GNU Radio in the Undergraduate Communications Curriculum

Peter Mathys
Department of ECEE
University of Colorado, Boulder

Software Defined Radio (SDR)

- The SDR evolution occurred roughly over the last decade.
- Wide variety of hardware available from DVB-T tuners (Realtek RTL2832U, $10) to the HackRF One (Great Scott Gadgets, $299) and the USRP E310 (Ettus Research, $2700).
- Typical RF frequency range is from a few 10 MHz to single digit GHz.
- Some software choices: GNU Radio, Matlab/Simulink

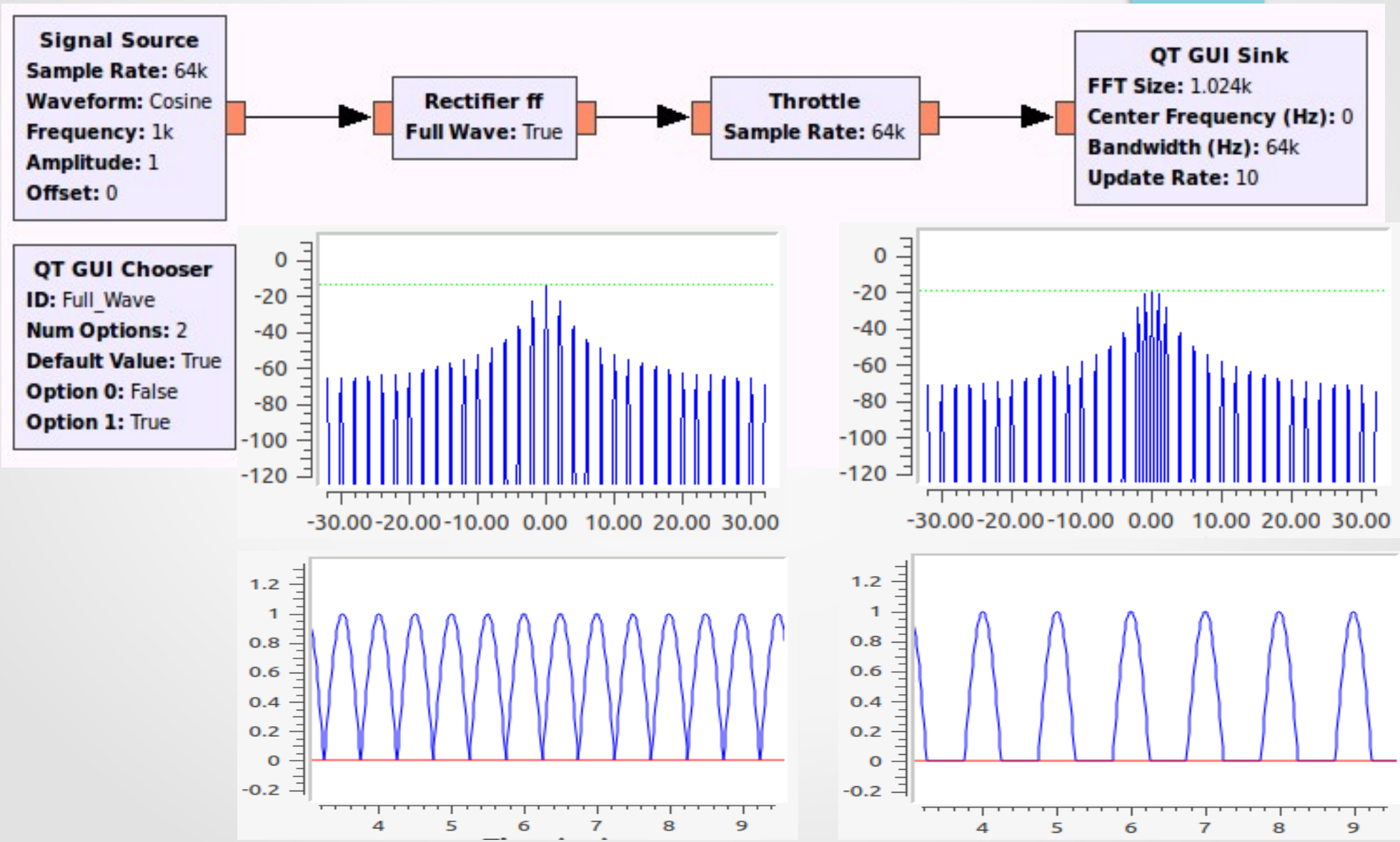
Software Defined Radio (SDR)

- Have SDRs changed the fundamental principles of communication theory (Shannon, 1948, “A Mathematical Theory of Communication”)?
- No! But it has completely changed the ways we go about experimenting with, implementing of, visualizing of, and understanding of new and more precise ways to create and receive sophisticated information carrying waveforms.

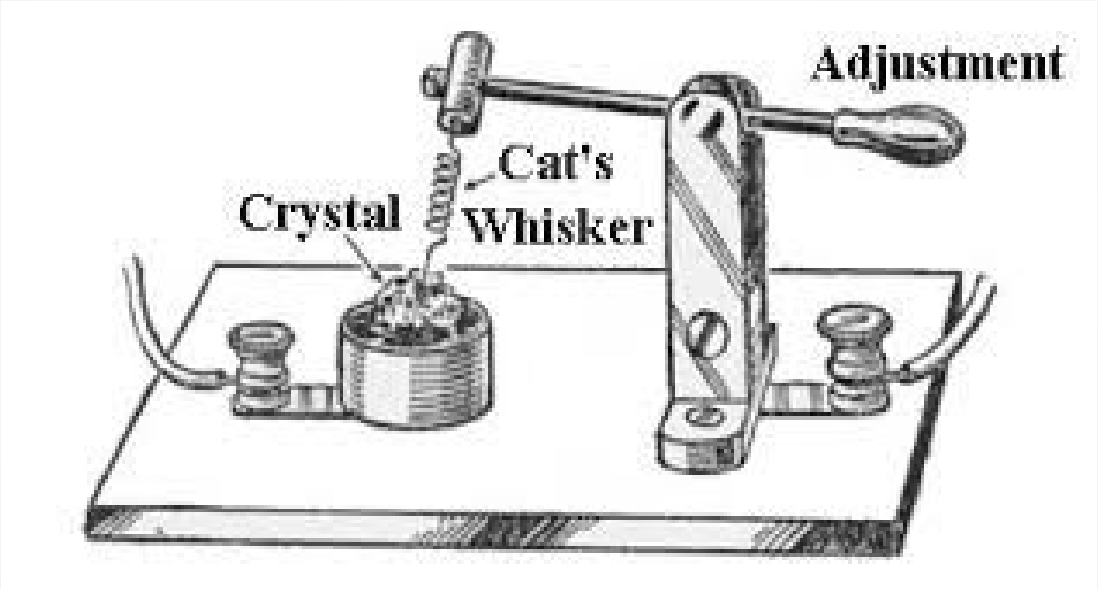
GNU Radio in the Classroom

- Expand GNU Radio user base in size and diversity.
- GNU Radio is affordable.
- GNU Radio is sophisticated.
- GNU Radio is open source and expandable.
- GNU Radio is a great tool to visualize the effects of signal processing blocks in real-time and ask what-if questions.

Example: Full/Half Wave Rectifier



Where is the Rectifier ff block in GNU Radio?



Rectifier ff is an OOT Module

- Out-of-tree modules are used to implement your own functions alongside the main GNU Radio code.
- gr_modtool is used to set up the framework.
- The main components of a typical OOT module are:
- The Python (or C++) QA test file: qa_rectifier_ff.py.
- The public header file: rectifier_ff.h
- The implementation header file: rectifier_ff_impl.h
- The implementation source file: rectifier_ff_impl.cc
- The xml block definition for GRC: test01_rectifier_ff.xml
- There is lots more behind the scenes!

From lib/rectifier_ff_impl.cc

```
int
rectifier_ff_impl::work(int noutput_items,
        gr_vector_const_void_star &input_items,
        gr_vector_void_star &output_items)
{
    const float *in = (const float *) input_items[0];
    float *out = (float *) output_items[0];

    // Do <+signal processing+>
    for (int i = 0; i < noutput_items; i++)
      if (in[i] >= 0.0)
        out[i] = in[i];
      else if (d_full_wave)
        out[i] = -in[i];
      else
        out[i] =0.0;

    // Tell runtime system how many output items we produced.
    return noutput_items;
}
```

From python/qa_rectifier_ff.py

```
def test_002_rectifier_ff_half (self):   # half wave rectifier test
    src_data = (0.0,  0.3827,  0.7071,  0.9239,  1.0,  0.9239,  0.7071,  0.3827,
                0.0, -0.3827, -0.7071, -0.9239, -1.0, -0.9239, -0.7071, -0.3827,
                0.0,  0.3827,  0.7071,  0.9239,  1.0,  0.9239,  0.7071,  0.3827,
                0.0, -0.3827, -0.7071, -0.9239, -1.0, -0.9239, -0.7071, -0.3827)
    exp_data = (0.0,  0.3827,  0.7071,  0.9239,  1.0,  0.9239,  0.7071,  0.3827,
                0.0,  0.0,     0.0,     0.0,     0.0,  0.0,     0.0,     0.0,
                0.0,  0.3827,  0.7071,  0.9239,  1.0,  0.9239,  0.7071,  0.3827,
                0.0,  0.0,     0.0,     0.0,     0.0,  0.0,     0.0,     0.0)
    src = blocks.vector_source_f(src_data)
    op = test01.rectifier_ff(False)
    dst = blocks.vector_sink_f()
    self.tb.connect(src, op, dst)
    self.tb.run ()
    result_data = dst.data()
    self.assertFloatTuplesAlmostEqual(exp_data, result_data, 5)
```

Why Write Your Own OOT Modules?

- GNU Radio is work in progress. Some functions may just simply not yet be available. Or you you may have come up with a great new idea that you would like to test. Or you have some specific needs, e.g., to educate undergraduate students.
- It's a great (but not painless) way to learn about GNU Radio.
- Where can I learn how to do it? That's a bit of a sticky point. There are some tutorials at https://gnuradio.org and there is the discuss-gnuradio mailing list, but expect to do a lot of trial and error.

The Undergraduate Curriculum in Communications at CU Boulder

- Communications track is two semester sequence for seniors:
- ECEN 4242, Communication Theory, taught in fall
- ECEN 4652, Communications Laboratory, taught in spring
- Prerequisites for ECEN 4242 are Linear Systems and Probability theory.
- Prerequisite for ECEN 4652 is ECEN 4242.

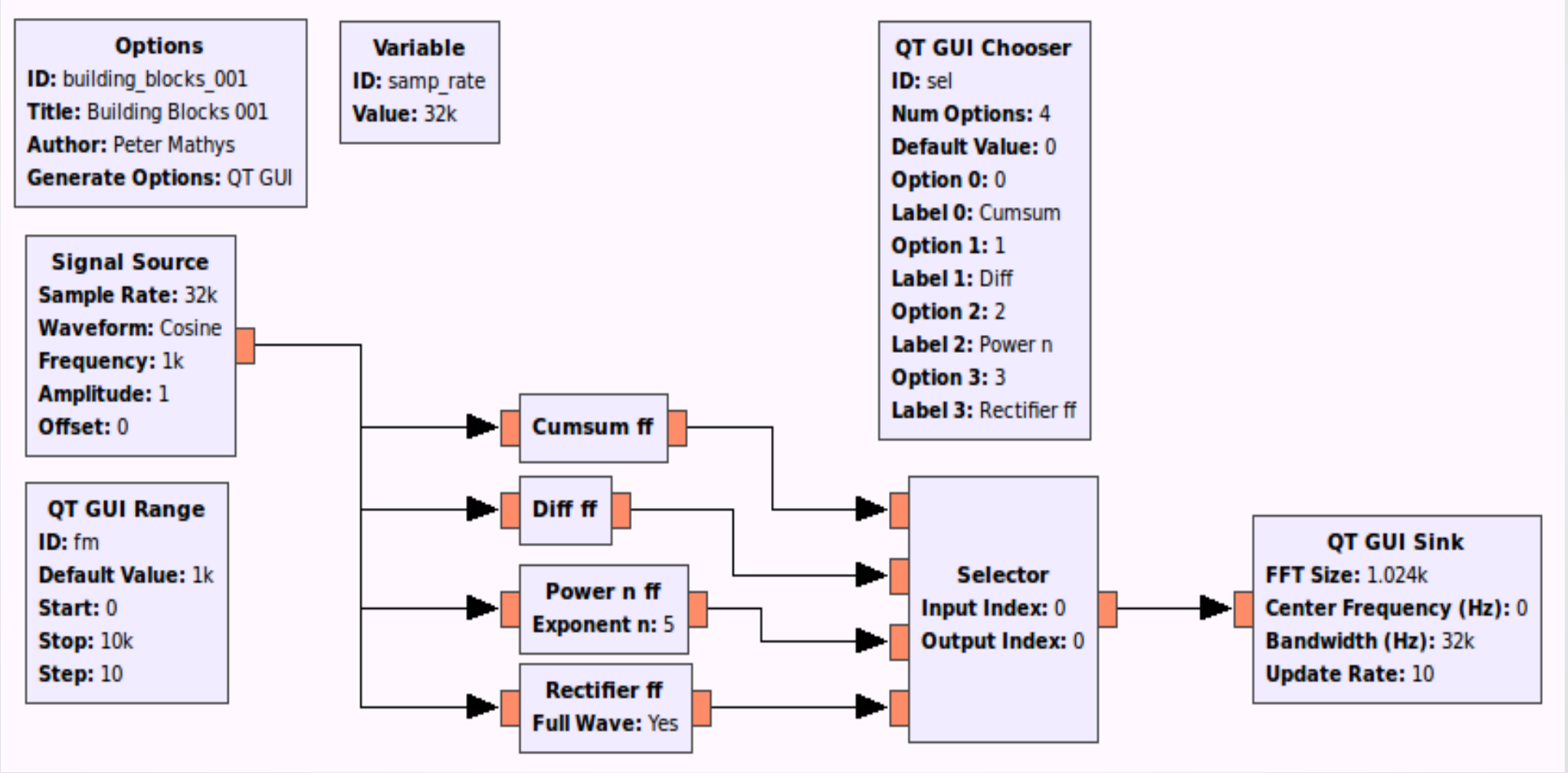
ECEN 4242 Communication Theory

1. Introduction
2. Linear Systems (review)
3. Amplitude Modulation
4. Angle Modulation
5. Probability Theory (review), Random Processes (introduction)
6. Noise in CM Modulation Systems
7. Transition from Analog to Digital (PAM, TDM, PPM, PCM)
8. Digital Baseband Communications (noise, ISI, matched filter, probability of error)
9. Digital Bandpass Communications (PSK, FSK, signal constellations)

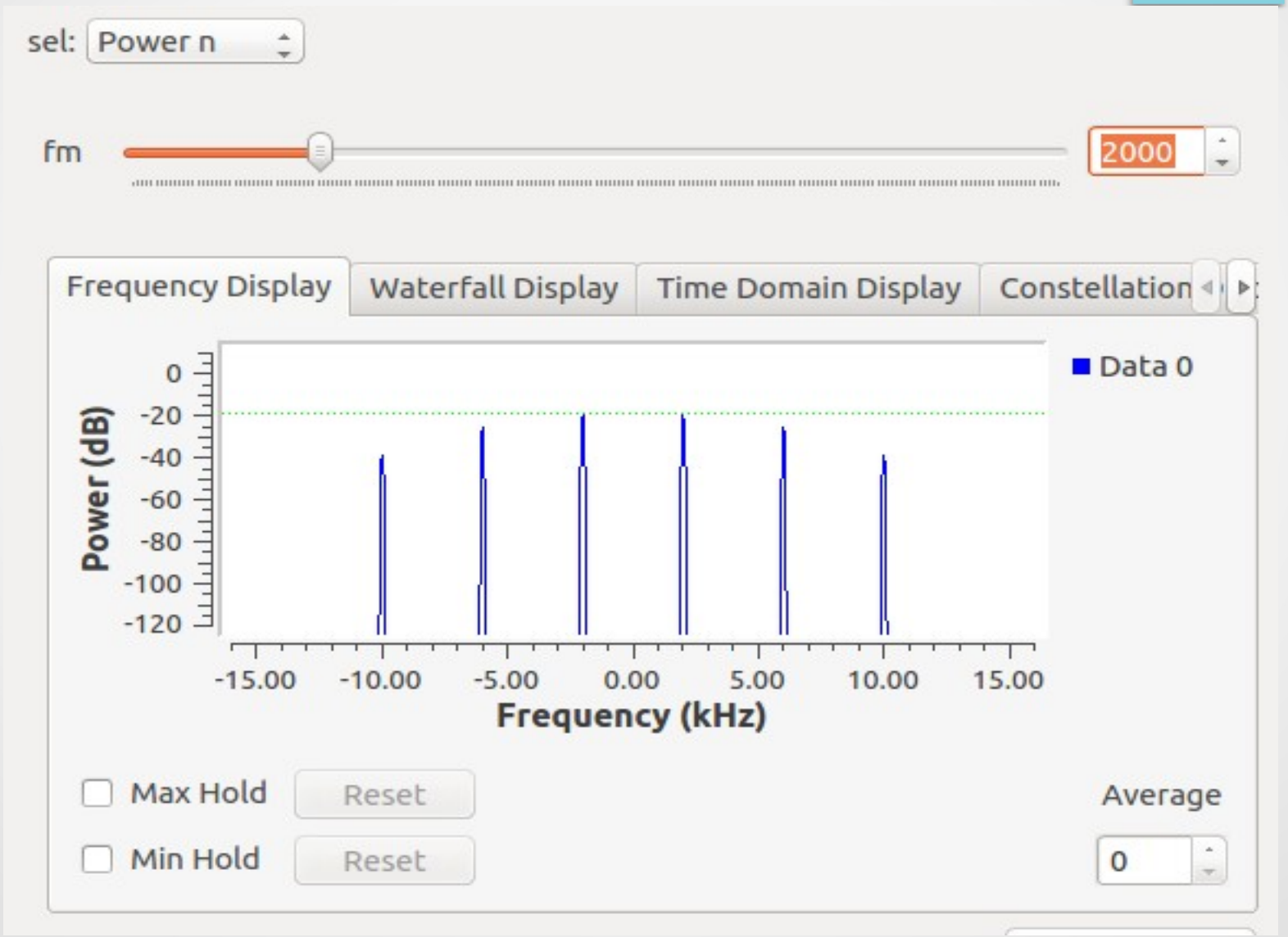
ECEN 4242 and GNU Radio

- The main goal of GNU Radio is to control SDRs, not to teach linear systems or to implement baseband communications.
- But to learn communication theory one needs to start with small building blocks and piece them together into larger systems while retaining the ability to probe the output of each of the building blocks.

Building Block Examples

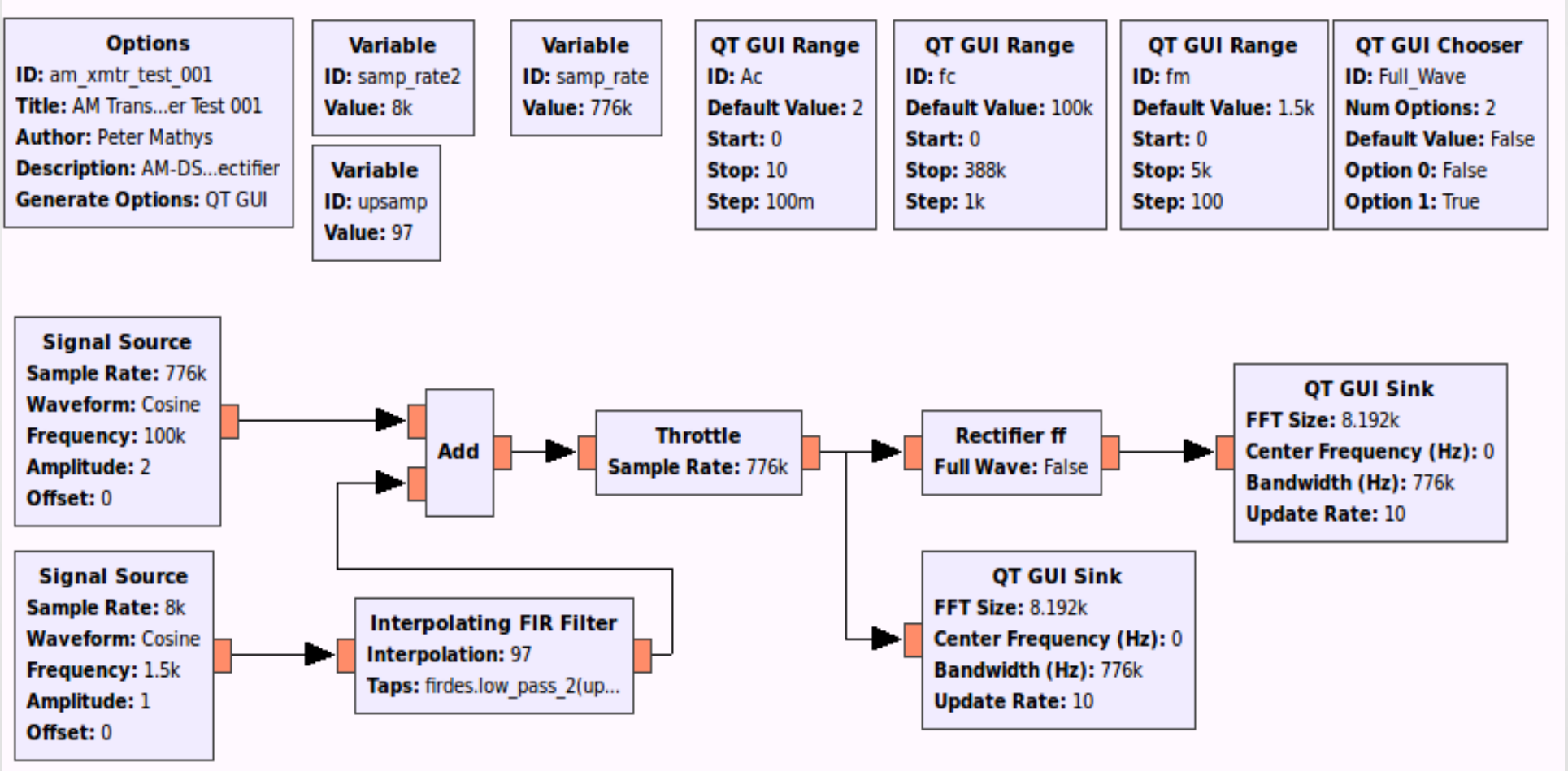


Building Block Examples

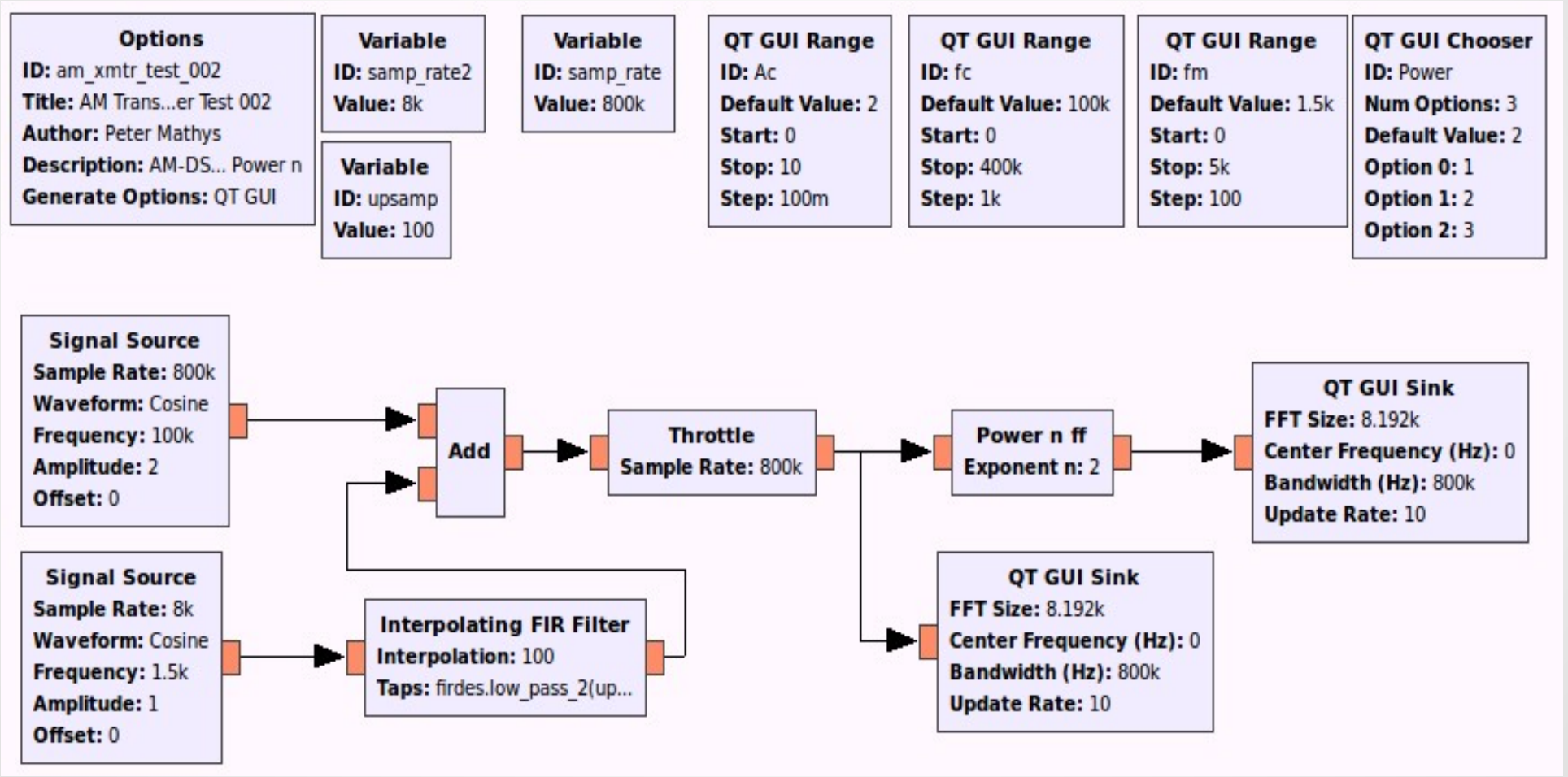


Amplitude Modulation

Amplitude Modulation

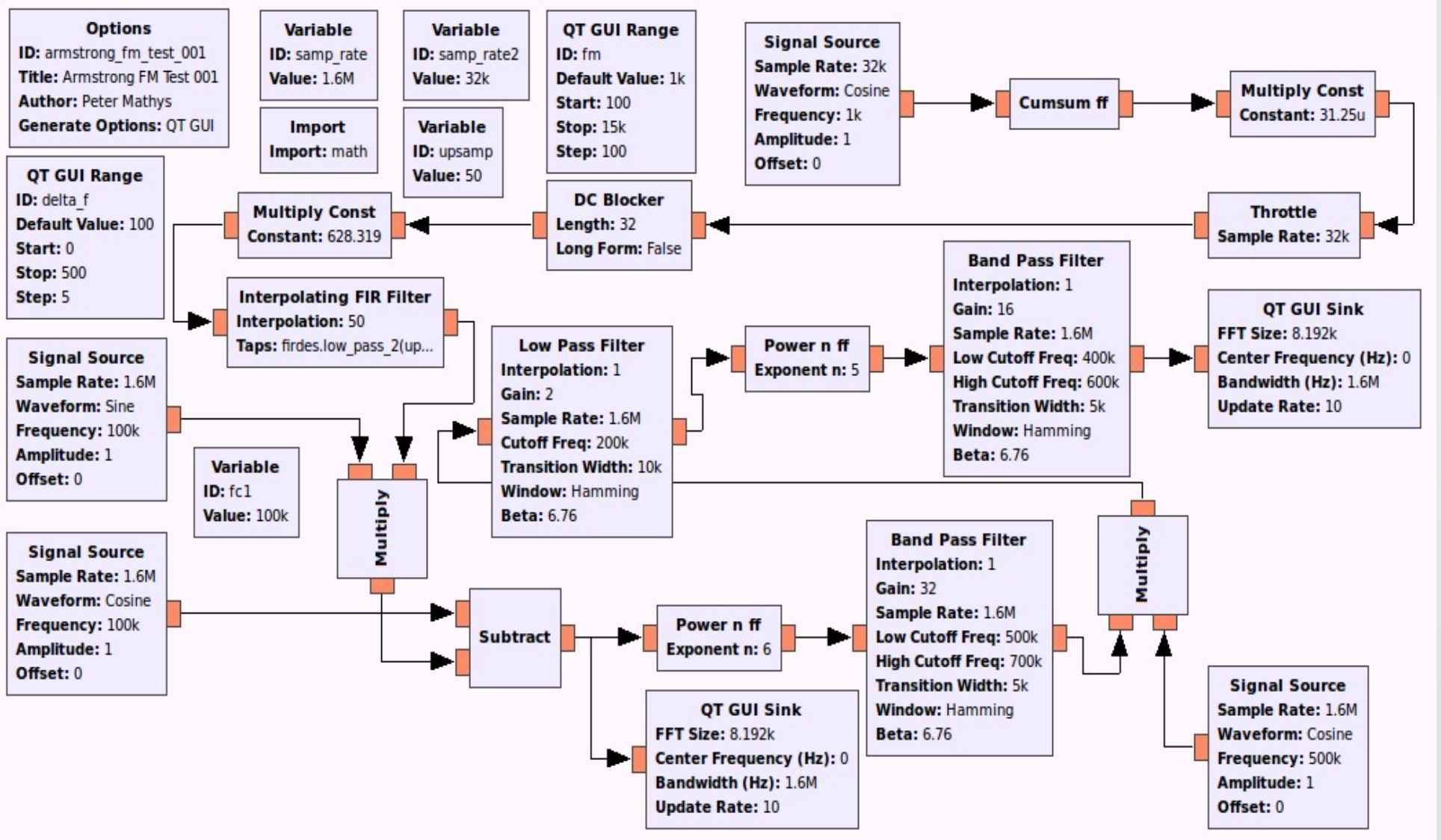


Amplitude Modulation

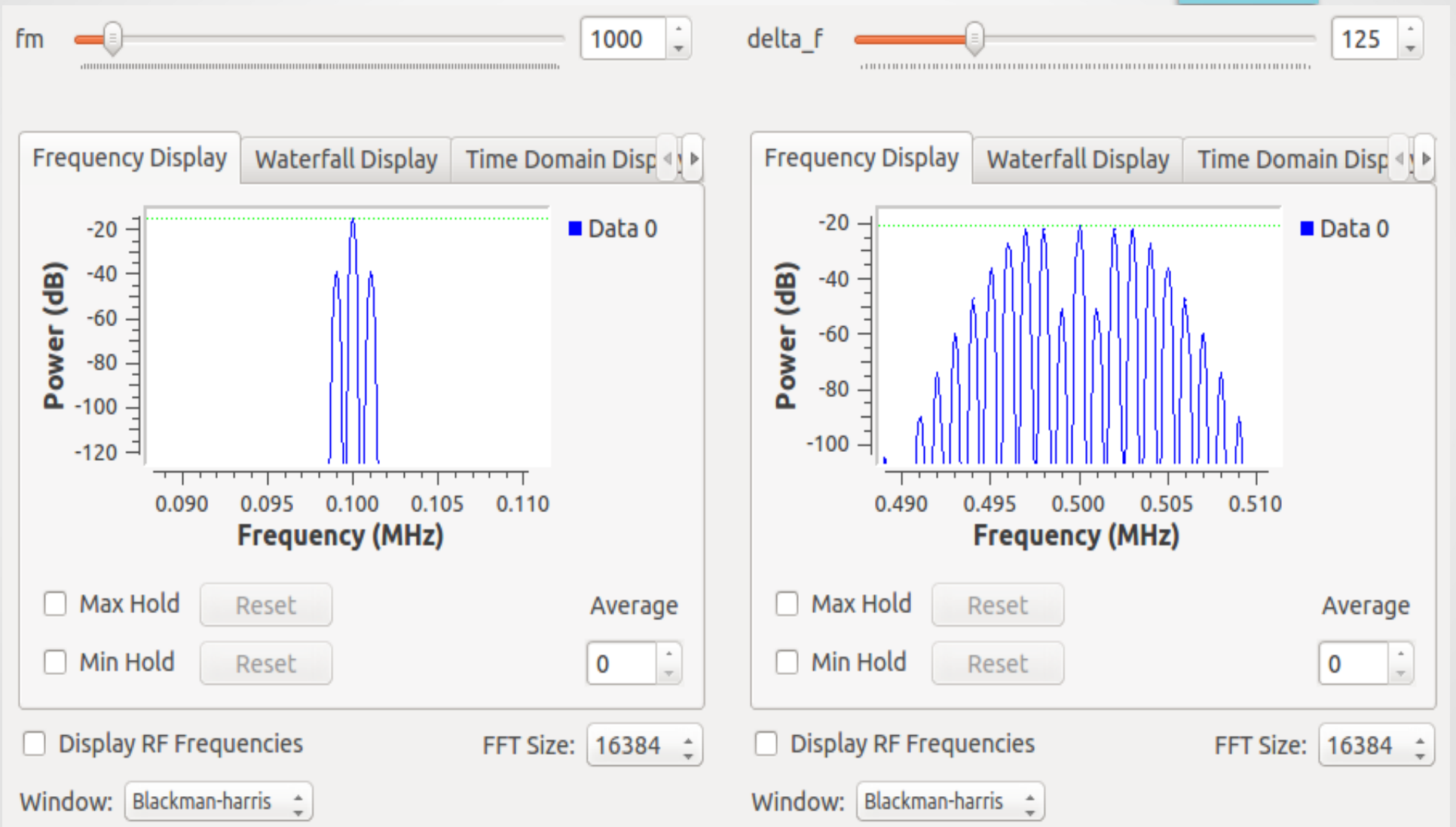


Frequency Modulation

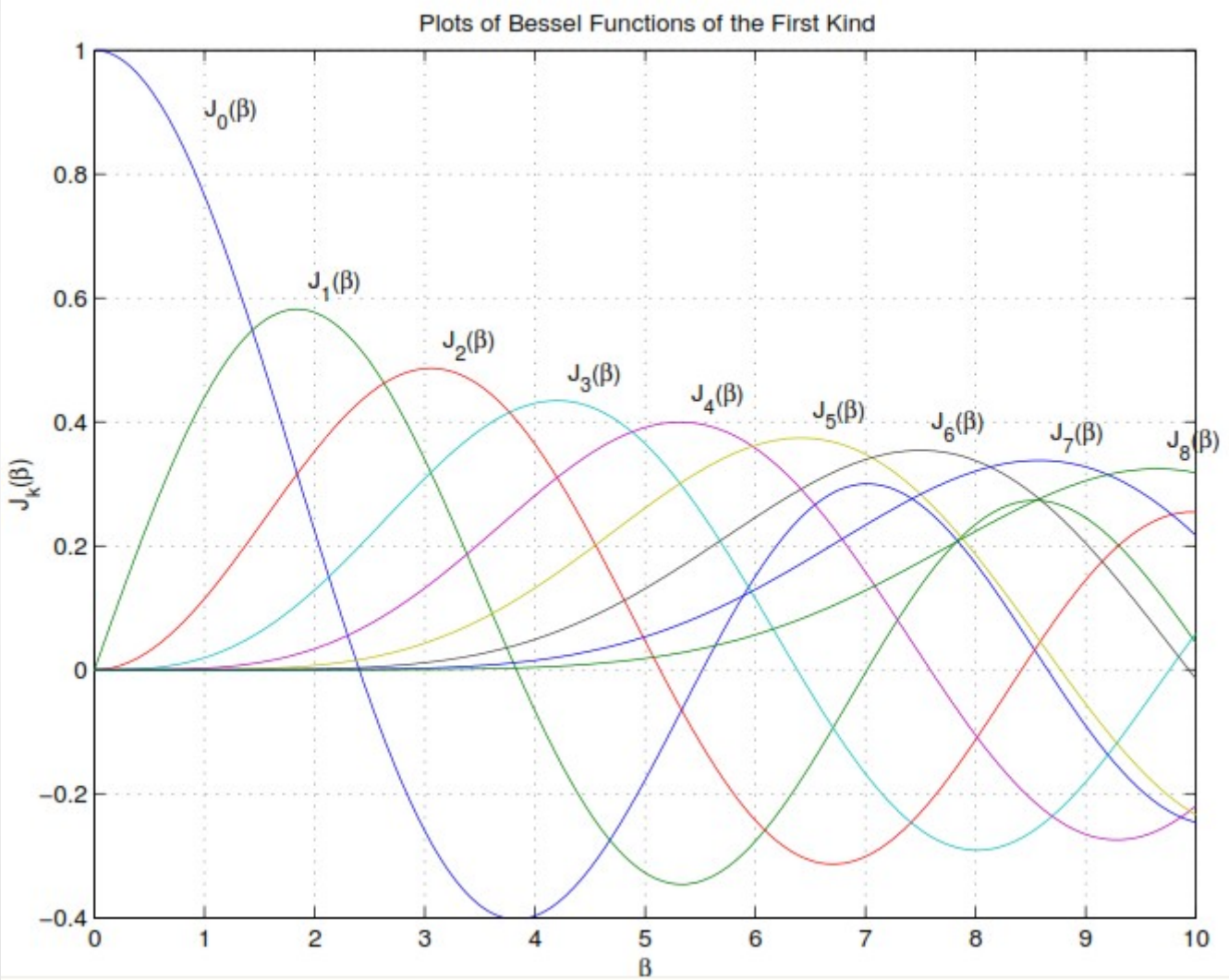
Frequency Modulation



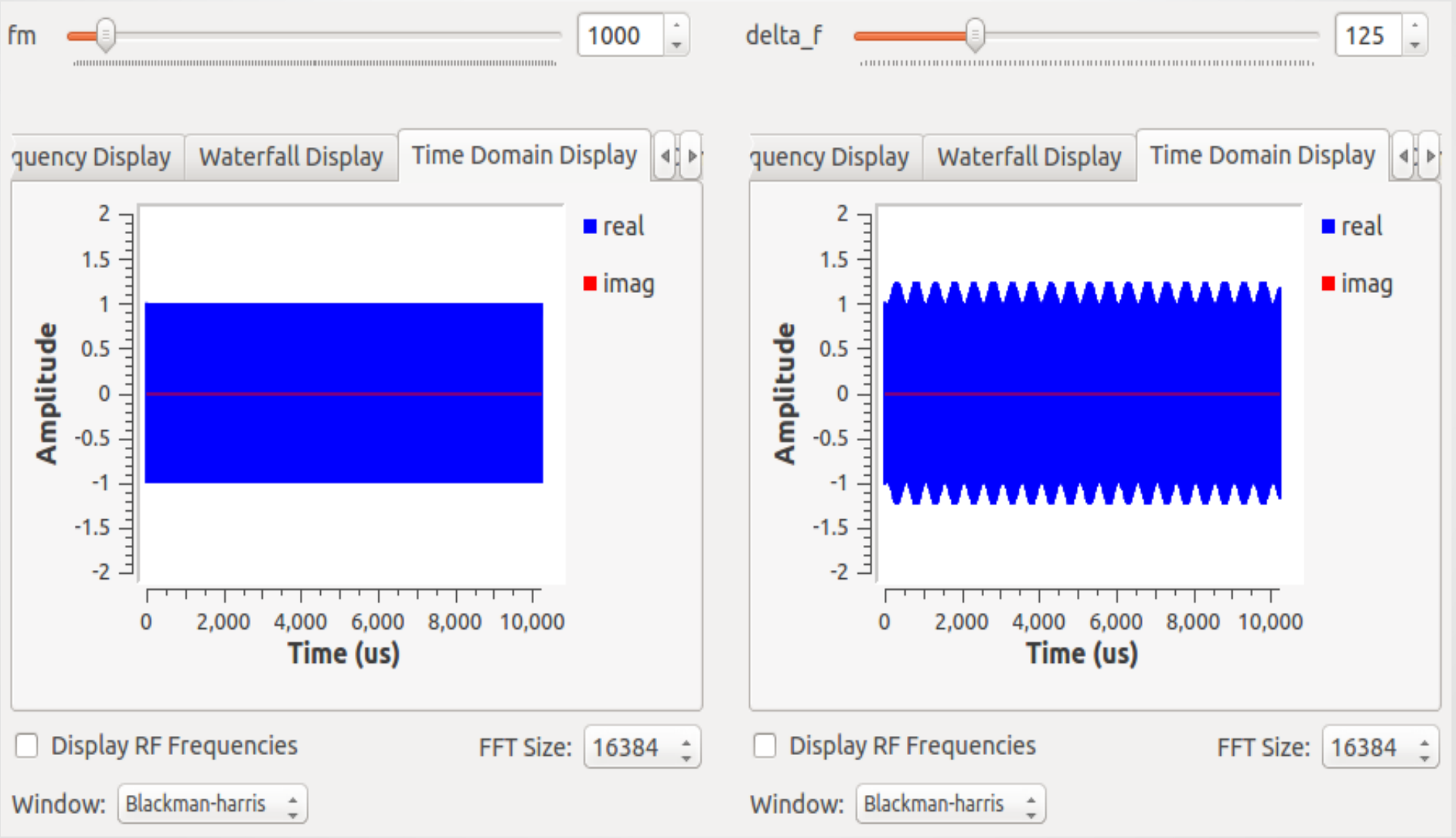
Frequency Modulation



Find beta using Bessel Functions



Frequency Modulation



ECEN 4652 Communications Lab

- Lab 1: CT and DT Signals, ASCII Code, Parallel to Serial to Parallel Conversion, Simple Rectangular PAM
- Lab 2: Fourier Transform Approximation by DFT/FFT, More General PAM (Rectangular, Triangular, Sinc Pulses)
- Lab 3: Sampling Theorem, Nyquist's Criterion, Eye Diagrams, ISI, Partial Response Signaling
- Lab 4: Random Processes, Power Spectral Density, Noise, Symbol Timing Information
- Lab 5: PAM Receiver with Matched Filter, SNR, Probability of Symbol Error

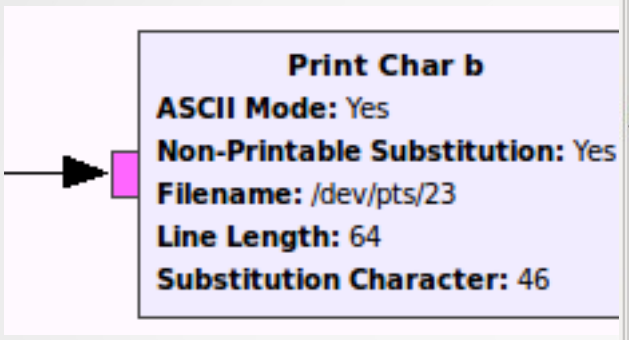
ECEN 4652 Communications Lab

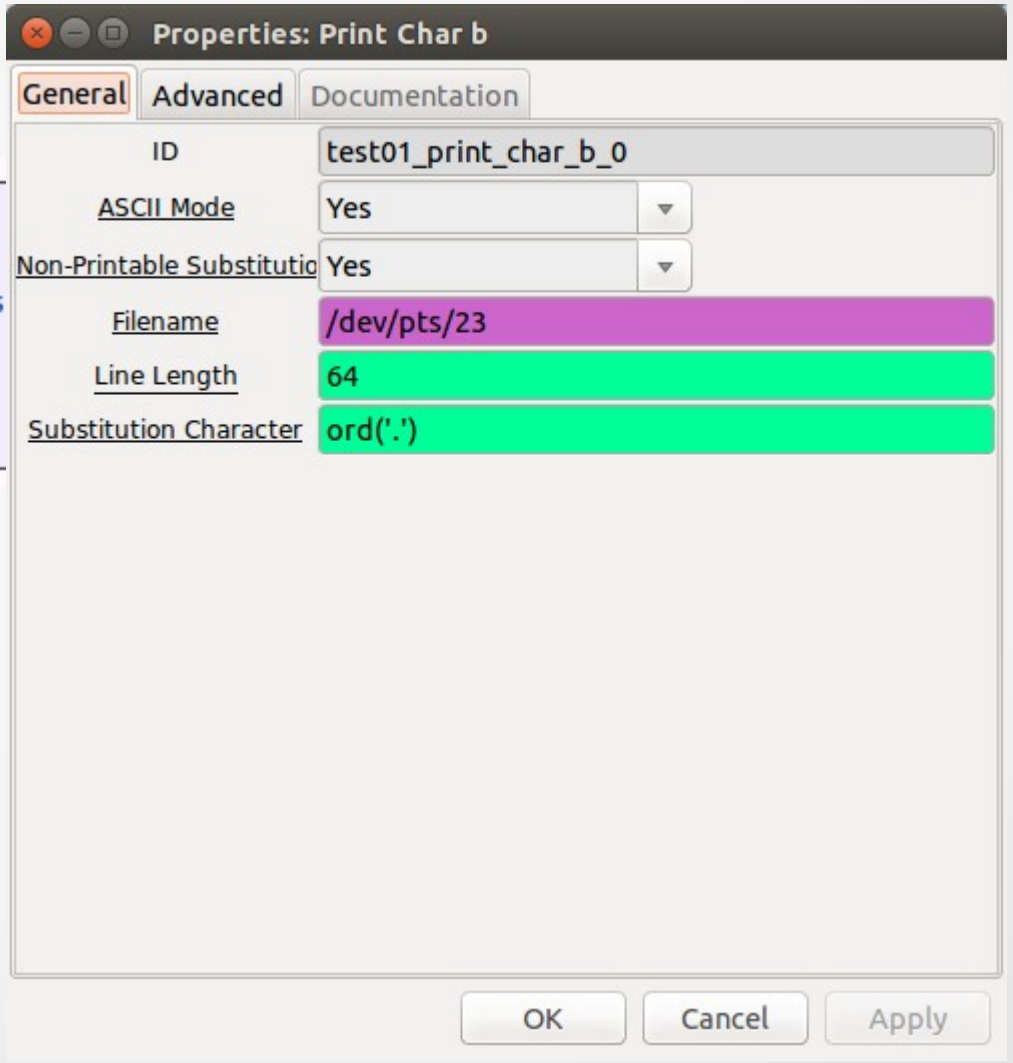
- Lab 6: Introduction to Software Defined Radio and GNU Radio
- Lab 7: Amplitude Modulation with Suppressed Carrier, Coherent Reception
- Lab 8: Amplitude Modulation with Transmitted Carrier, Noncoherent Receivers, FDM
- Lab 9: M-ary Amplitude and Frequency Shift Keying, Signal Space
- Lab 10: Real Bandpass and Complex Lowpass Signals, QAM, General Bandpass Filters
- Lab 11: Phase and Hybrid Amplitude/Phase Shift Keying, Carrier Synchronization
- URL: http://ecee.colorado.edu/~mathys/ecen4652

A Missing Piece in GNU Radio

- Serial bit input to ASCII text conversion with:
- LSB or MSB first selection,
- Selectable threshold for 0/1 decision,
- 0 ↔ 1, 1 ↔ 0 inversion,
- Selectable bit shift within bitstream
- Character substitution for non-printable ASCII
- Switchable between Hex and ASCII display

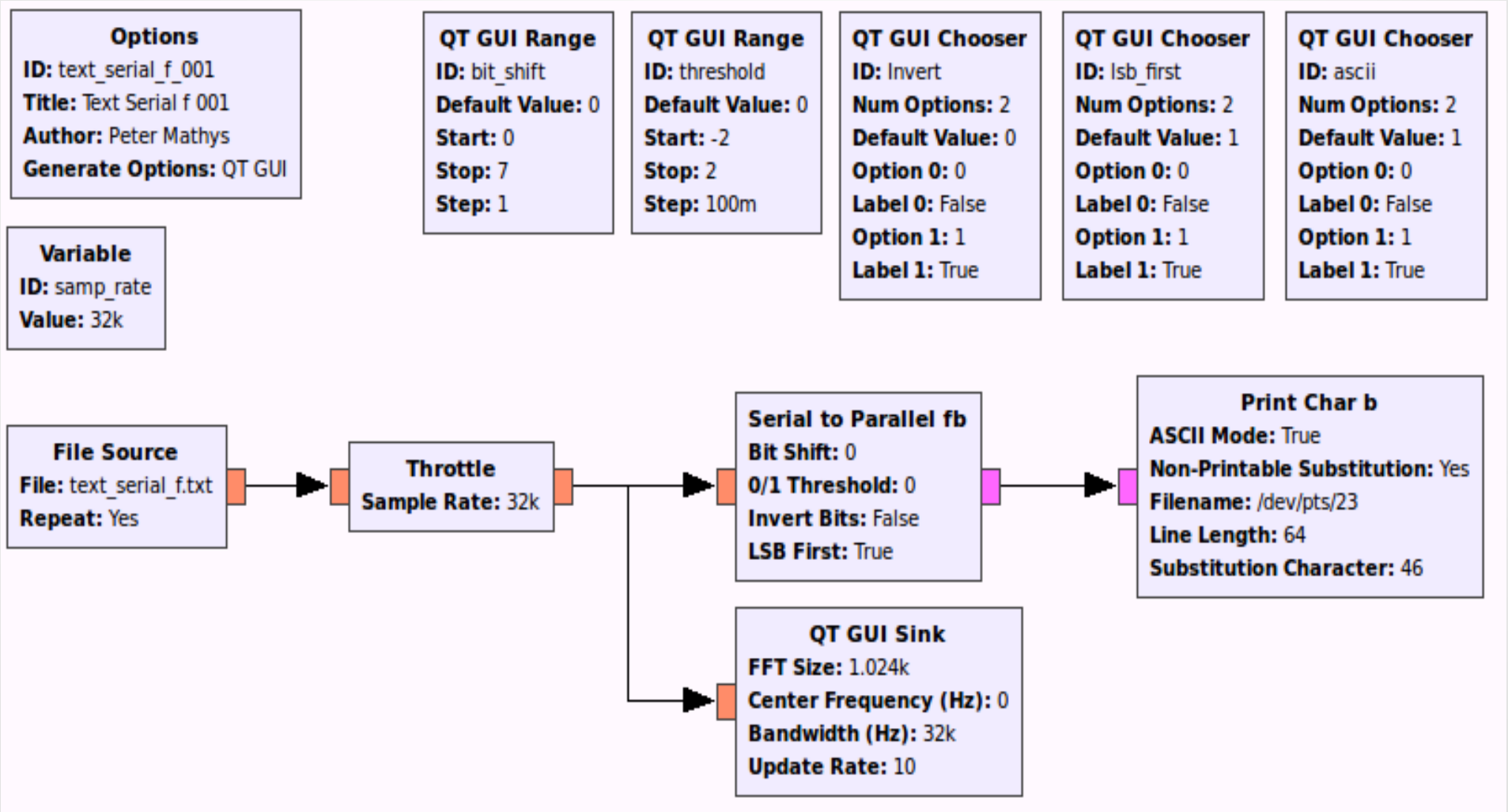
Our print_char_b OOT Block



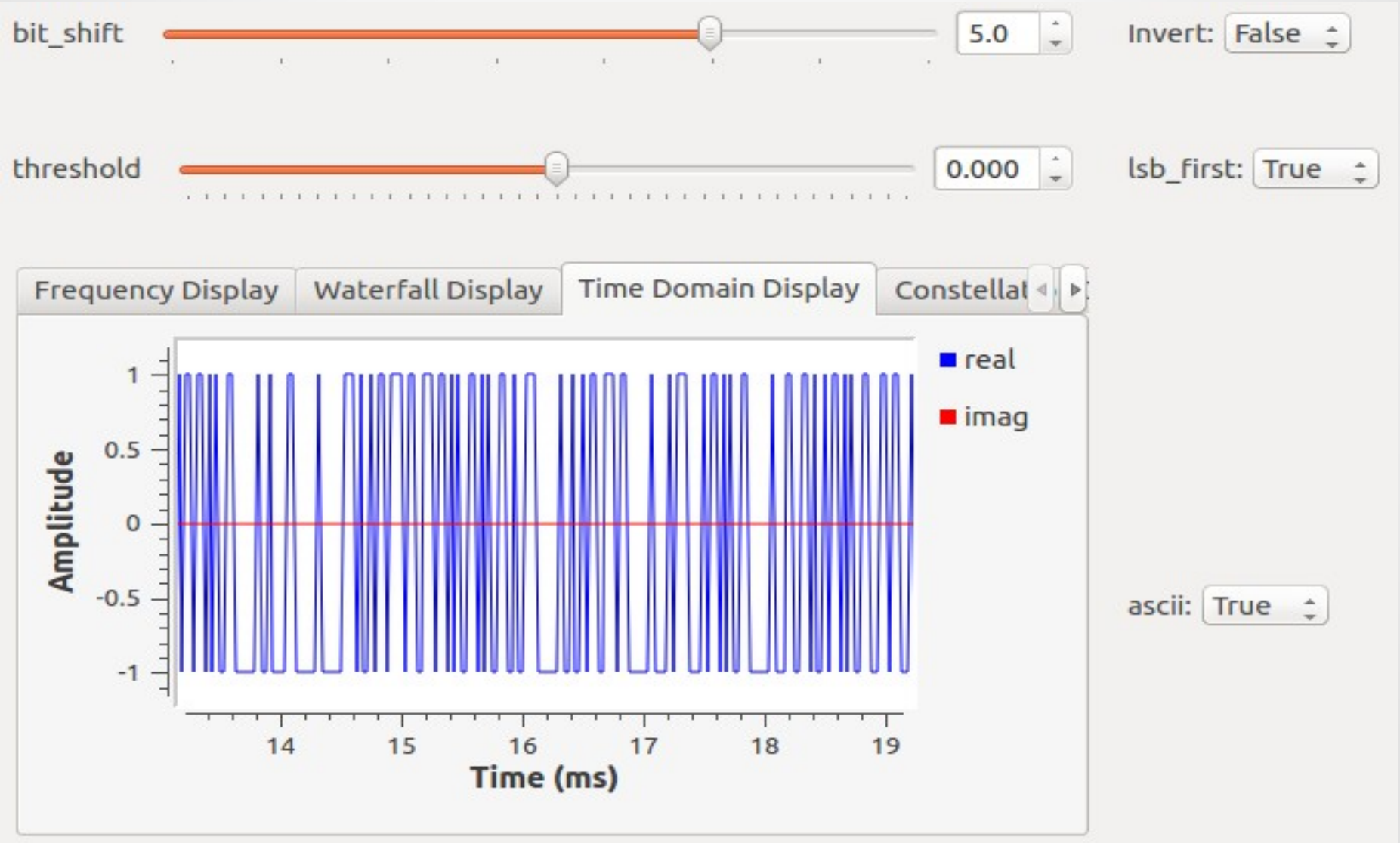


Digital Baseband Communications

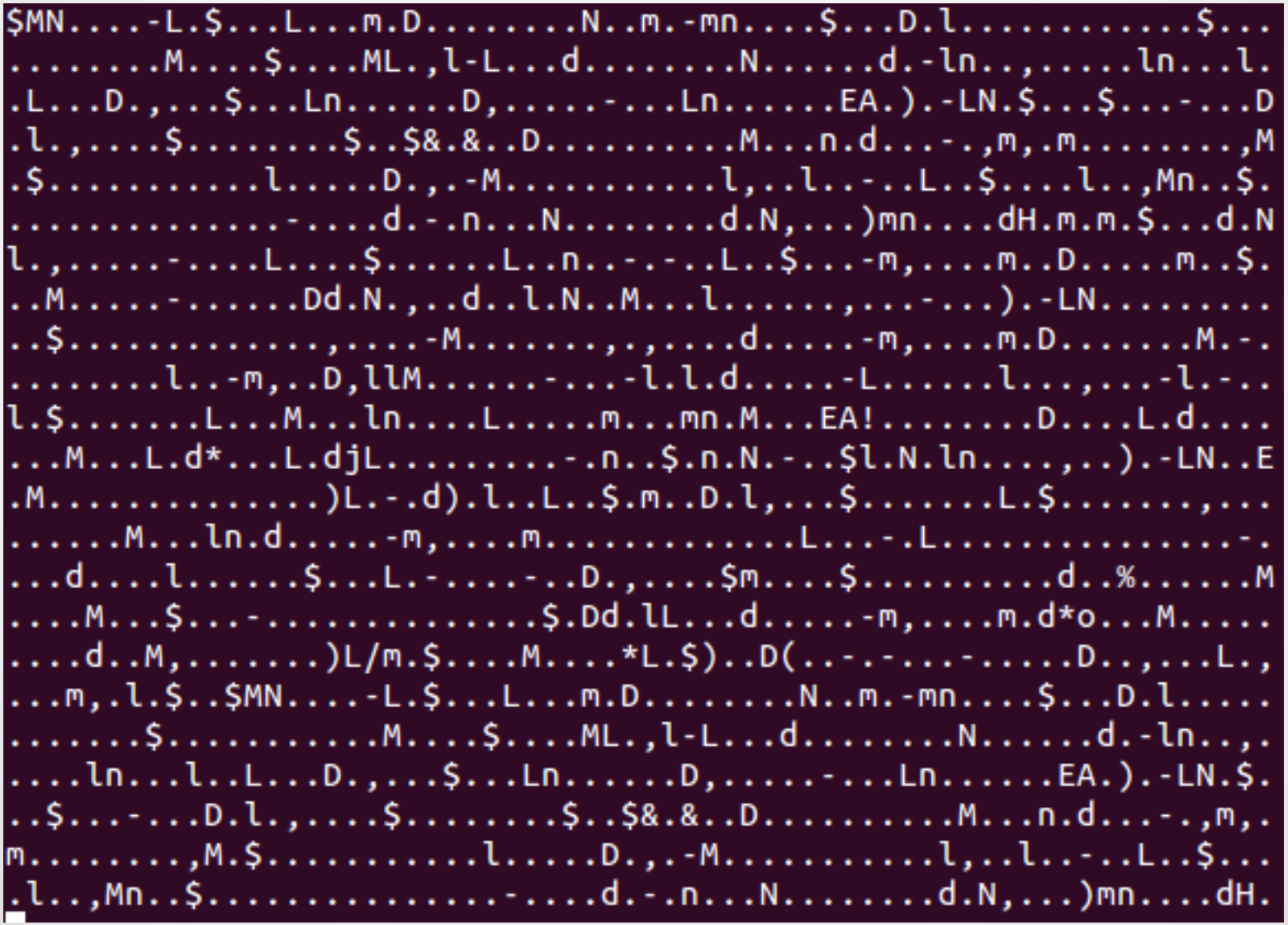
Visualizing a Serial Data Stream



Visualizing a Serial Data Stream



Wrong Bit Shift for ASCII



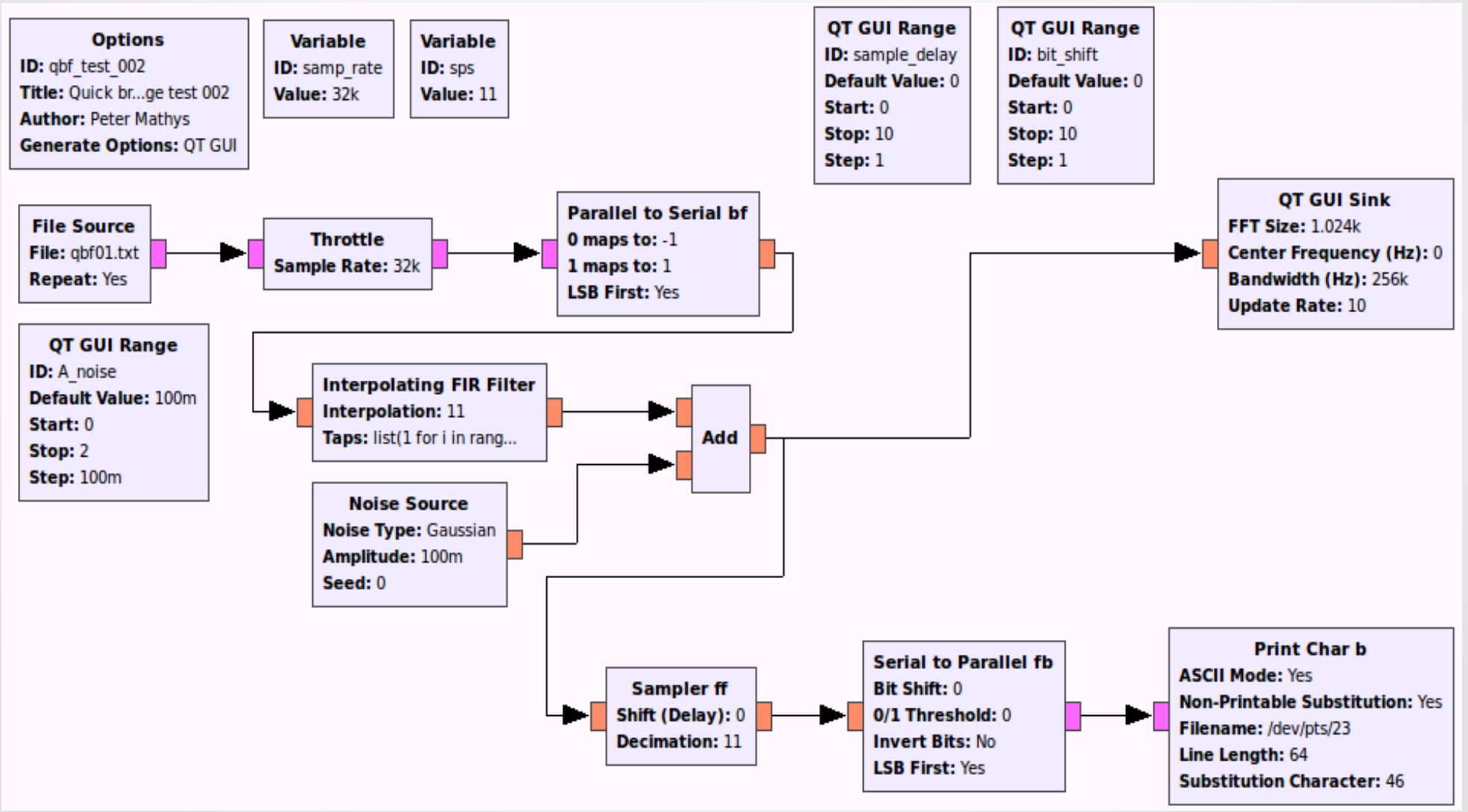
ASCII and Hex Display Options

```
 in numerous military applications. But most importantly, the "s
pread spectrum" technology that Lamarr helped to invent would ga
lvanize the digital communications boom, forming the technical b
ackbone that makes cellular phones, fax machines and other wirel
ess operations possible...Although better known for her Silver S
creen exploits, Austrian actress Hedy Lamarr (born Hedwig Eva Ma
ria Kiesler) also became a pioneer in the field of wireless comm
unications following her emigration to the United States. The in
ternational beauty icon, along with co-inventor George Anthiel,
developed a "Secret Communications System" to help combat the Na
zis in World War II. By manipulating radio frequencies at irregu
lar intervals between transmission and reception, the invention
formed an unbreakable code to prevent classified messages from b
eing intercepted by enemy personnel...Lamarr and Anthiel receive
d a patent in 1941, but the enormous significance of their inven
tion was not realized until decades later. It was first implemen
```

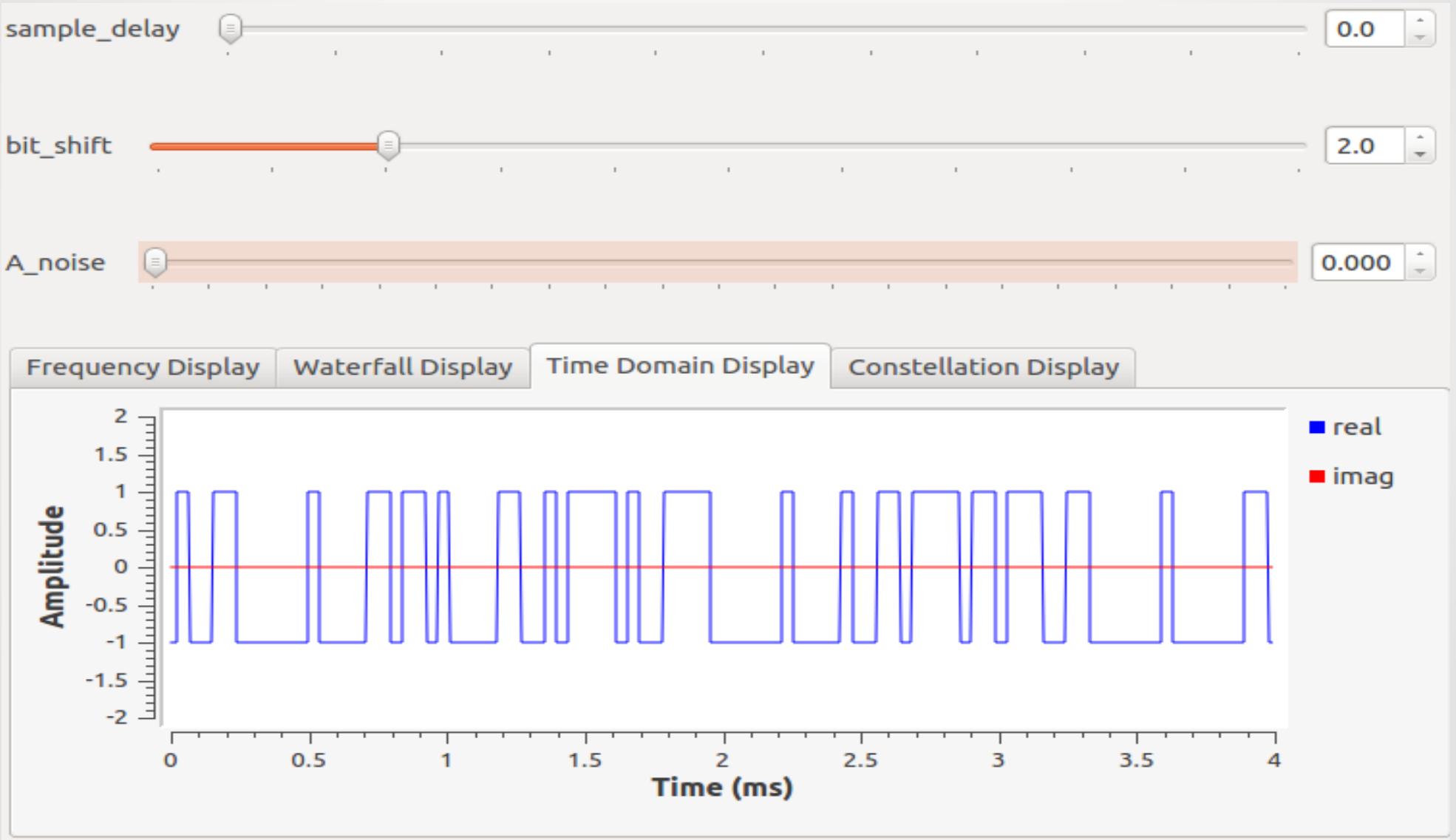
```
77,6E,20,66,6F,
65,6E,20,65,78,
61,63,74,72,65,
6F,72,6E,20,48,
69,65,73,6C,65,
70,69,6F,6E,65,
66,20,77,69,72,
```

```
65,6C,65,73,73,20,63,6F,6D,6D,75,6E,69,63,61,74,69,6F,6E,73,20,66,
6F,6C,6C,6F,77,69,6E,67,20,68,65,72,20,65,6D,69,67,72,61,74,69,6F,
6E,20,74,6F,20,74,68,65,20,55,6E,69,74,65,64,20,53,74,61,74,65,73,
2E,20,54,68,65,20,69,6E,74,65,72,6E,61,74,69,6F,6E,61,6C,20,62,65,
61,75,74,79,20,69,63,6F,6E,2C,20,61,6C,6F,6E,67,20,77,69,74,68,20,
63,6F,2D,69,6E,76,65,6E,74,6F,72,20,47,65,6F,72,67,65,20,41,6E,74,
68,69,65,6C,2C,20,64,65,76,65,6C,6F,70,65,64,20,61,20,22,53,65,63,
72,65,74,20,43,6F,6D,6D,75,6E,69,63,61,74,69,6F,6E,73,20,53,79,73,
```

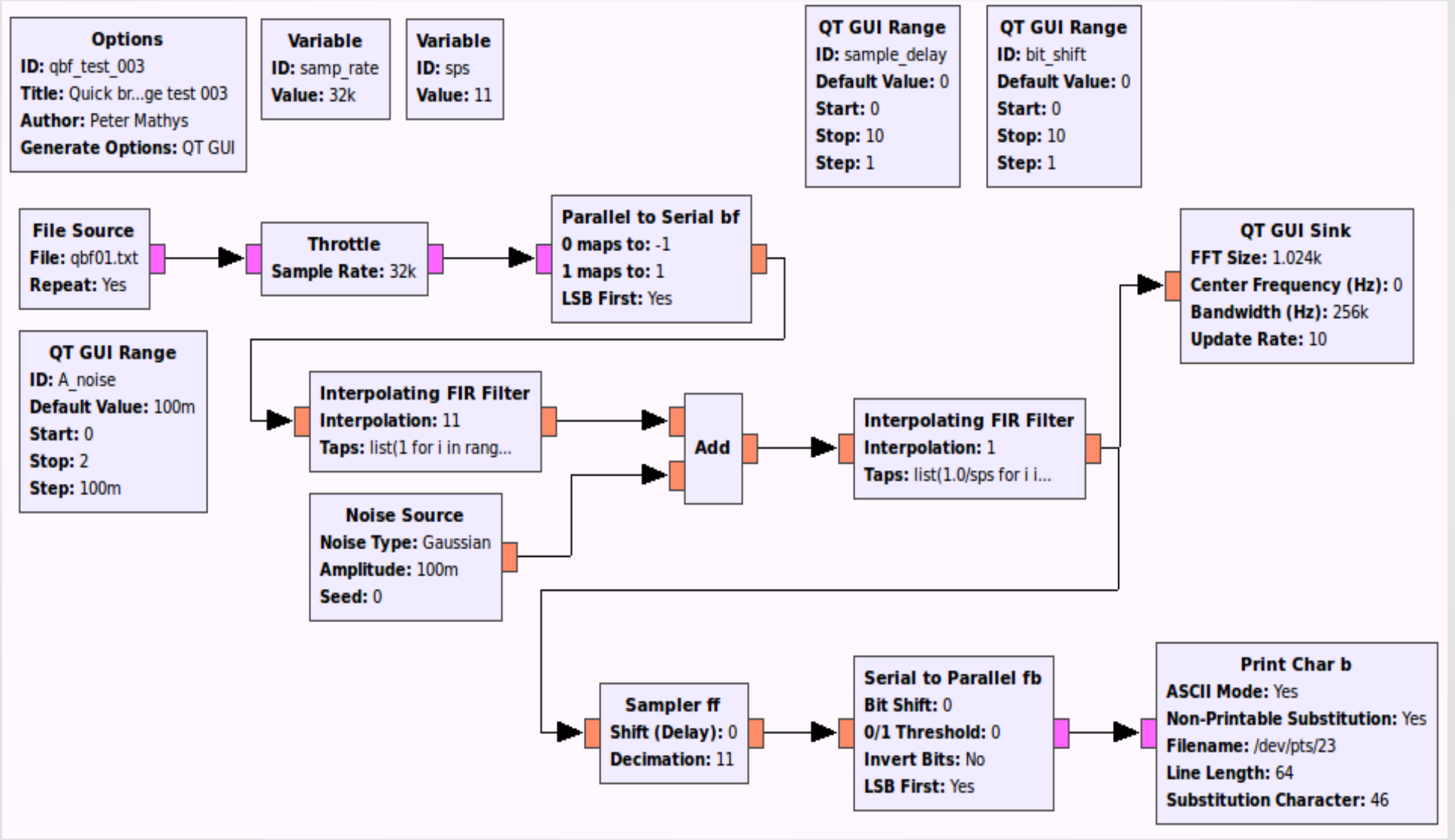
Rectangular PAM with Noise w/o MF



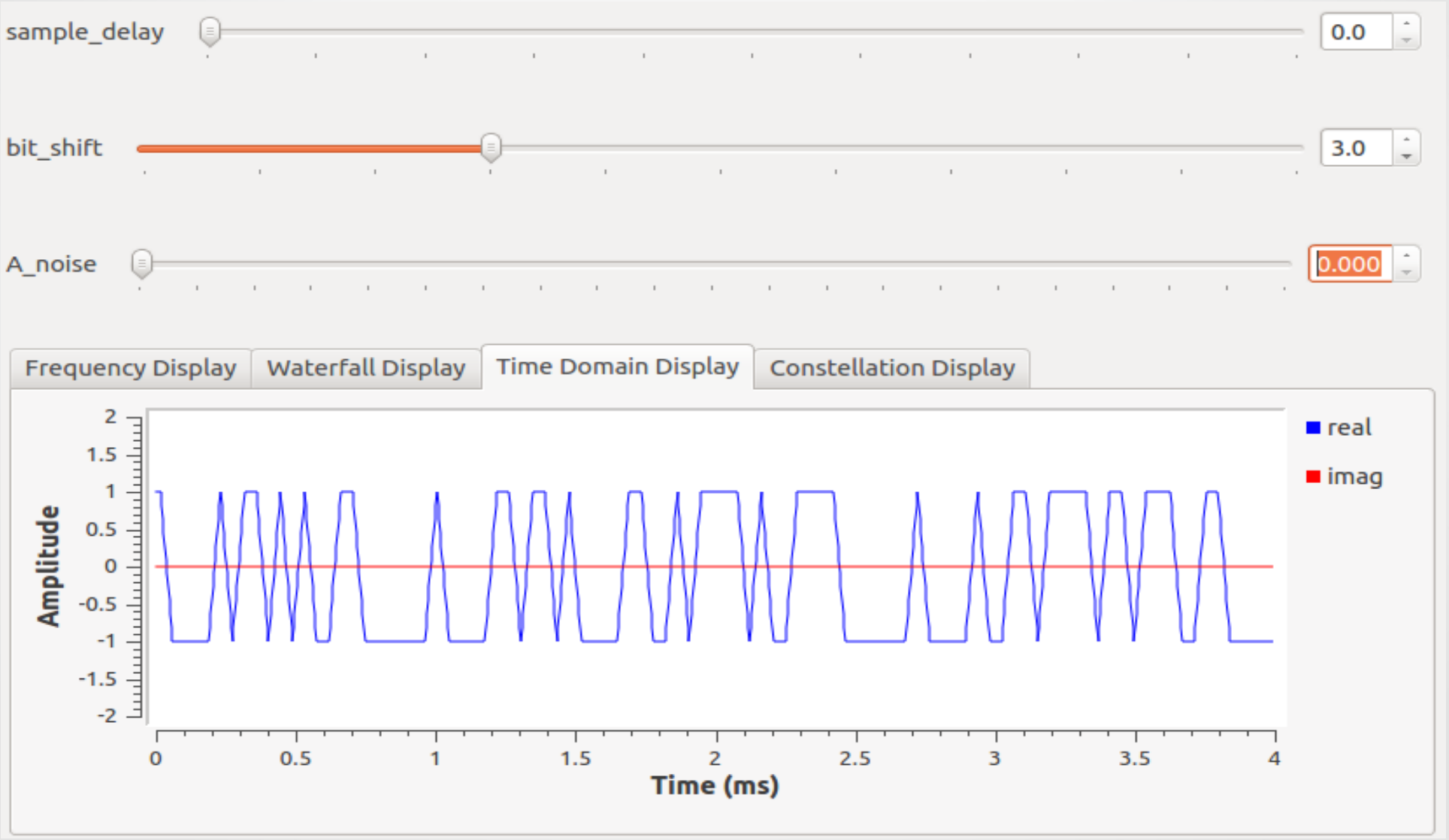
Rectangular PAM without MF



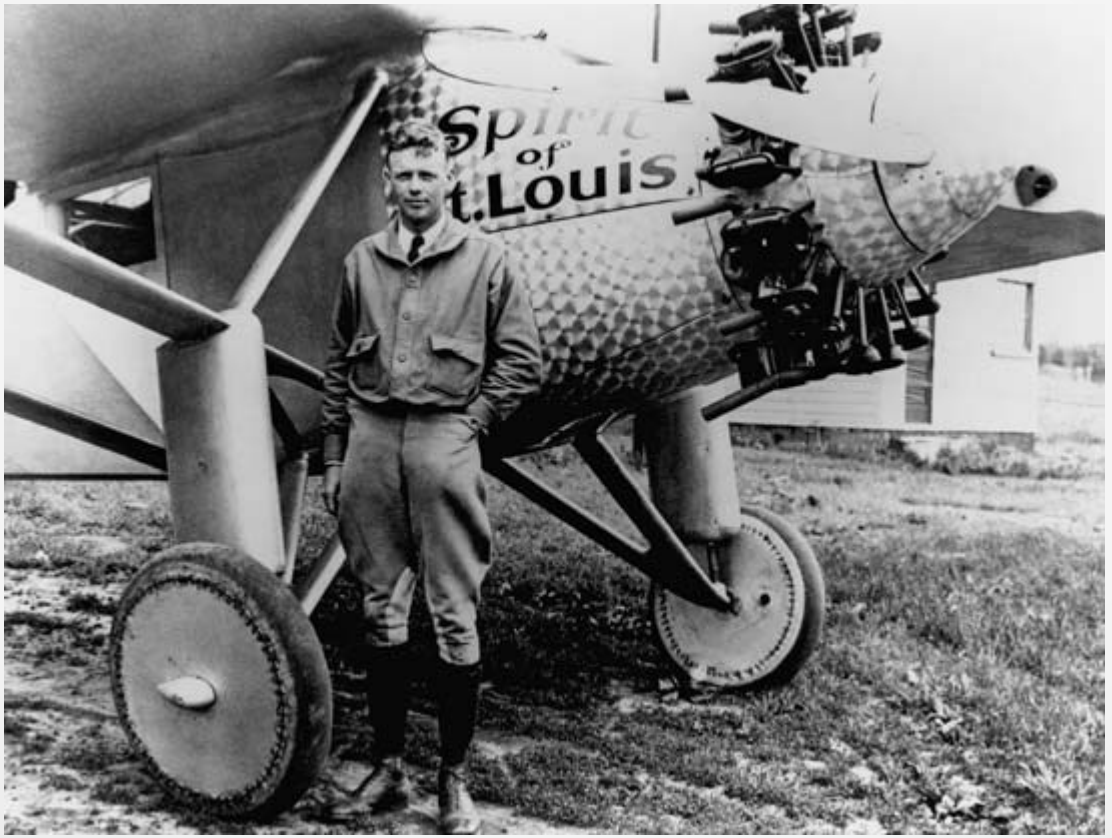
Rectangular PAM with Noise with MF



Rectangular PAM after MF



Flying by the Seat of our Pants



GNU Radio Installation

- **1'st attempt:**
- Install VirtualBox on either PC or Mac
- Install Ubuntu 14.04 in VirtualBox
- Install GNU Radio in Ubuntu using script at

  www.sbrac.org/files/build-gnuradio
- Problems:
- GNU Radio in VirtualBox is too slow
- Too much troubleshooting needed for individual students and their computers

GNU Radio Installation

- **2'nd attempt** (after seg fault in my own installation due to software conflicts):
- Put Ubuntu and GNU Radio on a USB flashdrive for both experimentation and for handing out to students
- Use the older MBR method (and not UEFI) to boot from the flash drive
- Use fast (USB 3) 32 GB flashdrive
- Do not format and use the full 32 GB if you want to make image copies of the flashdrive (some 32 GB drives are smaller than others!)

New to GNU Radio Development?

- **The (likely) unknowns:**
- Unix/Linux
- Python
- C/C++
- Object-Oriented Programming
- SWIG
- Cmake, Make
- Cheetah and Mako
- Boost
- Git, GitHub
- Sphinx, Doxygen
- XML
- FFTW
- Volk
- (Multirate) DSP
- Communication Theory
- Communication Standards

Learning How to Write OOT Modules

- Start from the square_ff tutorial (gr::sync_block)
- Look at code for add_const_vxx block to learn about float and gr-complex vectors and function arguments
- Look at code for pack_k_bits_bb to learn about sync_decimator blocks
- Look at code for unpack_k_bits_bb to learn about sync_interpolator blocks
- Look at code for moving_average_XX to learn about the set_history() feature

Conclusion

- GNU Radio is a great addition to the undergraduate communication curriculum.
- It is important to introduce GNU Radio early on to the next generation of communication engineers.
- GNU Radio must be working in the student environment with a minimum of hassle and overhead.
- The building blocks used for the basic education must be simple and their function must be transparent.
- Actual text and audio signals should be used whenever possible to emphasize the applicability of the theory.