

IIO – Devices

- Main structure
- Typically corresponds to a single physical hardware device
- Represented as directories in sysfs

/sys/

bus/
iio/
iio:device0/
iio:device1/
iio:deviceX/

©2022 Analog Devices, Inc. All rights reserved.
IIO – Attributes

- Describe hardware capabilities
- Allow to configure hardware features
  - SAMPLING_FREQUENCY
  - POWERDOWN
  - PLL_LOCKED
  - SYNC_DIVIDERS
  - etc.
- Represented as files in sysfs

```
# ls /sys/bus/iio/devices/
iio:device0  iio:device1  iio:device2  iio:device3  iio:device4
# cat /sys/bus/iio/devices/*/name
adm1177
ad9361-phy
xadc
cf-ad9361-dds-core-lpc
cf-ad9361-lpc
```

©2022 Analog Devices, Inc. All rights reserved.
IIO – Channels

- Representation of a data channel
- Has direction, type, index and modifier
  - Direction
    - IN
    - OUT
  - Type
    - IIO_VOLTAGE
    - IIO_TEMP, etc.
  - Index
    - 0..N
  - Modifier
    - IIO_MOD_I, IIO_MOD_Q
- Channel Attributes provide additional information
  - RAW
  - SCALE
  - OFFSET
  - FREQUENCY
  - PHASE
  - HARDWARE_GAIN
  - etc.

Example: Read voltage from ADC Channel X in mV

\[ \text{VoltageX\_mV} = (\text{in\_voltageX\_raw} + \text{in\_voltageX\_offset}) \times \text{in\_voltageX\_scale} \]
VGA/PGA Gain or DSA Attenuation Control

Shell Commands:

```
/sys/bus/iio/iio:device0  # cat name
ad8366
/sys/bus/iio/iio:device0  # echo 6 > out_voltage1_hardwaregain
/sys/bus/iio/iio:device0  # cat out_voltage1_hardwaregain
5.765000 dB
```
ADF4159
Direct Modulation/Fast Waveform Generating, 13 GHz, Fractional-N Frequency Synthesize

Shell Commands:

```
root@analog:~# iio_attr -c adf4159 altvoltage0
dev 'adf4159', channel 'altvoltage0' (output), attr 'frequency', value '6000000000'
dev 'adf4159', channel 'altvoltage0' (output), attr 'frequency_deviation_range', value '0'
dev 'adf4159', channel 'altvoltage0' (output), attr 'frequency_deviation_step', value '5960'
dev 'adf4159', channel 'altvoltage0' (output), attr 'frequency_deviation_time', value '0'
dev 'adf4159', channel 'altvoltage0' (output), attr 'powerdown', value '0'
dev 'adf4159', channel 'altvoltage0' (output), attr 'ramp_mode', value 'disabled'
dev 'adf4159', channel 'altvoltage0' (output), attr 'ramp_mode_available', value 'disabled continuous_sawtooth single_sawtooth_burst single_ramp_burst'
```
**iio_attr_sink**

- Generic writer for attributes of IIO devices.

- This block allows for updating of any IIO attribute that is writable. This includes channel, device, device buffer, device debug, and direct register attributes. All messages must be a **pmt dictionary** where the key is the attribute to update and the value is the value to be written. Messages can be an array of dictionaries or a single dictionary.
blocks_var_to_msg (since 3.10)

- This block will monitor a variable, and when it changes, generate a message.

- This block has a callback that will emit a **message pair** with the updated variable value when called. This is useful for monitoring a GRC variable and emitting a message when it is changed.
msg_to_iio_dict (custom Python block)

- Gr::basic_block

- This block will convert a PMT message pair, into a PMT dictionary.

- Useful to interface the blocks_var_to_msg with the iio_attr_sink block.
Conclusions

- You don't need to be a Linux expert to enable and use ready made IIO drivers
- Gr-iio much more than controlling a transceiver/digitizer
- Gr-iio iiio_attr blocks should also accept PMT message pairs

- More interest in gr-iio and next generation libiio?
  - Latest libiio v0.24 also supports HWMON devices
  - Attend: Libiio 1.0 design and how it will affect GNU Radio, Paul Cercueil, 9/28/22
    - Multibuffer, Async Protocol, DMABUF compatible interface, Timestamp API (WIP)
References

Ahhh, technology. We can't find that page.

Thanks
Q & A