**Transmitter**

**Base64 encoder**

This encoder encodes a string using Base64 table.



**For example:**

Suppose there is a string “hello”.

The hex will be 68|65|6C|6C|6F

The binary representation of hello will be

01101000|01100101|01101100|01101100|01101111

It will divide in group of six bits. Starting from left to right.

011010|000110|010101|101100|011011|000110|111100

The decimal will be

26|6|21|44|27|6|60

The answer will be

aGVsbG8

**Encode 6bto8b**

It will simply map 6bits to 8bits to achieve DC Balance. DC balance is a stream of data encoded to ensure an equal balance of 1’s and 0’s.

Input =aGVsbG8

Encoded 6b2\_8bmessage= [154, 198, 149, 172, 27, 198, 60, 106]

**Struct Pack**

It will returns a string containing values packed according to given format.

input= [154, 198, 149, 172, 27, 198, 60, 106]

Output=�ƕ�#�<j

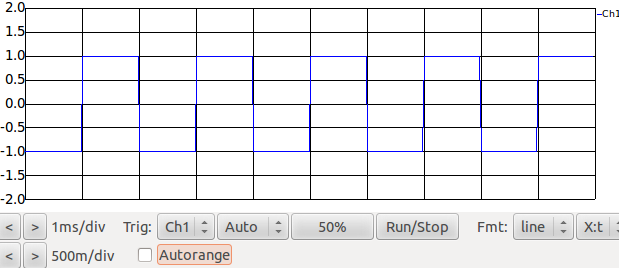
**Bytes to Symbols**

It will simply convert bytes to +/-1 symbol

gr.bytes\_to\_symbol()

Input = 10101010

Ouput=+1-1+1-1+1-1+1-1



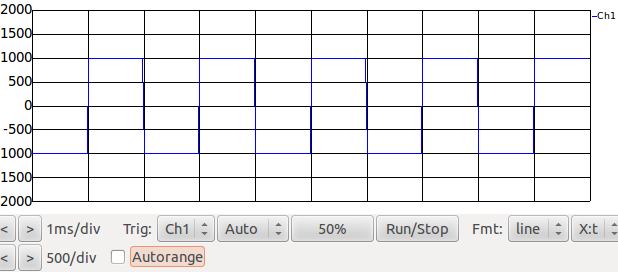
**Multiply constant**

It will simply multiply constant in the input

Output=Input x Const

gr.multiply\_const\_ff(1000)

It will simply multiply 1000 to the input.



**Add constant**

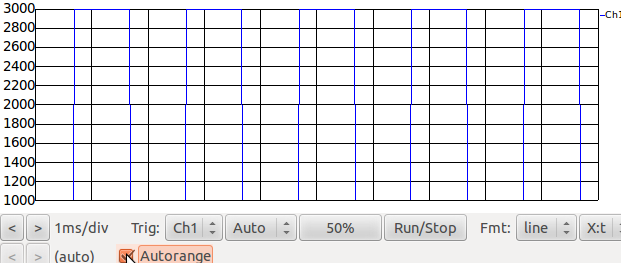
It will simply add constant in the input

Output=Input + Const

gr.add\_const\_ff(center\_frequency)

It will simply add 2000 center frequency to the input.

These Add and Multiply constant blocks are used to set the frequency of the wave.



**Repeater**

It will repeat a sample for the given time in output stream.

gr.repeat(gr.sizeof\_float,480)

so each symbol will have 480 samples.

**VCO**

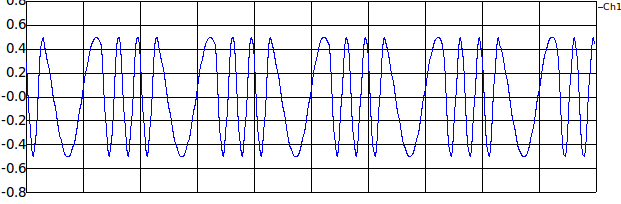
Voltage control oscillator is used to generate wave according to the input voltage.

fsk\_f = gr.vco\_f(audio\_rate, 2\*pi,0.5)

audio\_rate=Sampling time

Amplitude=0.5

frequency=2\*pi\*(input voltage)



**Audio Sink**

This block simply output soundwave to audio speaker

audio.sink(audio\_rate, "plughw:0,0")

input=Sampling rate

**Receiver**

In this section we will demodulate the received signal using quad demodulator to get PAM signal and to remove the carrier signal because we are using M&M clock recovery which is sensitive to carrier offsets. Quad demodulator needs I-phase and Q-phase.

**Band pass**

The signal is filter to remove the unwanted frequency signal.

bp\_coeff = gr.firdes.band\_pass(1,audio\_rate,1000,3000,100)

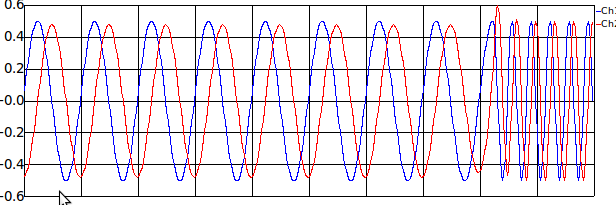
bpf = gr.fir\_filter\_fff(1,bp\_coeff)

**Hilbert Transform**

Hilbert transform is used to get I-phase and Q-phase component of the signal.

Parameter: No. of Taps

gr.hilbert\_fc(64)

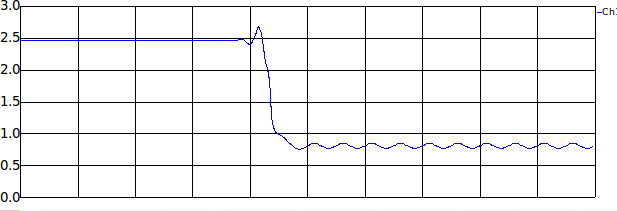


**Quad Demodulator**

This block is used to get the instantaneous frequency of the input signal

Parameter: Gain (2\*pi\*delta\_frequency)

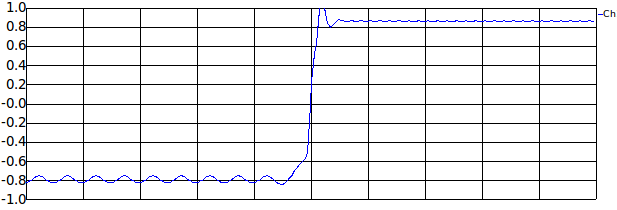
gr.quadrature\_demod\_cf(6.28)



**Add Constant**

This block is used to centre the signal at zero.

Parameter: -( centre Frequency)



**Clock Recovery**

This block is used to recover the clock. This is discrete error tracking based on Mueller and Muller algorithm .

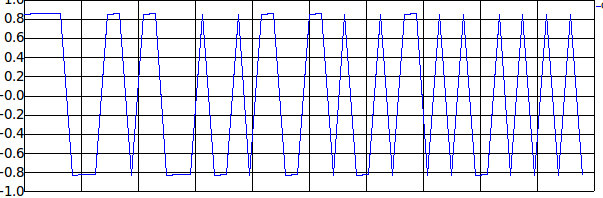
gr.clock\_recovery\_mm\_ff(480,0.000625,0.5,0.01,0.05)

omega= Samples per symbol

mu= Initial estimate of phase of sample

omega relative limit= limit on omega

gain\_mu = 0.01  
gain\_omega = .25 \* gain\_mu \* gain\_mu



It output one sample per symbol.

**Binary Slicer**

This block is decision maker it will assign 0 for input<0 and 1 for input>=0. Outputting 1 bit per sample. So Instead of having 480 samples we will have 1 sample per bit.

gr.binary\_slicer\_fb()

**Synchronous correlate**

It examine the input for the specified access code one bit at a time to give the output after specified access code.

gr.correlate\_access\_code\_bb("0100011101111000",0)

**Decode 8bto6b**

It will simply map 8bits to 6bits to achieve DC Balance.

Input= [154, 198, 149, 172, 27, 198, 60, 106]

Decoded 8b2\_6bmessage=aGVsbG8

**Base64 decoder**

This encoder encodes a string using Base64 table.

Input = aGVsbG8

The decimal will be

26|6|21|44|27|6|60

The binary representation will be

011010|000110|010101|101100|011011|000110|111100

It will divide in group of 8 bits. Starting from left to right.

01101000|01100101|01101100|01101100|01101111

The hex will be 68|65|6C|6C|6F

Using ASCII table output will be “hello”.