Implementation of AD-FMCOMMS1-EBZ module in GNU Radio

Daniel F. Contreras-Hernández[a], Víktor I. Rodríguez-Abdalá[a], Jorge Flores-Troncoso[a], Remberto Sandoval-Aréchiga[a], Salvador Ibarra-Delgado[a]

[a]*Universidad Autónoma de Zacatecas (UAZ), Unidad Académica de Ingeniería Eléctrica, Centro de Investigación, Innovación y Desarrollo en Telecomunicaciones*
Av. López Velarde 801, Col. Centro, Zacatecas , Zac., México, 98000.
{abdala,jflorest,rsandoval,sibarra}@uaz.edu.mx

2018 Published by $\mathcal{DIFU}_{100}ci@$ *http://difu100cia.uaz.edu.mx*

Abstract

The AD-FMCOMMS1-EBZ provides an analog front-end for a wide range of compute-intensive FPGA-based radio applications that addresses a broad range of research, academic, industrial and defense applications, and with GNU Radio framework the system modeling could be deployed in general purpose computers. This paper describes the implementation of a signal processing block in GNU Radio to allow the reception of signals from the evaluation board using libiio.

Keywords: AD-FMCOMMS1-EBZ, GNU Radio, libiio.

1. Introduction

Nowadays, in many communications systems the Software Defined Radio (SDR) has become the default standard due to the advantage of RF integrated circuits (RFICs) and FPGA technologies, which immediately increase the SDRs diversification in fields beyond military communications thus allowing an optimal use of its features. 4G LTE infrastructure is an example of such, which is based on RFICs and FPGAS, so it constitutes a great opportunity for the SDR development [1, 2, 3, 4].

The initial impulse given to 4G technology makes the SDR a platform on which the next generation on wireless communications will be developed. Therefore, technologies such as Internet of Things (IoT) or wireless sensor networks will require new generation hardware and software with SDRs embedded [4].

One of these technologies is the single-chip monolithic, which is used to reduce cost, size, weight and power (SWaP) in combination with new processors, these will allow FPGAs to merge with Analog to Digital Converters (ADCs) and Digital to Analog Converters (DACs) into the core processor, thus leaving the SDR limitations due software problems [4].

The aim of this paper is the implementation of a signal processing block for the RF module AD-FMCOMMS1-EBZ in the GNU Radio framework to perform design of communications systems based on SDR.

2. System Description

The AD-FMCOMMS1-EBZ evaluation board is used to develop last-generation aerospace, satellite, military and commercial telecommunications prototypes with SDR capabilities, but there is not an integration with GNU Radio in order to develop telecommunications systems. The GNU Radio integration with the Industrial I/O subsystem will facilitate the modeling of prototypes using the AD-FMCOMMS1-EBZ board with efficient performance for measurements and allowing modifications with the advantages of software design instead of hardware redesign.

2.1. GNU Radio

GNU Radio is a toolkit which provides signal processing blocks to implement SDR applications. Its main purpose is to facilitate the signal and data processing in modulation, demodulation, channel coding and intermediate frequency stages through a C++ native backend for critical signal processing, and a Python frontend for non-performance functions [5, 6, 7].

GNU Radio framework is based on a pattern design via graphs, as shown in Fig. 1, and its nodes are called blocks which are connected through ports. These blocks represent the digital signal processing to be applied upon the signal [5].

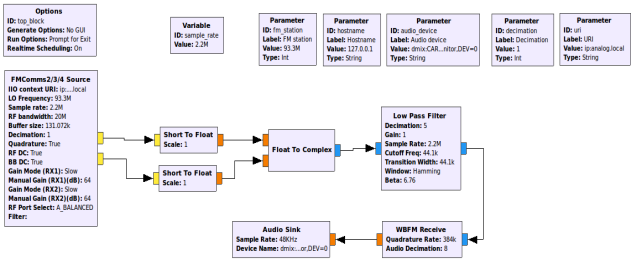

Figure 1. Example of a signal processing flow graph in GNU Radio.

2.2. Evaluation Board AD-FMCOMMS1-EBZ

The AD-FMCOMMS1-EBZ evaluation board is a high speed RF module that provides an analog frontend for multiple devices such as sensors and measuring devices. These are reconfigurable via software to a frequency range from 400 MHz to 4 GHz [8].

As shown in Fig. 2, the RF module has an ADC based on AD9643 for data reception, and a DAC based on AD9122 for data transmission [8, 9].

The received signal from the antenna is sent to ADL5380, a quadrature demodulator which operates between 400 and 6000 MHz with a 500 MHz bandwidth, in combination with the broadband synthesizer ADF4351, which operates from 35 MHz to 4400 MHz that delivers two reception channels. The Demodulated signal is regenerated via dual Variable Gain Amplifier (VGA) AD8366 in a 600 MHz operation limit. Finally, the signal is sent to the AD9643 ADC which generates the digital data for the FPGA [9].

For transmission, the digital data received from the FPGA is sent to the AD9122 DAC where the reconstructed analog signal is generated in two channels to the ADL5375 quadrature modulator. It operates from 35 MHz to 6000 MHz and the ADF4351 broadband synthesizer provides the modulated signal to the ADL5620 RF/IF amplifier which operates from 50 MHz to 4 GHz. The modulated signal is sent to the RF output [9].

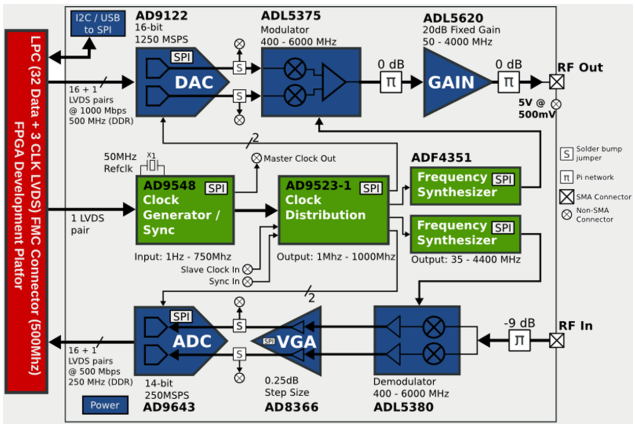

Figure 2. Functional block diagram of the AD-FMCOMMS1-EBZ board.

2.3. Linux Industrial I/O Subsystem

In Fig. 3 it is observed how the Analog Devices Industrial I/O (IIO) subsystem provides a layered communication model to ADCs and DACs through multiple platforms and operative systems [10].

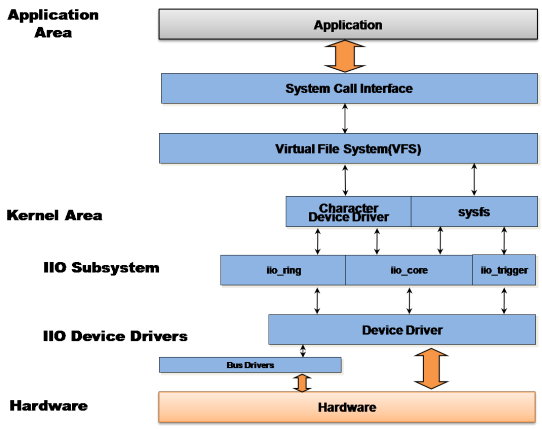

Figure 3. IIO Subsystem Overview.

The IIO Linux drivers are focused for PLL synthesizers with serial interface and provide a unified driver framework for various types of converters and sensors using multiple physical interfaces (i2c, spi, etc) as shown in Fig. 4 [10].

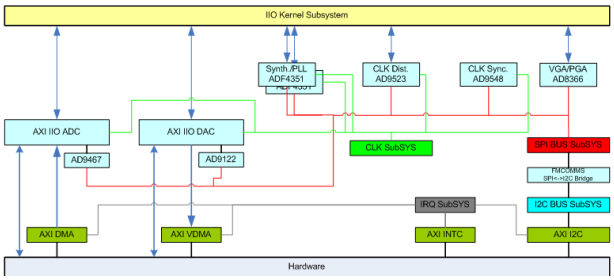

Figure 4. Linux Drivers for the AD-FMCOMMS1-EBZ board.

2.4. Linux Industrial I/O Library

The Industrial I/O subsystem library (libiio) allows the development of applications using IIO devices. It uses the IIO standardized interface of Linux kernel which is an interpreter between an application and the Linux kernel to support the IIO devices and their output channels [11, 12].

2.4.1. Backends

Fig. 5 shows the libiio backends, these are:

- XML backend.
- Local backend.
- Network backend.

```
__api struct iio_context * iio_create_local_context(void);
__api struct iio_context * iio_create_xml_context(const char *archivo_xml);
__api struct iio_context * iio_create_xml_context_mem(const char *archivo_xml, size_t longitud);
__api struct iio_context * iio_create_network_context(const char *host);
```

Figure 5. Backend prototypes of libiio.

The structure shown in Fig. 6 indicates that a backend is associated to an *iio_context* object, where the object has a pointer to the *iio_backend_ops* structure which contains a set of low-level functions according to the selected backend. These functions include: opening a device, reading the attributes of a device and accessing to the data flow, among others [11].

```
struct iio_backend_ops{
    ssize_t (*read)(const struct iio_device *dev, void *dst, size_t len, uint32_t *mask, size_t words);
    ssize_t (*write)(const struct iio_device *dev, const void *src, size_t len);
    ...
};
```

Figure 6. *iio_backend_ops* structure.

2.4.2. Local backend

The local backend is the only one that really interacts with the hardware through the *sysfs* interface of Linux kernel. Through this backend the *iio_create_local_context* function creates the *iio_context* object [11].

2.4.3. Network backend

The network backend characteristics are [11]:

- Allow to transmit and receive samples through the communications network to any IIO device.
- IIO applications can run directly in computers.
- Allow the parallel processing of dataflow samples from a device.
- Allow to run applications which require advanced characteristics of libiio without the administrator privileges as the local backend requires.

2.4.4. IIO Daemon

The network backend is connected to the IIO daemon server (IIOD) which manages the incoming network connections and allows remote communication to an IIO device.

A singular feature of IIOD is that it uses the libiio library underneath, as shown in the connections model in Fig. 7, which is interpreted as a daemon and at the same time is a component of the same library [11].

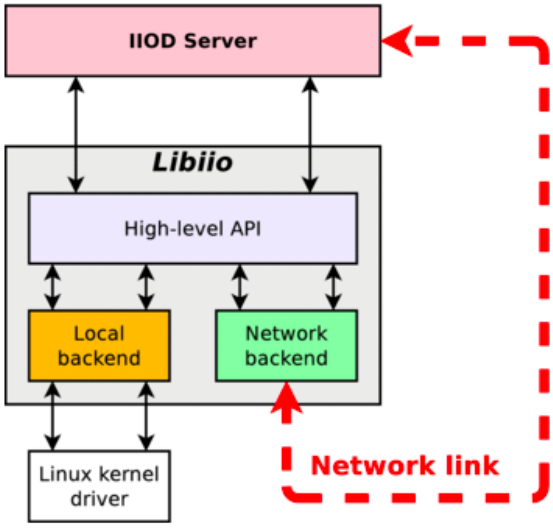

Figure 7. Network backend connections model.

3. Development

3.1. Libiio library linking

To link the libiio library to GNU Radio it must be declared in a CMake file as shown in Fig. 8, thus the library path can be recognized by the GNU Radio environment.

```
# Linking module with libiio library
INCLUDE(FindPkgConfig)
PKG_CHECK_MODULES(PC_IIO libiio)

# IIO headers search in listed paths.
FIND_PATH(
    IIO_INCLUDE_DIR
    NAMES iio.h
    HINTS ${PC_IIO_INCLUDE_DIR}/include
          ${IIO_DIR}/include
          /usr/local/include/
          /usr/local/src/libiio/
          /usr/src/linux-headers-4.4.0-21/include/linux/iio
)

# .so files search in listed paths
FIND_LIBRARY(
    IIO_LIBRARIES
    NAMES libiio.so.0.8
    HINTS $ENV{IIO_DIR}/lib
        ${PC_IIO_LIBDIR}
    PATHS ${CMAKE_INSTALL_PREFIX}/lib
          ${CMAKE_INSTALL_PREFIX}/lib64s
          /usr/local/lib
          /usr/local/lib64
          /usr/lib
          /usr/lib64
          /usr/pkgs64/lib
)
```

Figure 8. CMake file for libiio linking with GNU Radio.

3.2. Context creation

Libiio works with a structure called *context*, which is used to manage the device from inside its iio subsystem. This C structure must be declared as an external structure at the header file of the signal processing block to allow the use of the object *iio_context* by GNU Radio's runtime environment, so it can use the *create* function from the backend, which is necessary to access the low-level functions in the subsystem.

In the context creation, the GNU Radio block of the RF module does not use the local backend, instead, uses the network backend with the assigned FPGA IP address where the RF module is installed, these allow remote connections to access data configuration and the signals flow for transmission and reception. The remote connection from the block is accomplished by creating the context through the *iio_create_network_context* function.

3.3. RF module parameters

Fig. 2 shows the ADC and DAC devices of the RF module to signal reception/transmission. In order to identify the parameters and get the information needed of the IIO subsystem it is necessary to know:

- The IP address of the RF module.
- The ADC device ID labeled as "cf-ad9643-core-lpc" in the IIO subsystem.
- The physical channel of the ADC, labeled as "altvoltage0".
- The reception device ID, labeled as "adf4351-rx-lpc".
- The reception channels IDs, labeled as "voltage0" and "voltage1".

Once the communication with the RF Module is established, the following parameters can be modified: frequency, sample rate, bandwidth, buffer size and decimation, as shown in Fig. 9.

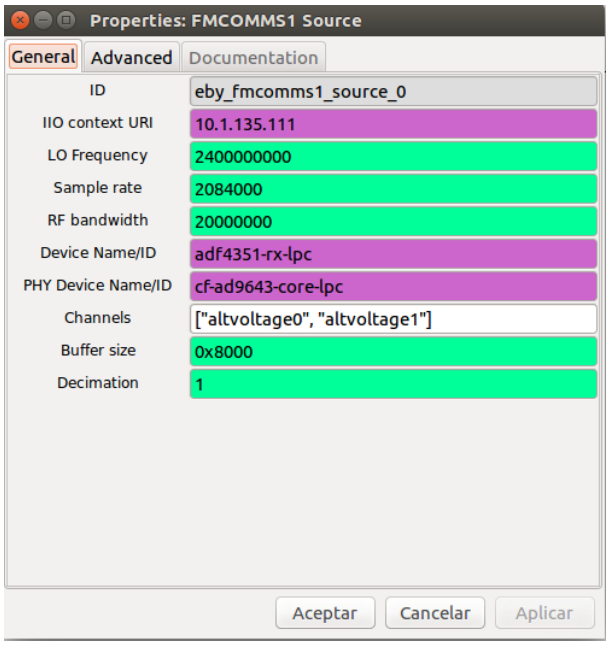

Figure 9. Configuration parameters in the GNU Radio source block.

4. Results

The source block for the RF module AD-FMCOMMS1-EBZ shown in Fig. 10 displays the data type given by the active channel, for this block is a 16-bit Short type.

Fig. 11 shows the received spectrum in the frequency of 887 MHz with a bandwidth of 200 MHz. There is a slightly visible carrier signal but due to hardware limitations of the RF module, the reception could be complemented with additional stages such low-noise amplifiers or high gain antenna.

As shown in Fig. 12, the IIO Oscilloscope tool from Analog Devices shows the same spectrum as the GNU Radio block.

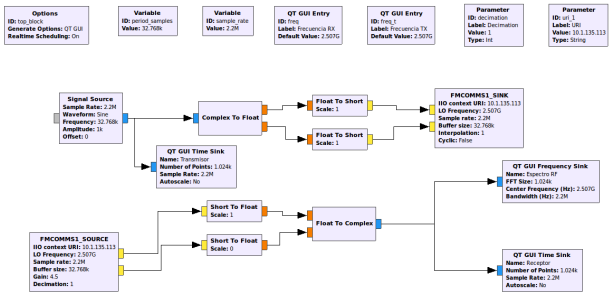

Figure 10. Flow graph of the AD-FMCOMMS1-EBZ board as receiver.

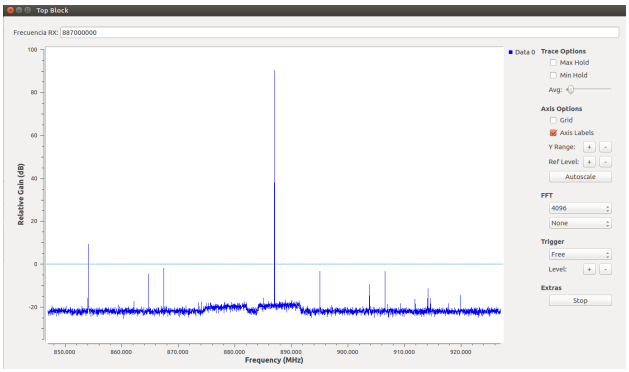

Figure 11. Spectrum of the received signal from the AD-FMCOMMS1-EBZ board in the GNU Radio block.

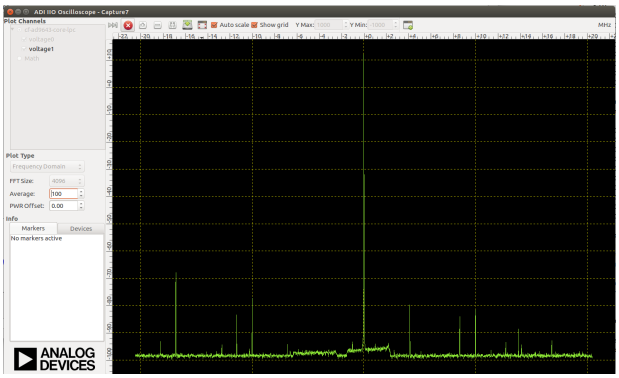

Figure 12. Spectrum of the received signal in the Analog Devices oscilloscope.

5. Conclusion

As seen in the section above, even with a basic configuration, GNU Radio and the AD-FMCOMMS1-EBZ evaluation board allows us to make a functional prototype of an SDR where is easy to modify settings like the IP address of the RF module, the ADC, even the physical channels using only the user interface given by GNU Radio, but it could be handled simply by the Linux Industrial I/O Library with a backend software programmed to do modifications automatically, given an event.

Nowadays the IoT concept of interconnection is taking a big relevance on the telecommunications area, so this work may add to the experience in working with SDR's on this concept since the reconfiguration capabilities of them allows to increase the range of configurations that can be applied to a single device.

References

[1] W. Tuttlebee. (2002). *Software defined radio: Enabling technologies.*. New York: J. Wiley & Sons.

[2] Reinhart, Richard, *"Space Communication and Navigation SDR Testbed, Overview and Opportunity for Experiments,"* Wireless Innovation Forum Technical Conference, January 2013.

[3] Sacchi, Claudio and Passerone, Roberto and others. (2012). *A new vision of software defined radio: from academic experimentation to industrial explotation,*, Universitat Politècnica de Catalunya

[4] *Software Defined Radio: Past, Present, and Future - National Instruments.* (2017). Ni.com. Retrieved 25 August 2017, from http://www.ni.com/white-paper/53706/en/

[5] Blossom, Eric and Corgan, Johnathan and Braun, Martin and Ettus, Matt and Rondeau, Tom *GNU software radio* (2017). GNU project. url: http://gnuradio.org/redmine/projects/gnuradio/wiki

[6] *GNU Radio* [Analog Devices Wiki]. (2017). Wiki.analog.com. Retrieved 23 March 2017, from https://wiki.analog.com/resources/tools-software/linux-software/gnuradio

[7] T. Schmid, *Gnu radio 802.15.4 en-and decoding* Networked & Embedded Systems Laboratory, UCLA, Technical Report TR-UCLANESL-200609-06, June 2006.

[8] *AD-FMCOMMS1-EBZ Introduction* [Analog Devices Wiki]. (2017). Wiki.analog.com. Retrieved 17 March 2017, from https://wiki.analog.com/resources/eval/user-guides/ad-fmcomms1-ebz/introduction

[9] *AD-FMCOMMS1-EBZ Functional Overview* [Analog Devices Wiki]. (2017). Wiki.analog.com. Retrieved 17 March 2017, from https://wiki.analog.com/resources/eval/user-guides/ad-fmcomms1-ebz/hardware/functional_overview

[10] *Linux Industrial I/O Subsystem* [Analog Devices Wiki]. (2017). Wiki.analog.com. Retrieved 17 January 2017, from https://wiki.analog.com/software/linux/docs/iio/iio

[11] *About libiio [Analog Devices Wiki].* (2017). Wiki.analog.com. Retrieved 17 March 2017, from https://wiki.analog.com/resources/tools-software/linux-software/libiio_internals

[12] Cercueil, P. (2015). *Demo. Proceedings Of The 2015 Workshop On Software Radio Implementation Forum - SRIF '15.* http://dx.doi.org/10.1145/2801676.2801684