Demonstration of GNU Radio High Data Rate QPSK Modem at 15.0 Mbps Real-Time with Multi-Core General Purpose Processor (GRCON 2022)

David T. Miller DAVE.TODD.MILLER@GMAIL.COM
Ashburn, VA 20147 USA

Abstract

This paper presents a GNU Radio Modulator/Demodulator (Modem) design and an associated test activity that demonstrates a GNU Radio modem can operate at 15.0 Mbps Real-Time with Quadrature Phase Shift Keying (QPSK), with GNU Radio version 3.10, and with a multi-core (8-core) General Purpose Processor (GPP) inside a relatively low cost Personal Computer (PC). The Modem demodulator design achieves the high date rate with a single GNU Radio flowgraph and without a Field Programmable Gate Array (FPGA) or Graphics Processor Unit (GPU). Specifically, the Modem demodulator design achieves the high data rate by breaking the incoming I/Q sample stream from a LimeSDR-mini into three "chunk" streams. Each chunk stream then flows to a separate Symbol Synchronizer (symbol synchronization) and Costas Loop (carrier synchronization) chain and each chain uses a separate GPP core. The GNU Radio Modem demodulator then "stitches" the original transmitted single stream back together by only using the frame Acquisition Synchronization Marker (ASM) and the known frame length of each frame.

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 PC or Server.
2. A continuous lowering of costs for a PC or Server with multi-core GPPs up to at least 64 cores.

Proceedings of the 12th GNU Radio Conference, Copyright 2022 by the author.

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

Demonstrating via actual testing that a GNU Radio Software Defined Radio (SDR) Demodulator can achieve 15.0 Mbps by using only GPP cores in parallel 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 real-time data rates up to 50 Mbps or even possibly 100 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 (FPGA) boards or Graphics Processor Unit (GPU) devices 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.

A feasibility only approach for conducting Binary Phase Shift Keying (BPSK) at 10.0 Mbps with a multi-core GPP was discussed and presented at the last GNU Radio Conference in September 2021 (Miller, 2021). However, the follow-on approach in this paper improves upon that design used during the BPSK feasibility demonstration in (Miller, 2021). The design and demonstration documented in this paper and the associated flowgraphs and code on github.com (gr-HighDataRate_Modem) can provide one potential path forward for greatly increasing the data rate performance of a GNU Radio modem by using multi-cores of a GPP in parallel.

2. Demonstration Test Objective

The primary objective of this demonstration test activity is to show that one can develop a practical GNU Radio HDR SDR Demodulator that can operate at a data rate of 15.0 Mbps in real-time with a relatively low cost 8-core GPP PC.

3. Scope of Demonstration Test

For Satellite communications, the Quadrature Phase Shift Keying (QPSK) modulation waveform is used extensively. Therefore, the author conducted a QPSK test case at 15.0 Mbps.

4. GNU Radio SDR Design

This section describes the details of the GNU Radio SDR design and implementation. However, please also refer to gr-HighDataRate_Modem on github.com for the required Out-Of-Tree (OOT) Blocks/Code, example flowgraphs, and a detailed Design Document for additional low level details.

The SDR design consists of an inexpensive Lenova IdeaPad 5 laptop (≈$650.00 in CY2021) containing an Advanced Micro Devices (AMD) Ryzen 7-4700U 8-core GPP, the free open source Linux/Ubuntu operating system (Version 20.04), the free open source GNU Radio software (Version 3.10), and an inexpensive (<$200.00 in CY 2021) 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 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 design in this paper eliminates the need for the frame counter in the high data rate BPSK feasibility design of (Miller, 2021). This new improved design can now support generic types of link framing by only requiring knowledge of the frame length and frame Acquisition Synchronization Marker (ASM) to stitch the frames back together again after they pass through the parallel synchronization chains.

Figure 1 depicts the GNU Radio Companion (GRC) Flowgraph Graphical User Interface (GUI) for the QPSK demodulator design and test demonstration for a continuous stream of received frames with a constant frame length of 4192 bits including a 32 bit ASM. Specifically, the design includes using the Consultative Committee for Space Data Systems (CCSDS) 32-bit ASM pattern (1ACFFC1D in hexadecimal) for the frame ASM.

The Demodulator Flowgraph in Figure 1 consists of three main parts:

1. Demodulator Front-End
2. Demodulator Parallel Synchronization Chains
3. Demodulator Back-End

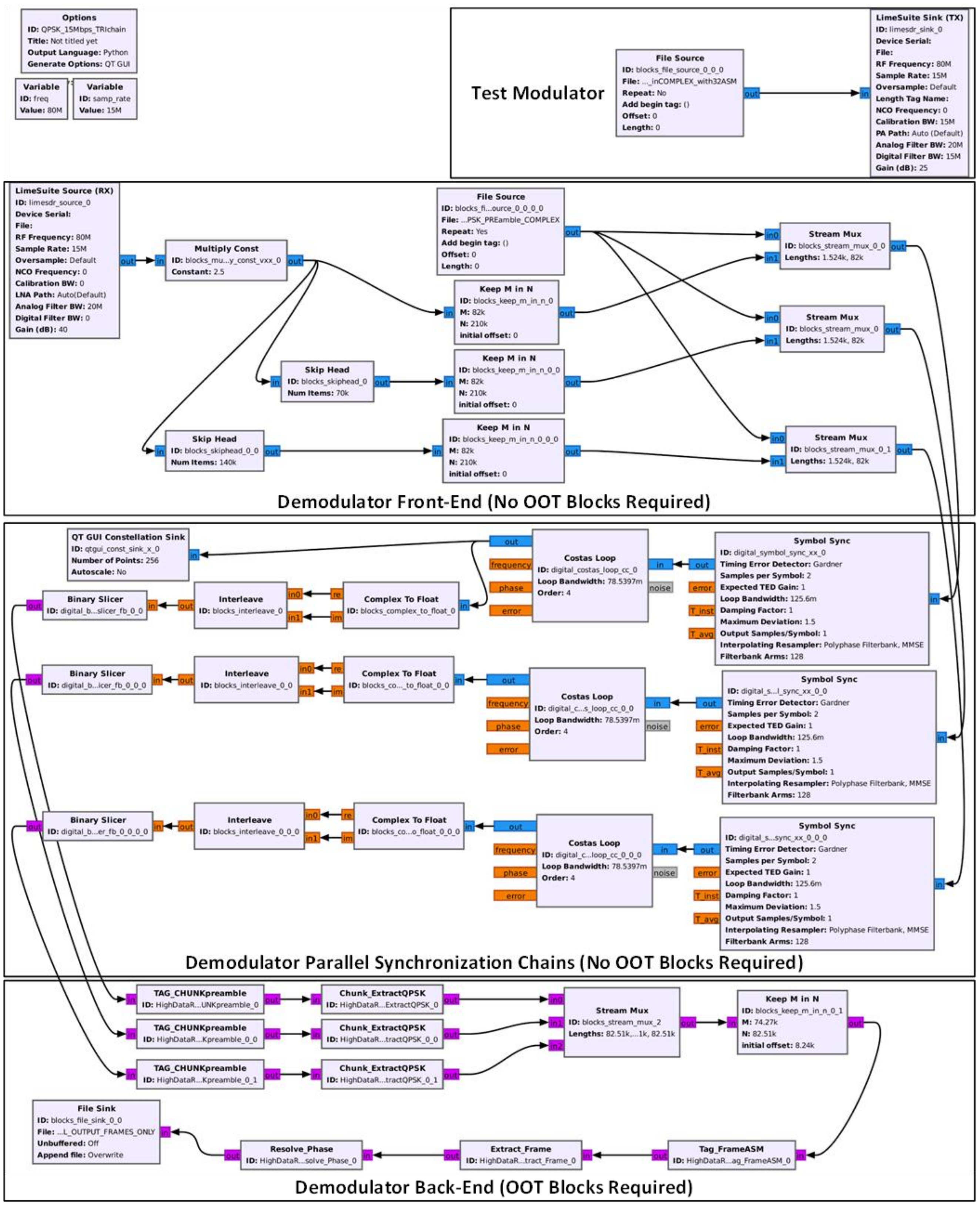
The design consists of mostly the GNU Radio blocks that were already available in the GNU Radio Block In-Tree library except for the final frame extraction and stitching blocks in the Demodulator Back-End: “TAG_CHUNKpreamble” block, “Chunk_ExtractQPSK” block, “Tag_FrameASM” block, Extract_Frame” block, and “Resolve_Phase” block.

The design replaces the frame counter to stitch the frames back together as used in (Miller, 2021) by using a new approach that includes adding a “Chunk Preamble” to the front of each chunk before the chunk enters one of the parallel synchronization chains. The demodulator can now use the added chunk preamble along with the known frame ASM and known frame length without a frame counter to recover the original frame data stream that the Radio Frequency (RF) transmitting source generated, modulated, and radiated.

(Grayver et al., 2020) introduced the term “chunk” in detail. A single chunk is one continuous stream of samples that enters a single symbol synchronizer block without a discontinuity: 82,000 samples as seen in the Demodulator Front-End of the flowgraph in Figure 1 (Three “Keep M in N” blocks with M set to 82,000).

4.1. Demodulator Front-End Functions and Data Flows

The Demodulator Front-End (see Figure 1) breaks the incoming single serial complex I/Q channel sample stream from the LimeSDR-Mini into parallel overlapping chunk streams and also adds a complex I/Q fixed pattern 1524 sample “Chunk Preamble” to the front of each individual chunk in each chunk stream before the chunk streams enter the next part of the Demodulator Flowgraph which is the Demodulator “Parallel Synchronization Chains” (see Figure 1). Each Synchronization Chain consists of a Symbol Synchronizer Block, Costas Loop Block, Complex to Float Block, Interleave Block, and Binary Slicer Block.


Test Modulator
Demodulator Front-End (No OOT Blocks Required)
Demodulator Parallel Synchronization Chains (No OOT Blocks Required)
Demodulator Back-End (OOT Blocks Required)
in
out
in0
in1
in2
re
im
frequency
phase
error
noise
T_inst
T_avg

Options
ID: QPSK_15Mbps_TRIchain
Title: Not titled yet
Output Language: Python
Generate Options: QT GUI

Variable
ID: freq
Value: 80M

Variable
ID: samp_rate
Value: 15M

File Source
ID: blocks_file_source_0_0_0
File: ..._inCOMPLEX_with32ASM
Repeat: No
Add begin tag: ()
Offset: 0
Length: 0

LimeSuite Sink (TX)
ID: limesdr_sink_0
Device Serial:
File:
RF Frequency: 80M
Sample Rate: 15M
Oversample: Default
Length Tag Name:
NCO Frequency: 0
Calibration BW: 15M
PA Path: Auto (Default)
Analog Filter BW: 20M
Digital Filter BW: 15M
Gain (dB): 25

LimeSuite Source (RX)
ID: limesdr_source_0
Device Serial:
File:
RF Frequency: 80M
Sample Rate: 15M
Oversample: Default
NCO Frequency: 0
Calibration BW: 0
LNA Path: Auto(Default)
Analog Filter BW: 20M
Digital Filter BW: 0
Gain (dB): 40

Multiply Const
ID: blocks_mu...y_const_vxx_0
Constant: 2.5

File Source
ID: blocks_fi...ource_0_0_0_0
File: ...PSK_PREamble_COMPLEX
Repeat: Yes
Add begin tag: ()
Offset: 0
Length: 0

Keep M in N
ID: blocks_keep_m_in_n_0
M: 82k
N: 210k
initial offset: 0

Skip Head
ID: blocks_skiphead_0
Num Items: 70k

Keep M in N
ID: blocks_keep_m_in_n_0_0
M: 82k
N: 210k
initial offset: 0

Skip Head
ID: blocks_skiphead_0_0
Num Items: 140k

Keep M in N
ID: blocks_keep_m_in_n_0_0_0
M: 82k
N: 210k
initial offset: 0

Stream Mux
ID: blocks_stream_mux_0_0
Lengths: 1.524k, 82k

Stream Mux
ID: blocks_stream_mux_0
Lengths: 1.524k, 82k

Stream Mux
ID: blocks_stream_mux_0_1
Lengths: 1.524k, 82k

QT GUI Constellation Sink
ID: qtgui_const_sink_x_0
Number of Points: 256
Autoscale: No

Costas Loop
ID: digital_costas_loop_cc_0
Loop Bandwidth: 78.5397m
Order: 4

Symbol Sync
ID: digital_symbol_sync_xx_0
Timing Error Detector: Gardner
Samples per Symbol: 2
Expected TED Gain: 1
Loop Bandwidth: 125.6m
Damping Factor: 1
Maximum Deviation: 1.5
Output Samples/Symbol: 1
Interpolating Resampler: Polyphase Filterbank, MMSE
Filterbank Arms: 128

Binary Slicer
ID: digital_b...slicer_fb_0_0

Interleave
ID: blocks_interleave_0

Complex To Float
ID: blocks_complex_to_float_0

Binary Slicer
ID: digital_b...icer_fb_0_0_0

Interleave
ID: blocks_interleave_0_0

Complex To Float
ID: blocks_co..._to_float_0_0

Costas Loop
ID: digital_c...s_loop_cc_0_0
Loop Bandwidth: 78.5397m
Order: 4

Symbol Sync
ID: digital_s...l_sync_xx_0_0
Timing Error Detector: Gardner
Samples per Symbol: 2
Expected TED Gain: 1
Loop Bandwidth: 125.6m
Damping Factor: 1
Maximum Deviation: 1.5
Output Samples/Symbol: 1
Interpolating Resampler: Polyphase Filterbank, MMSE
Filterbank Arms: 128

Binary Slicer
ID: digital_b...er_fb_0_0_0_0

Interleave
ID: blocks_interleave_0_0_0

Complex To Float
ID: blocks_co...o_float_0_0_0

Costas Loop
ID: digital_c...loop_cc_0_0_0
Loop Bandwidth: 78.5397m
Order: 4

Symbol Sync
ID: digital_s...sync_xx_0_0_0
Timing Error Detector: Gardner
Samples per Symbol: 2
Expected TED Gain: 1
Loop Bandwidth: 125.6m
Damping Factor: 1
Maximum Deviation: 1.5
Output Samples/Symbol: 1
Interpolating Resampler: Polyphase Filterbank, MMSE
Filterbank Arms: 128

TAG_CHUNKpreamble
ID: HighDataR...UNKpreamble_0

Chunk_ExtractQPSK
ID: HighDataR...ExtractQPSK_0

TAG_CHUNKpreamble
ID: HighDataR...Kpreamble_0_0

Chunk_ExtractQPSK
ID: HighDataR...tractQPSK_0_0

TAG_CHUNKpreamble
ID: HighDataR...Kpreamble_0_1

Chunk_ExtractQPSK
ID: HighDataR...tractQPSK_0_1

Stream Mux
ID: blocks_stream_mux_2
Lengths: 82.51k,...1k, 82.51k

Keep M in N
ID: blocks_keep_m_in_n_0_1
M: 74.27k
N: 82.51k
initial offset: 8.24k

Tag_FrameASM
ID: HighDataR...ag_FrameASM_0

Extract_Frame
ID: HighDataR...tract_Frame_0

Resolve_Phase
ID: HighDataR...solve_Phase_0

File Sink
ID: blocks_file_sink_0_0
File: ...L_OUTPUT_FRAMES_ONLY
Unbuffered: Off
Append file: Overwrite

Figure 1: GNU Radio Companion GUI Flowgraph for QPSK Test Case (Available on gr-HighDataRate_Modem)

Figure 2 depicts the Demodulator Front-End functional chunk creation for each chain in detail. Figure 2 functionally depicts the 12,000 sample overlap of the chunk chains (202, 206, and 209 in Figure 2) so that well more than one frame of overlap at the beginning and end of each chunk occurs relative to an adjacent chuck that will be on a different parallel synchronization chain. An overlap bit length beyond just one 4192 bit frame was also used because the Symbol Synchronizer blocks and Costas Loop blocks 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 82,000 sample chunk can vary randomly by a few symbols from chunk to chunk depending on the clock stability of the transmitter relative to the LimeSDR-Mini dongle clock. The design also uses a large overlap well beyond one frame so that an implementer can add support for different frame lengths in the future without the need to change the flowgraph blocks significantly.

Figure 3 depicts how the Demodulator Front-End functionally adds a “Chunk Preamble” to each 82000 sample chunk. The design uses the Chunk Preamble as a part of the frame stitching process in the Demodulator Back-End after the chunks pass through the three parallel Symbol Synchronizer and Costas Loop chains in the “Parallel Synchronization Chains” part of the Demodulator.

Specifically, this QPSK demodulator design adds the 1524 sample “Chunk Preamble” from a file source block in the Demodulator Front-End to the front of each 82000 sample chunk by using a “Stream Mux” block before the chunks enter the input of one of the Parallel Synchronization Chains as Figure 3 functionally depicts.

The added fixed “Chunk Preamble” stored in a prepared file starts with a complex I/Q pattern of 960 samples (corresponding to 960 bits based on 2 samples/symbol and 2 bits per symbol for QPSK) as follows: “1+j1, 1+j1, -1-j1, -1-j1, 1+j1, 1+j1, -1-j1, -1-j1, 1+j1, 1+j1, -1-j1, -1-j1, ..., 1+j1, 1+j1, -1-j1, -1-j1, 1+j1, 1+j1, -1-j1, -1-j1”. The next part of the chunk preamble is the 64 sample Chunk Preamble Marker (64 bits based on 2 samples/symbol and 2 bits/symbol) also in I/Q complex format and is a randomized pattern in bits for later chunk synchronization after the chunk streams exit the Parallel Synchronization Chains. One can use any 64 bit pattern (64 complex samples for QPSK) with quality randomization characteristics for use as a synchronization marker for the “Chunk Preamble Marker”. Following the preamble marker is the final part of the preamble which is a 500 zeros sample sequence also in complex format (0+j0 for each sample). The pre-created “Chunk Preamble” file for the file source in the Demodulator Front-End of Figure 1 is available on gr-HighDataRate_Modem located on github.com.

The “Chunk Preamble” samples before and after the chunk preamble marker assist with later symbol synchronization and carrier synchronization that is required for each new chunk and preamble marker. Each Parallel Synchronization Chain must conduct two re-synchronization processes for each passing 82,000 sample chunk and its preamble marker (synchronize on the preamble marker and then synchronize on the later chunk samples). This chunk preamble approach eliminates the need for a frame counter to “stitch” the original transmitted frame stream back together as discussed in (Miller, 2021).

Each Symbol Synchronizer block and each Costas Loop block was placed onto a dedicated single GPP core.

The final step for the Demodulator Front-End is to send the overlapping chunk streams with preambles to the Demodulator “Parallel Synchronization Chains”.

4.2. Demodulator Parallel Synchronization Chains Functions and Data Flows

The Demodulator “Parallel Synchronization Chains” conduct the processing intensive synchronization functions using parallel synchronization chains and GPP cores. The input of the Demodulator Parallel Synchronization Chains is a “complex” type. The output of the Demodulator Parallel Synchronization Chains is a “char” type.

4.3. Demodulator Back-End Functions and Data Flows

After the three Parallel Synchronization Chains, Figure 1 depicts how the Demodulator Back-End multiplexes the three parallel chunk streams (now bits not complex I/Q samples) into a final single stream of chunks in original order by using the Preamble Marker of each chunk, the “TAG_CHUNKpreamble” and “Chunk_ExtractQPSK” blocks, and a “Stream Mux” block. The “TAG_CHUNKpreamble” block in Figure 1 is an OOT block, but it is just a modified version of the “Correlate Access Code – Tag” In-Tree block created to include tags for all four possible QPSK access code (Chunk Preamble Marker) phases: 45°, 135°, 225°, and 315°.

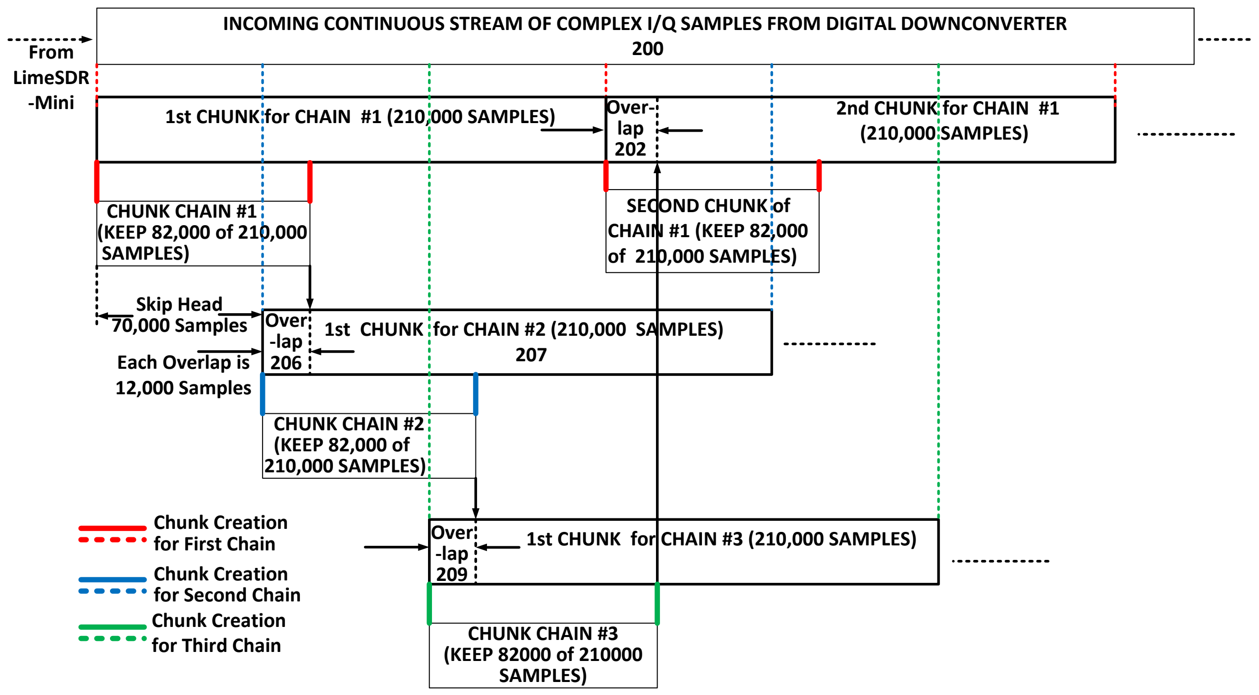

INCOMING CONTINUOUS STREAM OF COMPLEX I/Q SAMPLES FROM DIGITAL DOWNCONVERTER 200
From LimeSDR -Mini
1st CHUNK for CHAIN #1 (210,000 SAMPLES)
Over-lap 202
2nd CHUNK for CHAIN #1 (210,000 SAMPLES)
CHUNK CHAIN #1 (KEEP 82,000 of 210,000 SAMPLES)
SECOND CHUNK of CHAIN #1 (KEEP 82,000 of 210,000 SAMPLES)
Skip Head 70,000 Samples
Each Overlap is 12,000 Samples
Over-lap 206
1st CHUNK for CHAIN #2 (210,000 SAMPLES) 207
CHUNK CHAIN #2 (KEEP 82,000 of 210,000 SAMPLES)
Chunk Creation for First Chain
Chunk Creation for Second Chain
Chunk Creation for Third Chain
Over-lap 209
1st CHUNK for CHAIN #3 (210,000 SAMPLES)
CHUNK CHAIN #3 (KEEP 82000 of 210000 SAMPLES)

Figure 2: Demodulator Front-End Functional: Chunks for Each Symbol Synchronizer Created

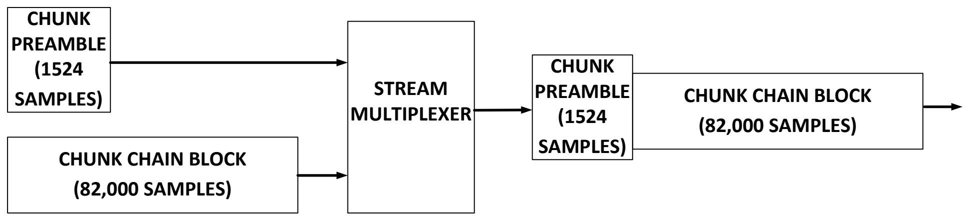
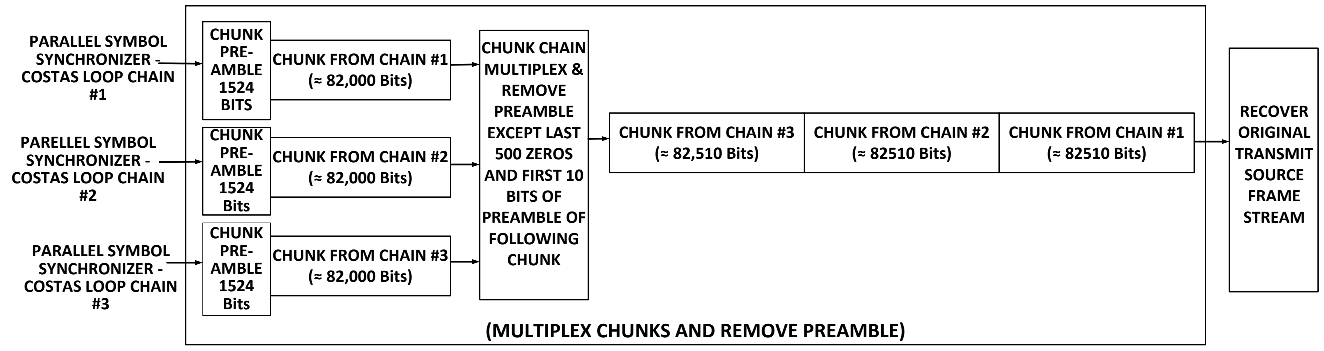
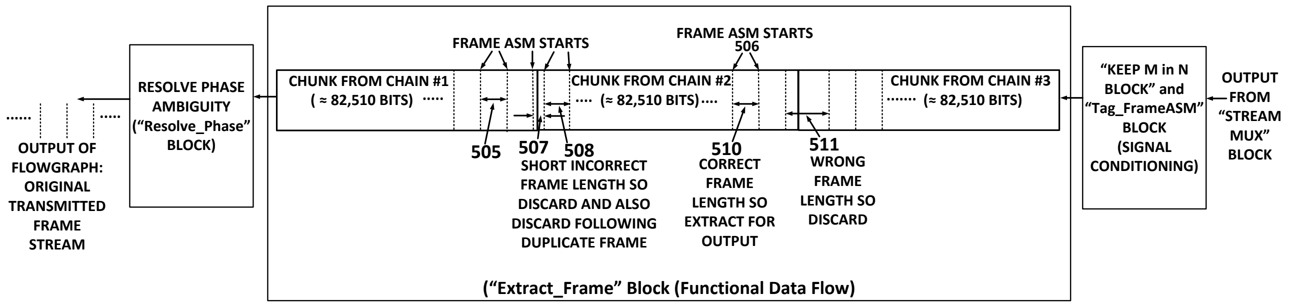

CHUNK PREAMBLE (1524 SAMPLES)
CHUNK CHAIN BLOCK (82,000 SAMPLES)
STREAM MULTIPLEXER
CHUNK PREAMBLE (1524 SAMPLES)
CHUNK CHAIN BLOCK (82,000 SAMPLES)

Figure 3: Demodulator Front-End Functional: Add Chunk Preamble to Each Chunk

Figure 4 functionally depicts the chunk multiplexing process. Then, the now single stream of chunk bits flows from the output of Figure 4 onto the final part of the Demodulator Back-End and the final part of the Demodulator where the Demodulator recovers the exact original frame stream that the test modulator transmitted.

The Demodulator Back-End “Keep M in N” block (Keep 74.27k of 82.51k) in the flowgraph performs the signal conditioning operation to remove the beginning overlap bits of each chunk except for the last overlap bits of length about 1.03 times the frame length. The next signal conditioning step is to tag each frame ASM (“Tag_FrameASM” Block). The “Tag_FrameASM” block in Figure 1 is an OOT block, but it is just a modified version of the “Correlate Access Code – Tag” In-Tree block created to include tags for all 4 possible QPSK access code (ASM) phases: 45°, 135°, 225°, and 315°. Figure 5 functionally depicts how the design extracts each frame from the now serial stream of chunks by using the “Extract_Frame” block in Figure 1 to recover the originally transmitted frame stream without extra bits, without extra frames, and without missing bits/frames. The “Extract_Frame” block passes a correct valid frame only when the frame has a correct frame length of 4192 bits as measured between the start of the frame ASM and the start of the next frame ASM ([505],[506] and [510] of Figure 5). When the distance between the start of the frame ASM and the next frame ASM is greater than the correct known frame length because of a chunk discontinuity/overlap, then the “Extract_Frame” block discards the frame ([511] of Figure 5). A duplicate frame also occurs occasionally because of the chunk overlap. That duplicate frame and the short frame that precedes it are identified by the incorrect short frame between two ASMs ([507] and [508] of Figure 5 depicts). Then, the “Extract_Frame” block discards both


PARALLEL SYMBOL SYNCHRONIZER - COSTAS LOOP CHAIN #1
PARELLEL SYMBOL SYNCHRONIZER - COSTAS LOOP CHAIN #2
PARALLEL SYMBOL SYNCHRONIZER - COSTAS LOOP CHAIN #3
CHUNK PRE-AMBLE 1524 BITS
CHUNK PRE-AMBLE 1524 Bits
CHUNK PRE-AMBLE 1524 Bits
CHUNK FROM CHAIN #1 (≈ 82,000 Bits)
CHUNK FROM CHAIN #2 (≈ 82,000 Bits)
CHUNK FROM CHAIN #3 (≈ 82,000 Bits)
CHUNK CHAIN MULTIPLEX & REMOVE PREAMBLE EXCEPT LAST 500 ZEROS AND FIRST 10 BITS OF PREAMBLE OF FOLLOWING CHUNK
CHUNK FROM CHAIN #3 (≈ 82,510 Bits)
CHUNK FROM CHAIN #2 (≈ 82510 Bits)
CHUNK FROM CHAIN #1 (≈ 82510 Bits)
RECOVER ORIGINAL TRANSMIT SOURCE FRAME STREAM
(MULTIPLEX CHUNKS AND REMOVE PREAMBLE)
Figure 4: Demodulator Back-End Functional: Multiplex Parallel Chain Chunk Streams


FRAME ASM STARTS
FRAME ASM STARTS 506
CHUNK FROM CHAIN #1 (≈ 82,510 BITS)
CHUNK FROM CHAIN #2 (≈ 82,510 BITS)
CHUNK FROM CHAIN #3 (≈ 82,510 BITS)
505
507
508
SHORT INCORRECT FRAME LENGTH SO DISCARD AND ALSO DISCARD FOLLOWING DUPLICATE FRAME
510
CORRECT FRAME LENGTH SO EXTRACT FOR OUTPUT
511
WRONG FRAME LENGTH SO DISCARD
"KEEP M in N BLOCK" and "Tag_FrameASM" BLOCK (SIGNAL CONDITIONING)
OUTPUT FROM "STREAM MUX" BLOCK
RESOLVE PHASE AMBIGUITY ("Resolve_Phase" BLOCK)
OUTPUT OF FLOWGRAPH: ORIGINAL TRANSMITTED FRAME STREAM
("Extract_Frame" Block (Functional Data Flow)
Figure 5: Demodulator Back-End Functional: Extract Original Transmitted Frame Stream

frames even though one of the frames (duplicate frame) has the correct frame length. The C++ "memcpy" function is used for speed in the "Extract_Frame" block.

Then the OOT "Resolve_Phase" block in Figure 1 uses the ASM to resolve the phase ambiguity: 45°, 135°, 225°, and 315°. The block rotates the bits in each frame depending on the ASM phase rotation of each frame. The output of the "Resolve_Phase" block is the original continuous frame data stream that the Radio Frequency (RF) transmitting source (Test Modulator) generated, modulated, and radiated.

For many of the OOT blocks, the GNU Radio outputs for each GNU Radio 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 call to the "Work" function. Specifically, the design code sets the minimum multiple value for the noutput_items parameter in order to guarantee each GNU Radio Scheduler "Work Call" processes at least 15-20 frames (One frame is 4192 bits in length). The long length blocks improve GNU Radio flowgraph speed and performance at high data rates.

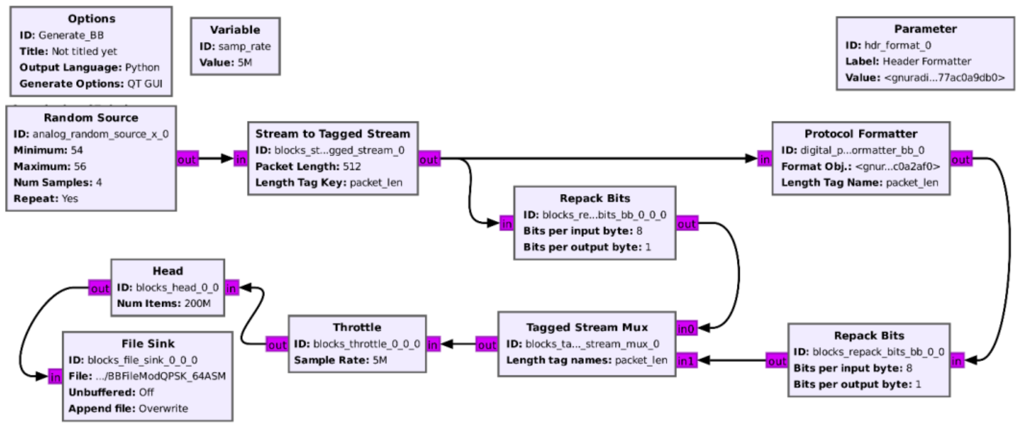
One can review the exact code for each OOT block at gr-HighDataRate_Modem.

4.4. Test Modulator

When running the transmit/receive loops in Figure 1, the transmit Test Modulator signal originates from a prepared modulation file so that the modulator running at 15.0 Megasamples per second (Msps) only requires one GPP core when testing the GNU Radio Demodulator.

Figure 6 depicts the creation of the modulator file in the initial baseband character format (4160 bits for each frame plus a 64 bit ASM that the Figure 7 flowgraphs will convert to a 32 bit ASM). The included header frame counter is available for post-test bit and frame error analysis.

Figure 7 depicts the last three modulator file creation flowgraphs that sequentially convert the baseband frame stream from the Figure 6 output into the final complex QPSK frame stream (with the 32 bit ASM) required by the Test Modulator File Source in Figure 1. The design includes a frame size of 4192 bits in length including a 32 bit ASM and 64 bit header after the ASM (with the header frame counter used for post-test analysis).



Options
ID: Generate_BB
Title: Not titled yet
Output Language: Python
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 5M

Parameter
ID: hdr_format_0
Label: Header Formatter
Value: <gnuradi...77ac0a9db0>

Random Source
ID: analog_random_source_x_0
Minimum: 54
Maximum: 56
Num Samples: 4
Repeat: Yes

Stream to Tagged Stream
ID: blocks_st...gged_stream_0
Packet Length: 512
Length Tag Key: packet_len

Protocol Formatter
ID: digital_p...ormatter_bb_0
Format Obj.: <gnur...c0a2af0>
Length Tag Name: packet_len

Repack Bits
ID: blocks_re...bits_bb_0_0_0
Bits per input byte: 8
Bits per output byte: 1

Head
ID: blocks_head_0_0
Num Items: 200M

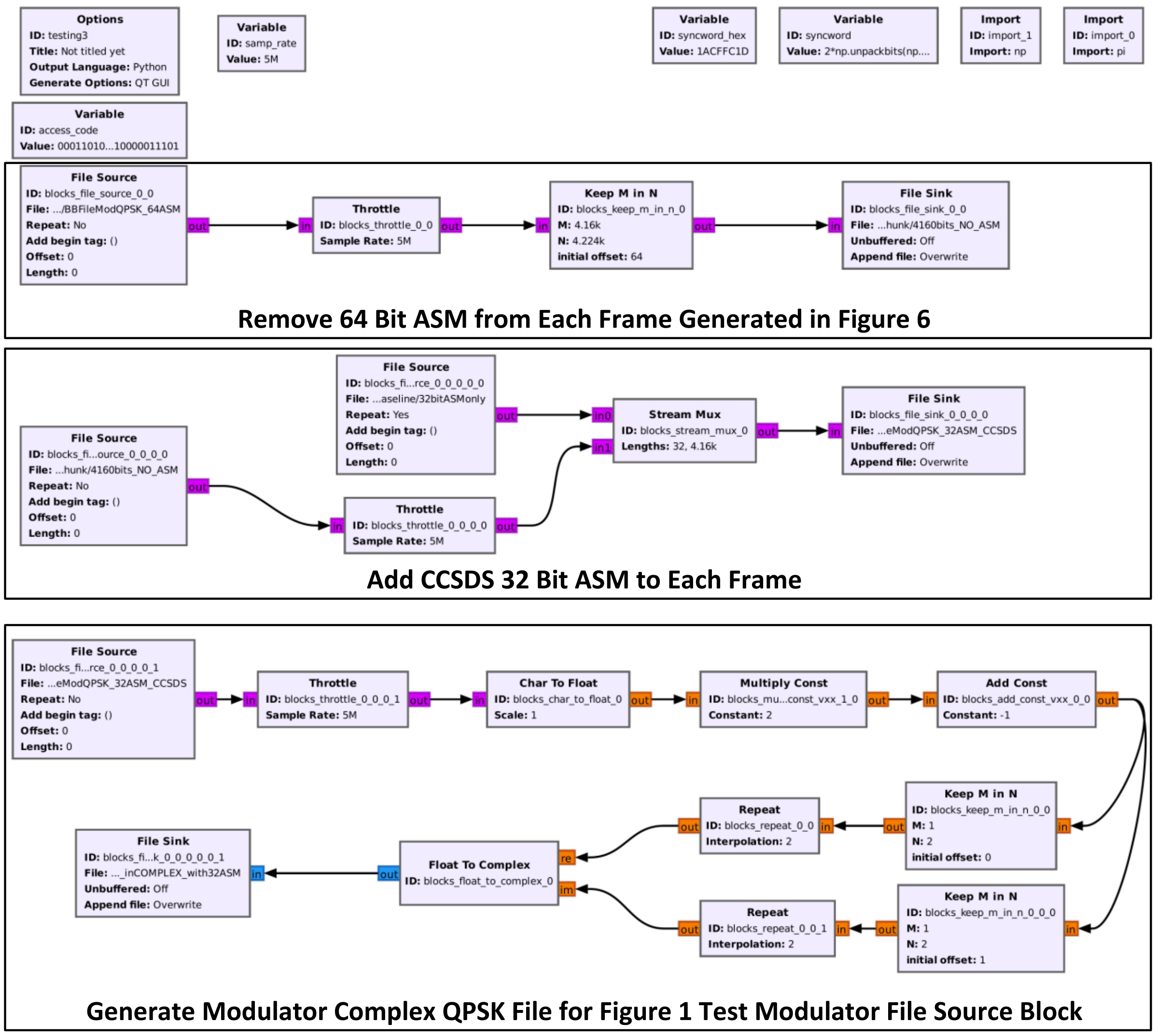
Throttle
ID: blocks_throttle_0_0_0
Sample Rate: 5M

Tagged Stream Mux
ID: blocks_ta..._stream_mux_0
Length tag names: packet_len

Repack Bits
ID: blocks_repack_bits_bb_0_0
Bits per input byte: 8
Bits per output byte: 1

File Sink
ID: blocks_file_sink_0_0_0
File: .../BBFileModQPSK_64ASM
Unbuffered: Off
Append file: Overwrite

Figure 6: GNU Radio Companion GUI Flowgraph for Test Modulator File Baseband Creation



Options
ID: testing3
Title: Not titled yet
Output Language: Python
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 5M

Variable
ID: syncword_hex
Value: 1ACFFC1D

Variable
ID: syncword
Value: 2*np.unpackbits(np....

Import
ID: import_1
Import: np

Import
ID: import_0
Import: pi

Variable
ID: access_code
Value: 00011010...10000011101

Remove 64 Bit ASM from Each Frame Generated in Figure 6

File Source
ID: blocks_file_source_0_0
File: .../BBFileModQPSK_64ASM
Repeat: No
Add begin tag: ()
Offset: 0
Length: 0

Throttle
ID: blocks_throttle_0_0
Sample Rate: 5M

Keep M in N
ID: blocks_keep_m_in_n_0
M: 4.16k
N: 4.224k
initial offset: 64

File Sink
ID: blocks_file_sink_0_0
File: ...hunk/4160bits_NO_ASM
Unbuffered: Off
Append file: Overwrite

Add CCSDS 32 Bit ASM to Each Frame

File Source
ID: blocks_fi...rce_0_0_0_0_0
File: ...aseline/32bitASMonly
Repeat: Yes
Add begin tag: ()
Offset: 0
Length: 0

File Source
ID: blocks_fi...ource_0_0_0_0
File: ...hunk/4160bits_NO_ASM
Repeat: No
Add begin tag: ()
Offset: 0
Length: 0

Stream Mux
ID: blocks_stream_mux_0
Lengths: 32, 4.16k

Throttle
ID: blocks_throttle_0_0_0_0
Sample Rate: 5M

File Sink
ID: blocks_file_sink_0_0_0_0
File: ...eModQPSK_32ASM_CCSDS
Unbuffered: Off
Append file: Overwrite

Generate Modulator Complex QPSK File for Figure 1 Test Modulator File Source Block

File Source
ID: blocks_fi...rce_0_0_0_0_1
File: ...eModQPSK_32ASM_CCSDS
Repeat: No
Add begin tag: ()
Offset: 0
Length: 0

Throttle
ID: blocks_throttle_0_0_0_1
Sample Rate: 5M

Char To Float
ID: blocks_char_to_float_0
Scale: 1

Multiply Const
ID: blocks_mu...const_vxx_1_0
Constant: 2

Add Const
ID: blocks_add_const_vxx_0_0
Constant: -1

Keep M in N
ID: blocks_keep_m_in_n_0_0
M: 1
N: 2
initial offset: 0

Repeat
ID: blocks_repeat_0_0
Interpolation: 2

Keep M in N
ID: blocks_keep_m_in_n_0_0_0
M: 1
N: 2
initial offset: 1

Repeat
ID: blocks_repeat_0_0_1
Interpolation: 2

Float To Complex
ID: blocks_float_to_complex_0

File Sink
ID: blocks_fi...k_0_0_0_0_0_1
File: ..._inCOMPLEX_with32ASM
Unbuffered: Off
Append file: Overwrite

Figure 7: GNU Radio Companion GUI Flowgraph for Modulator File Complex Creation with 32 Bit ASM

5. Demonstration Test Approach

The author conducted a 15.0 Mbps QPSK demonstration. The driving GNU Radio block parameter settings for the QPSK test case were as follows:

- RF Center Frequency: 80.0 MHz
- Sample Rate: 15.0 MegaSamples per Second
- Symbol Sync Input: 2.0 Samples per Symbol
- Symbol Sync Output: 1.0 Sample per Symbol
- Costas Loop Order: 4

The demonstration included using the "affinity" setting for each block in order to efficiently use the GPP 8-cores. Table 1 lists the affinity settings for each block (each block was assigned to a specific GPP core) during the test.

Table 1: GNU Radio Block Affinity Settings for Test

Flowgraph Block	Core/Affinity
Test Modulator File Source Block	1
LimeSuite Source (Receiver)	1
LimeSuite Sink (Modulator)	1
Demodulator Front-End Skip and Multiplier Blocks	2
Demodulator Front-End "Keep M in N" and "Stream Mux" Blocks	2
Demodulator Front-End File Source (Preamble) Block	2
Symbol Synchronizer/Costas Loop (Chunk Chain #1):	4
Symbol Synchronizer/Costas Loop (Chunk Chain #2)	5
Symbol Synchronizer/Costas Loop (Chunk Chain #3)	6
"Complex To Float" Blocks	4,5,6
"Binary Slicer" and "Interleave" Blocks	7
"TAG_CHUNKpreamble" Blocks	7
"Chunk_ExtractQPSK" Blocks	7
Demodulator Back-End "Stream Mux" Block	3
Demodulator Back-End "Keep M in N" Block	3
"Tag_FrameASM" Block	3
"Extract_Frame" Block	3
"Resolve_Phase" Block	3
Demodulator Back-End File Sink Block	3

Figure 8 depicts the demonstration loop test configuration with a 50 ohm coaxial cable between the LimeSDR-Mini transmit RF output and LimeSDR-Mini receive RF input.

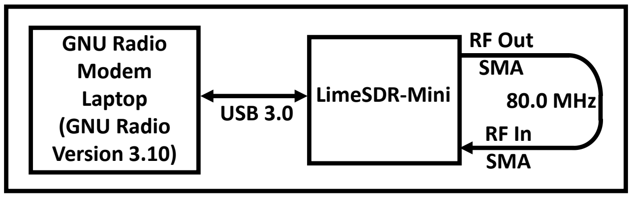


Figure 8: GNU Radio QPSK Demonstration Test Loop Configuration With LimeSDR-Mini

The GNU Radio modem flowgraph transmitted a repeating 32 bit pattern in the data portion of each frame as depicted in Figure 6. The flowgraph operated with 2 samples per symbol in order to achieve the demonstrated data rate of 15.0 Mbps with 15.0 MegaSamples per second/QPSK. The Binary Slicer blocks translated each soft bit into one hard decision bit for ASM tagging, frame stitching, and convenient file storage for post-test playback to check for bit errors and frame errors in non-real-time.

6. Demonstration Test Results

The following performance occurred during the QPSK demonstration test case:

- The GNU Radio SDR Demodulator successfully recovered the transmitted frame stream without bit errors, without missing frames, and without extra frames or extra bits.
- The GNU Radio SDR Demodulator successfully "stitched" (reassembled) the frame stream back together at the 15.0 Mbps high data rate in real-time.
- The GNU Radio SDR Demodulator successfully continually re-established Symbol Synchronizer Lock and Carrier Loop lock for each new "chunk" and each new "Chunk Preamble" marker on each chunk chain.

The author conducted this initial GNU Radio SDR Modem demonstration test phase without adding noise, therefore, perfect Bit Error Rate (BER) performance occurred.

The test results demonstrated that within the scope of this demonstration testing, the implemented GNU Radio SDR modem can achieve QPSK demodulation at a data rate of 15.0 Mbps in real-time by using GPP multi-cores in parallel.

7. Forward Work

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

- Expand the flowgraph capabilities for multiple frame length options rather than just the current fixed frame length capability of 4192 bits including the 32 bit ASM.
- Consider upgrading to a 32 core PC/server to demonstrate the feasibility of a 50 Mbps-100 Mbps QPSK real-time GNU Radio modem using only parallel GPP cores. 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.
- Conduct demonstrations with noise to characterize BER vs Eb/No performance.

8. Conclusions

- Within the scope of this initial demonstration testing phase, the GNU Radio SDR Modem Demodulator can support a data rate of 15.0 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 up to at least 50 Mbps or even 100 Mbps in real-time with a 32-core PC/server, GNU Radio, and a ≥100 Megasample per second “dongle” unit with a 10.0 Gigabit Ethernet interface.

References

Miller, David T. Demonstration of GNU Radio High Data Rate BPSK 10 Mbps Modem Real-Time with Only Multi-Core General Purpose Processors (GRCON 2021). *Proceedings of the 11th GNU Radio Conference, September 2021*.

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

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.