Building a Moderately Complex Modem with Spare Parts

Dan CaJacob

Director of Space Systems
HawkEye 360, Inc.

GNURadio
THE FREE & OPEN SOFTWARE RADIO ECOSYSTEM

GNURadio Conference 2017

Thursday 14 September 2017

Abstract

We present a QPSK modem with concatenated coding and IP encapsulation implemented entirely in GnuRadio. The modem is made with core GnuRadio components to the maximum extent possible, then relying on a few existing OOT modules and finally, after exhausting all other options, the author deigns to write a few custom blocks (in Python). Scary topics like constellation rotation/inversion correction and latency reduction are addressed in passing. Fun is had by all. An ill-advised live demo may be demanded. Hecklers point out that this is painfully close to a CCSDS recommended standard.

Overview

- Goal
- A Moderately Complex Modem?
- Spare Parts?
- Quick Success
- Modem
- A Few Custom Parts
- Five Stages of Avoiding Writing Code
- Defeating Latency
- Live Demo
- Future Work
- Acknowledgements

Goal

- Build a full-duplex modem with concatenated FEC codes corresponding closely to CCSDS recommended standard
- Support IP / UDP encapsulation
- Minimize latency
- Bonus: Add TCP acceleration via some PEP magic

GNURadio

THE FREE & OPEN SOFTWARE RADIO ECOSYSTEM

A Moderately Complex Modem?

CCSDS

The Consultative Committee for Space Data Systems

- Higher order modulation (QPSK)
- Concatenated Coding
  - ½ Rate Convolutional
  - RS + Scrambling and Interleaving
- Basically the CCSDS recommendation, without their insane mishmash of higher layers
- Phase Ambiguity Issues
- IP Encapsulation
  - Custom, toy protocol, but you could easily replace with HDLC, etc.
- Latency Reduction
- Do as much as possible in GRC

CCSDS
The Consultative Committee for Space Data Systems

Recommendation for Space Data System Standards

ENCAPSULATION SERVICE

RECOMMENDED STANDARD

CCSDS 133.1-B-2

Note: This current issue includes all updates through Technical Corrigendum 2, dated April 2014

BLUE BOOK
October 2009

CCSDS
The Consultative Committee for Space Data Systems

Recommendation for Space Data System Standards

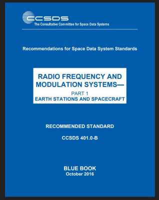
IP OVER CCSDS SPACE LINKS

RECOMMENDED STANDARD

CCSDS 702.1-B-1

Note: This current issue includes all updates through Technical Corrigendum 1, dated April 2014

BLUE BOOK
September 2012



CCSDS
The Consultative Committee for Space Data Systems

Recommendations for Space Data System Standards

RADIO FREQUENCY AND MODULATION SYSTEMS—
PART 1
EARTH STATIONS AND SPACECRAFT

RECOMMENDED STANDARD

CCSDS 401.0-B

BLUE BOOK
October 2016

CCSDS
The Consultative Committee for Space Data Systems

Recommendation for Space Data System Standards

TM SYNCHRONIZATION AND CHANNEL CODING

RECOMMENDED STANDARD

CCSDS 131.0-B-2

BLUE BOOK
August 2011

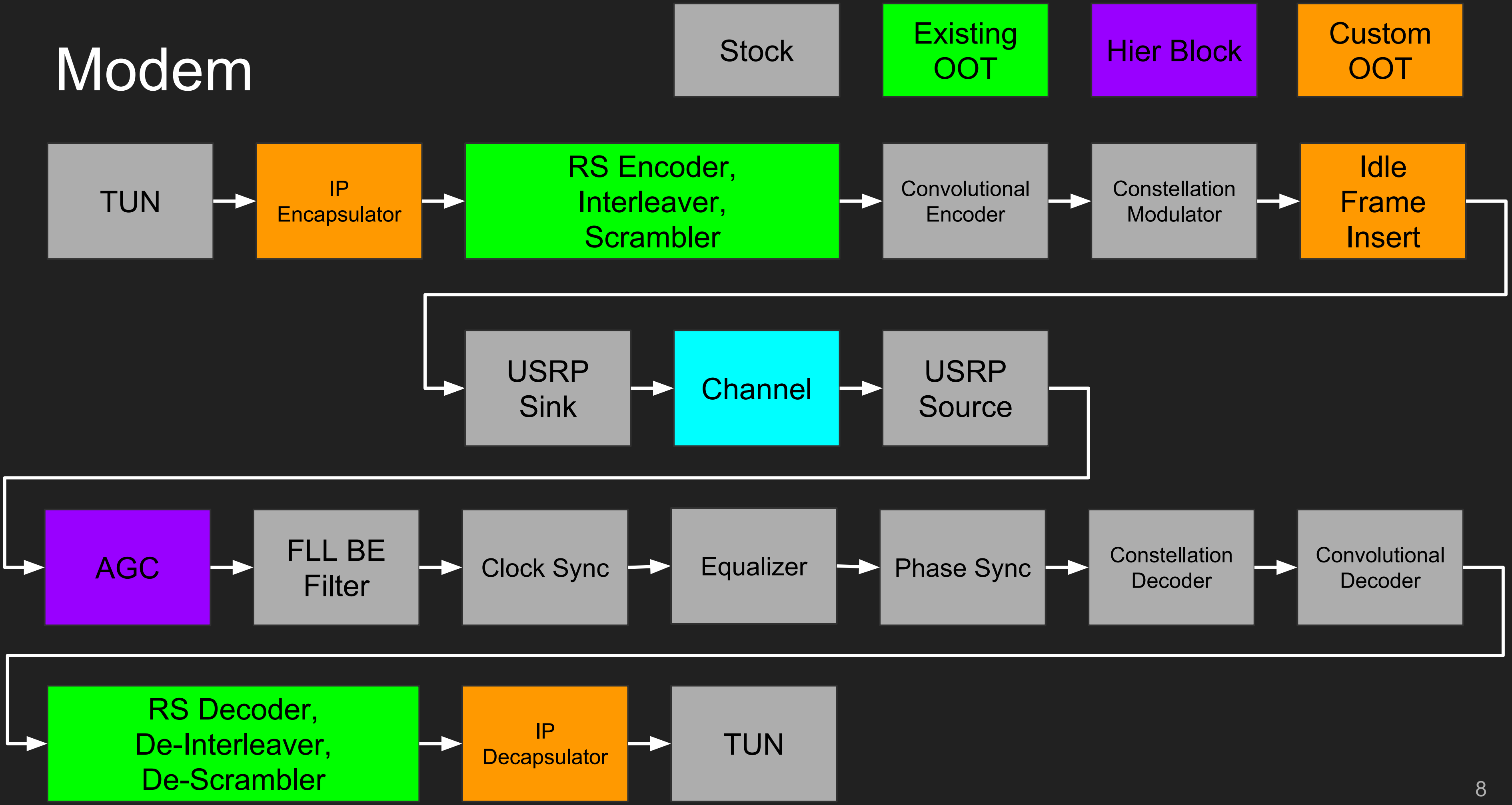
Spare Parts?

- Basically, I bend over backwards to not write a custom block (and fail)
- Because I’m lazy
- And because GNURadio largely makes this possible
- Also, there are some awesome OOT solutions out there
- And others that I completely missed
  - Sorry, gr-mapper

Quick Success

- Not too hard to build a modem with concatenated FEC codes
  - Thanks to awesome OOTs like André Løfaldli’s gr-ccsds
  - Experimented early with the old convolutional encoder block, quickly moved on to newer FEC API blocks
  - GR makes it easy to add Layers incrementally
- Use stock TUN block
  - Network interface to user-land code
- Latency was heinous (like 20 seconds)
  - Adding fill frames before modulator
- Works, but not really a good solution

Modem



Stock

Existing OOT

Hier Block

Custom OOT

TUN → IP Encapsulator → RS Encoder, Interleaver, Scrambler → Convolutional Encoder → Constellation Modulator → Idle Frame Insert

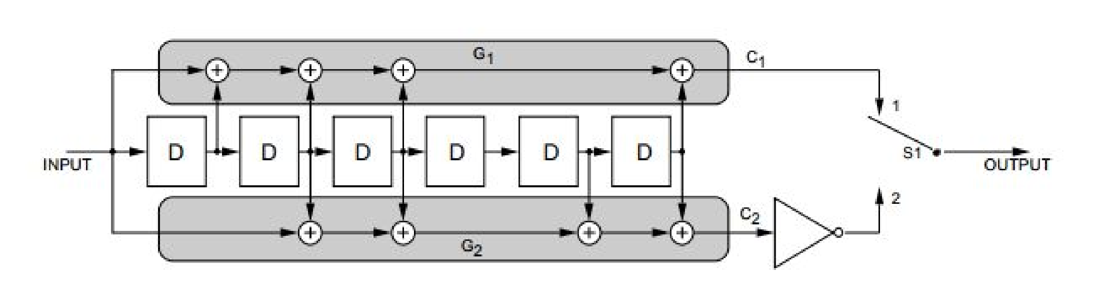
→ USRP Sink → Channel → USRP Source

→ AGC → FLL BE Filter → Clock Sync → Equalizer → Phase Sync → Constellation Decoder → Convolutional Decoder

→ RS Decoder, De-Interleaver, De-Scrambler → IP Decapsulator → TUN

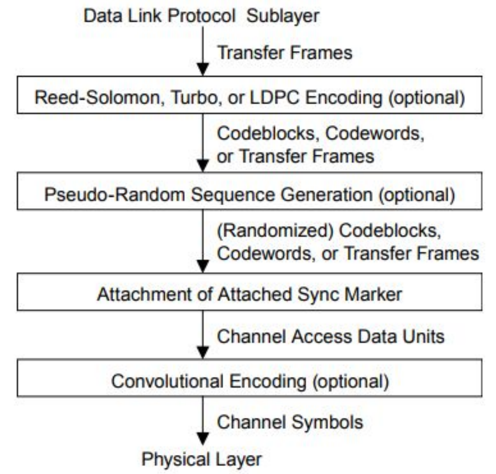
Modem: Convolutional Code

- CCSDS recommends R=½, K=7
  - Each output bit is dependent on 2(K-1) = 12 previous bits
  - So, we should ignore the first 12 bits sometimes
  - As a compromise for simplicity, ignore the first 16 bits
  - ASM is 32 bits, 64 bits after conv. code, so 52 of those are deterministically encoded
    - We just use 48 since it obeys byte boundaries
  - I’m actually not using the inversion yet, but Daniel Estevez made this easier to do, recently



Modem: Reed Solomon Code

- CCSDS recommends
  - RS (255, 223)
  - Interleaving depth = 5
  - Scrambling
- I use André Løfaldli’s gr-cssds OOT
  - Uses libfec (Phil Karn)



Data Link Protocol Sublayer

Transfer Frames

Reed-Solomon, Turbo, or LDPC Encoding (optional)

Codeblocks, Codewords, or Transfer Frames

Pseudo-Random Sequence Generation (optional)

(Randomized) Codeblocks, Codewords, or Transfer Frames

Attachment of Attached Sync Marker

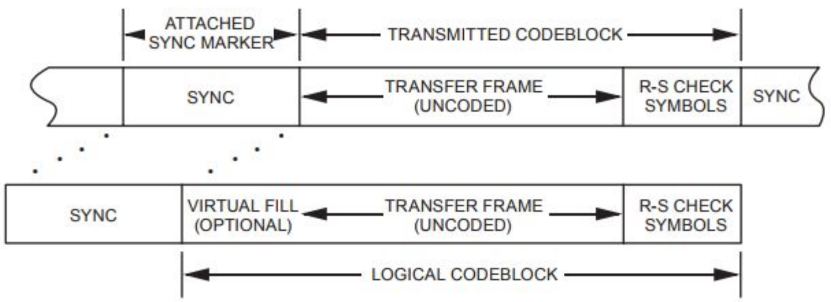
Channel Access Data Units

Convolutional Encoding (optional)

Channel Symbols

Physical Layer

Figure 2-2: Internal Organization of the Sublayer at the Sending End



ATTACHED SYNC MARKER

TRANSMITTED CODEBLOCK

SYNC

TRANSFER FRAME (UNCODED)

R-S CHECK SYMBOLS

SYNC

SYNC

VIRTUAL FILL (OPTIONAL)

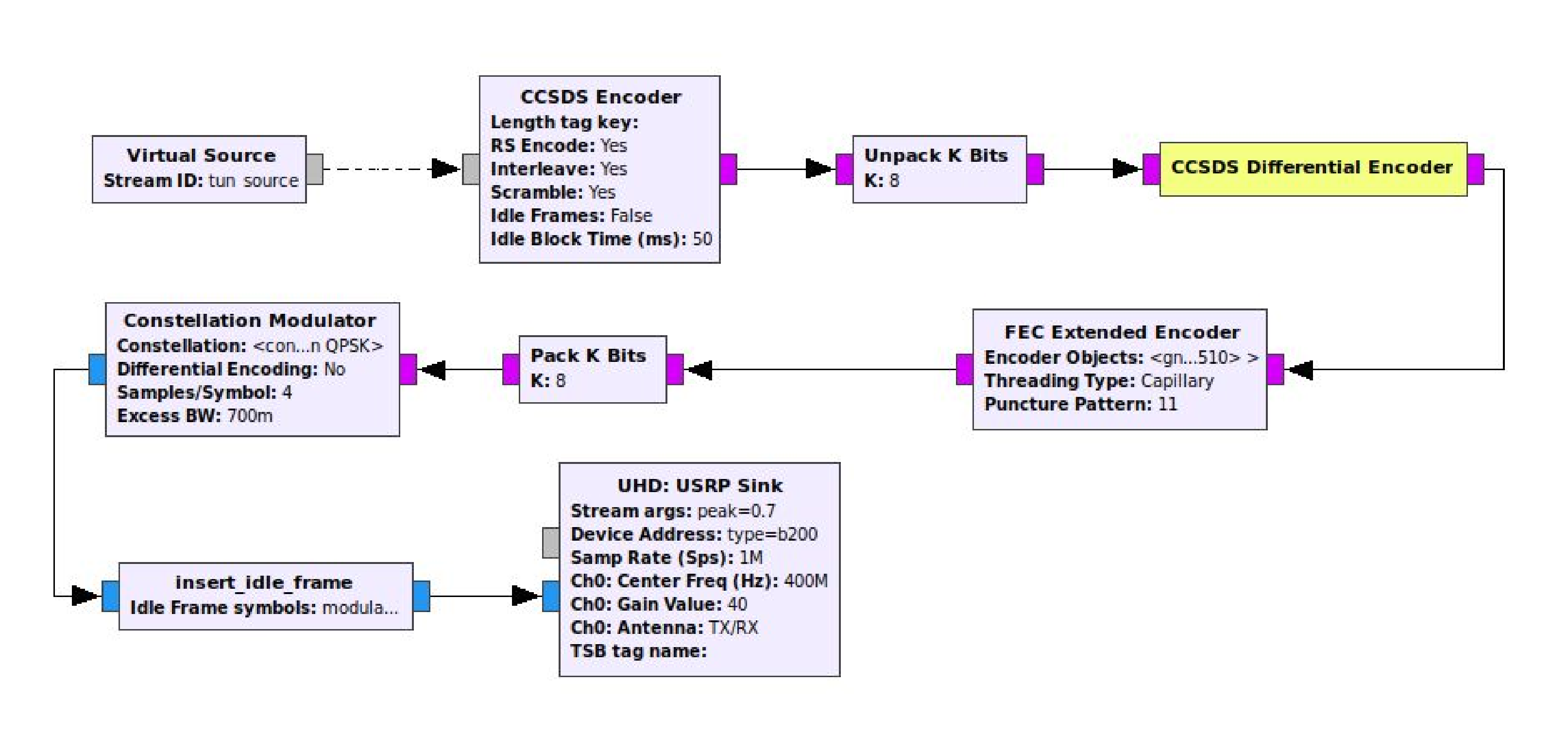
TRANSFER FRAME (UNCODED)

R-S CHECK SYMBOLS

LOGICAL CODEBLOCK

Figure 4-1: Reed-Solomon Codeblock Partitioning

Modem: Transmitter



Virtual Source
Stream ID: tun source

CCSDS Encoder
Length tag key:
RS Encode: Yes
Interleave: Yes
Scramble: Yes
Idle Frames: False
Idle Block Time (ms): 50

Unpack K Bits
K: 8

CCSDS Differential Encoder

FEC Extended Encoder
Encoder Objects: <gn...510> >
Threading Type: Capillary
Puncture Pattern: 11

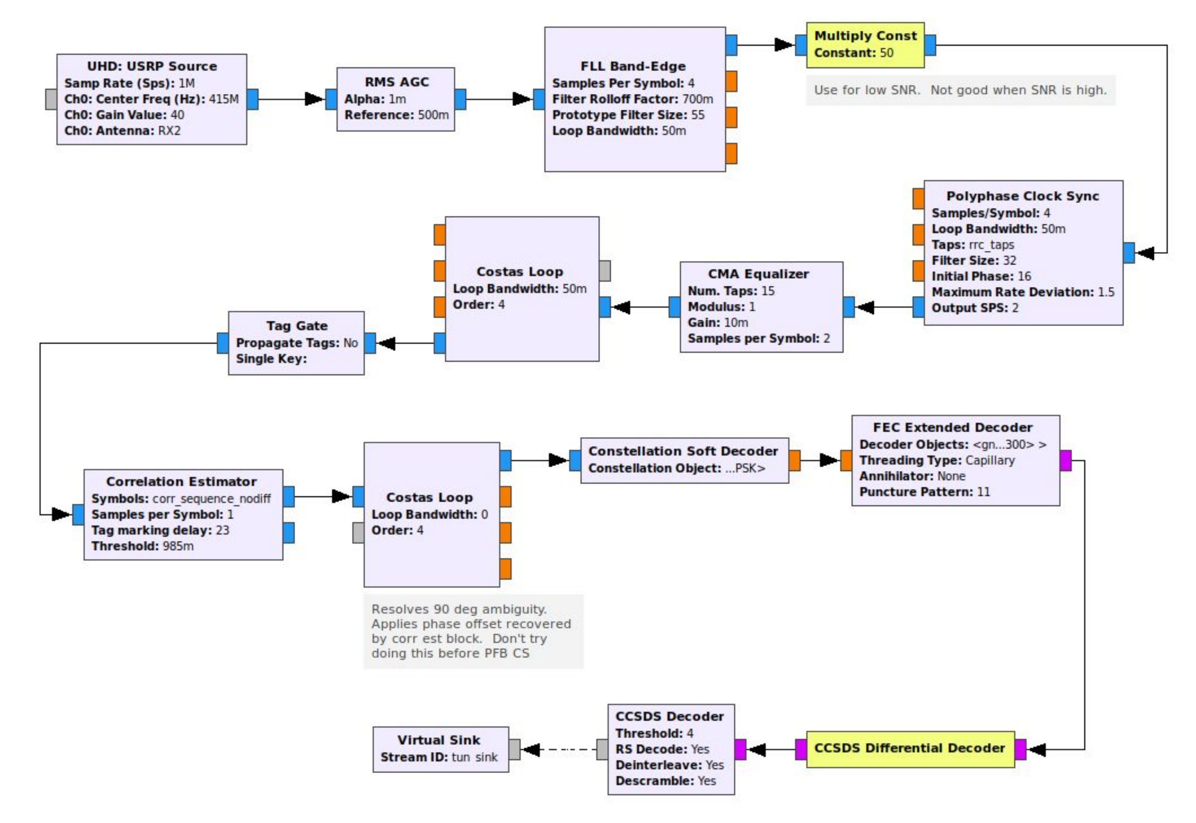
Pack K Bits
K: 8

Constellation Modulator
Constellation: <con...n QPSK>
Differential Encoding: No
Samples/Symbol: 4
Excess BW: 700m

insert_idle_frame
Idle Frame symbols: modula...

UHD: USRP Sink
Stream args: peak=0.7
Device Address: type=b200
Samp Rate (Sps): 1M
Ch0: Center Freq (Hz): 400M
Ch0: Gain Value: 40
Ch0: Antenna: TX/RX
TSB tag name:

Modem: Receiver



UHD: USRP Source
Samp Rate (Sps): 1M
Ch0: Center Freq (Hz): 415M
Ch0: Gain Value: 40
Ch0: Antenna: RX2

RMS AGC
Alpha: 1m
Reference: 500m

FLL Band-Edge
Samples Per Symbol: 4
Filter Rolloff Factor: 700m
Prototype Filter Size: 55
Loop Bandwidth: 50m

Multiply Const
Constant: 50

Use for low SNR. Not good when SNR is high.

Polyphase Clock Sync
Samples/Symbol: 4
Loop Bandwidth: 50m
Taps: rrc_taps
Filter Size: 32
Initial Phase: 16
Maximum Rate Deviation: 1.5
Output SPS: 2

CMA Equalizer
Num. Taps: 15
Modulus: 1
Gain: 10m
Samples per Symbol: 2

Costas Loop
Loop Bandwidth: 50m
Order: 4

Tag Gate
Propagate Tags: No
Single Key:

Correlation Estimator
Symbols: corr_sequence_nodiff
Samples per Symbol: 1
Tag marking delay: 23
Threshold: 985m

Costas Loop
Loop Bandwidth: 0
Order: 4

Resolves 90 deg ambiguity.
Applies phase offset recovered
by corr est block. Don't try
doing this before PFB CS

Constellation Soft Decoder
Constellation Object: ...PSK>

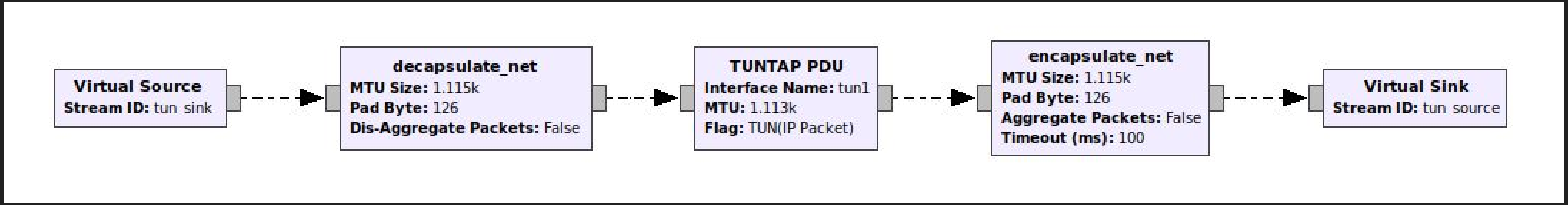
FEC Extended Decoder
Decoder Objects: <gn...300> >
Threading Type: Capillary
Annihilator: None
Puncture Pattern: 11

CCSDS Differential Decoder

CCSDS Decoder
Threshold: 4
RS Decode: Yes
Deinterleave: Yes
Descramble: Yes

Virtual Sink
Stream ID: tun_sink

Modem: IP Encap/Decapsulation Interface



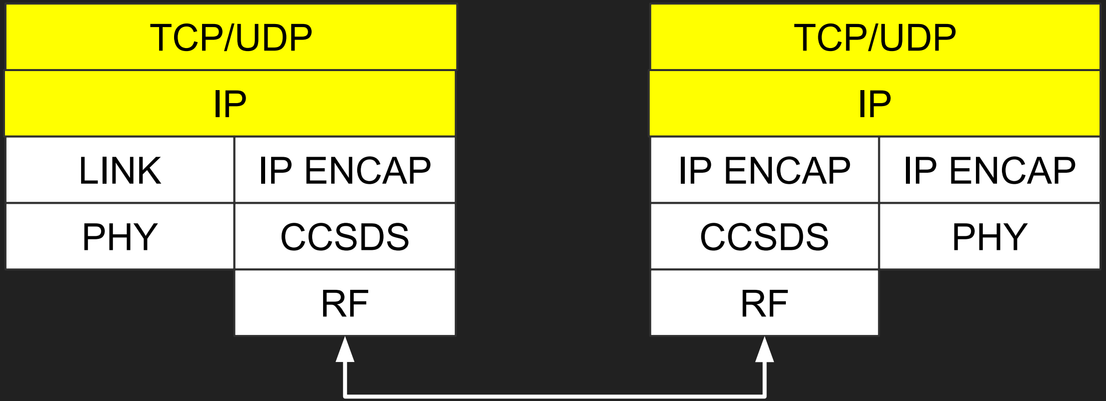
Virtual Source
Stream ID: tun sink

decapsulate_net
MTU Size: 1.115k
Pad Byte: 126
Dis-Aggregate Packets: False

TUNTAP PDU
Interface Name: tun1
MTU: 1.113k
Flag: TUN(IP Packet)

encapsulate_net
MTU Size: 1.115k
Pad Byte: 126
Aggregate Packets: False
Timeout (ms): 100

Virtual Sink
Stream ID: tun source



TCP/UDP
IP
LINK | IP ENCAP
PHY | CCSDS
RF

TCP/UDP
IP
IP ENCAP | IP ENCAP
CCSDS | PHY
RF

A Few Custom Parts

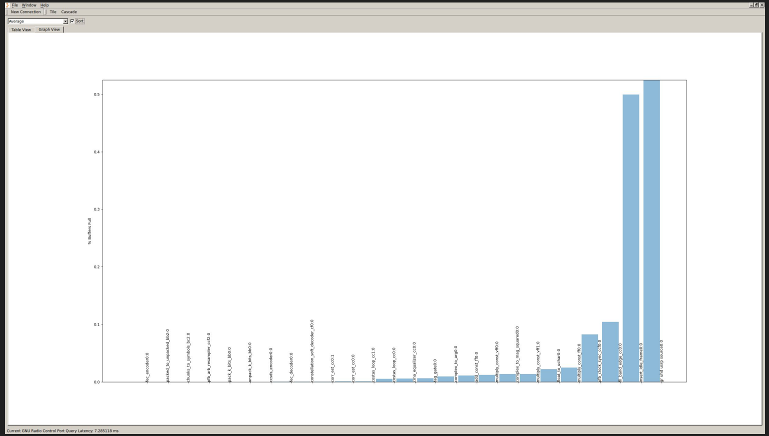
- Tricky problems force me to write some code
  - Phase lock ambiguity
    - Spent a lot of time spinning my wheels on this
    - Eventually settled on a compromise solution that uses spare parts
  - Fill frame insert
    - Stupidly simple
    - Can probably do this with spare parts
  - Post Modulator, Convolutional Encoder Fill frame stitching
    - Not really custom code, but still hard to get right
  - IP Encapsulator / Decapsulator
    - Also stupidly simple

Five Stages of ~~Grief~~ Avoiding Writing Code

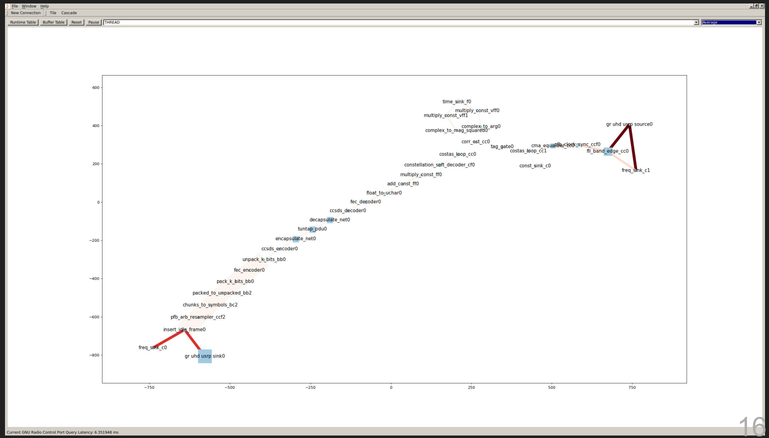
1. Denial
   - Surely this exists...
2. Anger
   - Why the hell doesn’t this exist?
3. Bargaining
   - Can you help me make this exist?
4. Depression
   - I’ll never make this work.
5. Acceptance
   - OK, I’ll write some code.
   - OK, I’ll do the convolutional code by hand (Thanks for your help, Darek)
6. My own addition
   - Can somebody help me write this code?

Latency Reduction

- Add fill frames / symbols at the last possible moment
  - Really... for reals
  - It makes phase synchronization a lot harder (because of the convolutional code)
- But go ahead and try ***everything else*** first
  - GR buffer manipulation (nope)
  - USB buffer buggery (kinda nope)
  - Bargaining (nope nope)
- Smaller gains
  - Tweak convolutional encoder / viterbi decoder work length (meh)
  - Increase the data rate (at least until the Os come)
- Gr-perf-monitorx keeps you honest

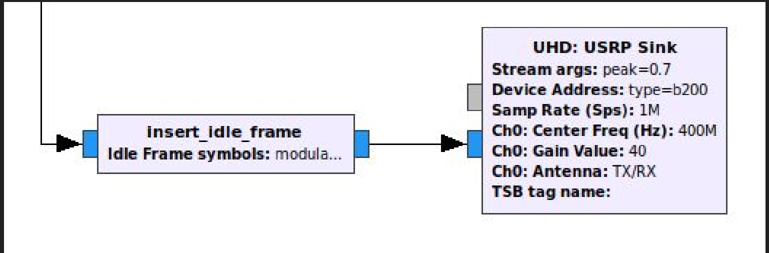


Current GNU Radio Control Port Query Latency: 7.285118 ms



Current GNU Radio Control Port Query Latency: 6.351948 ms

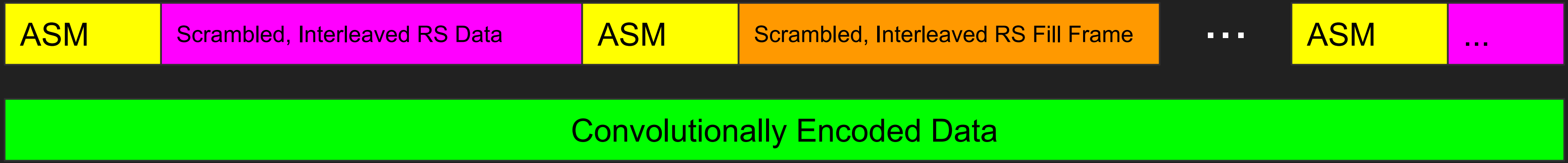
Add Fill Frames Last



insert_idle_frame
Idle Frame symbols: modula...

UHD: USRP Sink
Stream args: peak=0.7
Device Address: type=b200
Samp Rate (Sps): 1M
Ch0: Center Freq (Hz): 400M
Ch0: Gain Value: 40
Ch0: Antenna: TX/RX
TSB tag name:

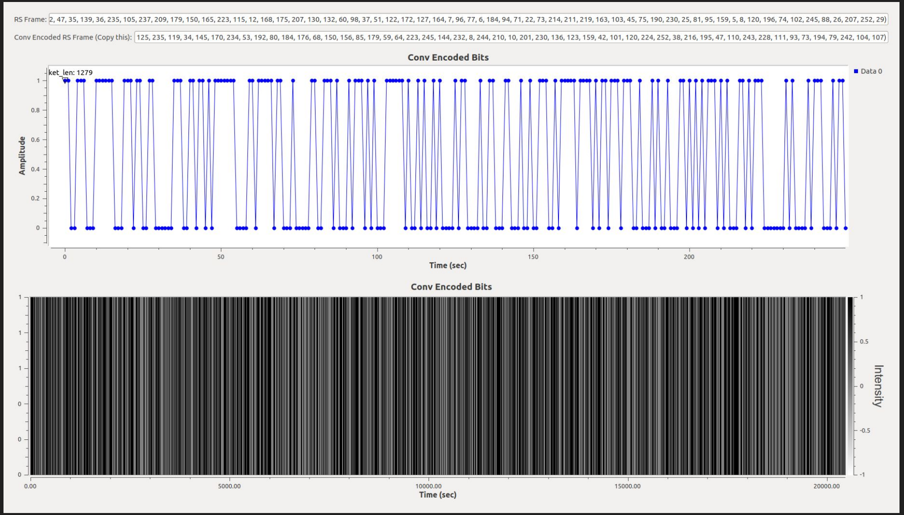
- Fill frames need to include all channel coding (convolutional coding!) and modulation
- If work is called and no items pending, send a fill frame
- Fill frame needs to be match up with previously sent data and data to come after (unpredictable)
    - Convolutional encoder and other blocks have memory, so this is hard!
    - But there’s no better way to defeat latency



ASM | Scrambled, Interleaved RS Data | ASM | Scrambled, Interleaved RS Fill Frame | ... | ASM | ...

Convolutionally Encoded Data

Building the Fill Frame



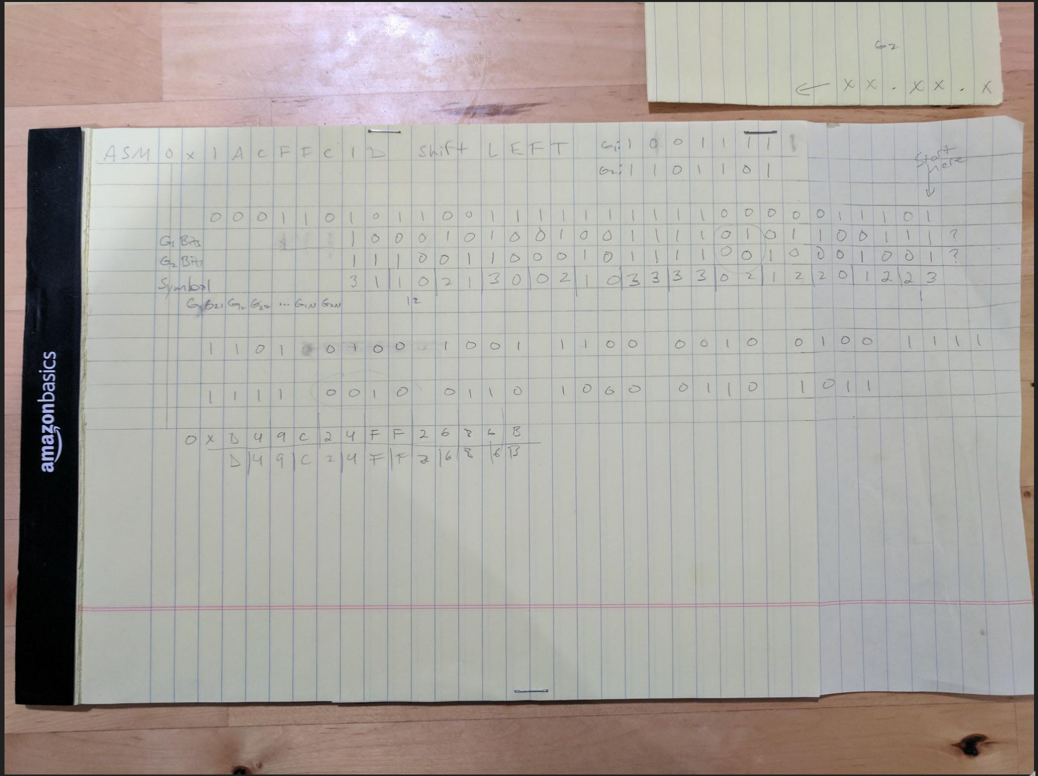
RS Frame: 2, 47, 35, 139, 36, 235, 105, 237, 209, 179, 150, 165, 223, 115, 12, 168, 175, 207, 130, 132, 60, 98, 37, 51, 122, 172, 127, 164, 7, 96, 77, 6, 184, 94, 71, 22, 73, 214, 211, 219, 163, 103, 45, 75, 190, 230, 25, 81, 95, 159, 5, 8, 120, 196, 74, 102, 245, 88, 26, 207, 252, 29)

Conv Encoded RS Frame (Copy this): 125, 235, 119, 34, 145, 170, 234, 53, 192, 80, 184, 176, 68, 150, 156, 85, 179, 59, 64, 223, 245, 144, 232, 8, 244, 210, 10, 201, 230, 136, 123, 159, 42, 101, 120, 224, 252, 38, 216, 195, 47, 110, 243, 228, 111, 93, 73, 194, 79, 242, 104, 107)

- Fill frames are just a data block of 0s
- Several support flowgraphs
  - Idle_frame_encode
    - Encode fill sequence with RS, interleaving, scrambling, convolutional code
  - Sub graph in main flowgraph
    - Use Modulate Vector block to generate a modulated IQ vector
- Zero pad to flush, slice intelligently to combat memory in certain blocks
- [0]*223*5 → ((-0.6960...0.9399...j), … (-0.9150...+0.7737...j))

Building the Fill Frame

- Hand encode ASM
  - After much prodding and help from colleagues
  - Crucial to getting to bit-accurate generation
  - Generator blocks have some memory, so need to trim intelligently



G2

← X X . X X . X

ASM 0 x 1 A C F F C 1 D Shift LEFT

G1: 1 0 0 1 1 1 1

G2: 1 1 0 1 1 0 1

Start here

0 0 0 1 1 0 1 0 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 1 1 1 0 1

G1 Bits 1 0 0 0 1 0 1 0 0 1 0 0 1 1 1 1 0 1 0 1 1 0 0 1 1 1 ?

G2 Bits 1 1 1 0 0 1 1 0 0 0 1 0 1 1 1 1 0 0 1 0 0 0 1 0 0 1 ?

Symbol 3 1 1 0 2 1 3 0 0 2 1 0 3 3 3 3 0 2 1 2 2 0 1 2 2 3

G11 G21 G12 G22 ... G1N G2N 12

1 1 0 1 0 1 0 0 1 0 0 1 1 1 0 0 0 0 1 0 0 1 0 0 1 1 1 1

1 1 1 1 0 0 1 0 0 1 1 0 1 0 6 0 0 1 1 0 1 0 1 1

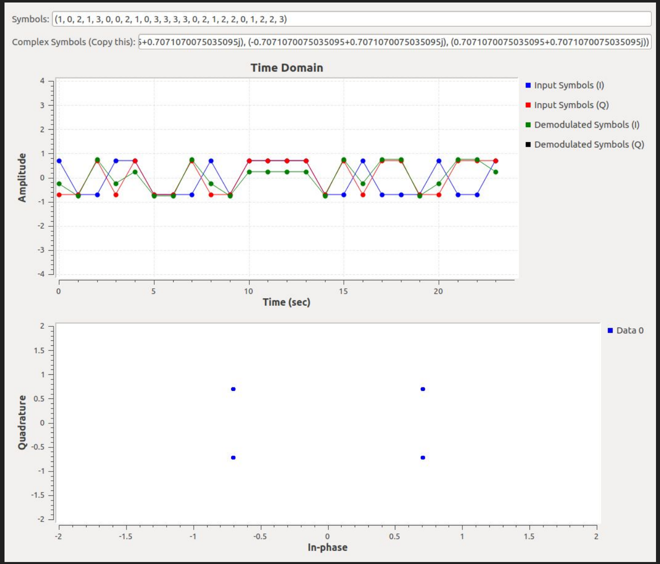
0 x D 4 9 C 2 4 F F 2 6 8 6 B

D 4 9 C 2 4 F F 2 6 8 6 B

Building the Correlation Sequence

- Several support flowgraphs
  - Asm_test
    - Encode ASM with convolutional code
  - Asm_post_costas
    - Map encoded ASM to complex symbols
- Zero pad to flush, slice intelligently to combat memory in certain blocks
- 0x1ACFFC1D →

  ((0.7071...-0.7071...j), … (0.7071...+0.7071...j))

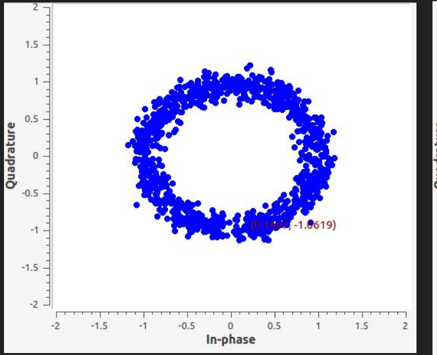


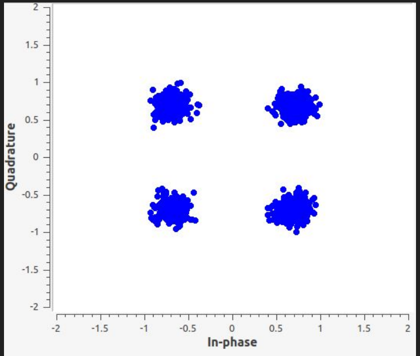
Symbols: (1, 0, 2, 1, 3, 0, 0, 2, 1, 0, 3, 3, 3, 3, 0, 2, 1, 2, 2, 0, 1, 2, 2, 3)

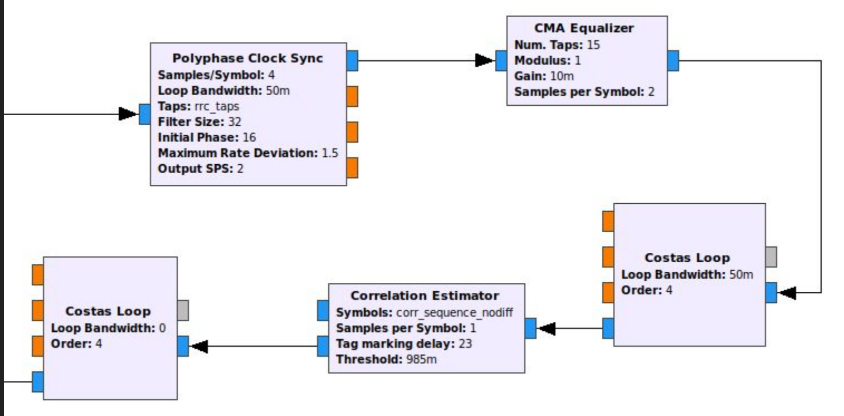
Complex Symbols (Copy this): 5+0.7071070075035095j), (-0.7071070075035095+0.7071070075035095j), (0.7071070075035095+0.7071070075035095j))

Phase Synchronization

- Harder than BPSK
- Costas loop will lock, but perhaps not to the right rotation, order
    - Simulations can fool you (unless you simulate real world effects)
    - Over the air, you might be right 50% or 25% of the time, depending on coding
- Correlation estimator and 2nd Costas Loop clean up the ambiguity
- There are better ways to do this







Polyphase Clock Sync
Samples/Symbol: 4
Loop Bandwidth: 50m
Taps: rrc_taps
Filter Size: 32
Initial Phase: 16
Maximum Rate Deviation: 1.5
Output SPS: 2

CMA Equalizer
Num. Taps: 15
Modulus: 1
Gain: 10m
Samples per Symbol: 2

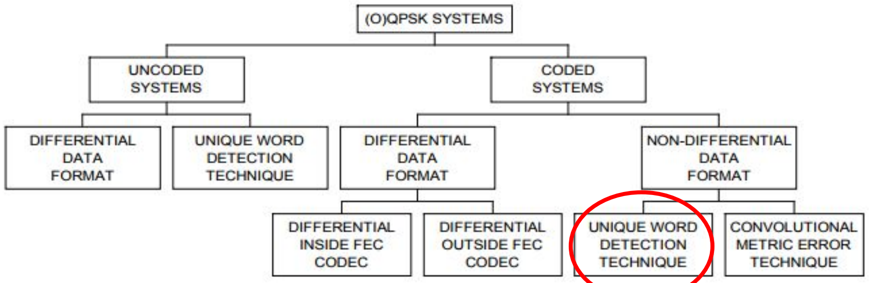
Costas Loop
Loop Bandwidth: 50m
Order: 4

Correlation Estimator
Symbols: corr_sequence_nodiff
Samples per Symbol: 1
Tag marking delay: 23
Threshold: 985m

Costas Loop
Loop Bandwidth: 0
Order: 4

Phase Synchronization

- There are better ways to do this



(O)QPSK SYSTEMS

UNCODED SYSTEMS

CODED SYSTEMS

DIFFERENTIAL DATA FORMAT

UNIQUE WORD DETECTION TECHNIQUE

DIFFERENTIAL DATA FORMAT

NON-DIFFERENTIAL DATA FORMAT

DIFFERENTIAL INSIDE FEC CODEC

DIFFERENTIAL OUTSIDE FEC CODEC

UNIQUE WORD DETECTION TECHNIQUE

CONVOLUTIONAL METRIC ERROR TECHNIQUE

Figure 2.4.11-1: List of Phase-Ambiguity Resolution Techniques

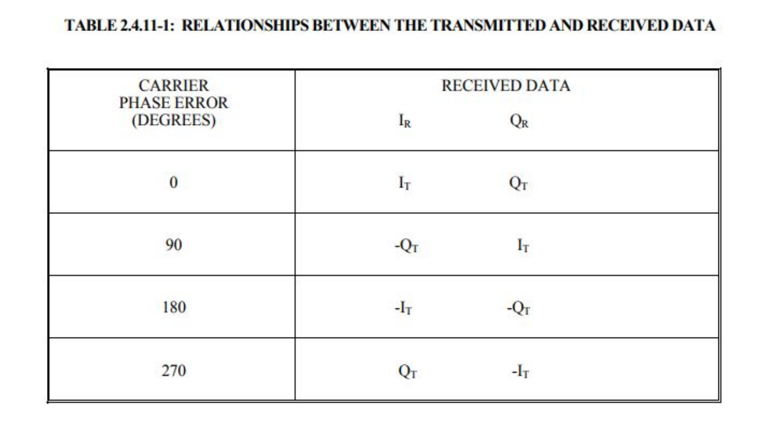


TABLE 2.4.11-1: RELATIONSHIPS BETWEEN THE TRANSMITTED AND RECEIVED DATA

CARRIER PHASE ERROR (DEGREES)	RECEIVED DATA	
	I_R	Q_R
0	I_T	Q_T
90	$-Q_T$	I_T
180	$-I_T$	$-Q_T$
270	Q_T	$-I_T$

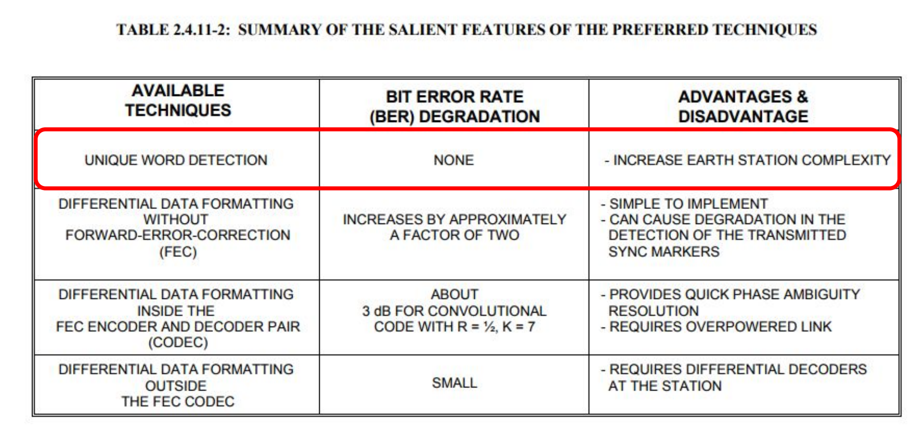
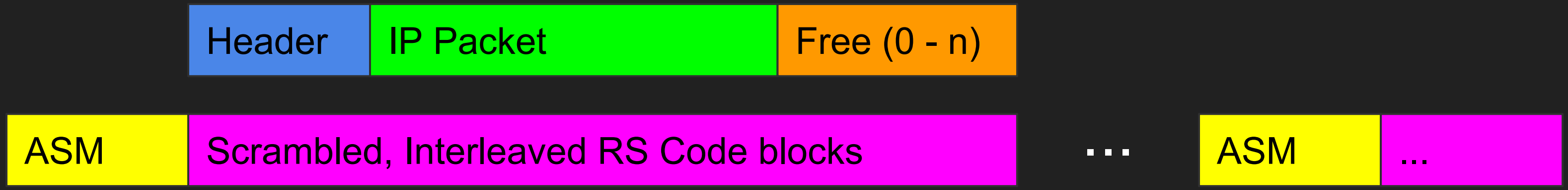


TABLE 2.4.11-2: SUMMARY OF THE SALIENT FEATURES OF THE PREFERRED TECHNIQUES

AVAILABLE TECHNIQUES	BIT ERROR RATE (BER) DEGRADATION	ADVANTAGES & DISADVANTAGE
UNIQUE WORD DETECTION	NONE	- INCREASE EARTH STATION COMPLEXITY
DIFFERENTIAL DATA FORMATTING WITHOUT FORWARD-ERROR-CORRECTION (FEC)	INCREASES BY APPROXIMATELY A FACTOR OF TWO	- SIMPLE TO IMPLEMENT - CAN CAUSE DEGRADATION IN THE DETECTION OF THE TRANSMITTED SYNC MARKERS
DIFFERENTIAL DATA FORMATTING INSIDE THE FEC ENCODER AND DECODER PAIR (CODEC)	ABOUT 3 dB FOR CONVOLUTIONAL CODE WITH R = ½, K = 7	- PROVIDES QUICK PHASE AMBIGUITY RESOLUTION - REQUIRES OVERPOWERED LINK
DIFFERENTIAL DATA FORMATTING OUTSIDE THE FEC CODEC	SMALL	- REQUIRES DIFFERENTIAL DECODERS AT THE STATION

IP Encapsulation / Decapsulation

- Toy implementation
  - Dead simple Python blocks
  - 1 IP / UDP packet per CCSDS frame
  - Just add a 2 byte header to define length of encapsulated packet
  - Enforce MTU size limit (2 header bytes + 1113 data bytes)
- Better solutions
  - HDLC encode to delimit packets
- Worse solutions
  - Actually follow CCSDS, don't do that

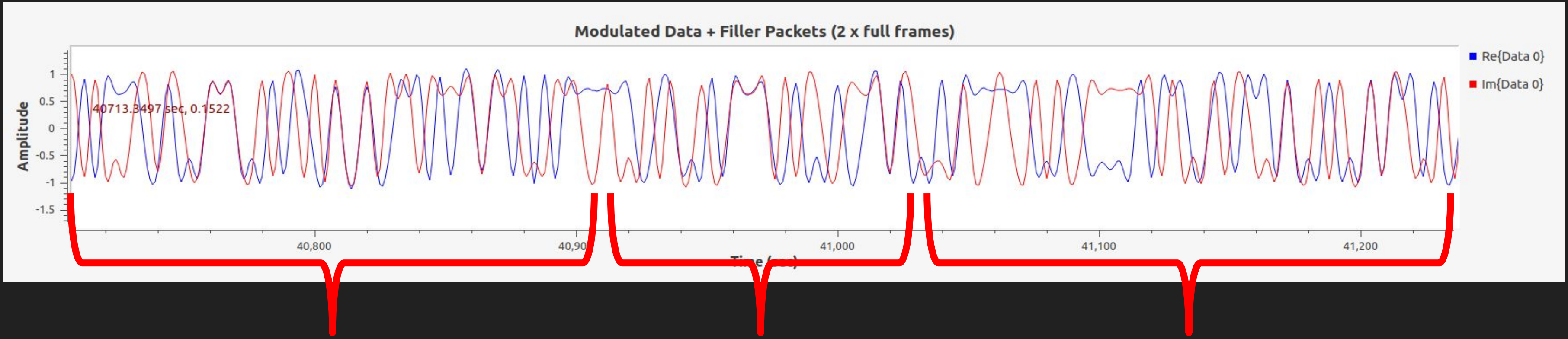


Header | IP Packet | Free (0 - n)

ASM | Scrambled, Interleaved RS Code blocks | ... | ASM | ...

Live Demo: Loopback Simulation

- Note unchanging fully encoded ASM symbols at start of frames
    - Accurate sticking of fully pre-encoded fill frames allows > 100x latency reduction
    - This was the principal challenge and success of this work



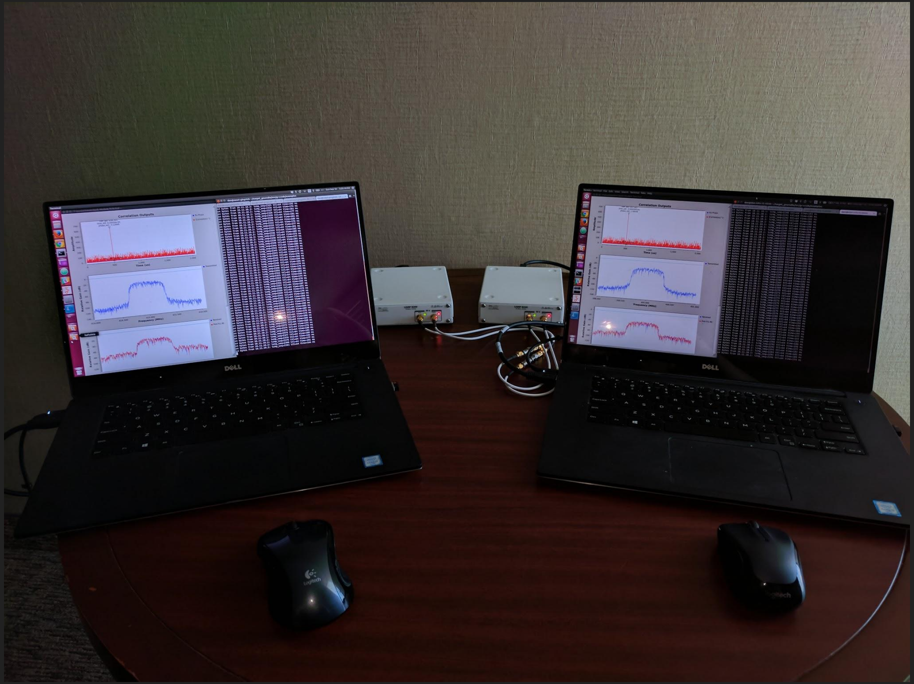
Previous frame
(changing data or
fill frame)

Encoded ASM
(static)

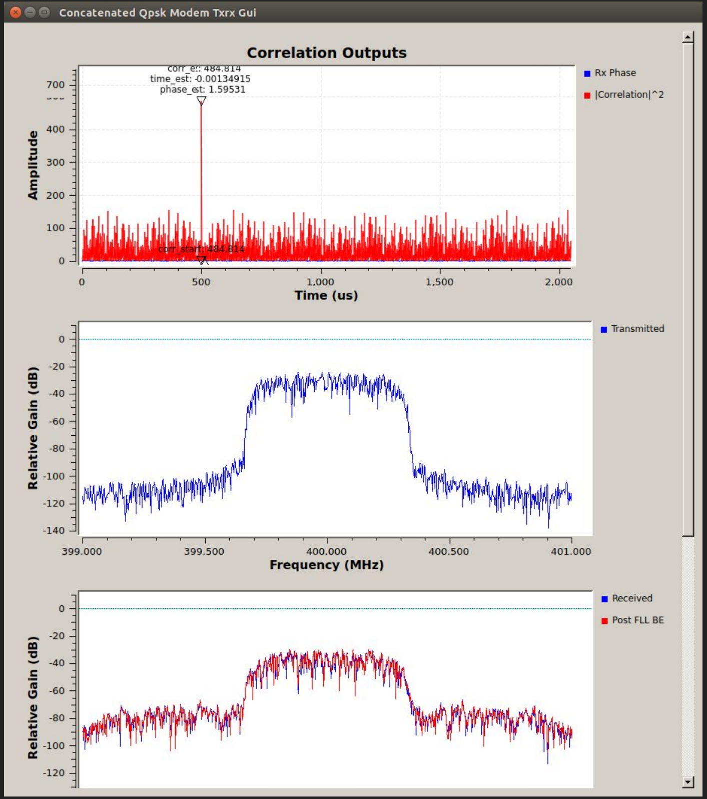
Next frame
(changing data or
fill frame)

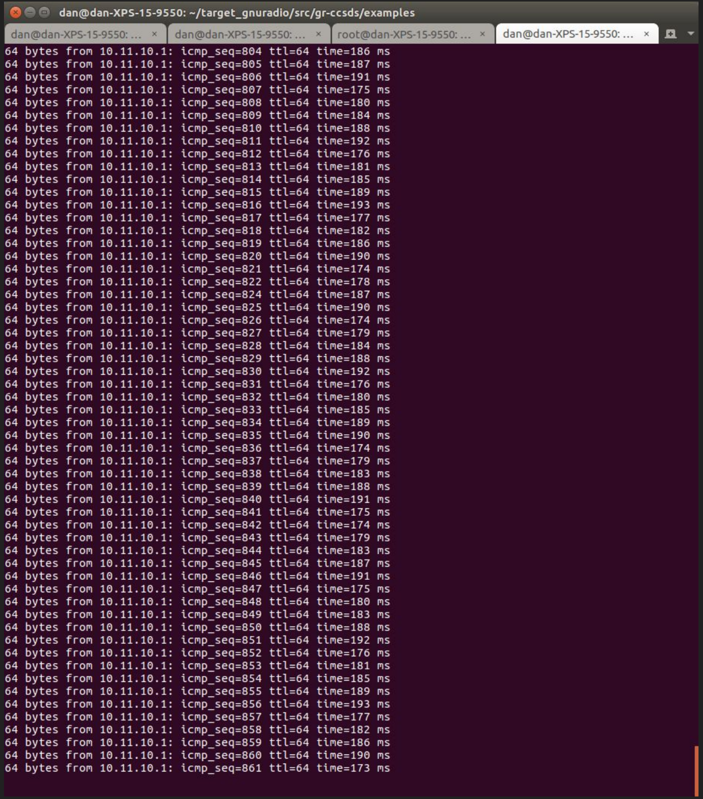
Live Demo: OTA

- Connect
- Ping
  - Highlight low latency
- Disconnect channel and demonstrate self-healing
- SSH / Mosh
  - Hollywood time!
- File Transfer?
- Demo PEP acceleration on a simulated high latency, high BER channel



Live Demo




dan@dan-XPS-15-9550: ~/target_gnuradio/src/gr-ccsds/examples
dan@dan-XPS-15-9550: ...
dan@dan-XPS-15-9550: ...
root@dan-XPS-15-9550: ...
dan@dan-XPS-15-9550: ...

```
64 bytes from 10.11.10.1: icmp_seq=804 ttl=64 time=186 ms
64 bytes from 10.11.10.1: icmp_seq=805 ttl=64 time=187 ms
64 bytes from 10.11.10.1: icmp_seq=806 ttl=64 time=191 ms
64 bytes from 10.11.10.1: icmp_seq=807 ttl=64 time=175 ms
64 bytes from 10.11.10.1: icmp_seq=808 ttl=64 time=180 ms
64 bytes from 10.11.10.1: icmp_seq=809 ttl=64 time=184 ms
64 bytes from 10.11.10.1: icmp_seq=810 ttl=64 time=188 ms
64 bytes from 10.11.10.1: icmp_seq=811 ttl=64 time=192 ms
64 bytes from 10.11.10.1: icmp_seq=812 ttl=64 time=176 ms
64 bytes from 10.11.10.1: icmp_seq=813 ttl=64 time=181 ms
64 bytes from 10.11.10.1: icmp_seq=814 ttl=64 time=185 ms
64 bytes from 10.11.10.1: icmp_seq=815 ttl=64 time=189 ms
64 bytes from 10.11.10.1: icmp_seq=816 ttl=64 time=193 ms
64 bytes from 10.11.10.1: icmp_seq=817 ttl=64 time=177 ms
64 bytes from 10.11.10.1: icmp_seq=818 ttl=64 time=182 ms
64 bytes from 10.11.10.1: icmp_seq=819 ttl=64 time=186 ms
64 bytes from 10.11.10.1: icmp_seq=820 ttl=64 time=190 ms
64 bytes from 10.11.10.1: icmp_seq=821 ttl=64 time=174 ms
64 bytes from 10.11.10.1: icmp_seq=822 ttl=64 time=178 ms
64 bytes from 10.11.10.1: icmp_seq=824 ttl=64 time=187 ms
64 bytes from 10.11.10.1: icmp_seq=825 ttl=64 time=190 ms
64 bytes from 10.11.10.1: icmp_seq=826 ttl=64 time=174 ms
64 bytes from 10.11.10.1: icmp_seq=827 ttl=64 time=179 ms
64 bytes from 10.11.10.1: icmp_seq=828 ttl=64 time=184 ms
64 bytes from 10.11.10.1: icmp_seq=829 ttl=64 time=188 ms
64 bytes from 10.11.10.1: icmp_seq=830 ttl=64 time=192 ms
64 bytes from 10.11.10.1: icmp_seq=831 ttl=64 time=176 ms
64 bytes from 10.11.10.1: icmp_seq=832 ttl=64 time=180 ms
64 bytes from 10.11.10.1: icmp_seq=833 ttl=64 time=185 ms
64 bytes from 10.11.10.1: icmp_seq=834 ttl=64 time=189 ms
64 bytes from 10.11.10.1: icmp_seq=835 ttl=64 time=190 ms
64 bytes from 10.11.10.1: icmp_seq=836 ttl=64 time=174 ms
64 bytes from 10.11.10.1: icmp_seq=837 ttl=64 time=179 ms
64 bytes from 10.11.10.1: icmp_seq=838 ttl=64 time=183 ms
64 bytes from 10.11.10.1: icmp_seq=839 ttl=64 time=188 ms
64 bytes from 10.11.10.1: icmp_seq=840 ttl=64 time=191 ms
64 bytes from 10.11.10.1: icmp_seq=841 ttl=64 time=175 ms
64 bytes from 10.11.10.1: icmp_seq=842 ttl=64 time=174 ms
64 bytes from 10.11.10.1: icmp_seq=843 ttl=64 time=179 ms
64 bytes from 10.11.10.1: icmp_seq=844 ttl=64 time=183 ms
64 bytes from 10.11.10.1: icmp_seq=845 ttl=64 time=187 ms
64 bytes from 10.11.10.1: icmp_seq=846 ttl=64 time=191 ms
64 bytes from 10.11.10.1: icmp_seq=847 ttl=64 time=175 ms
64 bytes from 10.11.10.1: icmp_seq=848 ttl=64 time=180 ms
64 bytes from 10.11.10.1: icmp_seq=849 ttl=64 time=183 ms
64 bytes from 10.11.10.1: icmp_seq=850 ttl=64 time=188 ms
64 bytes from 10.11.10.1: icmp_seq=851 ttl=64 time=192 ms
64 bytes from 10.11.10.1: icmp_seq=852 ttl=64 time=176 ms
64 bytes from 10.11.10.1: icmp_seq=853 ttl=64 time=181 ms
64 bytes from 10.11.10.1: icmp_seq=854 ttl=64 time=185 ms
64 bytes from 10.11.10.1: icmp_seq=855 ttl=64 time=189 ms
64 bytes from 10.11.10.1: icmp_seq=856 ttl=64 time=193 ms
64 bytes from 10.11.10.1: icmp_seq=857 ttl=64 time=177 ms
64 bytes from 10.11.10.1: icmp_seq=858 ttl=64 time=182 ms
64 bytes from 10.11.10.1: icmp_seq=859 ttl=64 time=186 ms
64 bytes from 10.11.10.1: icmp_seq=860 ttl=64 time=190 ms
64 bytes from 10.11.10.1: icmp_seq=861 ttl=64 time=173 ms
```

Future Work

- Replace toy IP encapsulation protocol with HDLC / MPoFR
- Figure out how to synchronize OQPSK
  - Technically, CCSDS recommends OQPSK, not QPSK
- Replace hacky phase synchronizer with more robust and more efficient implementation
  - Maybe try gr-mapper?
- Fix bug with long-running file transfers
  - ASM seen in data packet, confusing PLL?
- Script the correlation and idle frame generation
- Measure BER vs. E_S/N_0
- Bonus: Get all the OOT stuff in tree
  - RS → FEC API, etc.
- Get an intern to do it all for me

Acknowledgements

- Darek Kawamoto
- Nick McCarthy
- Andy Walls & the IRC gang
- André Løfaldli
  - gr-ccsds
- Daniel Estevez
  - Read his blog: http://destevez.net/
- HawkEye 360

GNURadio
THE FREE & OPEN SOFTWARE RADIO ECOSYSTEM

Ettus Research™
A National Instruments Company

HawkEye 360

Questions?

- HE360 is hiring!

HawkEye360

“We'd now like to open the floor to shorter speeches disguised as questions.”