GNU Radio Conference 2022

Demonstration of GNU Radio High Data Rate QPSK 15 Mbps Modem Real-Time with Only Multi-Core General Purpose Processors (Without FPGAs or GPUs)

September 26, 2022

David T. Miller
Dave.Todd.Miller@gmail.com

Background

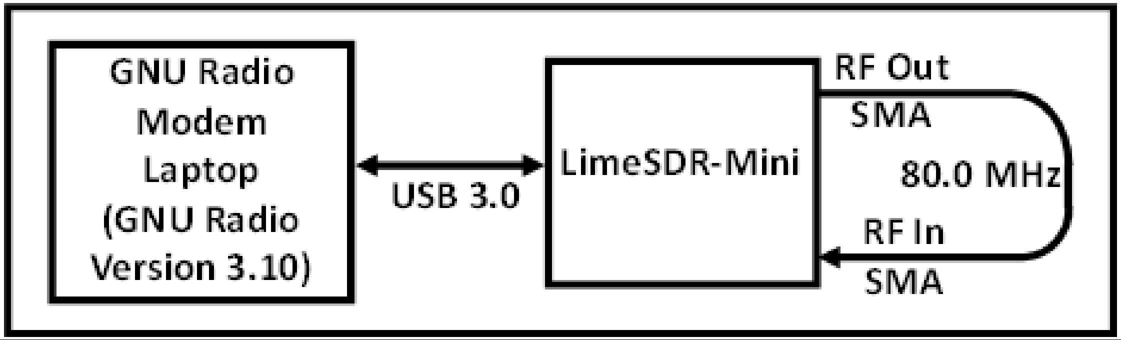
- **<u>2021 Conference Feasibility Approach</u>: Provided paper and Lightning talk at GNU Radio Conference 2021 on "Demonstration of GNU Radio High Data Rate BPSK 10 Mbps Modem Real-Time with Only Multi-Core General Purpose Processors, (Without FPGAs or GPUs)"**
  - ➔ <u>2022 GNU Radio Conference</u>: This presentation and associated paper and associated github site documents an improved design that includes support for QPSK modulation
- **Due to Moore's Law Stagnation for single core in a General Purpose Processor (GPP), GNU Radio Real-time limitation is about 6.0 Mbps for QPSK**
  - ➔ For example: One core per a symbol synchronizer block
  - ➔ Moore's Law continues only via multi-cores architecture approach
  - ➔ Increase data rate well beyond 6.0 Mbps when using only GNU Radio software by using approach and flowgraph that takes advantage of multi-cores

Purpose

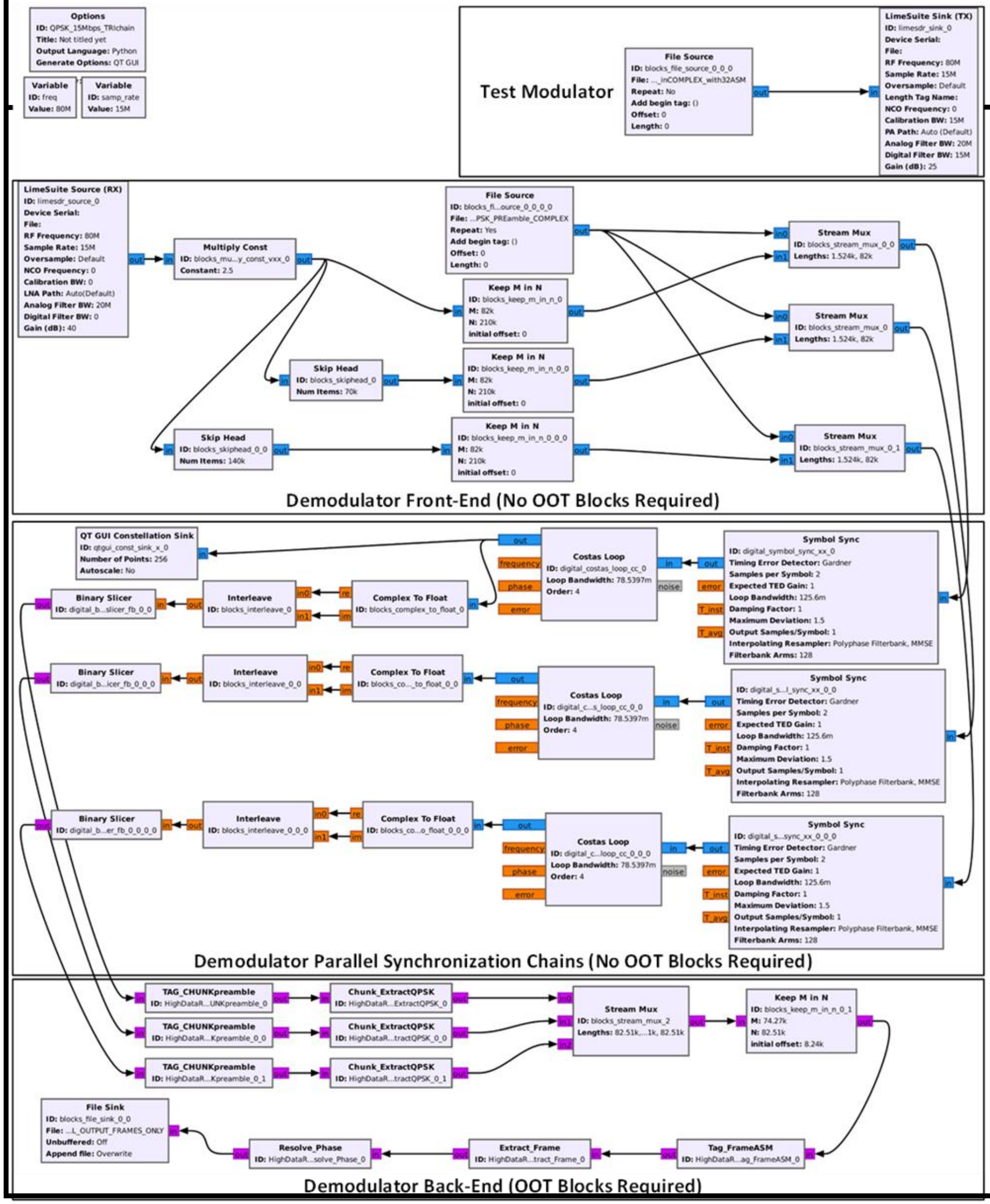
- **Implement practical GNU Radio approach to achieve data rates well beyond 6.0 Mbps without FPGA and/or Graphics Processor Unit (GPU)**
  - ➔ Solution With Multi-cores:
    - — Design breaks up received digital I/Q stream into overlapping “chunks” (blocks) of samples
    - — Then, processes chunks in parallel GPP cores
    - — and then, re-stiches demodulated chunks back together into original transmitted single stream of frames
      - No missing bits
      - No missing frames
      - And without duplicate bits and without duplicate frames due to overlapping approach

Scope

- **Operate at data rate of 15.0 Mbps with GNU Radio, QPSK, LimeSDR-Mini dongle, and parallel multi-core approach:**
  - ➔ QPSK at 15.0 Mbps (15.0 Megasamples per second)
  - ➔ Relatively inexpensive Lenova IdeaPad 5 laptop (≈$650.00 in CY2021) containing an Advanced Micro Devices (AMD) Ryzen 7-4700U 8-core GPP
  - ➔ GNU Radio software (version 3.10.3)
  - ➔ Linux/Ubuntu operating system (version 20.04)
  - ➔ Relatively Inexpensive LimeSDR-Mini dongle (<$200.00 in CY2021)
    - — High Rate Universal Serial Bus (USB) 3.0 interface
    - — >15.0 Megasamples per second capability
  - ➔ Loop back at 80.0 MHz RF frequency
  - ➔ See github site for code, documentation, flowgraphs, and relevant files: https://github.com/DavidToddMiller/gr-HighDataRate_Modem



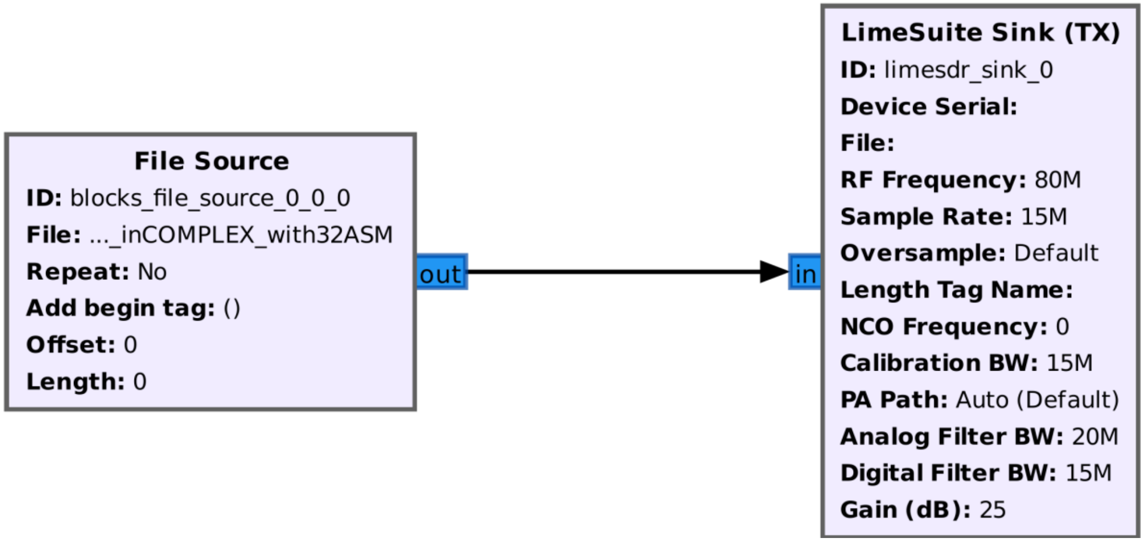
GNU Radio Transmit/Receive Flowgraph Overview



- **The Demodulator consists of 3 main parts:**
  - ➔ Demodulator Front-End
  - ➔ Demodulator Parallel Synchronization chains
  - ➔ Demodulator Back-End
- **Out-Of-Tree (OOT) blocks developed only for Demodulator Back-End to re-stitch original transmitter frame stream together in original order**
- **Test Modulator with pre-modulated complex I/Q file**
- **OOT Code, Flowgraphs, Detailed System Design Document, and test files available on: https://github.com/DavidToddMiller/gr-HighDataRate_Modem**
- **Also, see associated Conference paper for more details**

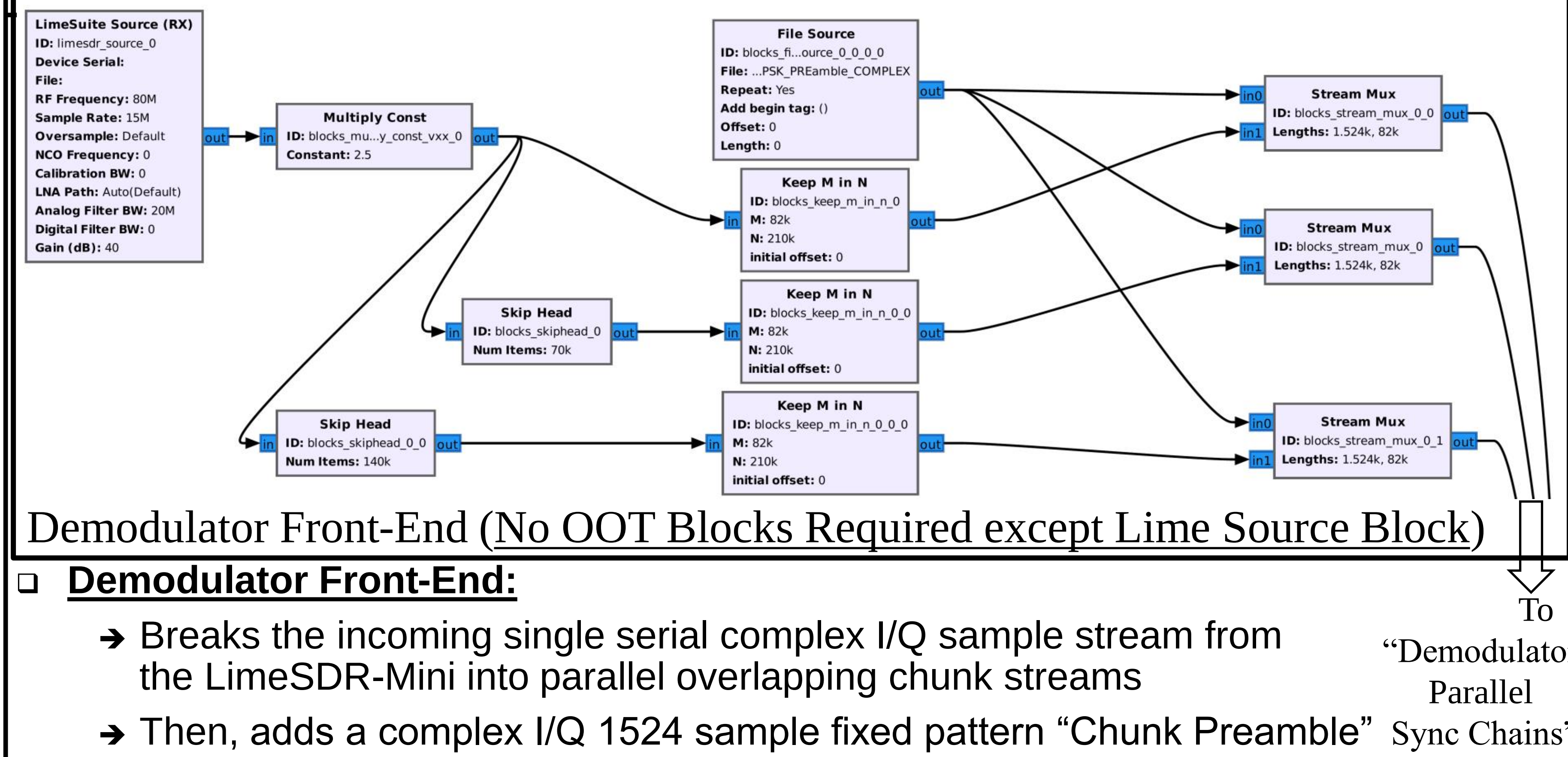
GNU Radio Transmit/Receive Flow Graph
("Zoom In" on Test Modulator)

Test Modulator
(No OOT Blocks
except Lime Sink)



- **<u>Test Modulator:</u>**
  - ➔ File Source provides the Pre-modulated Complex I/Q File for transmission during a loop test
    - — Approach requires just 1 core for modulator portion of modem
    - — Sample Frame stream files for File Source block provided on https://github.com/DavidToddMiller/gr-HighDataRate_Modem

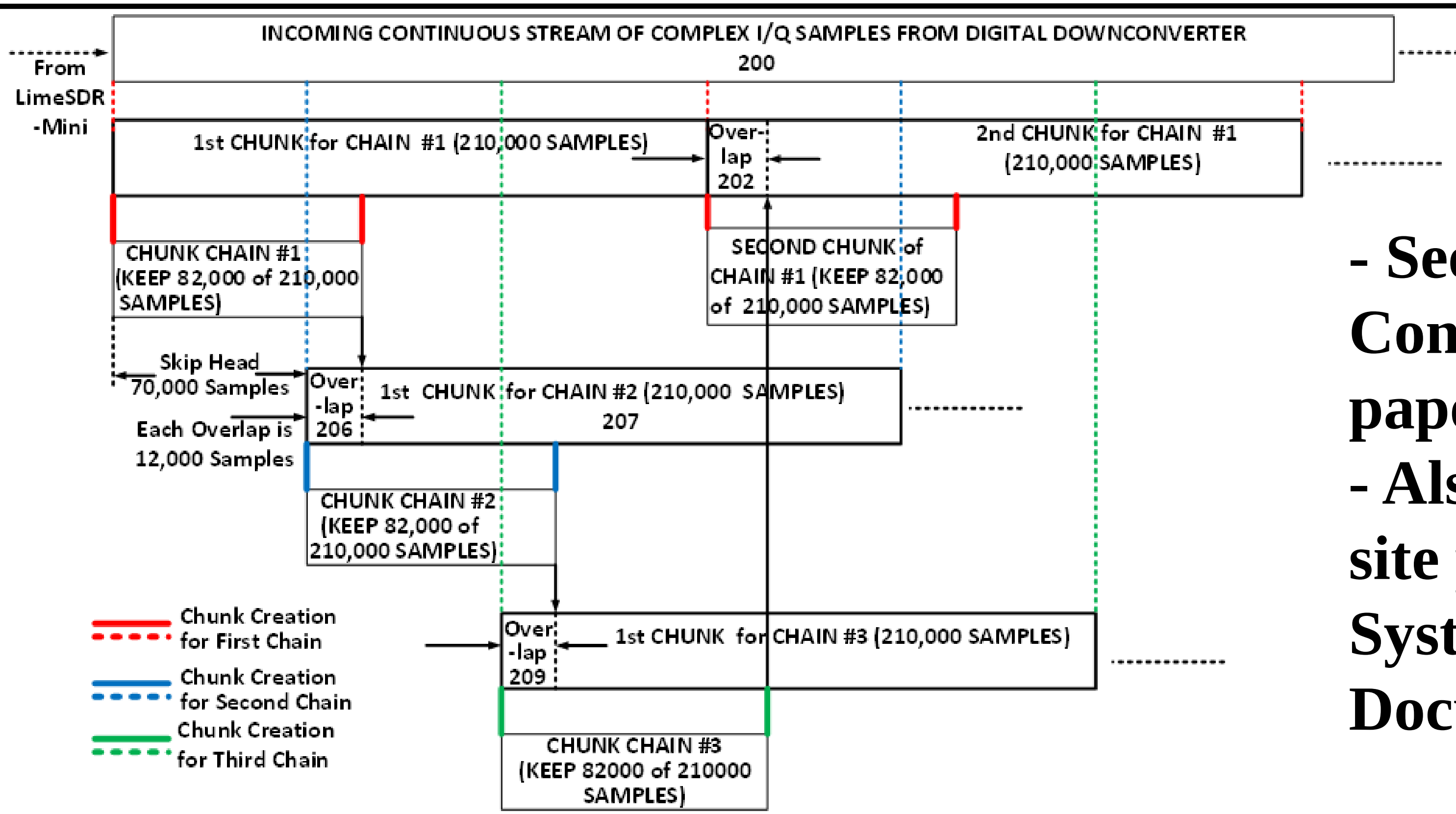
GNU Radio Transmit/Receive Flow Graph
("Zoom In" on "Demodulator Front-End")



Demodulator Front-End (No OOT Blocks Required except Lime Source Block)

- **Demodulator Front-End:**
  - ➔ Breaks the incoming single serial complex I/Q sample stream from the LimeSDR-Mini into parallel overlapping chunk streams
  - ➔ Then, adds a complex I/Q 1524 sample fixed pattern "Chunk Preamble" to front of each individual chunk in each chunk stream
    - — Chunk Preamble used for later frame stitching process in Demodulator Back-End after chunks pass through 3 parallel Symbol Synchronizer and Costas Loop chains
    - — 1524 fixed pattern sample file for File Source block provided on https://github.com/DavidToddMiller/gr-HighDataRate_Modem

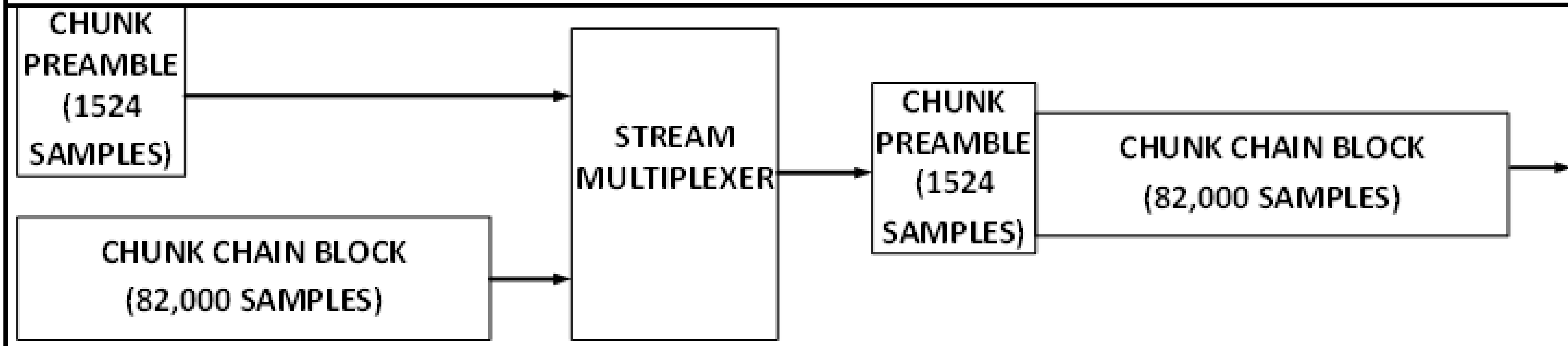
Functional: Create 3 parallel Chunk Streams with Chunk Overlap in "Demodulator Front-End"



- See Associated Conference paper
- Also, github site provides System Design Document

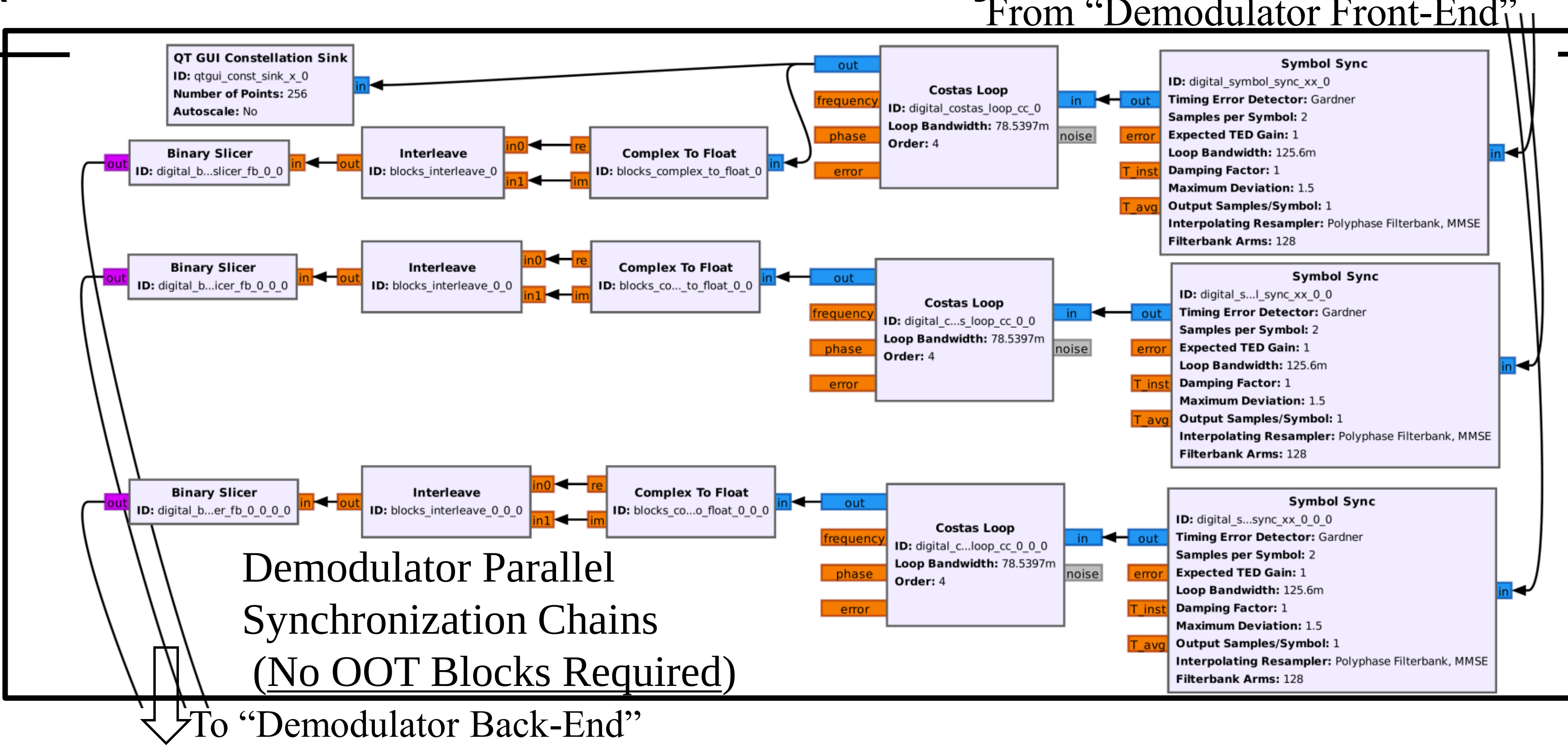
- ❑ 12,000 sample overlap at beginning and end of each chunk occurs relative to adjacent chuck (see 202, 206, and 209 in Figure)
  - ➔ Adjacent chunks will be on different parallel synchronization chains
- ❑ Reasons for overlap covered in later chart on "Demodulator Parallel Synchronization Chains"

Functional: Add Chunk Preamble to Each Chunk in "Demodulator Front-End"



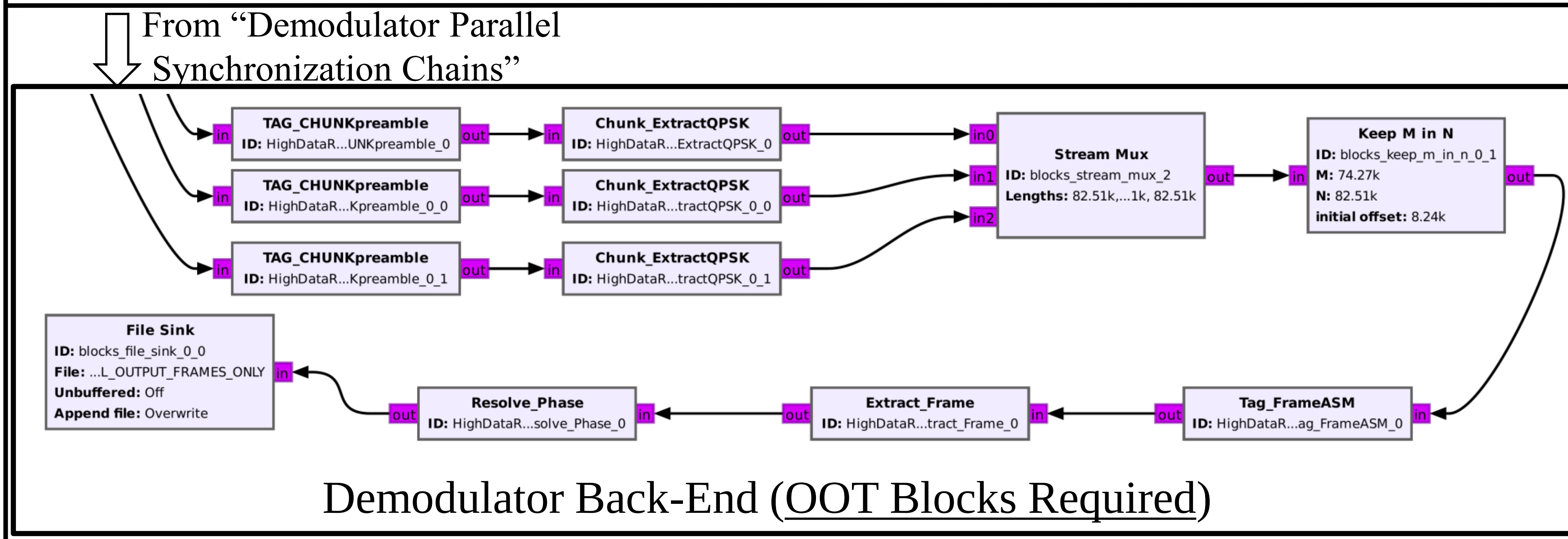
- **QPSK demodulator design adds the 1524 sample "Chunk Preamble" to each 82,000 sample chunk**
  - ➔ See actual "File Source" block and "Stream Mux" blocks in Demodulator Front-End on previous charts
- **"Chunk Preamble" stored in prepared file in complex I/Q format has 3 parts:**
  - ➔ Starts with complex I/Q pattern of 960 samples
    - — 960 bits based on 2 samples/symbol & 2 bits per symbol for QPSK) (-1-j1, -1-j1, 1+j1, 1+j1 …)
  - ➔ Next part of Chunk Preamble: 64 sample Chunk Preamble <u>Marker</u> in complex I/Q format
    - — 64 bits based on 2 samples/symbol and 2 bits/symbol
  - ➔ Final part of Preamble: 500 zeros sample sequence

GNU Radio Transmit/Receive Flow Graph
("Zoom In" on "Demodulator Parallel Synchronization Chains")



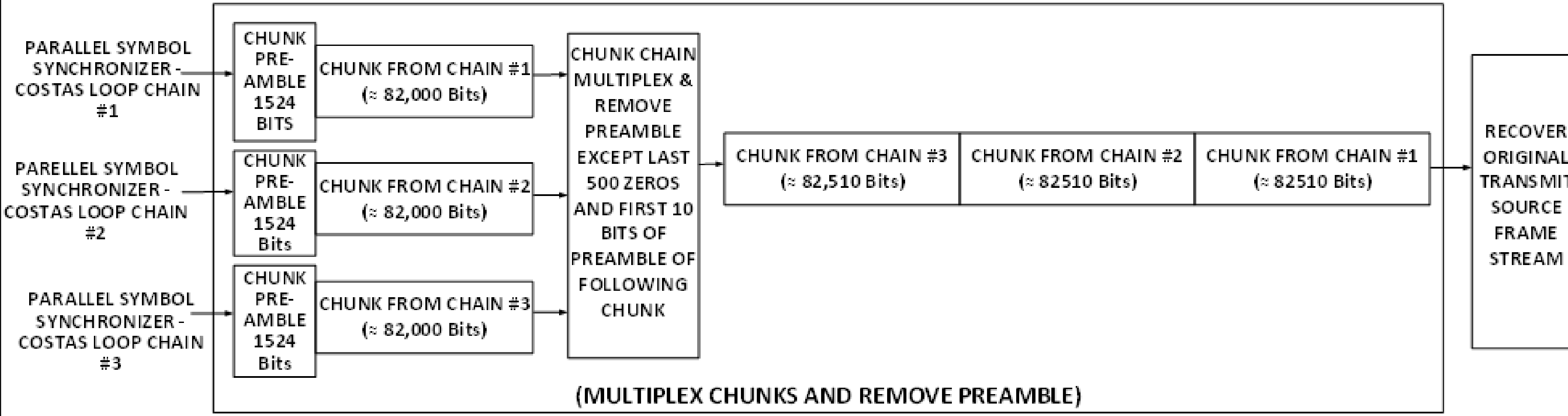
- **Demodulator Parallel Synchronization Chains**: Process 3 chunk streams in 3 parallel GPP cores
- **Chunk Overlap Required for 2 Reasons:**
  1) Symbols per 82,000 sample chunk can vary randomly by a few symbols from chunk to chunk depending on difference between transmitter and receiver (dongle) clock
  2) Symbol Sync & Costas Loop Blocks must continuously sync 2 times for each 82,000 sample chunk and its chunk preamble (error bits at start of each sync)

GNU Radio Transmit/Receive Flow Graph
("Zoom In" on "Demodulator Back-End")



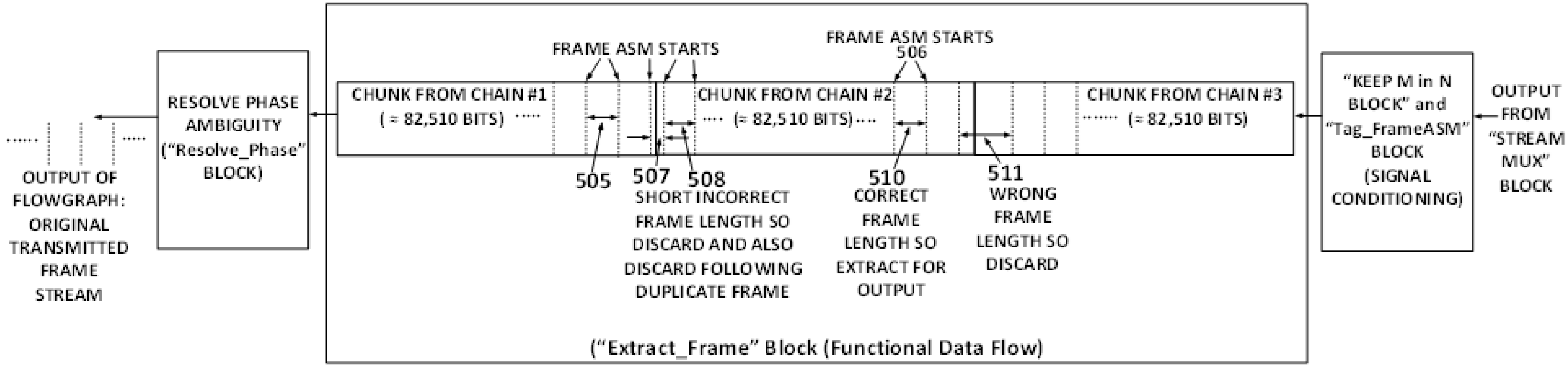
- **Demodulator Back-End:** **Re-stiches demodulated chunks back into the original transmitted frame stream**
- **OOT Blocks Required**:
  - "TAG_CHUNKpreamble" blocks and Tag _FrameASM block:
    - Modified "Correlate Access Code – Tag" In-Tree block to identify and tag all 4 possible QPSK Preamble Marker or ASM phases: 45°, 135°, 225°, and 315°
  - "Chunk_ExtractQPSK" blocks
  - "Extract_Frame" block
  - "Resolve_Phase" block: Rotates bits in entire frame depending on phase rotation of frame's ASM

Functional: “Demodulator Back-End” (“TAG_CHUNKpreamble & “Chunk_ExtractQPSK” Blocks)



- **OOT TAG_CHUNKpreamble & “Chunk_ExtractQPSK” Blocks to puts the chunks stream back into a single chunk stream:**
  - ➔ Identify beginning of each chunk with the chunk marker in the chunk pre-amble
  - ➔ Extract each chunk
  - ➔ Then, with In-Tree blocks (“Stream Mux” and “Keep M in N”), put chunks in order:
    - — Note: Overlap still exists

Functional: "Demodulator Back-End"
("TAG_FrameASM, Extract_Frame, & "Resolve_Phase" Blocks)



- Figure depicts functionally the Tag_FrameASM", Extract_Frame, and Resolve_Phase blocks with a functional flow right to left
- Distance between ASM markers is used to re-stitch the original frame stream back together without errors:
  - ➔ Correct valid frame only when the frame has a correct frame length of 4192 bits (510 above)
  - ➔ Delete bits between ASM markers when wrong frame length (511 above)
  - ➔ Occasional duplicate frames due to overlap are also identified and discarded (507 and 508 above)
- Resolve Phase: Rotate all bits in each frame appropriately based on rotation of bits in each ASM (resolve phase ambiguity)

Block Distribution on 8 GPP Cores
(GNU Radio Blocks have Affinity Setting Feature)

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

Results & Future Work

- **<u>Results:</u> Successfully operated real-time at 15.0 Mbps, QPSK with just GPP cores in parallel**
  - ➔ FPGAs and GPUs were not required for HDR performance
  - ➔ See associated paper in GNU Radio Conference 2022 Proceedings for details
  - ➔ See associated System Design Document details, OOT code, .grc flowgraph for operation with LimeSDR-Mini dongle, and prepared test files:
    - — https://github.com/DavidToddMiller/gr-HighDataRate_Modem
    - — Also, simulation flowgraph on github site for those without dongle who want to try parallel multicore approach
- **<u>Future Work</u>:**
  - ➔ Design should be scalable to data rates a lot higher than 15.0 Mbps (just add more cores and parallel chains)
    - — Requires PC with at least 16-24 cores to add coding, higher data rates, and real-time modulator