Demonstration of GNU Radio High Data Rate BPSK 10 Mbps Modem Real-Time with Only Multi-Core General Purpose Processors (GRCON 2021)

David T. Miller
Dave.Todd.Miller@gmail.com
Ashburn, VA 20147 USA

Abstract

This paper presents a GNU Radio Modem design that demonstrates the feasibility of achieving ≥10 Mbps Real-Time Binary Phase Shift Keying (BPSK) performance with a relatively low cost Personal Computer (PC) that contains an 8-core General Purpose Processor (GPP). The high date rate is achieved with a single GNU Radio flowgraph and without a Field Programmable Gate Array (FPGA) or Graphics Processor Unit (GPU). The high data rate is achieved by breaking the incoming I/Q sample stream from a LimeSDR-mini into four "chunk" streams with each chunk stream going to a separate Symbol Synchronizer (symbol synchronization) and Costas Loop (carrier synchronization) chain with each chain using a separate GPP core. The GNU Radio modem then "stitches" the original transmitted single stream back together by using the frame ASMs and the frame counter in the header of each frame. The approach is scalable, therefore much higher data rates (>50 Mbps) may be achievable also with more GPP cores.

1. Introduction

The feasibility of greatly expanding the real-time data rate capability of a GNU Radio modem at a reasonable cost now exists because of the following two trends in the Personal Computer (PC) and Server market:

1. A continuous improvement in the number of General Purpose Processor (GPP) cores in a single Personal Computer (PC) or Server.
2. A continuous lowering of costs for PCs/Servers with multi-core GPPs up to at least 64 cores.

Proceedings of the 11th GNU Radio Conference,

Moore's Law on increasing the speeds on an individual GPP single core has mostly stagnated for at least the last 10 years. However, the PC/Server industry trend to expand the number of cores in a GPP now provides a path forward still for applications like GNU Radio to greatly expand their performance speeds.

Demonstrating via actual testing that a GNU Radio Software Defined Radio (SDR) modem can achieve at least 10 Mbps with BPSK by using only GPP cores in parallel that are inside an 8-core GPP PC may unlock the potential for new GNU Radio high data rate (HDR) applications. For example, the design documented in this paper should be scalable to much higher data rates with more cores. It could be possible to achieve data rates >50 Mbps in real-time with a 32-48 core PC/server, GNU Radio, and a >100 Megasample per second "dongle" unit with a 10 Gigabit Ethernet interface. Field Programmable Gate Array (FPGAs) or Graphics Processor Units (GPUs) are not needed with this HDR parallel GPP multi-core approach.

One could even consider the feasibility of deploying GNU Radio on cloud servers with the digital complex I/Q stream originating from a ground station "hardware dongle" at a different geographic location.

Some of the challenges on increasing the performance of GNU Radio and SDRs in general with only GPPs is documented in (Bloessl et al., 2019). The approach in this paper using multi-cores in parallel and the demonstration documented in this paper can provide one potential path forward in order to overcome those challenges.

2. Demonstration Test Objective

The primary objective of this demonstration test activity is to show that a GNU Radio HDR SDR receiver can be developed that can achieve data rates of at least 10 Mbps with BPSK in real-time using a relatively low cost 8-core PC.

3. Scope of Demonstration Test

For Satellite communications, the Binary Phase Shift Keying (BPSK) modulation waveform is used extensively. Therefore, the author conducted a BPSK test case at 10 Mbps.

4. GNU Radio SDR Receiver Design

This section describes the details of the GNU Radio SDR design and implementation for 10 Mbps BPSK.

The author implemented the inexpensive GNU Radio SDR receiver with a Lenova IdeaPad 5 laptop (≈$650.00) containing an Advanced Micro Devices (AMD) Ryzen 7-4700U 8-core GPP, the free open source Linux/Ubuntu operating system, the free open source GNU Radio software (version 3.8.3), and an inexpensive <$200.00 Commercial Off-The-Shelf (COTS) LimeSDR-mini hardware transmit/receive dongle. The LimeSDR-Mini has a Universal Serial Bus (USB) 3.0 interface on one side for the connection to the Lenova laptop and about a 30 Megasample per second maximum capability. On its other side, the LimeSDR mini dongle has 50 ohm SubMiniature version A (SMA) transmit and receive Radio Frequency (RF) interfaces.

Please refer to the following for a detailed description of the LimeSDR-mini hardware dongle functions and design:
https://limemicro.com/products/boards/limesdr-mini

The author implemented the design with GNU Radio based on a non-GNU Radio parallel multi-core GPP approach documented in (Grayver et al., 2020).

Figure 1 depicts the GNU Radio Companion (GRC) Flowgraph Graphical User Interface (GUI) for the BPSK test demonstration.

Please refer to (Miller, 2019) for a detailed description on the basic BPSK flowgraph block settings when using GNU Radio up to 1.0 Mbps with only a single GPP core. This paper will focus on the additions of parallel Symbol Synchronizer/Costas Loop chains to greatly increase the real-time data rate capability of the GNU radio modem while only using a single PC with an eight-core GPP.

The author implemented the GNU Radio SDR HDR receiver with mostly the GNU Radio blocks that were already available in the GNU Radio Block In-Tree library except for the final frame stitching blocks ("myframer", "frame_stitcher", and "TagASM"). The author implemented the "myframer" and "Frame_stitcher" Out Of Tree (OOT) blocks via C++, however even those blocks were created by just modifying the code from the .cc file of an existing GNU Radio In-Tree block. For example, the In-Tree "tagged_stream_mux_impl.cc" code was modified to create the "myframer" and "frame_stitcher" blocks.

The "myframer" block just eliminates blocks that are not the correct length caused by the I/Q stream discontinuity break before each symbol synchronizer chain ("Keep M in N" block). The C++ "memcpy" function is used for speed.

The frame_stitcher block reassembles the frames into the correct single stream order via the frame header counter. The frame_stitcher block also deletes occasional duplicate frames that occur because of the chunk overlaps. Again the C++ "memcpy" function is used for speed in this block.

The "TagASM" blocks in Figure 1 are also OOT blocks, but they are just modified versions of the "Correlate Access Code – Tag" block created to include a phase ambiguity resolution feature using the ASM in addition to tagging each frame ASM.

For the OOT blocks, the GNU Radio inputs and outputs for each GNU Radio block Scheduler "Work" call were set at large minimum values using the "set_output_multiple()" function in the block code in order to provide long minimum input/output blocks during each "Work" call. Specifically, setting the minimum multiple value for the noutput_items parameter was done to guarantee at least 7 frames (One frame is 4224 bits in length as described below) were processed with each GNU Radio Scheduler "Work" call in order to improve GNU Radio flowgraph speed and performance at high data rates.

When running the transmit/receive loops in Figure 1, the transmit signal originates from a prepared modulation file so that the modulator running at 20 Megasamples per second only requires one GPP core for testing the HDR receiver.

Figure 2 depicts the creation of the modulator file. The modulator file including the ASMs and header counter were implemented with existing GNU Radio library blocks. No OOT blocks were needed in the development

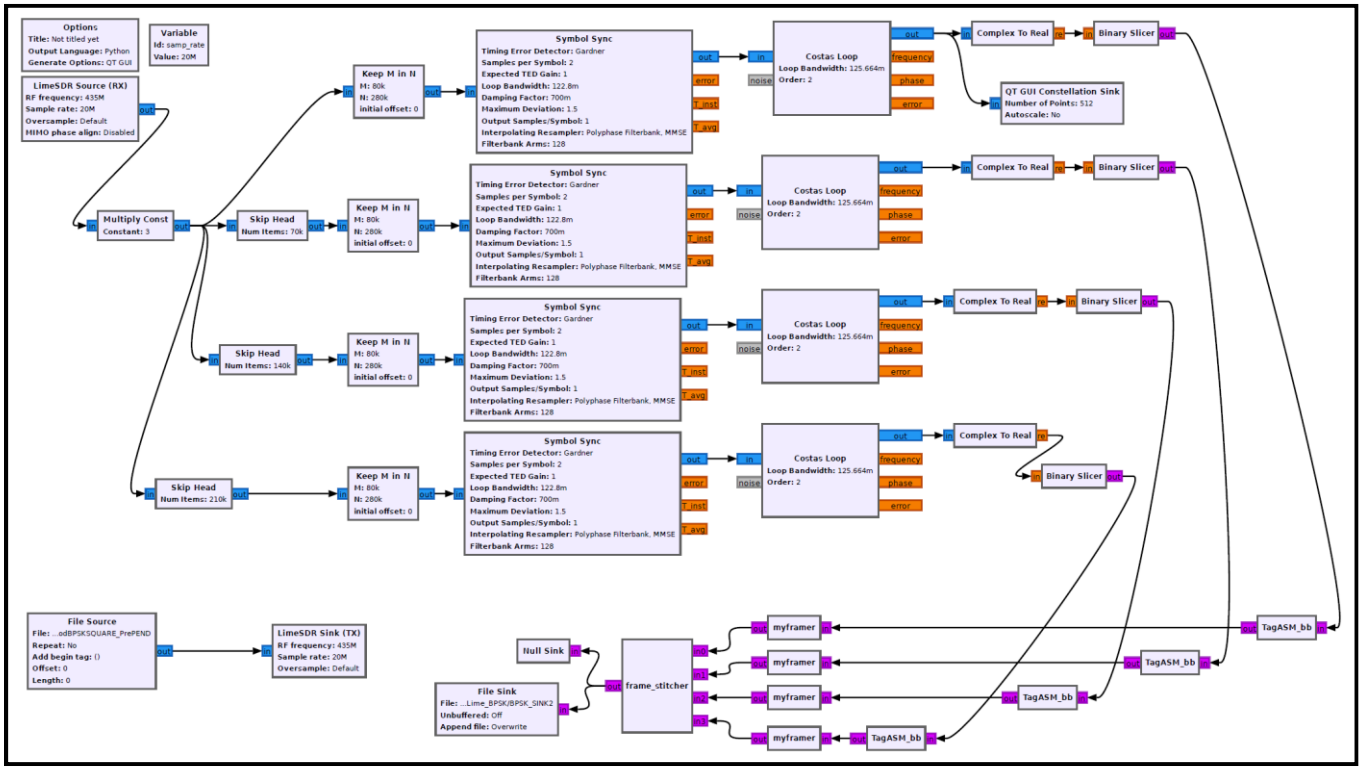

Figure 1: GNU Radio Companion GUI Flowgraph for Real-Time BPSK Test Case

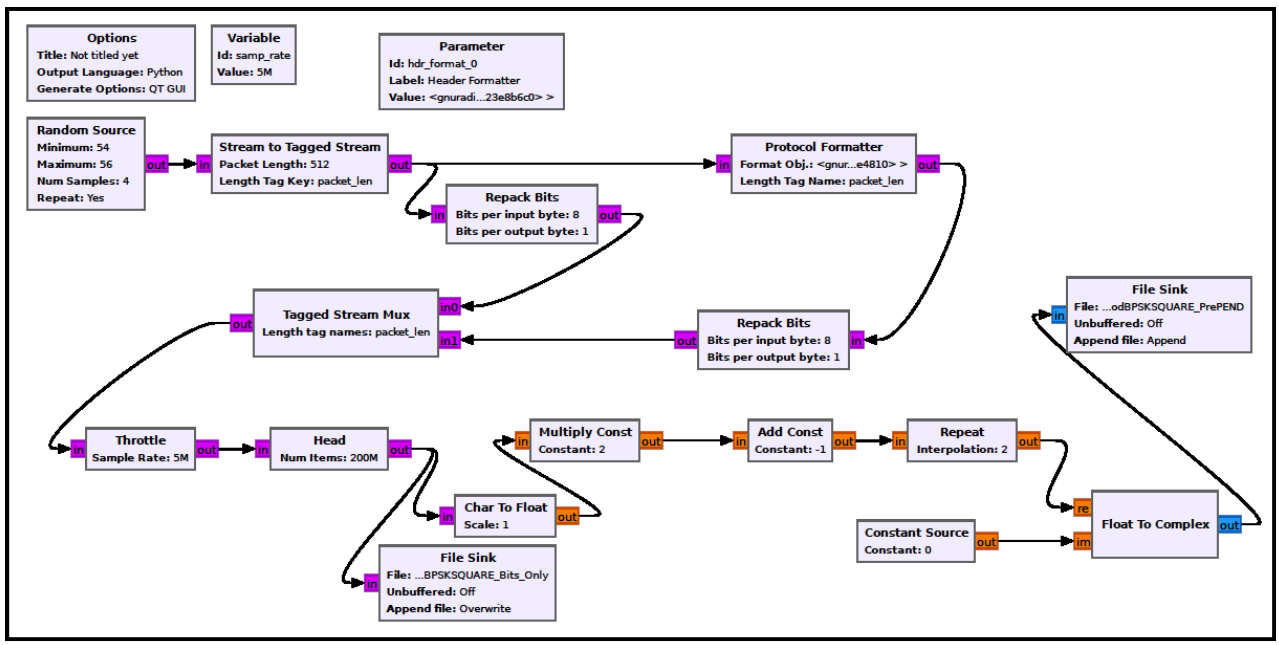

Figure 2: GNU Radio Companion GUI Flowgraph for Modulator File Creation

of the modulator file with the needed ASM, counter, and frame length. About 1.0 seconds of a 101010 repeating pattern for initial receiver synchronization was added to the beginning of the modulator file.

The frame size used for this demonstration was 4224 bits in length including a 64 bit ASM and 64 bit header with the counter.

The "affinity" setting of each block was used in order to efficiently use the PC 8-cores. Table 1 lists the affinity settings for each block (each block was assigned to a specific GPP core). In this paper, a term "chunk" is used that is introduced in detail in (Grayver et al., 2020). A single chunk is defined as one continuous stream of samples that enters a single symbol synchronizer block without a discontinuity: 80000 samples as seen in Figure 1 ("Keep M in N" block, "M" parameter setting). Each Symbol Synchronizer block and Costas Loop block chain for each chunk stream was placed onto a dedicated single GPP core. Figure 1 depicts the overlap of the chunk chains so that a little more than a one frame overlap exists at the beginning and end of each chunk relative to an adjacent chuck chain.

About an extra 800 bits overlap beyond just one 4224 bit frame was also used because the Symbol Synchronizer block and Costas Loop block need to re-sync for each new chunk due to the discontinuities between chunks on each chunk chain. Also, extra overlap is required for the variation in the transmitted symbol clock rate. For example, the symbols per one 80000 sample chunk can vary randomly by a few symbols from chunk to chunk depending on the clock stability of the transmitter.

5. Demonstration Test Approach

The author conducted the following specific demonstration test case with the test configuration of Figure 3:

- BPSK at 10.0 Mbps

Table 2 lists the driving GNU Radio block parameter settings for the BPSK test case.

The author configured the GNU Radio and LimeSDR-Mini for a 435.0 MHz RF test loop. Figure 3 depicts the demonstration loop test configuration with 50 ohm coaxial cables between the LimeSDR-Mini transmit RF output and LimeSDR-Mini receiver RF input.

The GNU Radio modem transmitted a repeating 32 bit pattern in the data portion of each frame as depicted in Figure 2. The LimeSDR-Mini source block was set to 20 Megasamples per second to achieve the demonstrated data rates with 2 samples per symbol.

The GNU Radio Symbol Synchronizer block was set to use the Gardner algorithm. With the Gardner algorithm, the Polyphase Filterbank setting was chosen for the block. For these demonstrations, the author used his own modulator set of blocks rather than a GNU Radio modulator block for better compatibility with the LimeSDR Sink block.

The Binary Slicer blocks translated each bit into one hard decision 8-bit byte for ASM tagging, frame stitching, and convenient file storage for post-test playback to check for bit errors in non-real-time.

6. Demonstration Test Results

The following performance occurred during the BPSK test case:

- The GNU Radio SDR modem successfully recovered the transmitted frame stream with the correct repeating bit pattern in the data field of each frame.
- The GNU Radio SDR successfully "stitched" (reassembled) the frame stream back together at the high data rate in real-time using the counter in the frame header.
- The GNU Radio SDR modem successfully continually maintained/re-established Symbol Synchronizer Lock and Carrier Loop lock for each new "chunk" on each chunk chain.

The author conducted this initial GNU Radio SDR demonstration test and feasibility phase without adding noise, therefore, as one would expect, perfect Bit Error Rate (BER) performance occurred. The author verified perfect BER performance by verifying that the final frames were re-ordered properly via the counter in the header of the frame and also by conducting post-test playback bit error measurements with saved files.

The test results demonstrated that within the scope of this initial testing phase, the implemented GNU Radio SDR modem can achieve BPSK data rates of at least 10 Mbps in real-time by using a flowgraph design that takes advantage of GPP multi-cores in parallel.

Table 1: GNU Radio Block Affinity Settings for Test Cases

GNU Radio Companion Blocks	GPP Core
Symbol Synchronizer/Costas Loop (Chunk Chain #1):	**4**
Symbol Synchronizer/Costas Loop (Chunk Chain #2)	**5**
Symbol Synchronizer/Costas Loop (Chunk Chain #3)	**6**
Symbol Synchronizer/Costas Loop (Chunk Chain #4)	**7**
LimeSDR Source (Receiver)	**1**
Lime SDR Sink (Modulator)	**0**
"Frame Stitcher" Block	**3**
"TagASM" Blocks	**3**
"myframer" Blocks	**3**
"Binary Slicer" and "Complex To Real" Blocks	**3**
"Keep M in N" Blocks	**2**
"Skip Head" Blocks	**2**

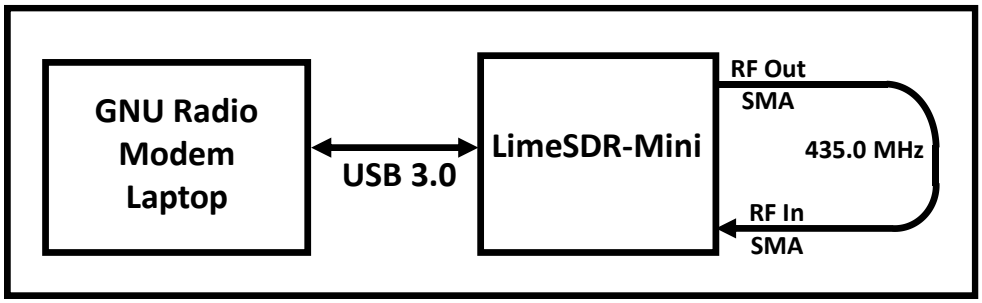

Figure 3: GNU Radio Demonstration Test Loop Configuration With LimeSDR-Mini

Table 2: GNU Radio Block Parameter Settings for BPSK Test Case

GNU Radio Companion Block	BPSK Test Case (10.0 Mbps)
LimeSDR-Mini Dongle Source Block:	
Center Frequency	435.0 MHz
Sample Rate	20.0 Msps
Costas Loop Blocks:	
Order	2
Symbol Synchronizer Blocks:	
Input samples per symbol	2
Output samples per symbol	1

7. Forward Work

Additional follow-on activities should include upgrading the GNU Radio modem as follows:

- Test the GNU Radio modem with QPSK at ≥20 Mbps.
- Consider upgrading to a 32-48 core PC/server to demonstrate the feasibility of >50Mbps BPSK and >100Mbps QPSK real-time using only parallel GPP cores with GNU Radio. At this time, no known showstoppers with GNU Radio or a Linux PC/Server would prevent scalability of the design in this paper to additional GPP cores and higher data rates by just adding more Symbol Synchronizer and Costas Loop parallel chains and obtaining a new hardware "dongle" with a higher sample rate capability.
- Investigate different transmit/receive carrier frequency and symbol frequency offsets.
- Add HDR block decoding using parallel GPP cores.
- Conduct demonstrations with noise to characterize BER vs Eb/No performance.

8. Conclusions

- Within the scope of this initial demonstration and feasibility testing phase, the GNU Radio modem can support BPSK data rates of at least 10.0 Mbps in real-time by just using GPP cores in parallel so that FPGAs and GPUs are not required for HDR performance.
- The design documented in this paper should be scalable to much higher data rates with more cores. It could be possible to achieve data rates >50 Mbps in real-time with a 32-48 core PC/server, GNU Radio, and a >100 Megasample per second "dongle" unit with a 10.0 Gigabit Ethernet interface. Also, Field Programmable Gate Array (FPGAs) or Graphics Processor Units (GPUs) are not needed with this HDR "GPP multi-cores" only approach.

References

Bloessl, Bastian, Müller, Marcus, and Hollick, Matthias. Benchmarking and Profiling the GNU Radio Scheduler. *Proceedings of the 9th GNU Radio Conference, September 2019*.

Grayver, Eugene and Utter, Alexander. Extreme Software Defined Radio – GHz in Real-time. *IEEE Aerospace Conference 2020*.

Miller, David T. Demonstration of GNU Radio Compatibility with a NASA Space Communications Network Modem. *Proceedings of the 9th GNU Radio Conference, September 2019*.

Biography

David T. Miller received a B.S. degree in electrical engineering from Virginia Tech and a M.S. degree in electrical engineering from Virginia Tech. He is currently employed as a NASA contractor with Peraton, Inc, but note that all information and opinions presented in this paper come only from the author's independent work and do not reflect the position or opinions in any way of NASA or Peraton.