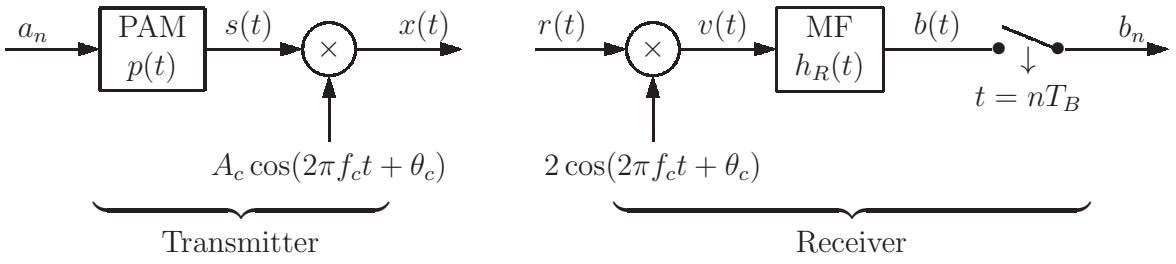
Lab 8: BPSK, Amplitude and Frequency Shift Keying, Signal Space

1 Introduction

Amplitude modulation (AM) can easily be used for the transmission of digital data if the (analog) message signal $m(t)$ is replaced by a (digitally) pulse amplitude modulated (PAM) signal $s(t)$. The simplest method to tansmit binary data is on-off keying (OOK), whereby the transmitter is off for sending 0's, and a carrier at frequency f_c is transmitted for sending 1's (or vice versa). If the carrier oscillator at the transmitter is just turned on and off, then the phase for each 1 is random and a non-coherent receiver must be used. If phase coherence is maintained at the transmitter, then coherent reception (with a smaller probability of error) is also possible. Instead of multiplying the carrier oscillator by 0 or 1 for binary data, it can be multiplied by -1 or $+1$, thereby producing phase changes by $\pm 180°$ at the carrier frequency f_c. This method is called binary phase shift keying (BPSK) and requires a coherent receiver. An extension of this is phase shift keying (PSK) with four phases, e.g., using $0°$, $\pm 90°$ and $180°$, for a 4-ary PAM signal. This is called quaternary phase shift keying (QPSK) and requires a coherent receiver. Instead of using a single carrier frequency f_c for transmitting 0's and 1's, two distinct frequencies, e.g., f_{c0} and f_{c1} can be used to transmit 0's and 1's, respectively. The resulting signal is a binary frequency shift keying (BFSK) signal. It can be easily demodulated with a non-coherent receiver. If phase coherence is maintained at the transmitter, it can also be demodulated (with a smaller probability of error) using a coherent receiver. Both, amplitude shift keying (ASK) and frequency shift keying (FSK) can be extended in a straightforward manner from the binary to the M-ary case by using M amplitudes (e.g., $0, 1, 2, 3$ or $-3, -1, 1, 3$ for $M = 4$) or M carrier frequencies.

1.1 Amplitude Shift Keying

The name "Amplitude Shift Keying" (ASK) refers to a digital bandpass modulation method, whereby the amplitude of a carrier frequency f_c takes on different discrete values (at the sampling time instants), depending on the transmitted DT sequence a_n. The blockdiagram of a coherent ASK communication system looks as follows.



At the transmitter a PAM signal

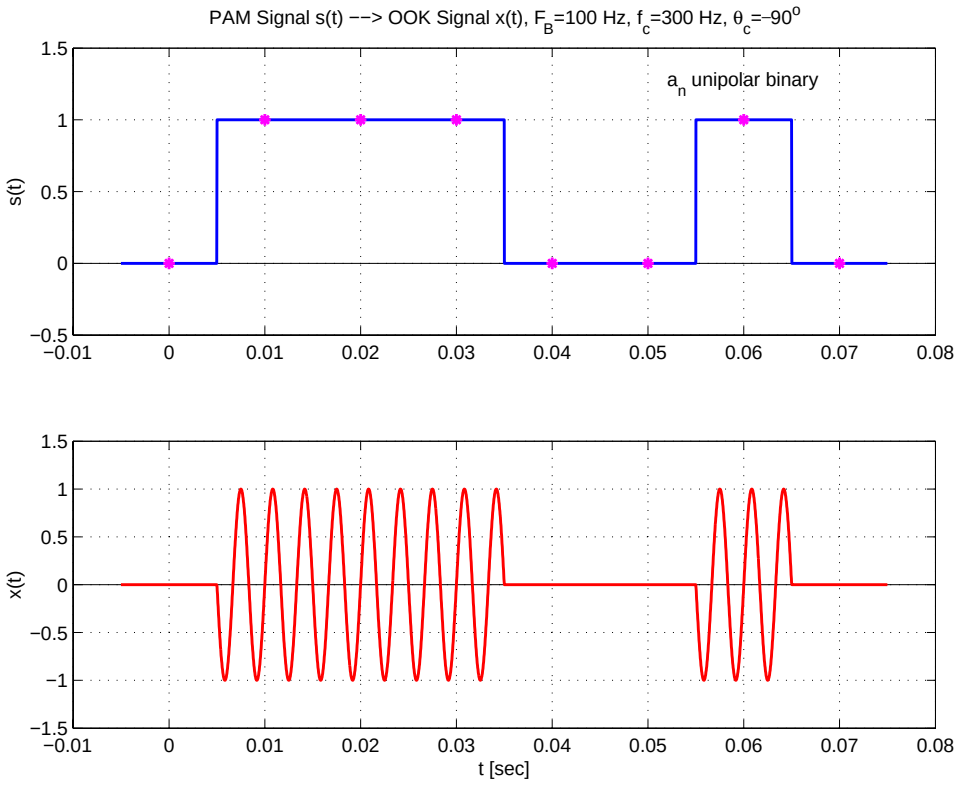
$$s(t) = \sum_{n=-\infty}^{\infty} a_n \, p(t - nT_B) \, ,$$

with baud rate $F_B = 1/T_B$ and pulse $p(t)$ is generated from the (digital) DT sequence a_n. The CT signal $s(t)$ then modulates a carrier with frequency f_c and phase θ_c, so that the transmitted signal is

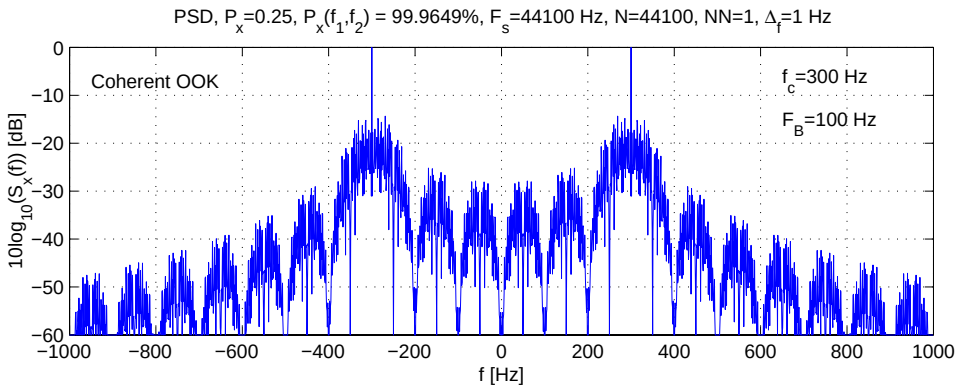
$$x(t) = A_c \sum_{n=-\infty}^{\infty} a_n \, p(t - nT_B) \, \cos(2\pi f_c t + \theta_c) \, .$$

At the receiver the signal $r(t)$ is demodulated using a local oscillator that replicates the carrier signal $\cos(2\pi f_c t + \theta_c)$ with exact frequency and phase synchronization. After that, a regular baseband PAM receiver with matched filter (MF) $h_R(t)$, matched to $p(t) * h_{CL}(t)$, where $h_{CL}(t)$ is the lowpass-equivalent channel response (i.e., $h_C(t)$ shifted down in frequency by f_c), is used.

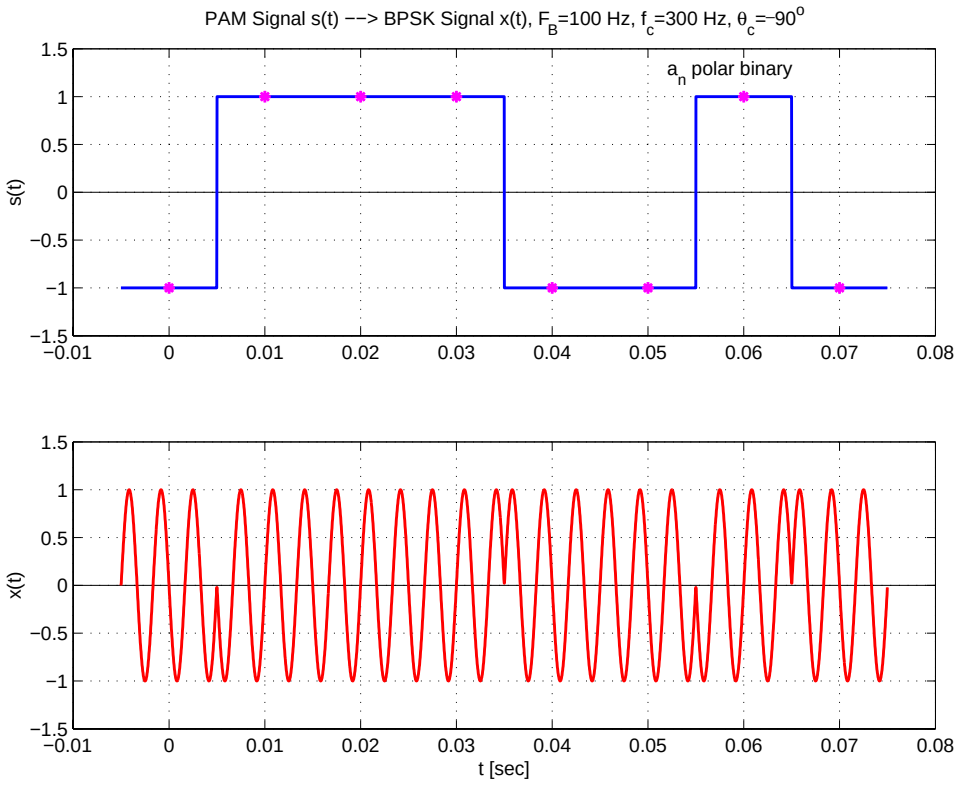
The simplest case of ASK is **on-off keying (OOK)**, which is obtained when a_n is a unipolar binary sequence (i.e., $a_n \in \{0, 1\}$). The following two graphs show $s(t)$ and $x(t)$ for $\{a_n\} = \{0, 1, 1, 1, 0, 0, 1, 0\}$, $F_B = 100$ Hz, rectangular $p(t)$, $f_c = 300$ Hz, and $\theta_c = -90°$.



The PSD of a coherent OOK signal (i.e., an OOK signal for which θ_c does not change over time) with rectangular $p(t)$, $f_c = 300$ Hz, and $F_B = 100$ Hz is shown in the next graph.



If a_n is a polar binary sequence, e.g., $\{a_n\} = \{-1, +1, +1, +1, -1, -1, +1, -1\}$, then $s(t)$ and $x(t)$ look as follows.



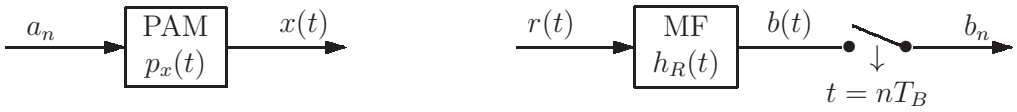
Assuming a rectangular $p(t)$, the carrier is multiplied by either $+1$ or -1, depending on the data to be transmitted. This leads to phase changes by $180°$ and because of it this version of ASK is usually called **binary phase shift keying (BPSK)**. Note that for BPSK it is crucial that the transmitter and receiver are phase-synchronized, i.e., BPSK requires a coherent receiver.

Rather than generating a baseband PAM signal first and then using amplitude modulation to make it a bandpass signal at f_c, it is possible to directly generate a bandpass PAM signal

using a pulse $p_x(t)$ that is already shifted to f_c in the frequency domain. Mathematically, this can be expressed as

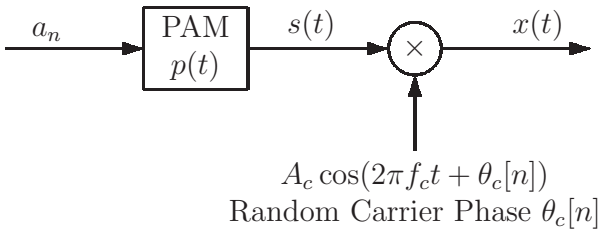
$$x(t) = \sum_{n=-\infty}^{\infty} a_n \, p_x(t - nT_B) \, , \qquad \text{where} \quad p_x(t) = p(t) \, \cos(2\pi f_c t + \theta_c) \, .$$

At the receiver one can then use a MF $h_R(t)$ that is directly matched to $p_x(t) * h_c(t)$ where $h_c(t)$ is the impulse response of the channel. The blockdiagram of an ASK transmission system that uses $p_x(t)$ directly at f_c is shown below.



$p_x(t) = p(t) \, \cos(2\pi f_c t + \theta_c)$ $\qquad$ $h_R(t)$ matched to $p_x(t) * h_c(t)$

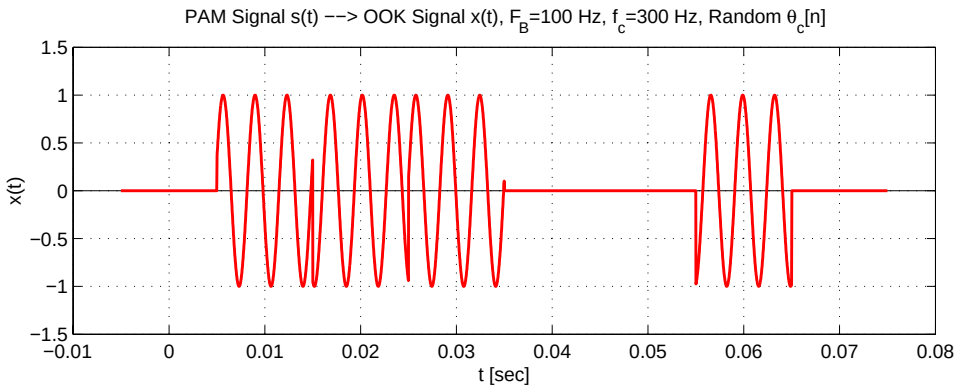
The simplest method to achieve wireless data transmission is to turn an oscillator with frequency f_c on or off, depending on whether a one or a zero is transmitted. Using the Morse code for on-off keying, this is how Marconi demonstrated in 1901 that transatlantic wireless communication was possible. What is likely to happen, however, if the oscillator at f_c is turned on and off, is that the carrier phase θ_c takes on random values between 0 and 2π. A model for this is shown in the following blockdiagram.



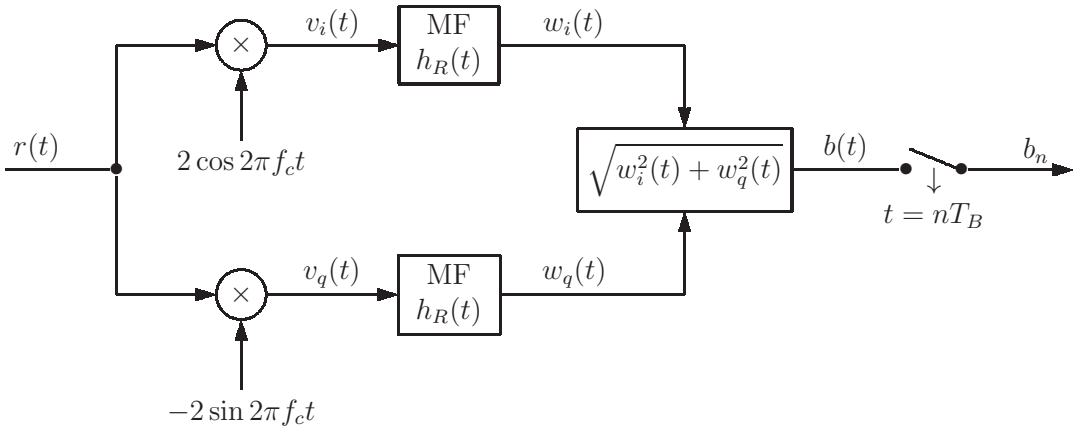
Here it is assumed that for each new symbol a_n a random carrier phase value $\theta_c[n]$ is chosen (even if the oscillator is not turned off between two consecutive ones). Thus

$$\begin{aligned} x(t) &= A_c \sum_{n=-\infty}^{\infty} a_n \, p(t - nT_B) \, \cos(2\pi f_c t + \theta_c[n]) \\ &= A_c \Big[\sum_{n=-\infty}^{\infty} a_n \cos\theta_c[n] \, p(t - nT_B) \, \cos 2\pi f_c t - \sum_{n=-\infty}^{\infty} a_n \sin\theta_c[n] \, p(t - nT_B) \, \sin 2\pi f_c t \Big] \, , \end{aligned}$$

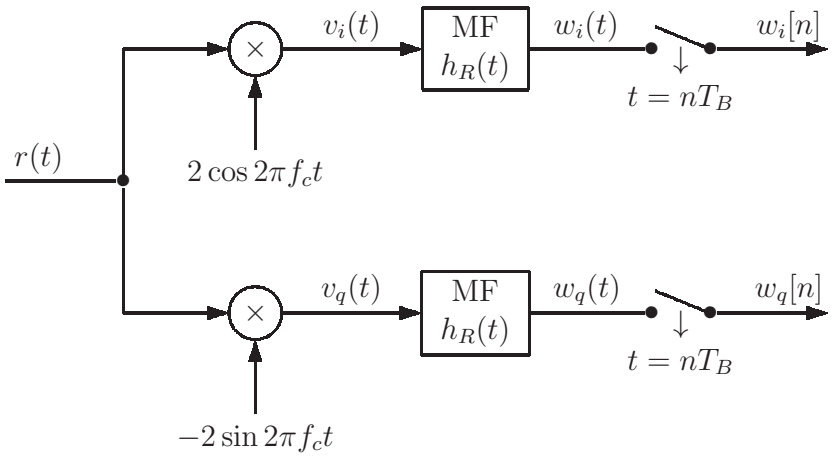
where $\theta_c[n] \in [0, 2\pi)$. Because the transmitted carrier phase of such a signal is not constant, the resulting ASK signal will be called **non-coherent ASK**. An example when $\{a_n\} = \{0, 1, 1, 1, 0, 0, 1, 0\}$, $F_B = 100$ baud, and $f_c = 300$ Hz is shown in the next graph.



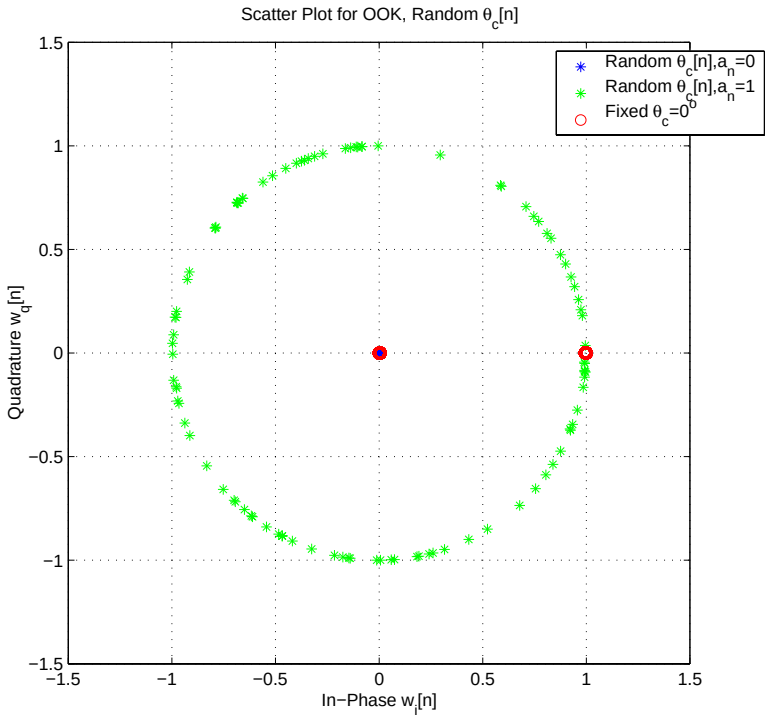
To demodulate a signal with (unknown) random phase, a **matched filter envelope detector (MFED)**, such as the one shown in the following blockdiagram, can be used.



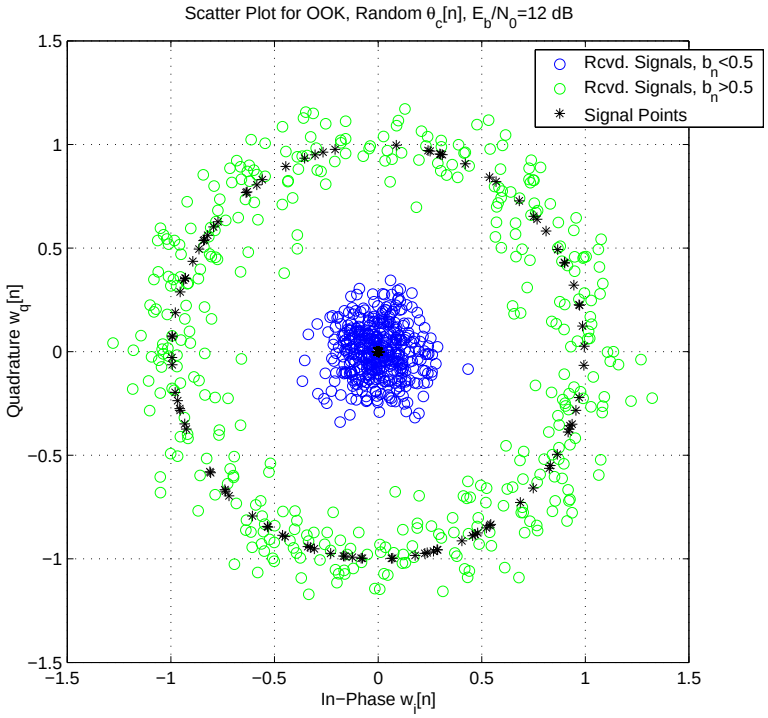
In the process of computing $b(t) = \sqrt{w_i^2(t) + w_q^2(t)}$, the dependence on the carrier phase (and in fact the dependence on the exact carrier frequency f_c) drops out. The interesting observation to be made is that, because of the orthogonality of $\cos 2\pi f_c t$ and $\sin 2\pi f_c t$, the in-phase component $w_i(t)$ and the quadrature component $w_q(t)$ span a 2-dimensional space. Upon sampling at times $t = nT_B$, as shown in the blockdiagram below, this becomes a **2-dimensional signal space** spanned by $w_i[n]$ (real part in complex plane) and $w_q[n]$ (imaginary part in complex plane).



Thus, coherent ASK uses a one-dimensional signal space and noncoherent ASK uses a two-dimensional signal space. This is shown for OOK in the following figure which is called a **scatter plot** or **constellation plot**.



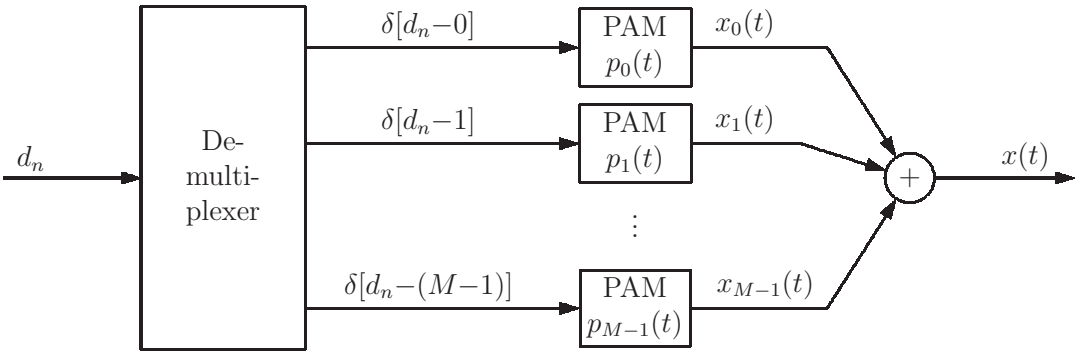
Note that for noncoherent ASK the signal "points" are actually points on concentric circles. If the received signal is noisy, then the noise manifests itself as deviations of these points from the concentric circles, as shown for OOK with SNR $E_b/\mathcal{N}_0 = 12$ dB in the scatter plot below.



The black stars are the signal points of the noncoherent ASK signal at the transmitter. The blue and green circles are the received noisy signal points with blue indicating a decision for a received 0 and green indicating a decision for a received 1. As the SNR decreases, the received signal points deviate more and more from their nominal positions at the origin or on the circle with radius 1, and the probability of symbol error, $P_s(\mathcal{E})$, approaches 0.5.

1.2 Frequency Shift Keying

Rather than using a digital DT sequence to change the amplitude of a carrier, the frequency of the carrier can be changed by the data sequence to be transmitted. The resulting digital bandpass signaling method is called **frequency shift keying (FSK)**. One way to implement M-ary FSK is shown in the following block diagram.



An M-ary DT sequence d_n, with $d_n \in \{0, 1, \ldots, M-1\}$, is first demultiplexed into M binary sequences, the first one having 1's in all positions where d_n is zero and 0's everywhere else, the second one having 1's in all positions where d_n is one and 0's everywhere else, etc. The resulting M sequences are then converted to OOK signals of the form

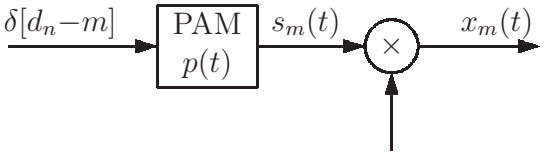
$$x_m(t) = \sum_{n=-\infty}^{\infty} \delta[d_n - m]\, p_m(t - nT_B)\,, \quad m = 0, 1, \ldots, M-1\,, \quad \delta[k] = \begin{cases} 1\,, & k = 0\,, \\ 0\,, & \text{otherwise}\,, \end{cases}$$

and added up to produce the FSK signal $x(t)$ as

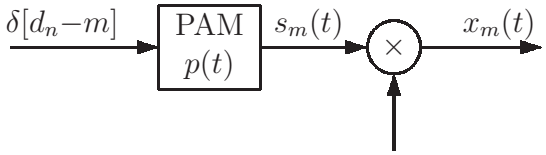
$$x(t) = \sum_{m=0}^{M-1} x_m(t) = \sum_{m=0}^{M-1} \sum_{n=-\infty}^{\infty} \delta[d_n - m]\, p_m(t - nT_B)\,.$$

Depending on whether the individual OOK signals are coherent or non-coherent, one can distinguish between **coherent FSK** and **non-coherent FSK**. The blockdiagrams below show how to implement the individual bandpass PAM functions $p_m(t)$, $m = 0, 1, \ldots, M-1$, in each case.

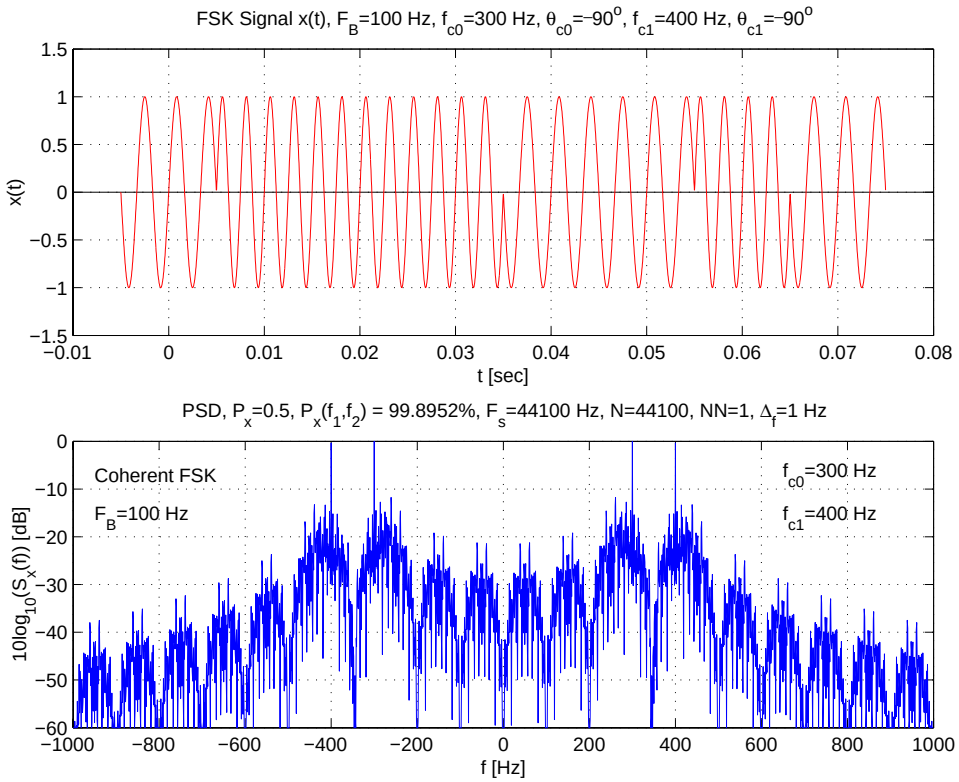
PAM $p_m(t)$ for Coherent FSK



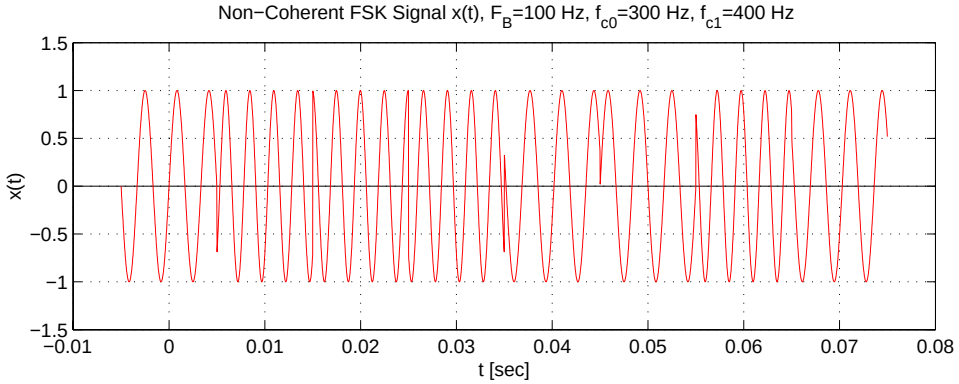
PAM $p_m(t)$ for Non-Coherent FSK



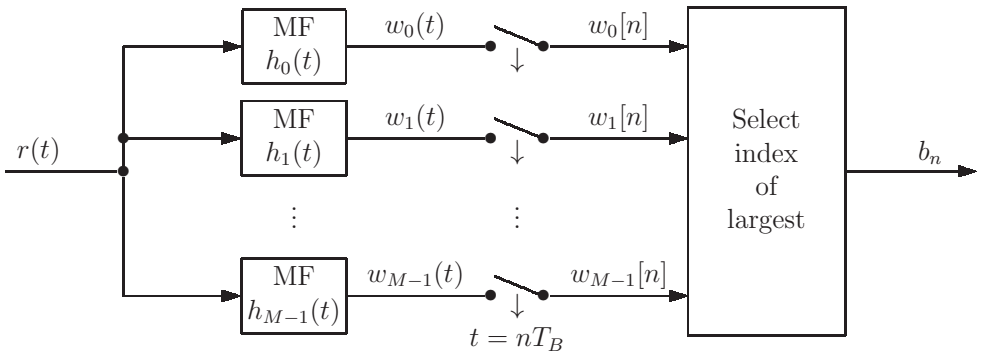
Thus, for coherent FSK $p_m(t) = A_c\, p(t)\, \cos(2\pi f_{cm} t + \theta_{cm})$, and for non-coherent FSK $p_m(t) = A_c\, p(t)\, \cos(2\pi f_{cm} t + \theta_{cm}[n])$. Note that, when the FSK signal is made up from the outputs of individual oscillators, $x(t)$ may contain phase jumps at the symbol boundaries, as can be seen in the following graph that shows an example of coherent binary FSK with rectangular $p(t)$, $F_B = 100$ baud, $f_{c1} = 300$ Hz, $\theta_{c1} = -90°$, $f_{c2} = 400$ Hz, $\theta_{c2} = -90°$.



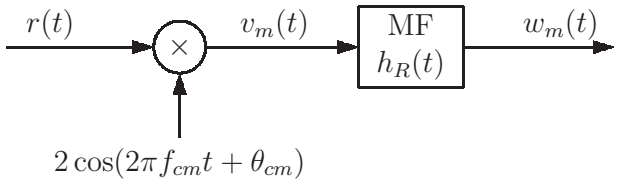
The upper graph shows $x(t)$ in the time domain for $\{d_n\} = \{0, 1, 1, 1, 0, 0, 1, 0\}$, and the lower graph shows the PSD $S_x(f)$ when d_n is a random binary signal with equally likely 0's and 1's. The next graph shows $x(t)$ for the same data and parameters, but using a non-coherent FSK transmitter.



The block diagram below shows the basic structure of a receiver for M-ary FSK.



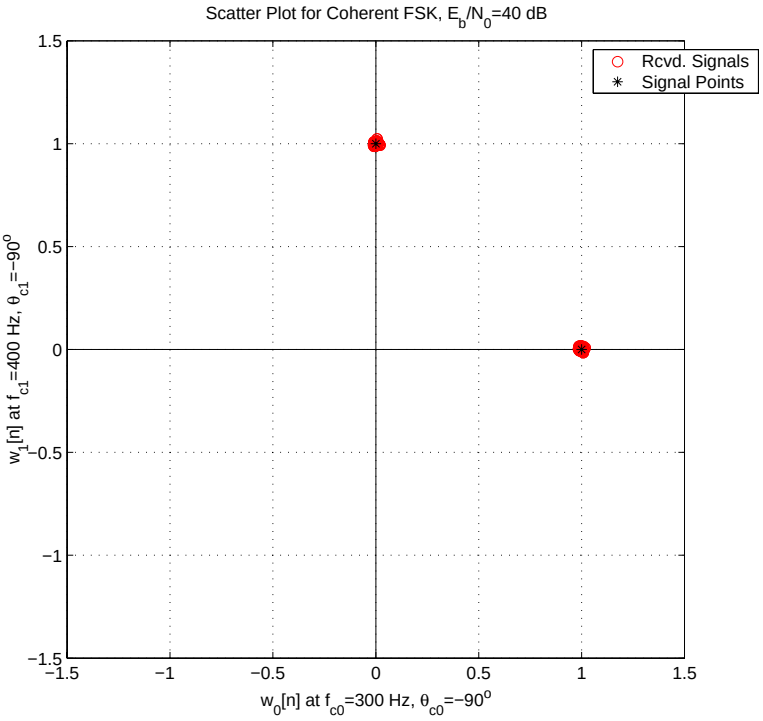
There is a matched filter at each of the M FSK frequencies $f_{c0}, \ldots, f_{cM-1}$. Coherent FSK can be received with coherent matched filters. One way to implement a coherent MF when f_{cm} and θ_{cm} are fixed and known is shown in the following blockdiagram.



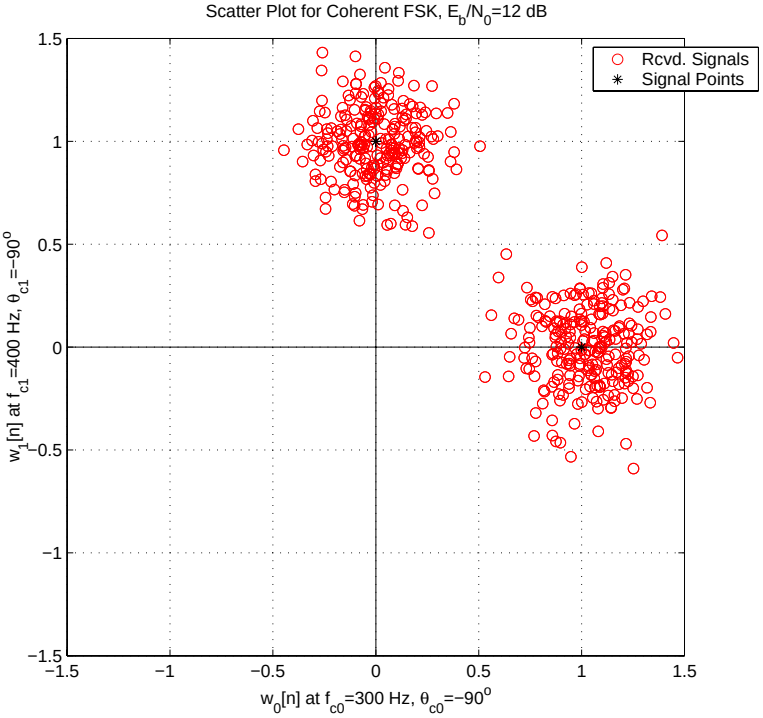
An interesting question is how close together any two adjacent frequencies, say $f_{c\ell}$ and $f_{cm} = f_{c\ell} + \Delta_f$, should be chosen. By making Δ_f small, the total bandwidth used by M-ary FSK can be reduced, but if Δ_f is too small, then the normalized distance (normalized wrt average energy per bit) between symbols using different frequencies becomes too small and thus the probability of error increases. As a general criterion, one usually tries to make all M symbols in M-ary FSK **orthogonal**, i.e., assuming $p(t)$ is a rectangular pulse of width T_B, one selects Δ_f in $f_{cm} = f_{c\ell} + \Delta_f$ such that

$$\int_{(n-1/2)T_B}^{(n+1/2)T_B} \cos(2\pi f_{c\ell}t + \theta_{c\ell})\, \cos(2\pi f_{cm}t + \theta_{cm})\, dt = 0\,, \qquad \text{for all integers } n.$$

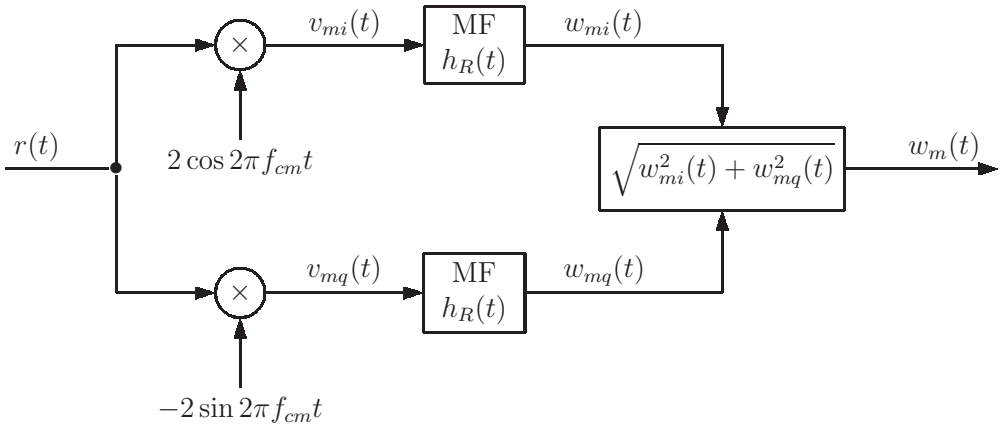
The scatter or constellation plot in the following figure shows $w_0[n]$ versus $w_1[n]$ for coherent binary FSK satisfying the orthogonality condition.



For the graph above a FSK signal with high SNR (40 dB) was used so that the nominal signal points (black *) and the received signal points (red o) coincide. The graph below shows how this changes when the SNR is reduced to $E_b/\mathcal{N}_0 = 12$ dB.

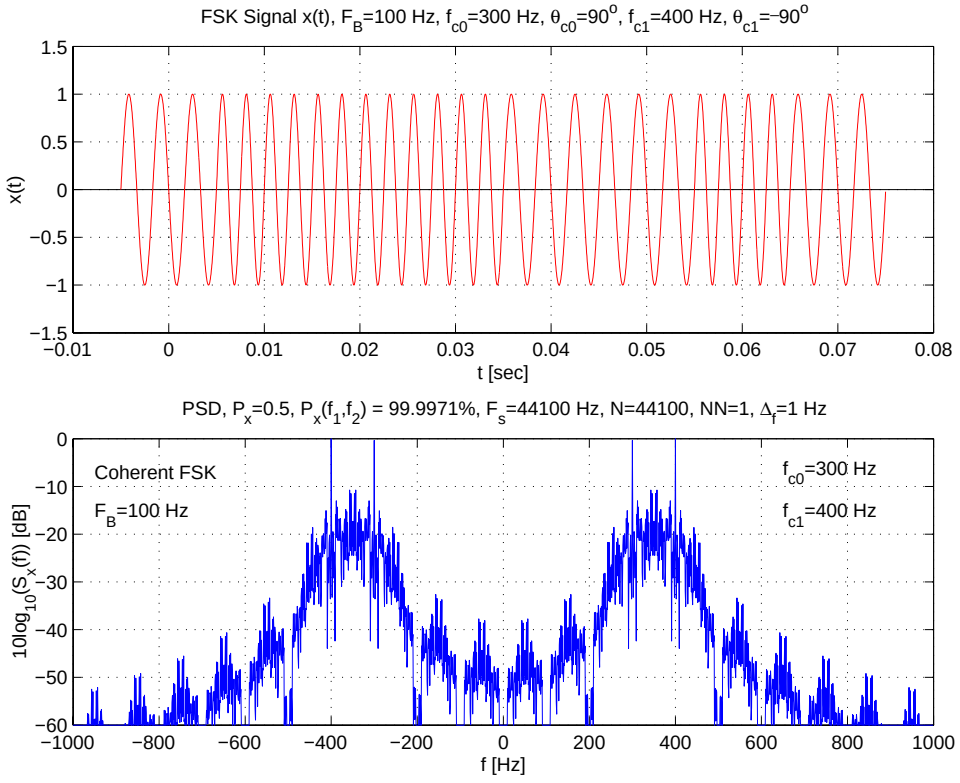


If the carrier phase is random and/or unknown to the receiver, then matched filter envelope detectors (MFED) of the form shown in the blockdiagram below need to be used.



In this case, a scatter plot of binary FSK would have to be drawn in 4-dimensional space. Looking at just $w_0 i[n]$ versus $w_0 q[n]$, or $w_1 i[n]$ versus $w_1 q[n]$ yields signal "points" that either lie at the origin or on a circle, centered at the origin, in the same way as for noncoherent OOK.

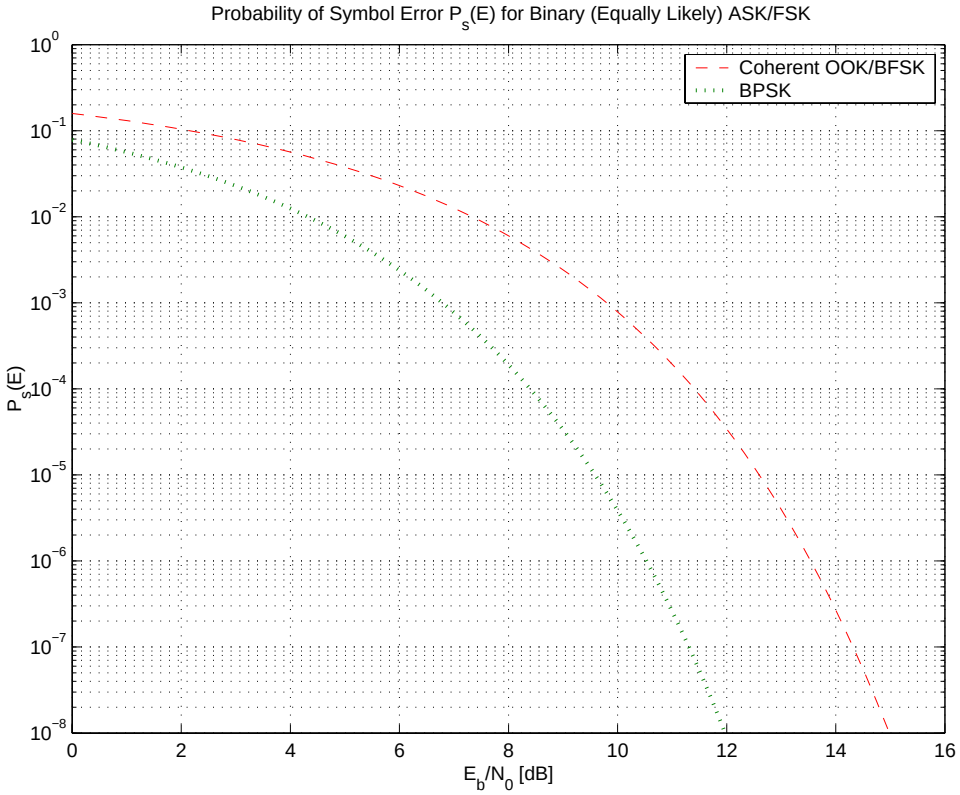
One more version of FSK that is actually preferred in practice is **continuous phase FSK (CPFSK)**. This can be obtained from coherent FSK by choosing the frequencies f_{cm} and the phases θ_{cm} in such a way that phase jumps are avoided at the symbol boundaries. An example of binary CPFSK and its PSD is shown in the following two graphs.



As can be seen in the lower graph, the absence of phase jumps in $x(t)$ yields a PSD $S_x(f)$ that decreases rapidly for frequencies below f_{c0} and above f_{cM-1} (in the M-ary case), which reduces interference between communication systems that use adjacent bands.

1.3 Probability of Error

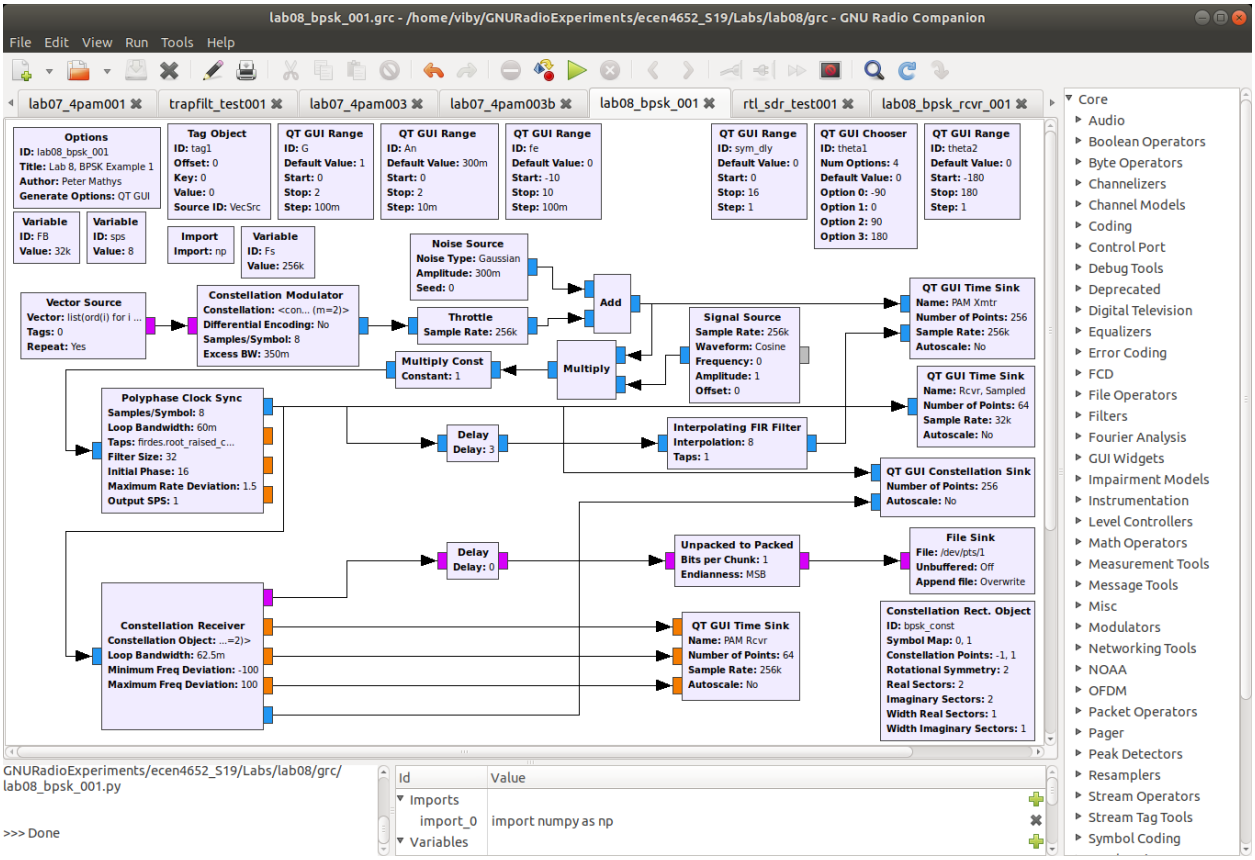
The graph below shows the probability $P_s(\mathcal{E})$ of error for OOK, BPSK and BFSK when coherent receivers are used.

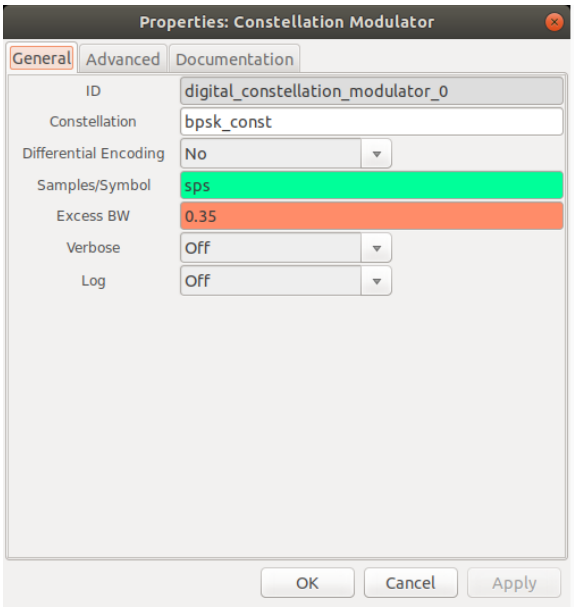


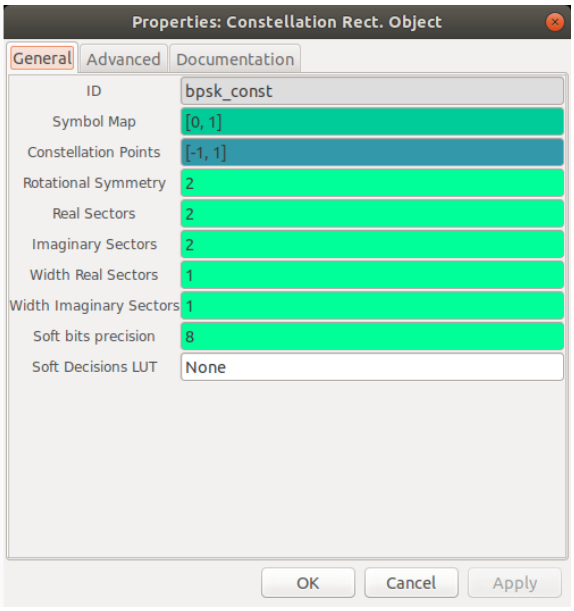
In general a non-coherent receiver will have a larger $P_s(\mathcal{E})$ for a given $E_b/\mathcal{N}_0$. The reason for this is that a coherent OOK/ASK receiver at some carrier frequency f_c (or f_{ci} for the i-th FSK frequency) only needs to look at the in-phase component of the demodulation, whereas a non-coherent receiver needs to look at both the in-phase and the quadrature components. Thus, a coherent receiver only picks up noise from the in-phase channel, whereas a non-coherent receiver picks up noise from both the in-phase and the quadrature channels.

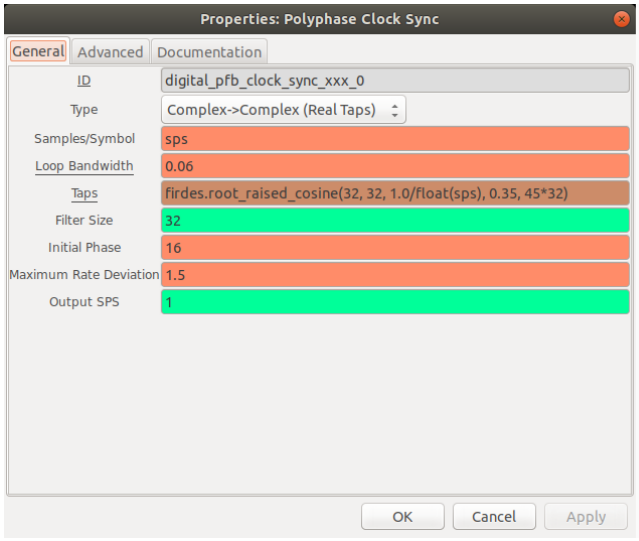
1.4 BPSK and QPSK in GNU Radio

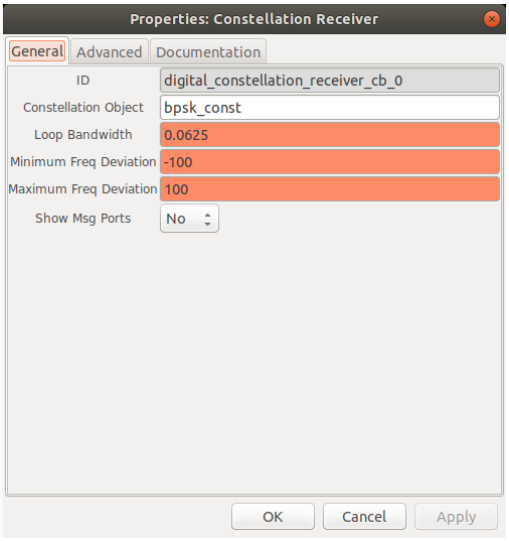
The flowgraph of a "loopback" BPSK communication system in GNU radio is shown in the figure below.

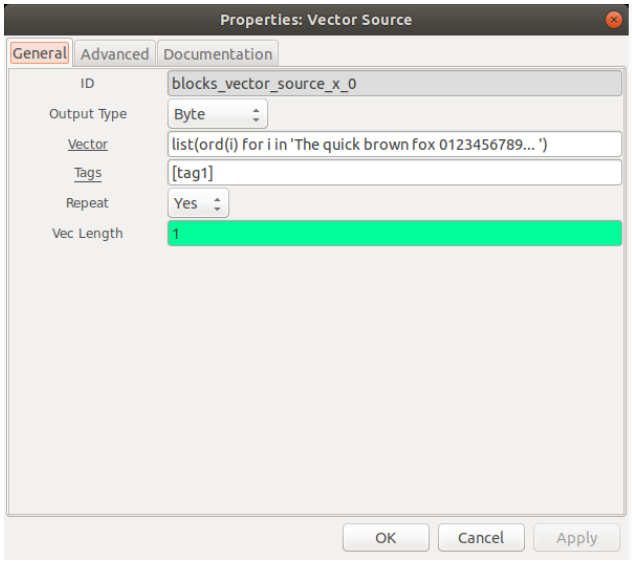

lab08_bpsk_001.grc - /home/viby/GNURadioExperiments/ecen4652_S19/Labs/lab08/grc - GNU Radio Companion
File Edit View Run Tools Help
lab07_4pam001
trapfilt_test001
lab07_4pam003
lab07_4pam003b
lab08_bpsk_001
rtl_sdr_test001
lab08_bpsk_rcvr_001
Options
Tag Object
QT GUI Range
QT GUI Chooser
Variable
Import
Noise Source
Vector Source
Constellation Modulator
Throttle
Add
Multiply
Multiply Const
Signal Source
QT GUI Time Sink
Polyphase Clock Sync
Delay
Interpolating FIR Filter
QT GUI Constellation Sink
Unpacked to Packed
File Sink
Constellation Receiver
Constellation Rect. Object
Core
Audio
Boolean Operators
Byte Operators
Channelizers
Channel Models
Coding
Control Port
Debug Tools
Deprecated
Digital Television
Equalizers
Error Coding
FCD
File Operators
Filters
Fourier Analysis
GUI Widgets
Impairment Models
Instrumentation
Level Controllers
Math Operators
Measurement Tools
Message Tools
Misc
Modulators
Networking Tools
NOAA
OFDM
Packet Operators
Pager
Peak Detectors
Resamplers
Stream Operators
Stream Tag Tools
Symbol Coding
GNURadioExperiments/ecen4652_S19/Labs/lab08/grc/
lab08_bpsk_001.py
>>> Done
Id
Value
Imports
import_0
import numpy as np
Variables

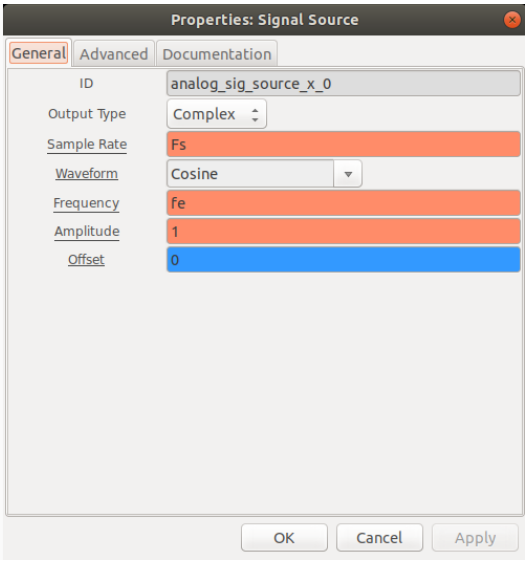

Properties: Constellation Modulator
General
Advanced
Documentation
ID
digital_constellation_modulator_0
Constellation
bpsk_const
Differential Encoding
No
Samples/Symbol
sps
Excess BW
0.35
Verbose
Off
Log
Off
OK
Cancel
Apply

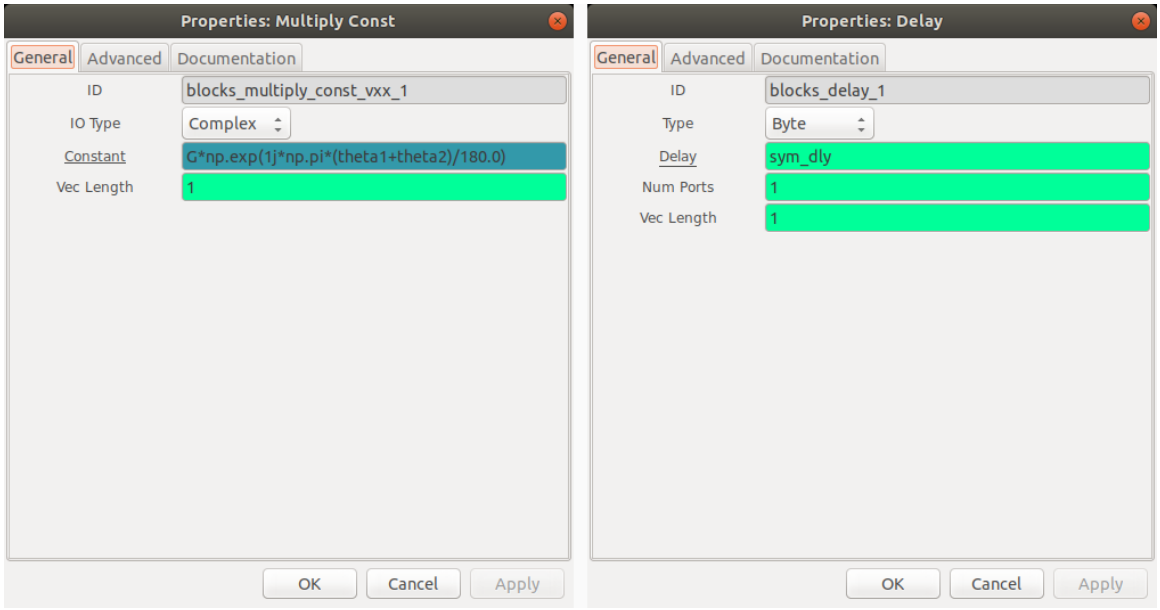

Properties: Constellation Rect. Object
General
Advanced
Documentation
ID
bpsk_const
Symbol Map
[0, 1]
Constellation Points
[-1, 1]
Rotational Symmetry
2
Real Sectors
2
Imaginary Sectors
2
Width Real Sectors
1
Width Imaginary Sectors
1
Soft bits precision
8
Soft Decisions LUT
None
OK
Cancel
Apply

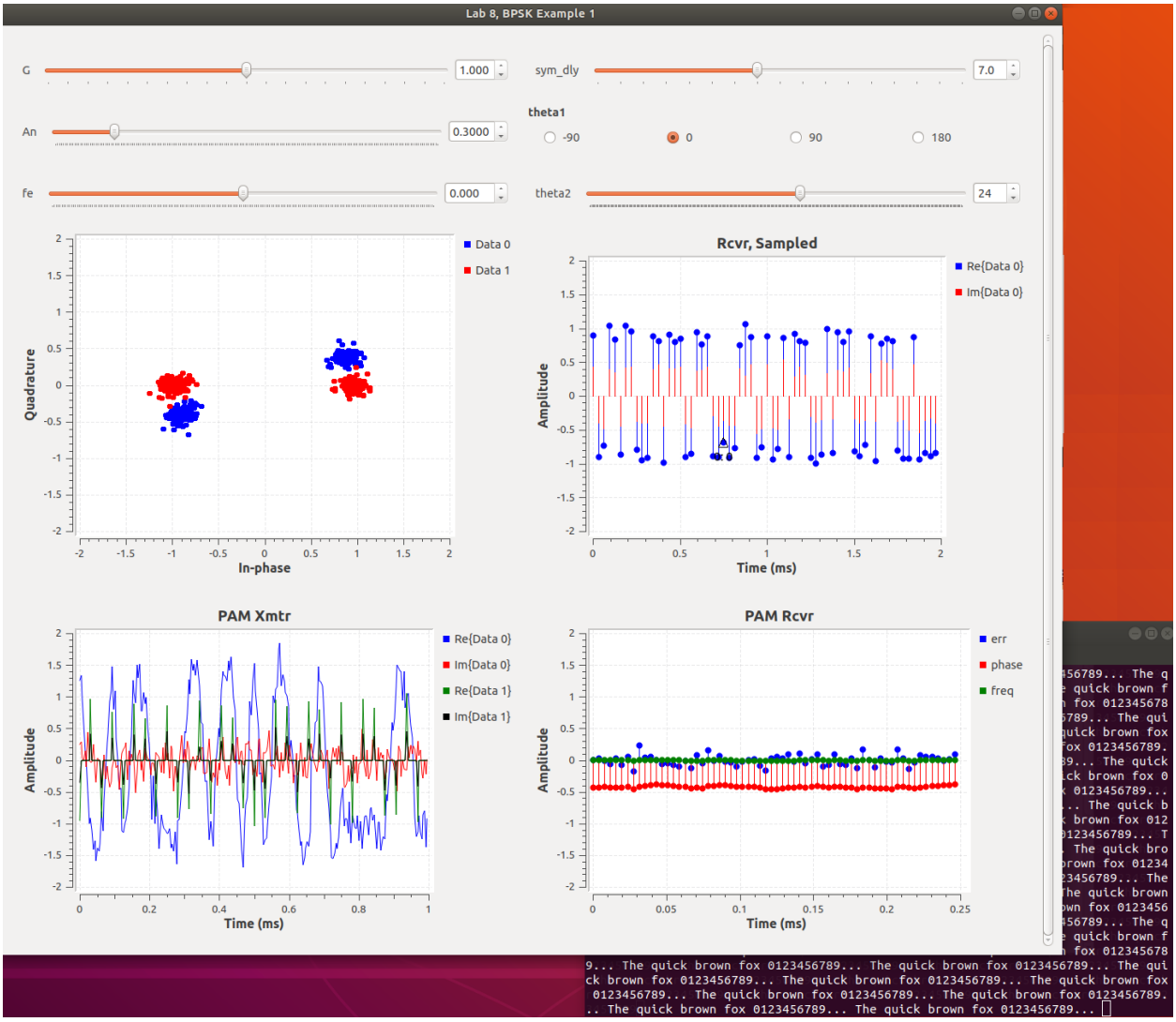

Properties: Polyphase Clock Sync
General
Advanced
Documentation
ID
digital_pfb_clock_sync_xxx_0
Type
Complex->Complex (Real Taps)
Samples/Symbol
sps
Loop Bandwidth
0.06
Taps
firdes.root_raised_cosine(32, 32, 1.0/float(sps), 0.35, 45*32)
Filter Size
32
Initial Phase
16
Maximum Rate Deviation
1.5
Output SPS
1
OK
Cancel
Apply


Properties: Constellation Receiver
General
Advanced
Documentation
ID
digital_constellation_receiver_cb_0
Constellation Object
bpsk_const
Loop Bandwidth
0.0625
Minimum Freq Deviation
-100
Maximum Freq Deviation
100
Show Msg Ports
No
OK
Cancel
Apply

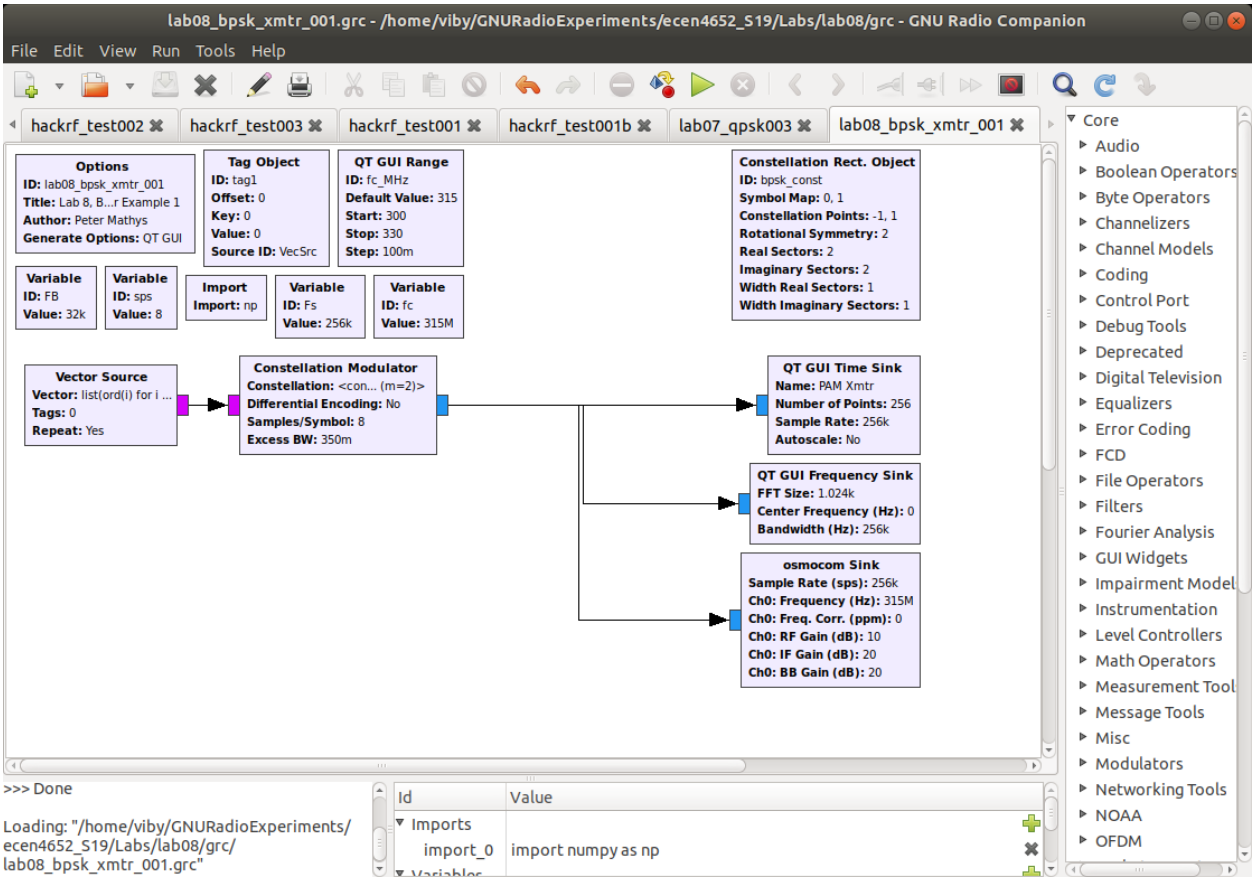

Properties: Vector Source
General
Advanced
Documentation
ID
blocks_vector_source_x_0
Output Type
Byte
Vector
list(ord(i) for i in 'The quick brown fox 0123456789... ')
Tags
[tag1]
Repeat
Yes
Vec Length
1
OK
Cancel
Apply

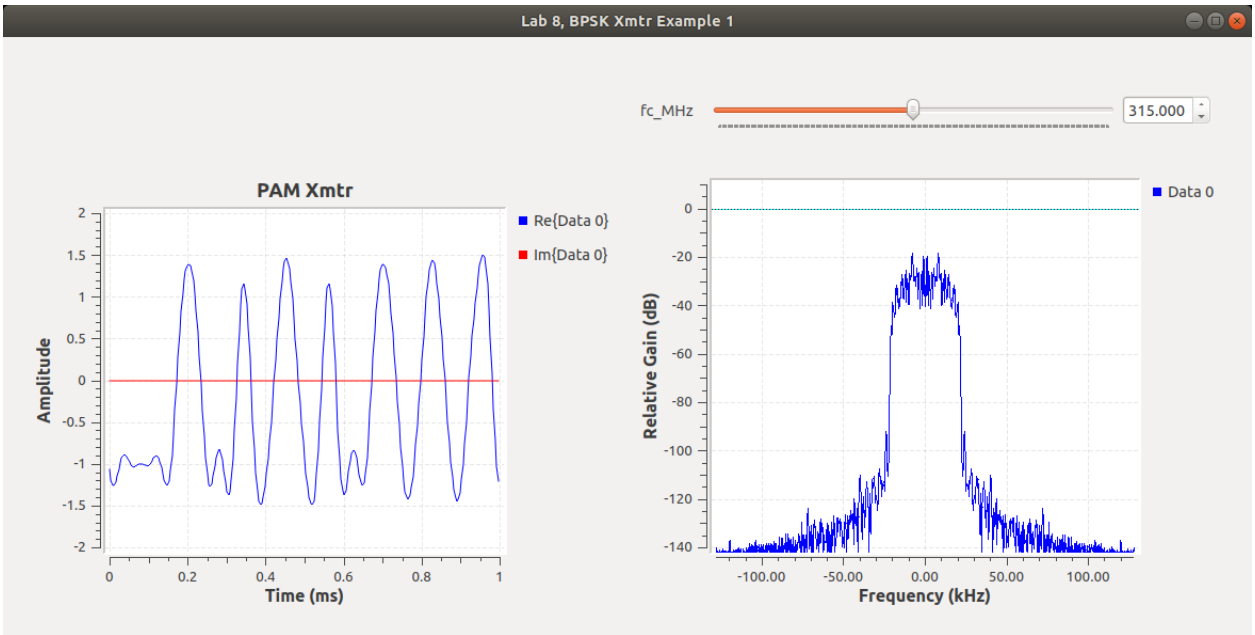

Properties: Signal Source
General
Advanced
Documentation
ID
analog_sig_source_x_0
Output Type
Complex
Sample Rate
Fs
Waveform
Cosine
Frequency
fe
Amplitude
1
Offset
0
OK
Cancel
Apply


Properties: Multiply Const
General
Advanced
Documentation
ID
blocks_multiply_const_vxx_1
IO Type
Complex
Constant
G*np.exp(1j*np.pi*(theta1+theta2)/180.0)
Vec Length
1
OK
Cancel
Apply
Properties: Delay
General
Advanced
Documentation
ID
blocks_delay_1
Type
Byte
Delay
sym_dly
Num Ports
1
Vec Length
1
OK
Cancel
Apply

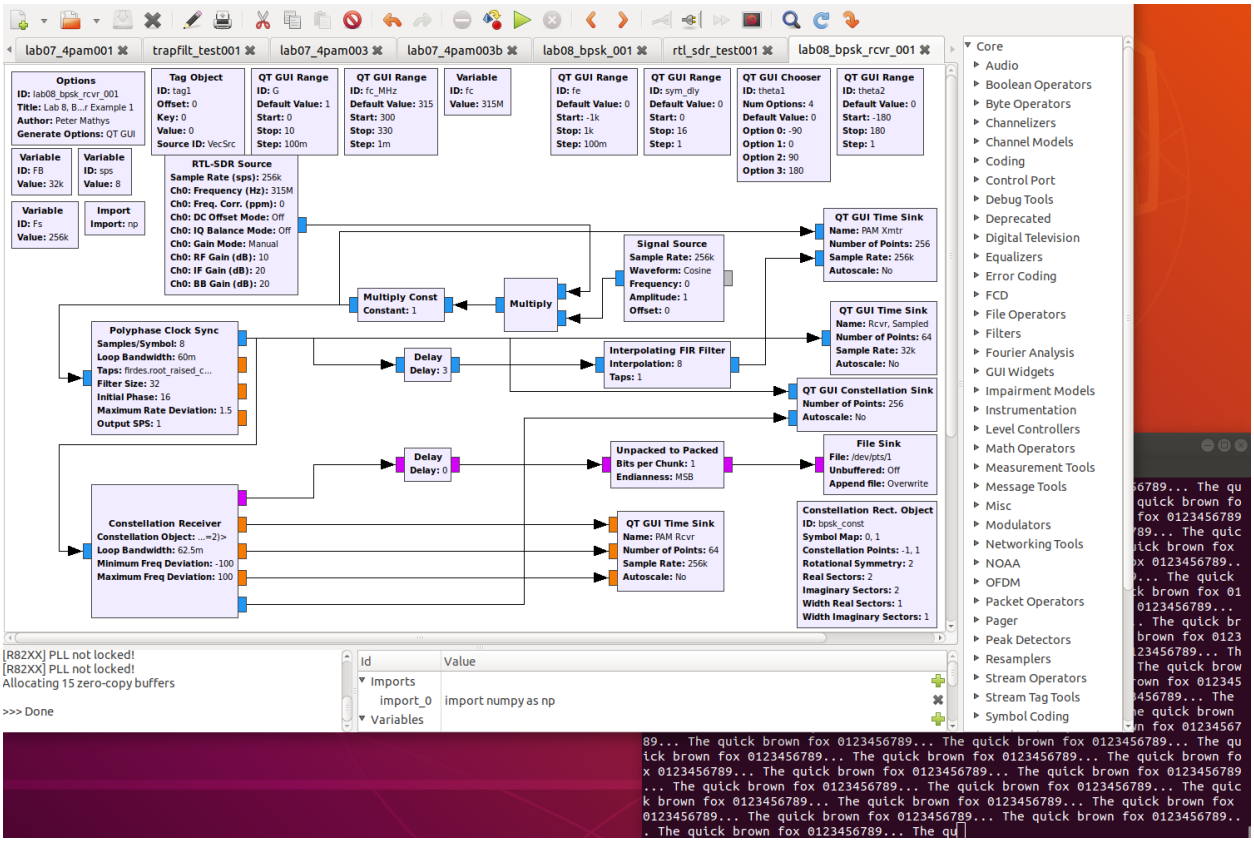


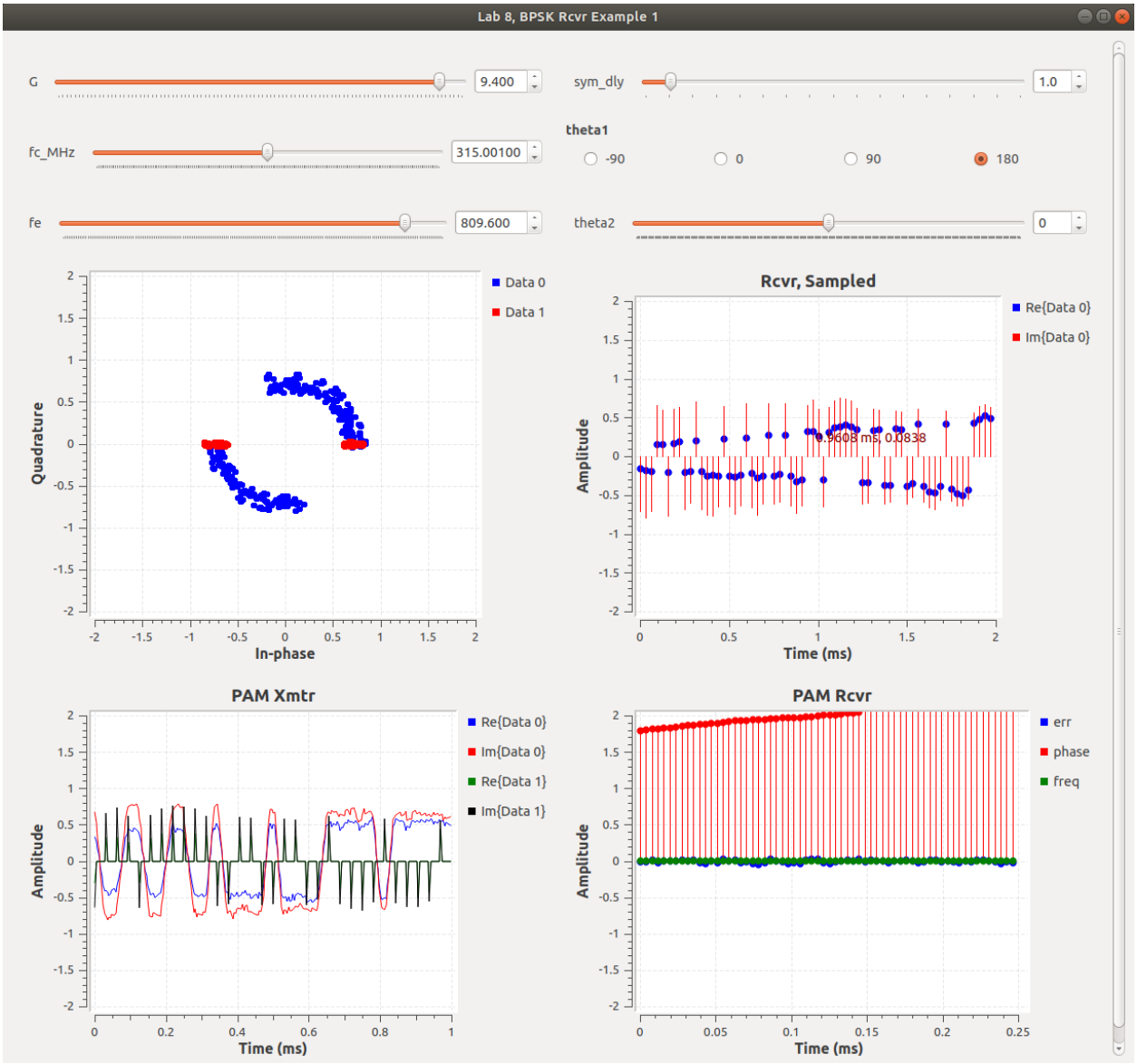
For free-space transmission we separate the transmitter and receiver parts. For the transmitter (using a HackRF One SDR) we use a `osmocom Sink` block (requires that the gr-osmosdr out-of-tree module is installed in GNU Radio).





For the receiver (using a rtl-sdr DVB-T USB dongle) we use a `RTL-SDR Source` block (from the gr-osmosdr module).

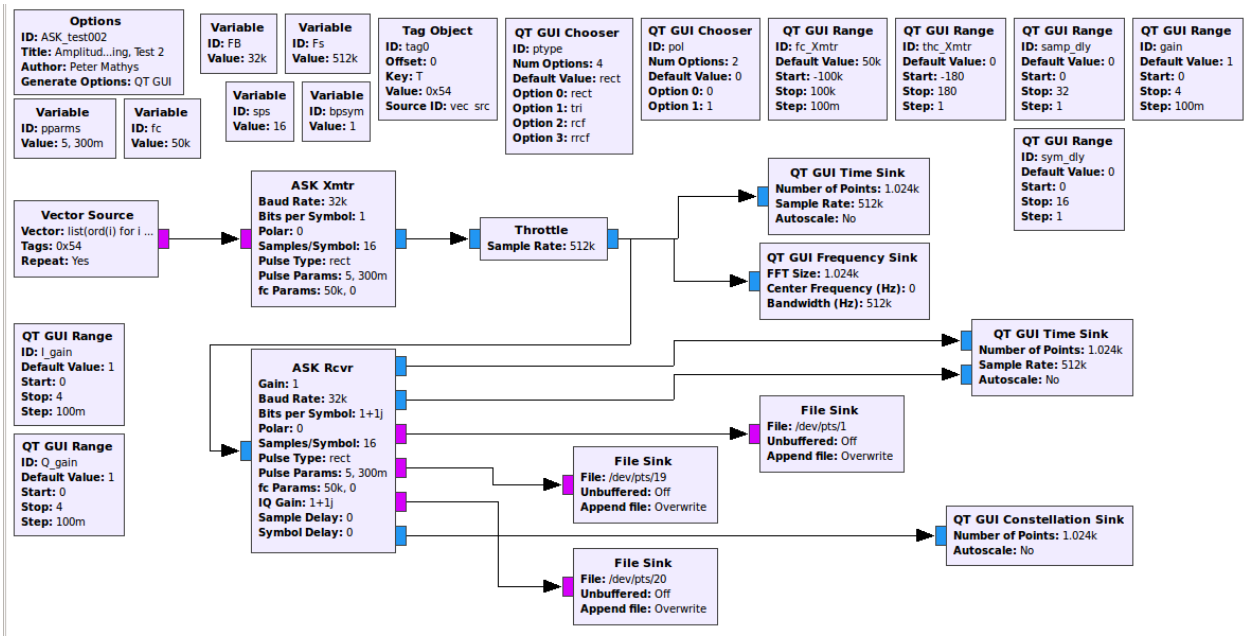

lab07_4pam001
trapfilt_test001
lab07_4pam003
lab07_4pam003b
lab08_bpsk_001
rtl_sdr_test001
lab08_bpsk_rcvr_001
Options
Tag Object
QT GUI Range
QT GUI Range
Variable
QT GUI Range
QT GUI Range
QT GUI Chooser
QT GUI Range
RTL-SDR Source
Multiply Const
Multiply
Signal Source
QT GUI Time Sink
QT GUI Time Sink
Polyphase Clock Sync
Delay
Interpolating FIR Filter
QT GUI Constellation Sink
File Sink
Unpacked to Packed
Constellation Receiver
QT GUI Time Sink
Constellation Rect. Object
[R82XX] PLL not locked!
[R82XX] PLL not locked!
Allocating 15 zero-copy buffers
>>> Done
Id
Value
Imports
import_0
import numpy as np
Variables
Core
Audio
Boolean Operators
Byte Operators
Channelizers
Channel Models
Coding
Control Port
Debug Tools
Deprecated
Digital Television
Equalizers
Error Coding
FCD
File Operators
Filters
Fourier Analysis
GUI Widgets
Impairment Models
Instrumentation
Level Controllers
Math Operators
Measurement Tools
Message Tools
Misc
Modulators
Networking Tools
NOAA
OFDM
Packet Operators
Pager
Peak Detectors
Resamplers
Stream Operators
Stream Tag Tools
Symbol Coding

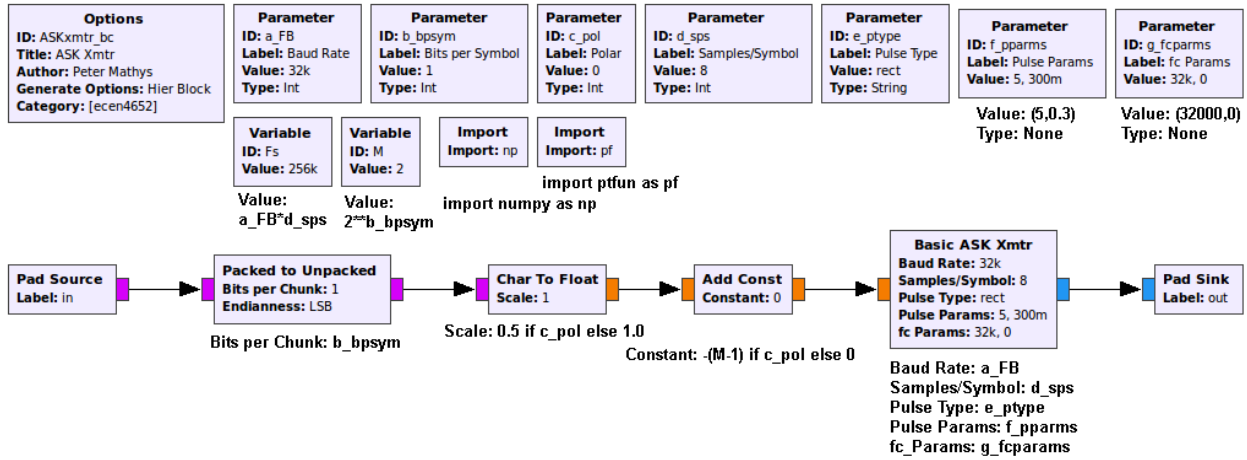


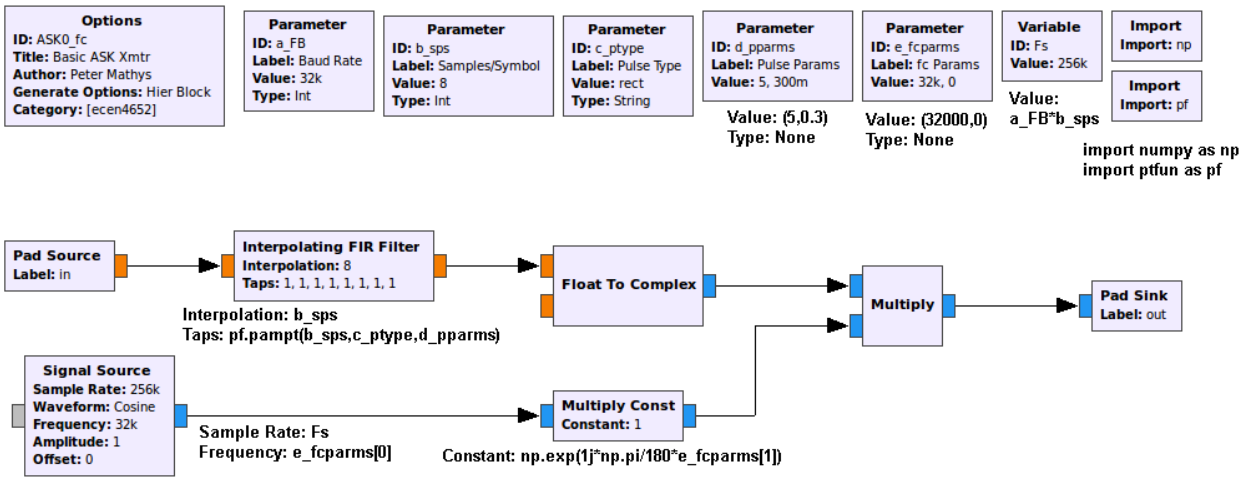
1.5 Digital Modulation in GNU Radio

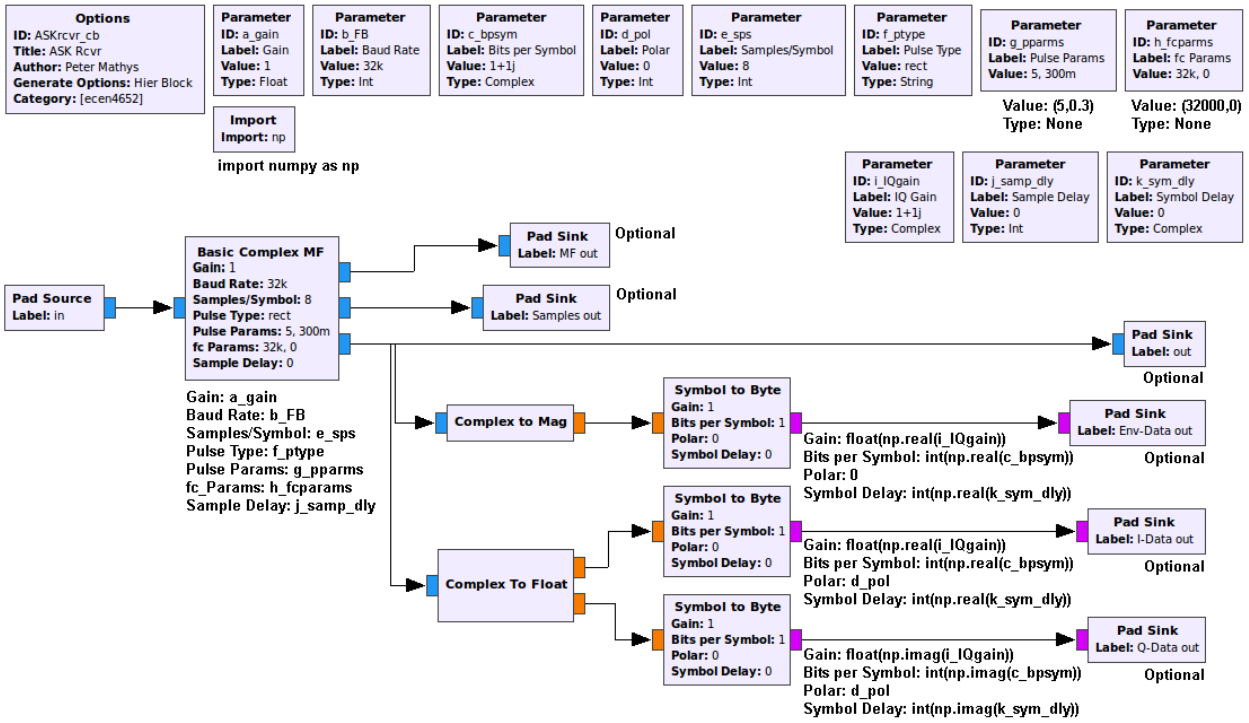
The GNU Radio flowgraph of a digital communication system made up from the basic ASK or FSK components introduced in this lab gets crowded very quickly. Luckily, the GNU Radio Companion has a feature that allows us to build our own Hierarchical Blocks that can be used in a flowgraph like the blocks that come together with GNU Radio. Thus, we can build a basic ASK transmitter (hierarchical) block that takes in a sequence a_n, converts it to a PAM signal with PAM pulse $p(t)$ and then puts the result on a carrier with frequency f_c and phase θ_c by multiplying with $e^{j2\pi f_c t}\, e^{j\theta_c}$. This basic ASK transmitter can then be used in another hierarchical block to define a ASK transmitter that accepts ASCII code (bytes)

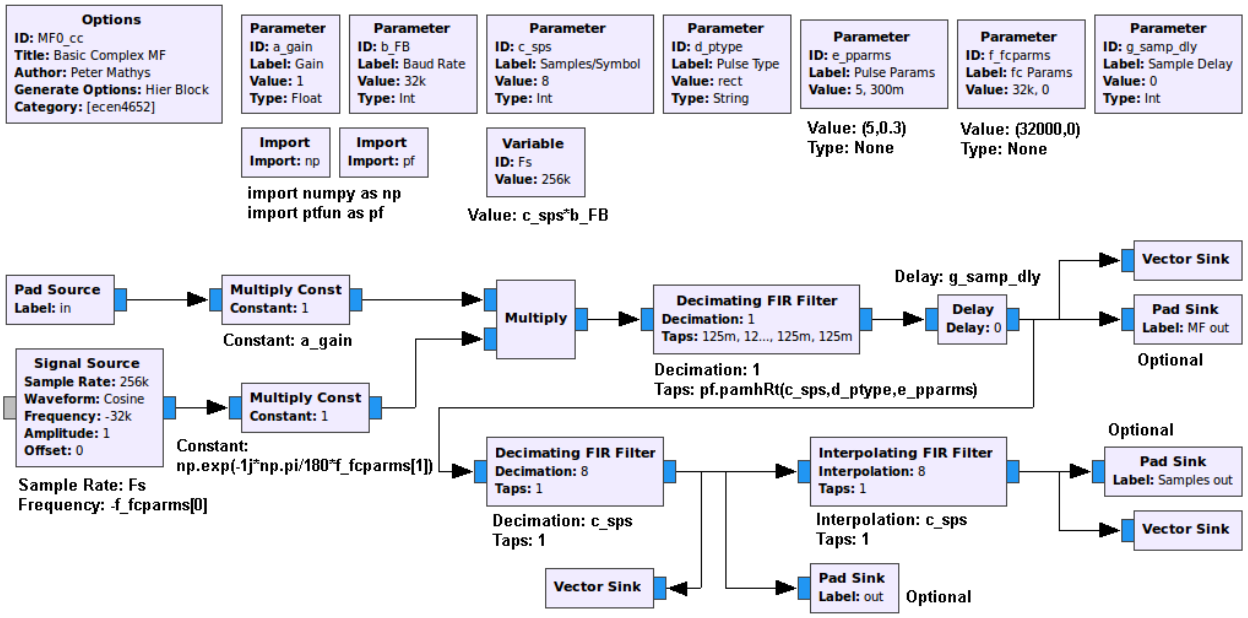
at the input and produces an M-ary polar or unipolar ASK signal at the output. Such an ASK transmitter can then be used, together with an ASK receiver that is similarly built from hierarchical blocks, to make a complete ASK transmission system like the one shown in the flowgraph below.

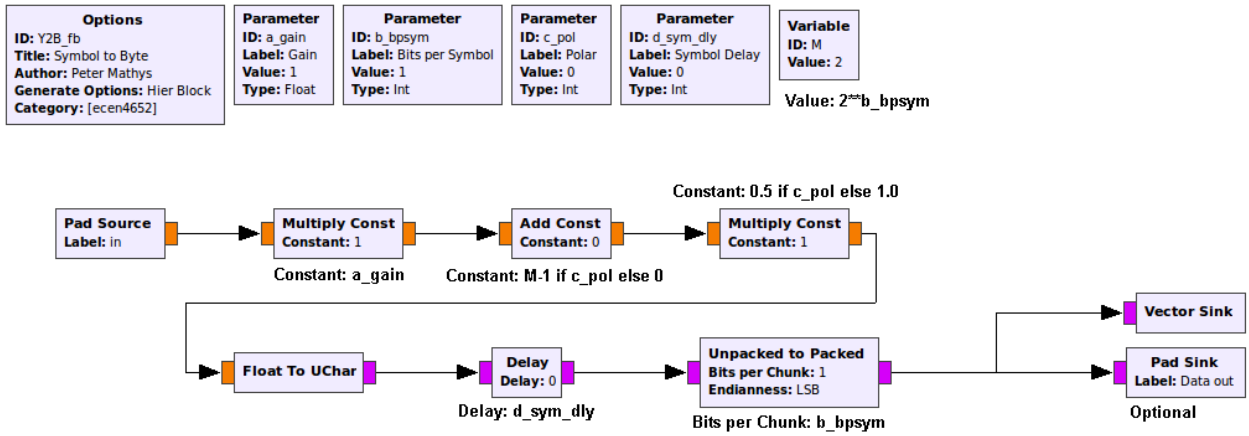





Options
ID: ASK0_fc
Title: Basic ASK Xmtr
Author: Peter Mathys
Generate Options: Hier Block
Category: [ecen4652]
Parameter
ID: a_FB
Label: Baud Rate
Value: 32k
Type: Int
Parameter
ID: b_sps
Label: Samples/Symbol
Value: 8
Type: Int
Parameter
ID: c_ptype
Label: Pulse Type
Value: rect
Type: String
Parameter
ID: d_pparms
Label: Pulse Params
Value: 5, 300m
Value: (5,0.3)
Type: None
Parameter
ID: e_fcparms
Label: fc Params
Value: 32k, 0
Value: (32000,0)
Type: None
Variable
ID: Fs
Value: 256k
Value:
a_FB*b_sps
Import
Import: np
Import
Import: pf
import numpy as np
import ptfun as pf
Pad Source
Label: in
Interpolating FIR Filter
Interpolation: 8
Taps: 1, 1, 1, 1, 1, 1, 1, 1
Interpolation: b_sps
Taps: pf.pampt(b_sps,c_ptype,d_pparms)
Float To Complex
Multiply
Pad Sink
Label: out
Signal Source
Sample Rate: 256k
Waveform: Cosine
Frequency: 32k
Amplitude: 1
Offset: 0
Sample Rate: Fs
Frequency: e_fcparms[0]
Multiply Const
Constant: 1
Constant: np.exp(1j*np.pi/180*e_fcparms[1])


Options
ID: ASKrcvr_cb
Title: ASK Rcvr
Author: Peter Mathys
Generate Options: Hier Block
Category: [ecen4652]
Parameter
ID: a_gain
Label: Gain
Value: 1
Type: Float
Parameter
ID: b_FB
Label: Baud Rate
Value: 32k
Type: Int
Parameter
ID: c_bpsym
Label: Bits per Symbol
Value: 1+1j
Type: Complex
Parameter
ID: d_pol
Label: Polar
Value: 0
Type: Int
Parameter
ID: e_sps
Label: Samples/Symbol
Value: 8
Type: Int
Parameter
ID: f_ptype
Label: Pulse Type
Value: rect
Type: String
Parameter
ID: g_pparms
Label: Pulse Params
Value: 5, 300m
Value: (5,0.3)
Type: None
Parameter
ID: h_fcparms
Label: fc Params
Value: 32k, 0
Value: (32000,0)
Type: None
Import
Import: np
import numpy as np
Parameter
ID: i_IQgain
Label: IQ Gain
Value: 1+1j
Type: Complex
Parameter
ID: j_samp_dly
Label: Sample Delay
Value: 0
Type: Int
Parameter
ID: k_sym_dly
Label: Symbol Delay
Value: 0
Type: Complex
Pad Source
Label: in
Basic Complex MF
Gain: 1
Baud Rate: 32k
Samples/Symbol: 8
Pulse Type: rect
Pulse Params: 5, 300m
fc Params: 32k, 0
Sample Delay: 0
Gain: a_gain
Baud Rate: b_FB
Samples/Symbol: e_sps
Pulse Type: f_ptype
Pulse Params: g_pparms
fc_Params: h_fcparms
Sample Delay: j_samp_dly
Pad Sink
Label: MF out
Optional
Pad Sink
Label: Samples out
Optional
Pad Sink
Label: out
Optional
Complex to Mag
Symbol to Byte
Gain: 1
Bits per Symbol: 1
Polar: 0
Symbol Delay: 0
Gain: float(np.real(i_IQgain))
Bits per Symbol: int(np.real(c_bpsym))
Polar: 0
Symbol Delay: int(np.real(k_sym_dly))
Pad Sink
Label: Env-Data out
Optional
Complex To Float
Symbol to Byte
Gain: 1
Bits per Symbol: 1
Polar: 0
Symbol Delay: 0
Gain: float(np.real(i_IQgain))
Bits per Symbol: int(np.real(c_bpsym))
Polar: d_pol
Symbol Delay: int(np.real(k_sym_dly))
Pad Sink
Label: I-Data out
Optional
Symbol to Byte
Gain: 1
Bits per Symbol: 1
Polar: 0
Symbol Delay: 0
Gain: float(np.imag(i_IQgain))
Bits per Symbol: int(np.imag(c_bpsym))
Polar: d_pol
Symbol Delay: int(np.imag(k_sym_dly))
Pad Sink
Label: Q-Data out
Optional





– To be completed –

2 Lab Experiments

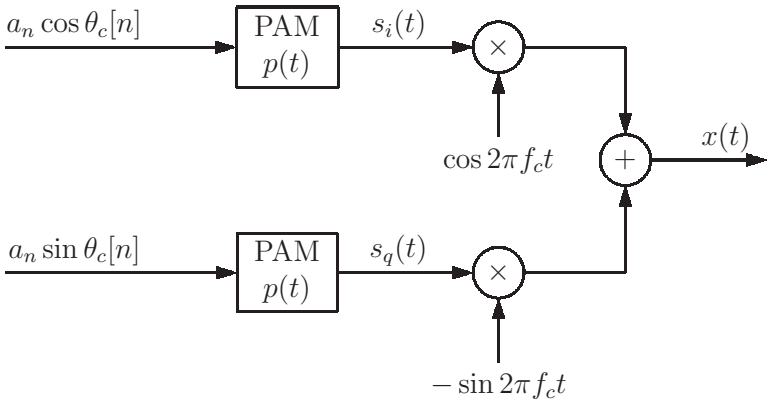
E1. Amplitude Shift Keying. **(a)** Start a new Python module called `keyfun.py` and use your **pam12** function from the `pamfun` module as a building block to complete the ASK transmitter function `askxmtr` whose header is shown below.

```
# File: keyfun.py
# Functions for amplitude/frequency/phase shift keying
# ASK, FSK, PSK and hybrid APSK
import numpy as np
import comsig
import pamfun
import filtfun

def askxmtr(sig_an,Fs,ptype,pparms,xtype,fcparms):
    """
    Amplitude Shift Keying (ASK) Transmitter for
    Coherent ('coh') and Non-coherent ('noncoh') ASK Signals
    >>>>> sig_xt,sig_st = askxmtr(sig_an,Fs,ptype,pparms,xtype,fcparms) <<<<<
    where  sig_xt: waveform from class sigWave
           sig_xt.signal():   transmitted ASK signal, sampling rate Fs
                    x(t) = s(t)*cos(2*pi*fc*t+(pi/180)*thetac)
           sig_xt.timeAxis(): time axis for x(t), starts at t=-TB/2
           sig_st: waveform from class sigWave
           sig_st.signal(): baseband PAM signal s(t) for 'coh'
           sig_st.signal(): st = sit + 1j*sqt  for 'noncoh'
           sit:     PAM signal of an*cos(pi/180*thetacn)
           sqt:     PAM signal of an*sin(pi/180*thetacn)
           xtype:   Transmitter type from list {'coh','noncoh'}
           sig_an: sequence from class sigSequ
           sig_an.signal() = [an]            for {'coh'}
           sig_an.signal() = [[an],[thetacn]]    for {'noncoh'}
           an:      N-symbol DT input sequence a_n, 0<=n<N
           thetacn: N-symbol DT sequence theta_c[n] in degrees,
                    used instead of thetac for {'noncoh'} ASK
           sig_an.get_FB(): baud rate of a_n (and theta_c[n]), TB=1/FB
           Fs:      sampling rate of x(t), s(t)
           ptype:   pulse type from list
                    ['man','rcf','rect','rrcf','sinc','tri']
           pparms = []   for {'man','rect','tri'}
           pparms = [k, alpha]  for {'rcf','rrcf'}
           pparms = [k, beta]   for {'sinc'}
           k:       "tail" truncation parameter for {'rcf','rrcf','sinc'}
                    (truncates at -k*TB and k*TB)
           alpha:   Rolloff parameter for {'rcf','rrcf'}, 0<=alpha<=1
           beta:    Kaiser window parameter for {'sinc'}
           fcparms = [fc, thetac]  for {'coh'}
           fcparms = [fc]          for {'noncoh'}
           fc:      carrier frequency in Hz
           thetac:  carrier phase in deg (0: cos, -90: sin)
    """
```

To generate noncoherent ASK signals with a random carrier sequence $\theta_c[n]$, use in-phase

and quadrature PAM signals $s_i(t)$ and $s_q(t)$ and QAM modulation as shown in the following blockdiagram.



Test your transmitter by generating a short (about 10 symbol times) random coherent OOK signal and a short random noncoherent OOK signal and displaying them in the time domain. Use $F_s = 44100$ Hz, $F_B = 100$ baud, $f_c = 300$ Hz, and a rectangular pulse $p(t)$ of width $T_B = 1/F_B$. Generate a uniformly distributed random phase for noncoherent OOK using the `rand` function.

(b) In the `keyfun.py` module, complete the following ASK receiver function, called `askrcvr`. The goal is to be able to use it to receive coherent and noncoherent ASK signals, and to make scatter plots.

```
def askrcvr(sig_rt,rtype,fcparms,FBparms,ptype,pparms):
    """
    Amplitude Shift Keying (ASK) Receiver for
    Coherent ('coh') and Non-coherent ('noncoh') ASK Signals
    >>>>> sig_bn,sig_bt,sig_wt,ixn =
                        askrcvr(sig_rt,rtype,fcparms,FBparms,ptype,pparms) <<<<<
    where  sig_bn: sequence from class sigSequ
           sig_bn.signal(): received DT sequence b[n]
           sig_bt: waveform from class sigWave
           sig_bt.signal(): received 'CT' PAM signal b(t)
           sig_wt: waveform from class sigWave
           sig_wt.signal(): wt = wit + 1j*wqt
           wit:      in-phase component of b(t)
           wqt:      quadrature component of b(t)
           ixn:      sampling time indexes for b(t)->b[n], w(t)->w[n]
           sig_rt: waveform from class sigWave
           sig_rt.signal():   received (noisy) ASK signal r(t)
           sig_rt.timeAxis(): time axis for r(t)
           rtype:    receiver type from list ['coh','noncoh']
           fcparms = [fc, thetac]  for {'coh'}
           fcparms = [fc]          for {'noncoh'}
           fc:       carrier frequency in Hz
           thetac:   carrier phase in deg (0: cos, -90: sin)
           FBparms = [FB, dly]
           FB:       baud rate of PAM signal, TB=1/FB
           dly:      sampling delay for b(t)->b[n], fraction of TB
                     sampling times are t=n*TB+t0 where t0=dly*TB
           ptype:    pulse type from list
                     ['man','rcf','rect','rrcf','sinc','tri']
           pparms = []  for 'man','rect','tri'
           pparms = [k, alpha]  for {'rcf','rrcf'}
           pparms = [k, beta]   for {'sinc'}
           k:       "tail" truncation parameter for {'rcf','rrcf','sinc'}
                    (truncates at -k*TB and k*TB)
           alpha:   Rolloff parameter for {'rcf','rrcf'}, 0<=alpha<=1
           beta:    Kaiser window parameter for {'sinc'}
    """
```

Use the `pamrcvr10` function in module `pamfun` as part of the ASK receiver. Test `askrcvr` together with `askxmtr` using random (unipolar) binary data and the parameters given in part (a).

(c) Let $F_s = 44100$ Hz, $F_B = 100$ baud, $f_c = 2100$ Hz, and let $p(t)$ be a rectangular pulse of width T_B. Use random binary data to produce (i) a coherent OOK signal, (ii) a noncoherent OOK signal, and (iii) a BPSK signal, each of duration 2 sec. Plot and compare the PSDs for all three cases. Then use the `wt` and `wn=wt[ixn]` outputs of the `askrcvr` function to make scatter plots for the three signals and compare them. Are there any interesting spectral lines

if you look at the PSDs of the squared ASK signals? How do things change if you use a triangular pulse $p(t)$ (of total width $2T_B$ from $-T_B$ to $+T_B$) instead of the rectangular pulse?

(d) The wav files `asksig801.wav`, `asksig802.wav` and `asksig803.wav` are binary ASK signals which contain 8-bit, LSB-first, ASCII signals. Analyze the three signals and extract the text messages. **Hint:** The signals in the wav files have been scaled so that their amplitude (including noise) is less than or equal to one. Use eye diagrams and/or scatter plots to determine the proper threshold below which the receiver decides that a 0 was received and above which it decides that a 1 was received.

(e) In the GNU Radio Companion, build the "loopback" BPSK communication system flowgraph shown in the lab description. Use the `sym_dly` slider to to establish ASCII byte synchronization in the receiver until you see the "The quick brown fox 0123456789..." message in the receiver terminal. Experiment with different signal to noise ratios (by changing `An`), with different gains/attenuations between transmitter and receiver (by changing `G`), with different phase shifts between transmitter and receiver (by changing `theta1` for phase jumps and `theta2` for gradual phase changes), and with different error frequencies between transmitter and receiver (by changing `fe`). Describe the effects of each of the adjustments on the quality and the ability to receive the transmitted message signal.

(f) Free-space reception of transmitted BPSK signals. In the lab there is a wireless transmitter that transmits two BPSK signals, one at $f_c = 315.0$ MHz with $F_B = 32000$ baud and one in the vicinity of this f_c with a different baud rate. Both signals use 'rrcf' PAM pulses with $\alpha = 0.35$. Use an rtl-sdr receiver together with the GNU Radio flowgraph given in the lab description to receive the ASCII text in the two messages. A capture of the wireless signal for off-line processing is also available in the file `lab08_multi_rcvr_001f_rec001.bin`. The file was recorded with $F_s = 256$ kHz and $f_c = 315.0$ MHz.

E2. Frequency Shift Keying. (a) For the Python module `keyfun.py`, write a function called `fskxmtr` which implements an M-ary FSK transmitter for coherent and noncoherent FSK. The header of this function is shown below.

```
def fskxmtr(M,sig_dn,Fs,ptype,pparms,xtype,fcparms):
    """
    M-ary Frequency Shift Keying (FSK) Transmitter for
    Choherent ('coh') and Non-coherent ('noncoh') FSK Signals
    >>>>> sig_xt = fskxmtr(M,sig_dn,Fs,ptype,pparms,xtype,fcparms) <<<<<
    where  sig_xt: waveform from class sigWave
           sig_xt.signal():   transmitted FSK signal, sampling rate Fs
           sig_xt.timeAxis(): time axis for x(t), starts at t=-TB/2
           M:       number of distinct symbol values in d[n]
           xtype:   Transmitter type from set {'coh','noncoh'}
           sig_dn: sequence from class sigSequ
           sig_dn.signal() = [dn]           for ['coh']
           sig_dn.signal() = [[dn],[thetacn]]  for ['noncoh']
           dn:      M-ary (0,1,..,M-1) N-symbol DT input sequence d_n
           thetacn: N-symbol DT sequence theta_c[n] in degrees,
                    used instead of thetac0..thetacM-1 for {'noncoh'} FSK
           sig_dn.get_FB(): baud rate of d_n (and theta_c[n]), TB=1/FB
           Fs:      sampling rate of x(t)
           ptype:   pulse type from set
                    {'man','rcf','rect','rrcf','sinc','tri'}
           pparms = []         for {'man','rect','tri'}
           pparms = [k alpha]  for {'rcf','rrcf'}
           pparms = [k beta]   for {'sinc'}
           k:       "tail" truncation parameter for {'rcf','rrcf','sinc'}
                    (truncates p(t) to -k*TB <= t < k*TB)
           alpha:   Rolloff parameter for {'rcf','rrcf'}, 0<=alpha<=1
           beta:    Kaiser window parameter for {'sinc'}
           fcparms = [[fc0,fc1,...,fcM-1],[thetac0,thetac1,...,thetacM-1]]
                     for {'coh'}
           fcparms = [fc0,fc1,...,fcM-1]  for {'noncoh'}
           fc0,fc1,...,fcM-1:   FSK (carrier) frequencies
                                for {'coh','noncoh'}
           thetac0,thetac1,...,thetacM-1: FSK (carrier) phases in deg
                                (0: cos, -90: sin) for {'coh'}
    """
```

Test `fskxmtr` by recreating the three (time domain) sample graphs for $F_B = 100$ baud, $f_{c0} = 300$ Hz and $f_{c1} = 400$ Hz, which were given in the introduction for binary coherent FSK and noncoherent FSK. Use $d_n = \{0, 1, 1, 1, 0, 0, 1, 0\}$ and (for the second graph) $\theta_c[n] = \{270°, 225°, 4°, 135°, 250°, 90°, 40°, 240°\}$.

(b) In the `keyfun.py` module, implement the FSK receiver function `fskrcvr` whose header is given below.

```
def fskrcvr(M,sig_rt,rtype,fcparms,FBparms,ptype,pparms):
    """
    M-ary Frequency Shift Keying (FSK) Receiver for
    Coherent ('coh') and Non-coherent ('noncoh') FSK Reception
    >>>>> sig_bn,sig_wt,ixn =
                  fskrcvr(M,sig_rt,rtype,fcparms,FBparms,ptype,pparms) <<<<<
    where  sig_bn: sequence from class sigSequ
           sig_bn.signal(): received DT sequence b[n]
           sig_wt: waveform from class sigWave
           sig_wt.signal(): wt = [[w0it+1j*w0qt],[w1it+1j*w1qt],...,
                                                    [wM-1it+1j*wM-1qt]]
           wmit:     m-th in-phase matched filter output
           wmqt:     m-th quadrature matched filter output
           ixn:      sampling time indexes for b(t)->b[n], w(t)->w[n]
           M:        number of distinct FSK frequencies
           sig_rt: waveform from class sigwave
           sig_rt.signal():   received (noisy) FSK signal r(t)
           sig_rt.timeAxis(): time axis for r(t)
           rtype:    receiver type from list {'coh','noncoh'}
           fcparms = [[fc0,fc1,...,fcM-1],[thetac0,thetac1,...,thetacM-1]]
                     for {'coh'}
           fcparms = [fc0,fc1,...,fcM-1]  for {'noncoh'}
           fc0,fc1,...,fcM-1:   FSK (carrier) frequencies
                                for {'coh','noncoh'}
           thetac0,thetac1,...,thetacM-1: FSK (carrier) phases in deg
                                (0: cos, -90: sin) for {'coh'}
           FBparms = [FB, dly]
           FB:      baud rate of PAM signal, TB=1/FB
           dly:     sampling delay for wm(t)->wm[n], fraction of TB
                    sampling times are t=n*TB+t0 where t0=dly*TB
           ptype:   pulse type from list
                    {'man','rcf','rect','rrcf','sinc','tri'}
           pparms = []  for {'man','rect','tri'}
           pparms = [k, alpha]  for {'rcf','rrcf'}
           pparms = [k, beta]   for {'sinc'}
           k:       "tail" truncation parameter for {'rcf','rrcf','sinc'}
                    (truncates at -k*TB and k*TB)
           alpha:   Rolloff parameter for {'rcf','rrcf'}, 0<=alpha<=1
           beta:    Kaiser window parameter for {'sinc'}
    """
```

Depending on the choice of the `rtype`, the receiver should perform demodulation using either M coherent MFs or M MFEDs. Test both receiver modes with the signals that you generated in (a) when you were testing `fskxmtr`.

(c) Generate a coherent binary FSK signal from random data with equally likely 0's and 1's, using a rectangular $p(t)$, $F_B = 100$ baud, $f_{c0} = 300$ Hz, $\theta_{c0} = 0°$, $f_{c1} = 400$ Hz, and $\theta_{c1} = 0°$. Use a coherent demodulator to produce a scatter plot of $w_0[n]$ versus $w_1[n]$. Change the

phase θ_{c1} from 0° to 180° and check whether the signals transmitted at f_{c0} and at f_{c1} remain orthogonal in the signal space spanned by $w_0[n]$ and $w_1[n]$. Is it possible to reduce the frequency spacing $\Delta_f = f_{c1} - f_{c0}$ to a value less than F_B while maintaining orthogonality? Try changing f_{c1} to 350 Hz and vary the phase θ_{c1} again from 0° to 180°.

(d) Use uniformly distributed random M-ary data of length about 2 sec to generate PSD plots of $M = 2$ and $M = 4$ coherent and noncoherent FSK signals with rectangular $p(t)$. Determine the -40 dB bandwidth for both cases. Use $F_B = 100$ baud, $f_{c0} = 2100$ Hz, and $f_{cm} = f_{c0} + mF_B$. For coherent FSK set $\theta_{cm} = 0$ for all m.

(e) The wav files `fsksig801.wav` and `fsksig802.wav` contain binary FSK signals made from 8-bit, LSB-first, ASCII encoded characters. Analyze the two signals and extract the text messages.

E3. CPFSK, ASK and FSK Signals in GNU Radio. (a) For the Python module `keyfun.py`, write a function called `cpfskxmtr` which implements an M-ary FSK transmitter for continuous phase FSK (CPFSK). The header of this function is shown below.

```
def cpfskxmtr(M,sig_dn,Fs,ptype,pparms,fcparms):
    """
    M-ary Frequency Shift Keying (FSK) Transmitter for
    Continuous Phase FSK Signals
    >>>>> sig_xt = cpfskxmtr(M,sig_dn,Fs,ptype,pparms,fcparms) <<<<<
    where  sig_xt: waveform from class sigWave
           sig_xt.signal():   transmitted FSK signal, sampling rate Fs
           sig_xt.timeAxis(): time axis for x(t), starts at t=-TB/2
           M:       number of distinct symbol values in d[n]
           sig_dn: sequence from class sigSequ
           sig_dn.signal() = dn
           dn:      M-ary (0,1,..,M-1) N-symbol DT input sequence d_n
           sig_dn.get_FB(): baud rate of d_n, TB=1/FB
           Fs:      sampling rate of x(t)
           ptype:   pulse type from set
                    {'man','rcf','rect','rrcf','sinc','tri'}
           pparms = []         for {'man','rect','tri'}
           pparms = [k alpha]  for {'rcf','rrcf'}
           pparms = [k beta]   for {'sinc'}
           k:       "tail" truncation parameter for {'rcf','rrcf','sinc'}
                    (truncates p(t) to -k*TB <= t < k*TB)
           alpha:   Rolloff parameter for {'rcf','rrcf'}, 0<=alpha<=1
           beta:    Kaiser window parameter for {'sinc'}
           fcparms = [fc, deltaf]
           fc:      carrier frequency for {'cpfsk'}
           deltaf:  frequency spacing for {'cpfsk'}
                    for dn=0 -> fc, dn=1 -> fc+deltaf,
                    dn=2 -> fc+2*deltaf, etc
    """
```

Test `cpfskxmtr` by recreating the (time domain) sample graph for $F_B = 100$ baud, $f_{c0} = 300$ Hz and $f_{c1} = 400$ Hz, which was given in the introduction for CPFSK.
(b) Use uniformly distributed random M-ary data of length about 2 sec to generate PSD plots of $M = 2$ and $M = 4$ CPFSK signals with rectangular $p(t)$. Determine the -40 dB bandwidth. Use $F_B = 100$ baud, $f_{c0} = 2100$ Hz, and $f_{cm} = f_{c0} + mF_B$. Compare to the -40 dB bandwiths of coherent and non-coherent FSK signals in E2.

(c) In GNU Radio, build the hierarchical blocks `ASKxmtr_bc` and `ASKrcvr_cb` (and the hierarchical subblocks within these blocks) that are given in the lab description. Test the blocks using a flowgraph similar to `ASK_test002` which is shown in the lab description. Include at least two plots of running the flowgraph, one with a unipolar `'rect'` signal, and one with a polar `'rrcf'` signal.

(d) The binary GNU Radio file `digMod_805.bin` contains several (complex-valued) binary communication signals. Use the GNU Radio Companion to find the signals recorded in the file and determine their properties such as the type of modulation and the carrier frequencies. Each of the signals contains an ASCII coded (MSB first) message. Try to demodulate the signals and extract the messages.

