Modular, Open-Source Software Transceiver for PHY/MAC Research

John Malsbury
Ettus Research
1043 N Shoreline Blvd.
(650)417-6019
john.malsbury@ettus.com

Abstract

The USRP™ (Universal Software Radio Peripheral) is a software-defined radio platform that has been widely adopted for wireless research in cognitive radio, cellular networks, and other application areas. USRP devices are often used with GNU Radio, a free and open-source DSP framework that allows designers to prototype with a combination of C++, Python, and graphical tools. This paper will investigate various methods that can be used to build complete communications stacks within GNU Radio. These methods will leverage advanced features of UHD™ (USRP Hardware Driver) and GNU radio to implement TDMA, CSMA and FHSS transceivers that can be modified in GNU Radio Companion - a graphical development environment. The implementation will also show how to interact with upper network and application layers, all within GNU Radio. The implementation presented in this paper will be open-source. It can serve as an educational resource, or as a basis for additional research.

Categories and Subject Descriptors

C.2.0 **[General]**: Open Systems Interconnection Reference Model (OSI)

C.2.1 **[Network Architecture and Design]**: Wireless Communication

General Terms

Design, Experimentation, Algorithms

SRIF'13, August 12, 2013, Hong Kong, China.

Keywords

GNU Radio, PHY/MAC, cognitive radio, SDR programming model, SDR architecture, SDR reference design, prototyping, graphical design

1. Introduction

GNU Radio is an open-source DSP framework that has seen widespread adoption for PHY/MAC prototyping. In many cases, GNU Radio is used with the Ettus Research USRP products. The combination of low-cost hardware and open software has drastically lowered the barrier to entry for wireless prototyping. There are on-going improvements that continue to make this tool valuable to the community.

The framework uses an elegant architecture that provides high performance DSP capability, while offering development tools and APIs that facilitate experimentation. GNU Radio generally runs on general-purpose processors (GPP). In most cases the DSP blocks are written in C++, and DSP chains are built graphically or in Python.

The USRP product family contains several devices with a range of capabilities. Generally speaking, developers connect a USRP to a host-computer via USB 2.0 or Gigabit Ethernet. UHD™ (USRP Hardware Driver) provides a common API, including member functions to configure the USRP (frequency, gain) and stream baseband samples to and from the host-computer. This API is used by GNU Radio, and other frameworks, to provide access to the USRP functionality.

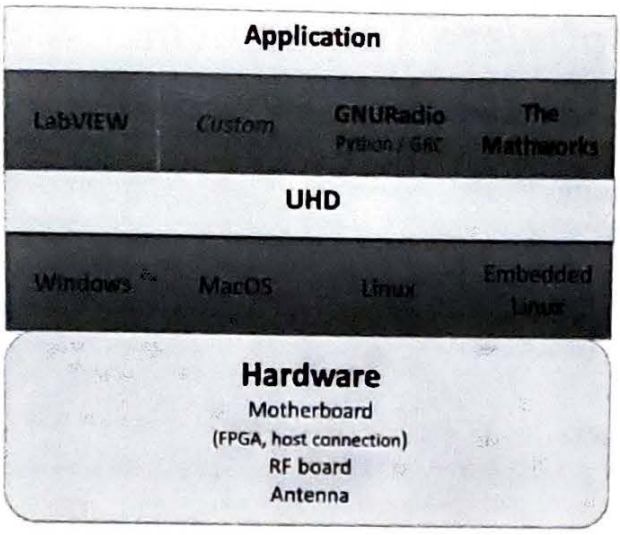

Figure 1 - USRP and GNU Radio Software Stack

2. Development Challenges with Existing PHY/MAC Examples

GNU Radio and the USRP product family have served as a common and widely adopted prototyping platform for wireless systems. In many cases, developers adapt existing implementations, such as tunnel.py or benchmark_rx.py to their research needs.

These applications serve as basic examples. New users are able to observe the coding structure of GNU Radio, and gain an understanding of interactions with the USRP. The applications have been modified to support unique research. However, there are still some weaknesses to be overcome. The applications generally rely on an architecture shown in Figure 2.

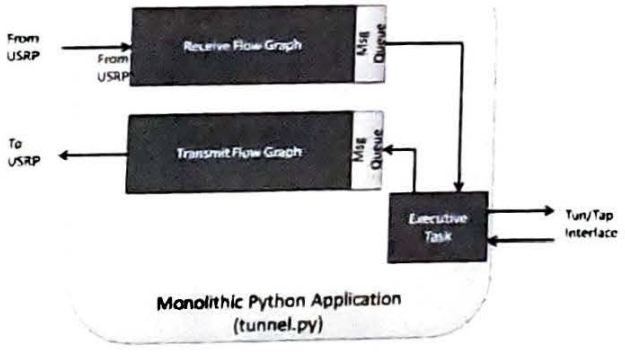

Figure 2 - Tunnel.py Architecture

While these applications serve as straight-forward implementation examples of a CSMA transceiver, there are a few drawbacks to this approach. First, there is a lack of modularity. For example, it is not easy to modify the functionality of the CSMA MAC, and implement TDMA without modifying major sections of the code. In some cases, for the novice developer it may be advantageous to 'swap' a CSMA implementation block for a TDMA implementation block.

As one might expect, many novice developers also prefer graphical tools for at least some of their research. The implementation of tunnel.py prevents the use of GNU Radio companion for further development or extension.

3. A New Framework

Recent advances in GNU Radio have enabled new approaches to developing complete PHY/MAC solutions. Recently integrated message passing functionality can allow the developer to integrate basic MAC-layer functionality in the flow-graph itself.

The remainder of this paper will examine a simple framework that has been used to build modular transceivers in GNU Radio companion.

3.1 The USRP Architecture

To proceed with a discussion on these features, it helps to have an understanding of the USRP architecture. The USRP system contains a few major components:

- USRP motherboard with FPGA, clock management, power, ADCs, DACs and host-interface
- RF Daughterboard for up/down conversion
- Host-computer

The FPGA performs a number of functions. In the receive direction, the FPGA accepts clocked samples from the dual ADCs. The sampled I/Q data passes through DC offset compensation, a CORDIC for high-resolution offset tuning, scaling multipliers and several decimating filters. This chain ultimately reduces the sample rate until the stream can be passed through the host interface. The samples are framed with VITA-49, passed along to a packet-router, and onto the host-interface. A similar process occurs in reverse on the transmit chain.

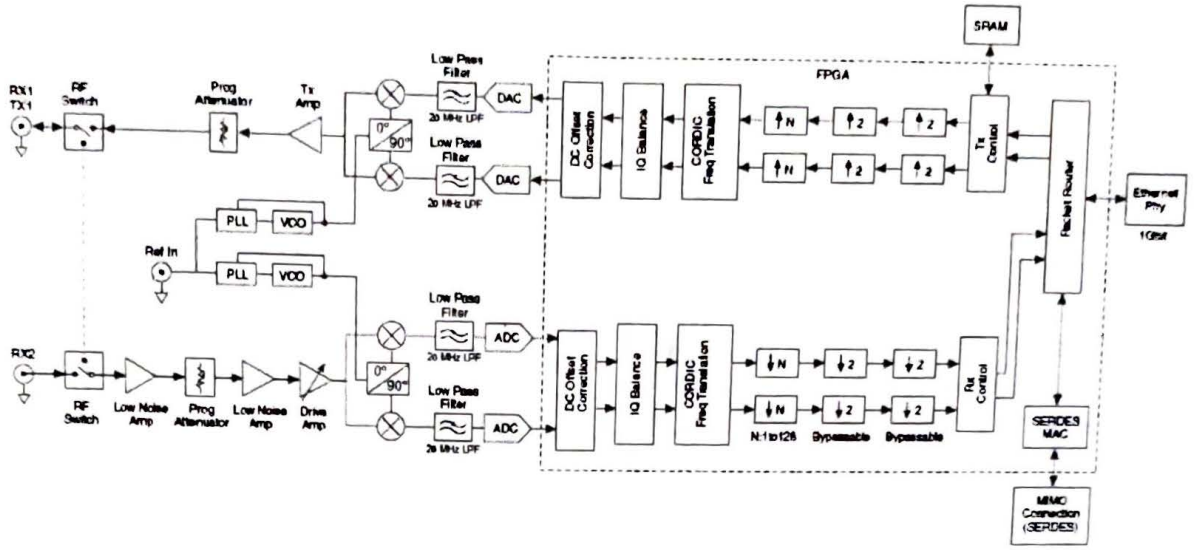

Figure 3 - USRP Block Diagram

3.2 Meta-Data and Timed Commands

While it is not clearly shown in the diagram, the USRP FPGA provides temporal alignment capability for reception, transmission, tuning and other operations. Leveraging UHD, it is possible to couple metadata to transmit samples.

The meta-data can indicate the start of a burst, the end-of-a-burst, and other conditions. The meta-data can also specify precise times for samples to be transmitted, which allows tight control of outgoing packets. This is useful for TDMA or FHSS implementations. Figure 4 includes a basic example of how to use these metadata features.

```
//send first packet of samples in stream at seconds_in_future
uhd::tx_metadata_t md;
md.start_of_burst = false;
md.end_of_burst = false;
md.has_time_spec = true;
md.time_spec = uhd::time_spec_t(seconds_in_future);
tx_stream->send( &buff.front(), samps_to_send, md, to);
```

Figure 4 - C++ Example for Timed Tx

The FPGA also accepts command packets from the host-computer. These commands set and get register values. They also set other components that are external to the FPGA, such as phase-locked loops (PLLs) and general purpose inputs and outputs (GPIO). PLL tuning and GPIO can be controlled in a timed-manner, allowing for deterministic frequency hopping or precise switching of external amplifiers.

3.3 GNU Radio Architecture

As mentioned, GNU Radio provides high-performance DSP blocks using C++, and a user-friendly API in Python that allows users to connect these blocks easily. This combination allows developers to build and modify applications rapidly while maintaining reasonable performance on commodity computing hardware.

The default case relies on C++ for the implementation of blocks. Recent feature additions allow developers to develop blocks in Python, which can be easier for new developers that are learning to code. Python, being an interpreted language, also allows the developer to avoid compilation steps. The developer must consider trade-offs between performance and ease-of-use as he/she selects Python or C++ for various functions.

Compared to C++, Python will generally provide lower throughput for a given algorithm. Radios with the MAC layer written in Python and Physical layer in C++ routinely achieved data rates of 500-1000 kb/s on a mid-range Quad-Core, I7 PC.

Based on these empirical results, some recommendations for implementation language are given for various PHY/MAC operations. In general, functions that operate on a frame-by-frame basis can be implemented in Python with reasonable performance. Anything that operates on a symbol or sample basis should generally be implemented in C++.

Type of Operation	Python	C++
PDU Generation	X	X
Filtering Received PDUs by Address	X	X
ARQ	X	X
Block Encoding		X
Mod/Demod		X
Filtering		X

Table 1 - Recommended Language for Various Functions

3.4 Stream Tags – Meta-Data in the GNU Radio Universe

GNU Radio allows samples to be tagged with arbitrary pieces of meta-data. For example, the UHD sink (transmitter) block, which uses the UHD API, accepts the following meta-data – Transmit Start-of-Burst (tx_sob), Transmit Start Time (tx_time), and Transmit End-of-Burst (tx_eob). These three tags allow a GNU Radio application to produce a burst transmission that occurs at a precise time. An illustration of how these tags are applied to the data is shown in Figure 5.

In this example, the USRP receives stream of stamples, the first of which is tagged with tx_sob and tx_time that are. When the internal counter, which has a resolution of 10 ns in the USRP N210, reaches tx_time, the USRP will enable the transmit switch in the RF chain and begin clocking-out I/Q samples to the DAC. After receiving the tx_eob, the USRP will stop clock out samples, turn-off the transmit chain. If the USRP is running in half-duplex mode, the FPGA will also switch the RX/TX input from the transmit chain to the receive chain.

Stream-tags are a poly-morphic type. They can contain an arbitrary amount and type of information. In this example, PMTs of PMT_T and a tuple are used to convey the necessary information.

3.5 Asynchronous Messages vs. Stream Tags

The GNU Radio scheduler calls DSP functions of each block based on the availability of space and samples at the outputs and inputs of each block. The sample streams are unidirectional, connections can only be made downstream. This allows the GNU Radio runtime to pipeline operations, and ensure that the proper number of samples are produced and consumes. Functionally speaking, stream tags are appended to samples, and move with the tagged sample as it passes through each block of the DSP chain.

Asynchronous messages operate in a different manner. Rather than relying on buffers between DSP blocks that operate synchronously, messages are passed between blocks through queues. When a message is pushed to a queue, the schedule raises the corresponding handler functions of connected blocks. It is possible to pass messages to up or downstream blocks. Also, the timing of the message passing is independent of any sample streams, unless the designer enforces some alignment policy in his block implementation.

While stream tags can be used for operations that are synchronous to a sample stream, asynchronous messages can be used to move information in the flow-graph with more flexibility. For example, a MAC-layer component such as a framer or ARQ manager may interface with an application layer component such as a user typing text messages. By nature, many application layer processes generate and consume data in an asynchronous and unpredictable manner.

Asynchronous message passing may also be used to implement control-plane functions. A data-link management block might pass along information to set the frequency of the USRP to a clear channel, or adjust the data-rate, etc.

4. SDR Transceiver Architecture

Modern wireless research is sometimes focused on cross-layer approaches that begin to diverge from more conventional protocol layering definitions. In order to make this design straight-forward and easy to understand, the implementers chose a structure that is roughly in line with the OSI Model.

In GNU Radio, hierarchal blocks are used to create macro-functional modules that are easily swapped with other modules. A clear line is drawn between the data-link and network layer.

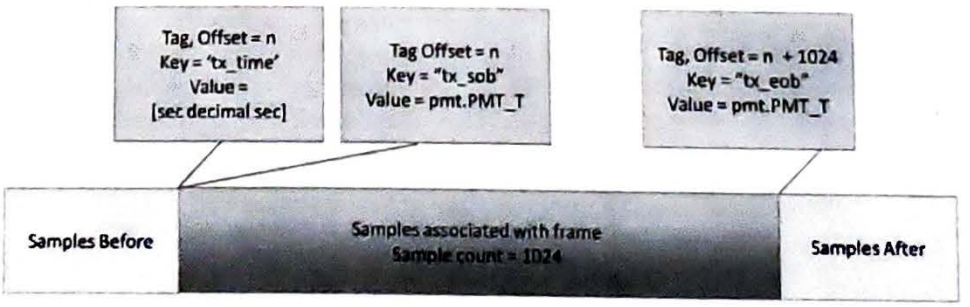

Figure 5 - Stream Tag Example - Burst Tx With USRP

The physical and data-link layer functionality is implemented in top-level MODEM blocks. Various types of modems have been implemented: CSMA, FHSS, TDMA, etc. Implementations include full-duplex and half-duplex modes.

The network and transport layers are currently implemented in a MAC layer block. This block, termed "Simple MAC" provides addressing, network layer framing, and stop-and-wait ARQ functionality.

The upper layers - session, presentation, and application – are based on flexible interconnection of various blocks. In a simple example, which can be seen in Figure 6**Error! Reference source not found.**, the application layer is implemented with a TCP/IP interface. This TCP/IP interface could be swapped with a TUN/TAP interface, an audio processing layer, or a myriad of other application layer functions.

In general, operations in the MAC, session, presentation and application layers rely on asynchronous messages to pass information from block to block. This allows the flowgraph to interact with various processes that may not operate synchronously with the transceiver.

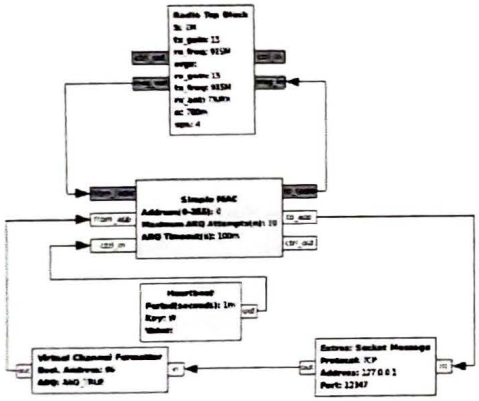

Figure 6 - PHY/MAC Implementation

The block labeled "Radio Top Block" is a hierarchal block that includes datalink and physical layer functionality. A more detailed look at the contents of the Radio Top Block can be seen in Figure 7. This block serves several important functions, which will be described in more detail.

4.1 General Modem Implementation

The Radio Top Block provides a transition from the asynchronous message-based domain to synchronous sample streams. In the implementation shown in Figure 7, the packet framer block accepts asynchronous messages with payload data. It extracts the payload bytes, appends synchronization words and CRC, scrambles the data, and outputs the resulting bytes in a sample stream. In this process, it also tags the last sample with a tx_eob tag. In TDMA and FHSS implementations, the framer block accepts more complicated async messages that include transmit burst times in addition to the payload data. The packet framer appends this information to the first sample of a frame with stream tags to assure the USRP transmits the packet at the correct time.

A modulator accepts a *stream* of bytes from the packet encoder and modulates the data, bit-by-bit.. The result is a stream of complex (I/Q) samples that can be sent to the USRP source for transmission. In this example, a GMSK modulator is shown. The GMSK modulator can be replaced arbitrarily with QAM, PSK, OFDM or anything else that suits the researcher. The samples and appended stream tags are buffered and sent to the USRP hardware/FPGA with a UHD sink. A similar chain is used for the reception process.

In addition to simple random CSMA implementations, this framework also demonstrates TDMA and FHSS. There are minor differences in the implementation of all of these blocks. The TDMA and FHSS blocks make use of more stream-tagging features of the USRP sink and source blocks. These stream-tagging and time-commands provide alignment between sample transmission, tuning, and T/R switching.

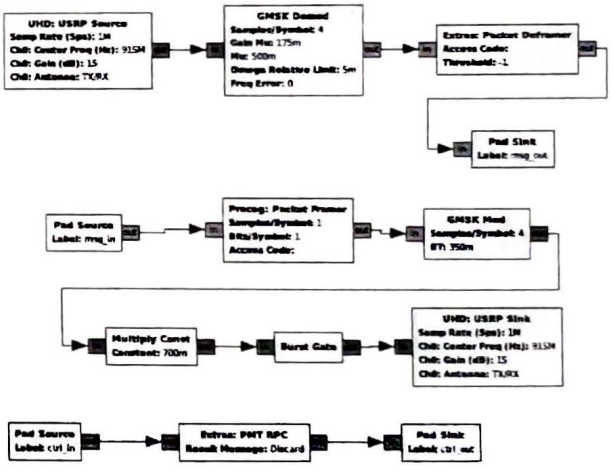

Figure 7 - Physical Layer Implementation

4.2 Physical, Network, and Application Layer Modularity

Looking at Figure 6, it is evident that the CSMA block can be easily replaced by the FHSS or TDMA blocks without affecting the design and operation of the upper protocols layers (network/application).

Likewise, the simple message-based interface can provide latitude for the network layer implementation. The Simple

MAC block can be replaced with one or more blocks of similar functionality. Due the latency of the USRP interface, stop-and-wait provide relatively poor performance. The author has also developed selective repeat ARQ. Future USRP devices will provide a lower latency PCIe interface, which will improve performance. The developer may integrate additional blocks for link management and control or implement upper cognitive functions at this layer.

There is also significant flexibility in the application layer. **Error! Reference source not found.** The TCP/IP can be replaced or multiplexed with audio and other tactical data.

Given this modularity, researchers are able to experiment with one or all of the protocol layers. For example, the developer that is concerned with developing unique physical layer implementations can leverage existing MAC and application layer functionality, while concentrating efforts on modulation, demodulation, and spectrum sensing algorithms. Conversely, the researcher who is primarily concerned with performance of many-node RF networks, might use a fixed-functionality OFDM modem, but design blocks that would replace the Simple MAC block.

5. Demonstration and Source Code

The concepts and implementations discussed in this paper will be demonstrated at SRIF 2013. The demonstrations will include CSMA, TDMA, and FHSS transceiver implementations. The audience will be able to observe various types of data such as text messages or audio data being transferred through a wireless network using these various implementations. Through these demonstrations, researchers will become familiar with the overall transceiver architecture, and gain some insight on how various components can be added or modified for their own research.

All code for this reference design will be posted online, prior to the conference.

6. Future Work

To improve performance, many of the blocks could be re-implemented in C++. While Python offers a fast path to prototype the MAC-layer blocks, C++ offers higher throughput capability.

As advanced wireless research continues to drive requirements for lower latency and higher processing throughput, developers may need to make better use of the FPGA for digital signal processing. Future implementations will likely maintain the overall architecture and protocol layering. The physical layer functionality provided by the "Radio Top Block" in Figure 6 could be replaced by an FPGA PHY. In this case, the PDUs from the MAC layer would be based through the host-interface to the SDR hardware. The FPGA IP on the SDR would provide modulation, demodulation, and frame alignment functionality. This would leave the upper-network functionality to the host.

7. Acknowledgements

The author would like to thank Josh Blum for pioneering many of the message passing features with gr-extras, and the GNU Radio community as a whole for building and maintaining such a capable, free software-framework for wireless prototyping.

8. References

[1] GNU Radio – FAQ. Retrieved May 29, 2013, from GNU Radio: http://gnuradio.org/redmine/projects/faq

[2] GNU Radio - Overview. Retrieved May 29, 2013, from GNU Radio: http://gnuradio.org/redmine/projects/gnuradio

[3] GNU Radio – Message Passing. Retrieved May 29, 2013, from GNU Radio: http://gnuradio.org/doc/doxygen-3.6.3/page_msg_passing.html

[4] Reed, J.H., Software Radio: *A Modern Approach to Radio Engineering*. Prentice Hall, 2002