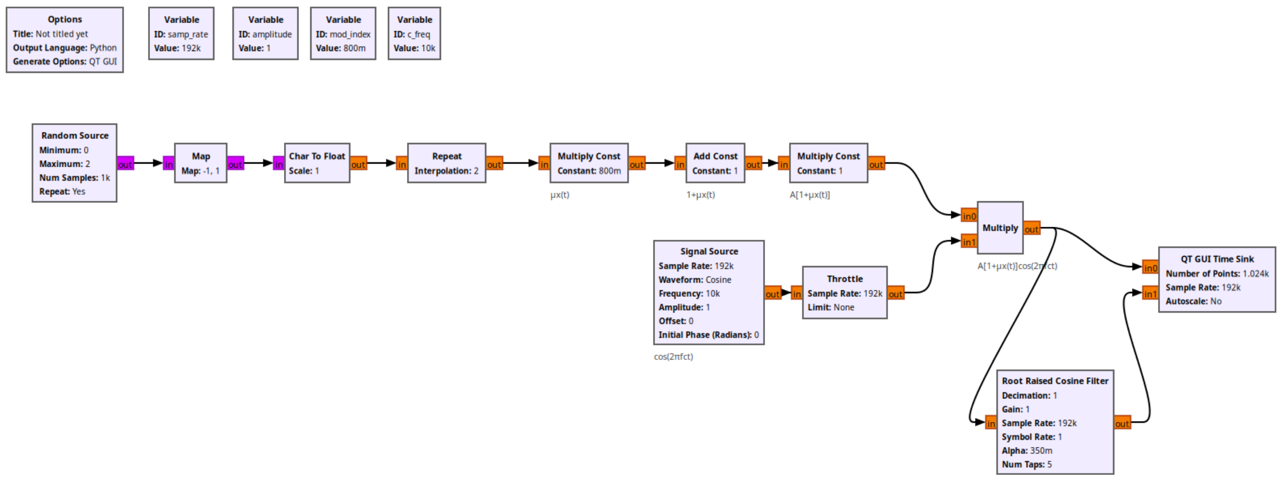
HY330 – Telecommunication Systems

Chris Papastamos | csd4569

Assignment 5

Exercise 1

For the purpose of this exercise I created the following flowgraph:



Options
Title: Not titled yet
Output Language: Python
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 192k

Variable
ID: amplitude
Value: 1

Variable
ID: mod_index
Value: 800m

Variable
ID: c_freq
Value: 10k

Random Source
Minimum: 0
Maximum: 2
Num Samples: 1k
Repeat: Yes

Map
Map: -1, 1

Char To Float
Scale: 1

Repeat
Interpolation: 2

Multiply Const
Constant: 800m

Add Const
Constant: 1

Multiply Const
Constant: 1

Multiply

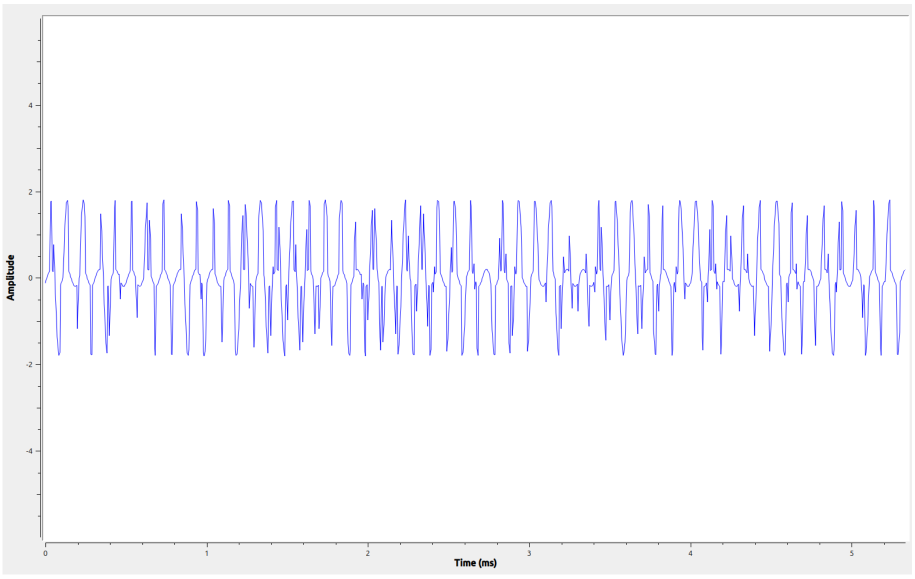
Signal Source
Sample Rate: 192k
Waveform: Cosine
Frequency: 10k
Amplitude: 1
Offset: 0
Initial Phase (Radians): 0

Throttle
Sample Rate: 192k
Limit: None

QT GUI Time Sink
Number of Points: 1.024k
Sample Rate: 192k
Autoscale: No

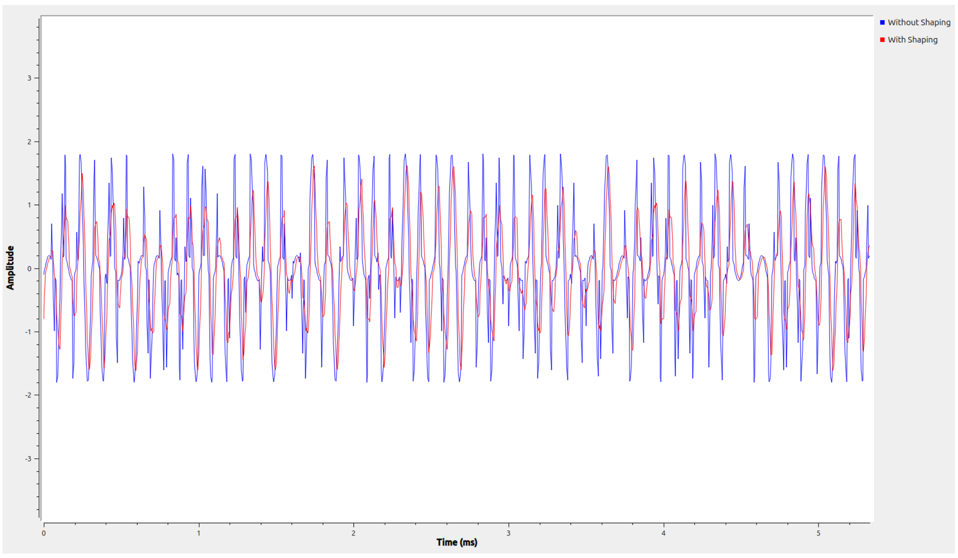
Root Raised Cosine Filter
Decimation: 1
Gain: 1
Sample Rate: 192k
Symbol Rate: 1
Alpha: 350m
Num Taps: 5

This flowgraph modulates the output of the random source (1,-1) with Amplitude Modulation. It uses the variable modulation index to distance the two amplitude modulation points. The output of the flowgraph looks like this:



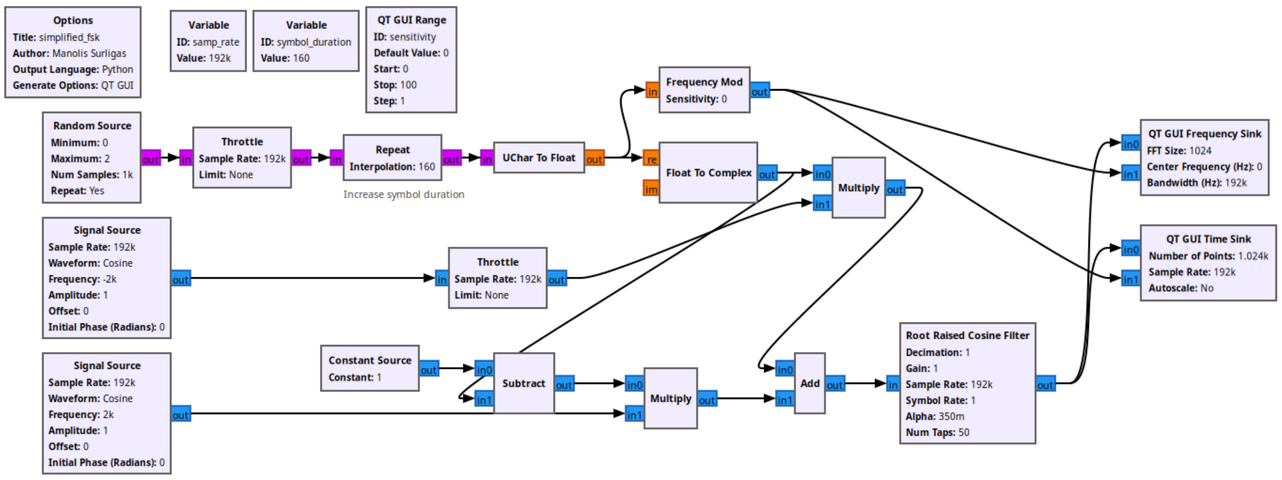
We can see that the signal has the two amplitude modulation points at 0.2 and at 1.8 (1±0.8).

For the shaping of the signal, we can add a Root Raised Cosine Filter to shape our signal. The unshaped and shaped output can be seen below:



Exercise 2

For the 2nd exercise I edited the simplified_fsk.grc by applying the following modifications: I altered the samp rate to 192kSPS and changed the symbol duration to 160 points in order to achieve a bitrate of 1200SPS. The final flowgraph is the following:



Options
Title: simplified_fsk
Author: Manolis Surligas
Output Language: Python
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 192k

Variable
ID: symbol_duration
Value: 160

QT GUI Range
ID: sensitivity
Default Value: 0
Start: 0
Stop: 100
Step: 1

Random Source
Minimum: 0
Maximum: 2
Num Samples: 1k
Repeat: Yes

Throttle
Sample Rate: 192k
Limit: None

Repeat
Interpolation: 160

Increase symbol duration

UChar To Float

Frequency Mod
Sensitivity: 0

Float To Complex

Multiply

QT GUI Frequency Sink
FFT Size: 1024
Center Frequency (Hz): 0
Bandwidth (Hz): 192k

QT GUI Time Sink
Number of Points: 1.024k
Sample Rate: 192k
Autoscale: No

Signal Source
Sample Rate: 192k
Waveform: Cosine
Frequency: -2k
Amplitude: 1
Offset: 0
Initial Phase (Radians): 0

Throttle
Sample Rate: 192k
Limit: None

Signal Source
Sample Rate: 192k
Waveform: Cosine
Frequency: 2k
Amplitude: 1
Offset: 0
Initial Phase (Radians): 0

Constant Source
Constant: 1

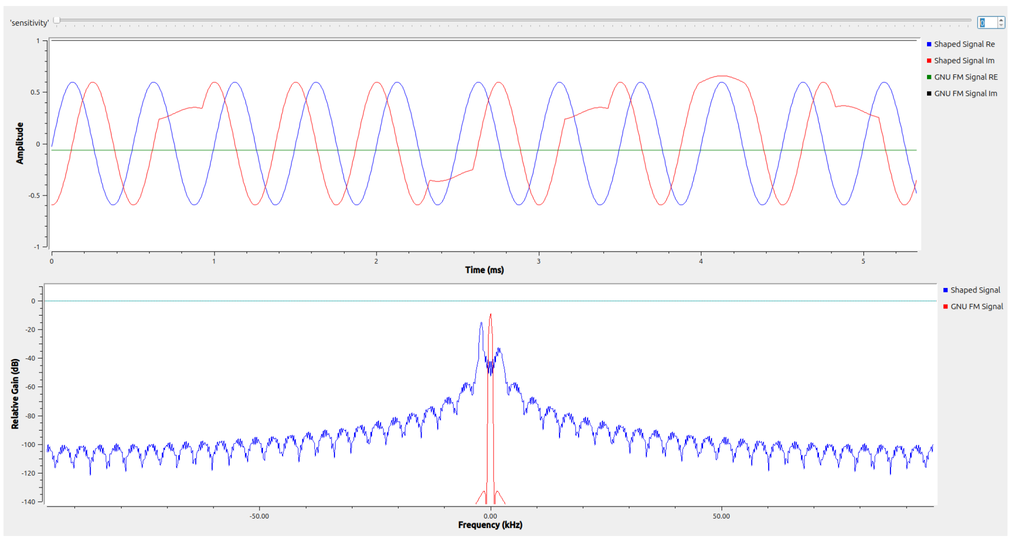
Subtract

Multiply

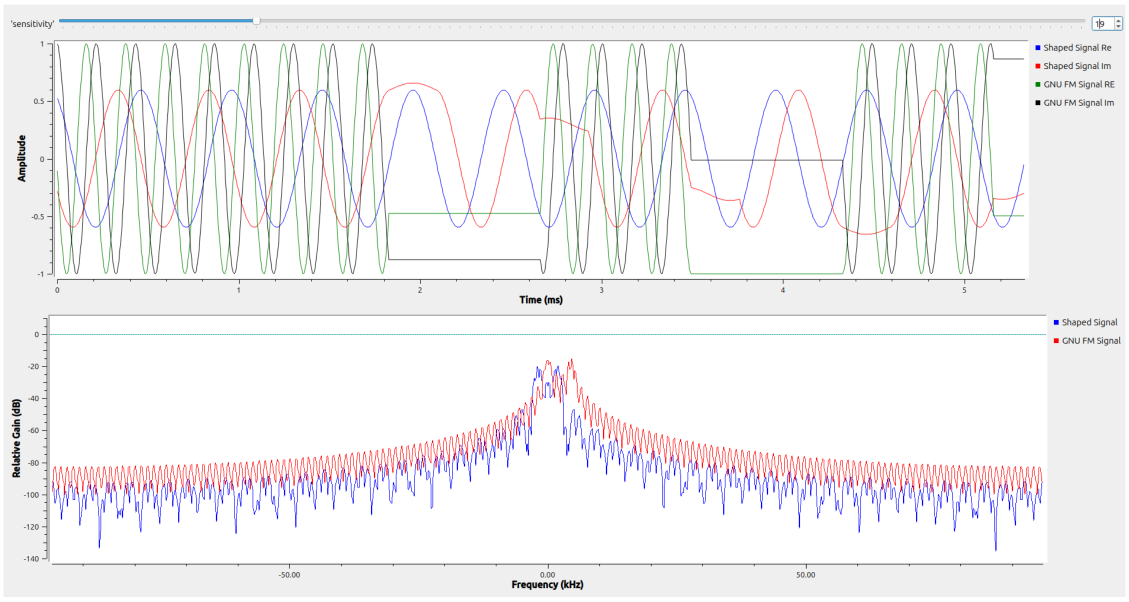
Add

Root Raised Cosine Filter
Decimation: 1
Gain: 1
Sample Rate: 192k
Symbol Rate: 1
Alpha: 350m
Num Taps: 50

In order to shape the output of the signal I used the Root Raised Cosine Filter block. I also used the Frequency Mod block to apply FM to the signal. The output of the flowgraph is the following:



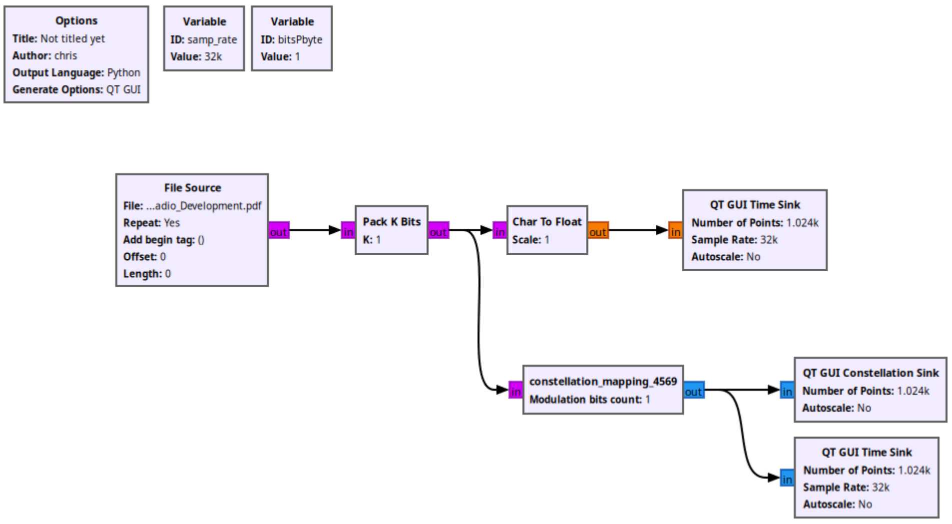
By raising the Frequency Mod sensitivity, we observe the following output:



Exercise 2

In this exercise I had to create a new block called constellation_mapping_4569. For this I used gr-modtool to create a new module (called “chris_module”) and then I created the constellation_mapping_4569 block.

For the purpose of demonstrating the new block I created the following flowgraph:



Options
Title: Not titled yet
Author: chris
Output Language: Python
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 32k

Variable
ID: bitsPbyte
Value: 1

File Source
File: ...adio_Development.pdf
Repeat: Yes
Add begin tag: ()
Offset: 0
Length: 0

Pack K Bits
K: 1

Char To Float
Scale: 1

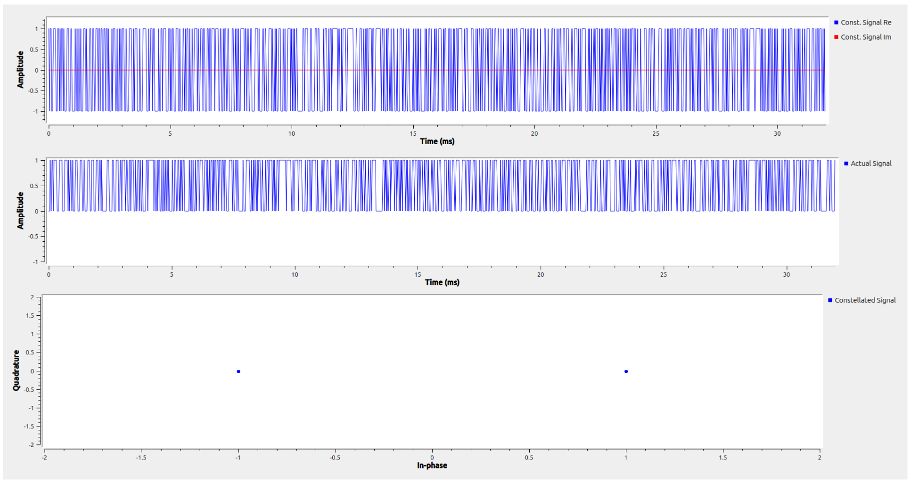
QT GUI Time Sink
Number of Points: 1.024k
Sample Rate: 32k
Autoscale: No

constellation_mapping_4569
Modulation bits count: 1

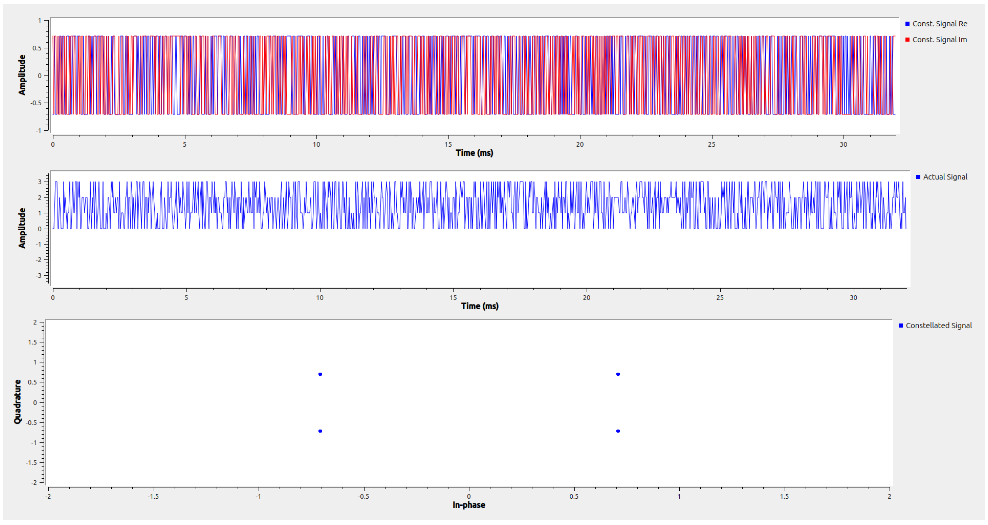
QT GUI Constellation Sink
Number of Points: 1.024k
Autoscale: No

QT GUI Time Sink
Number of Points: 1.024k
Sample Rate: 32k
Autoscale: No

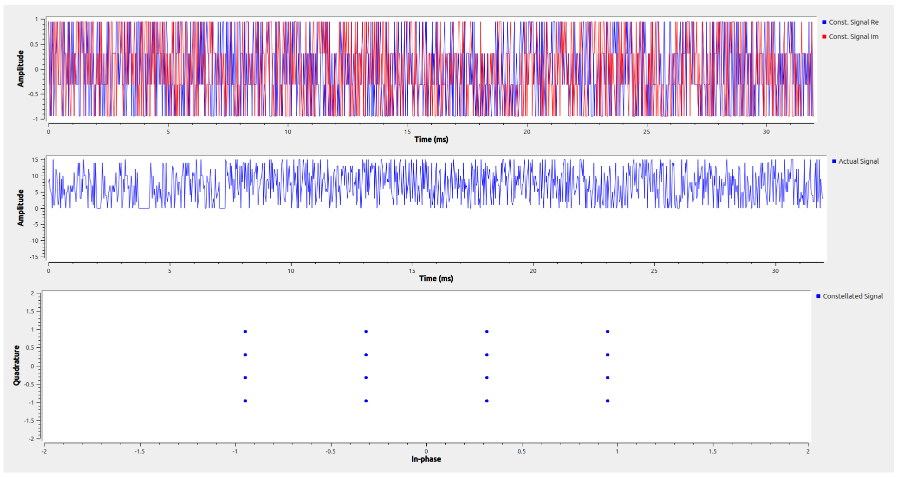
Next up we can see a screenshot of a run for each value of bitsPbyte [1, 2, 4, 3 (or any other value)]:



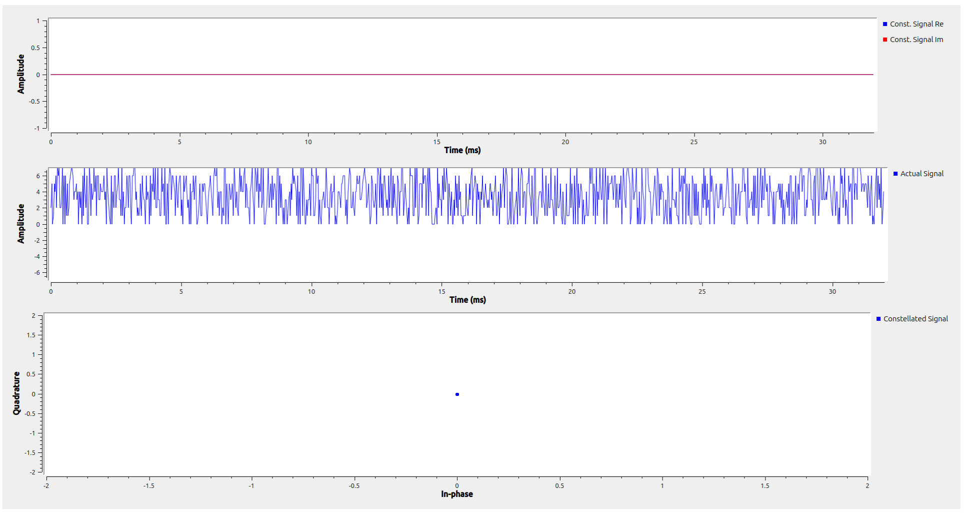
BPSK (1 bitPB)



QPSK (2 bitsPB)

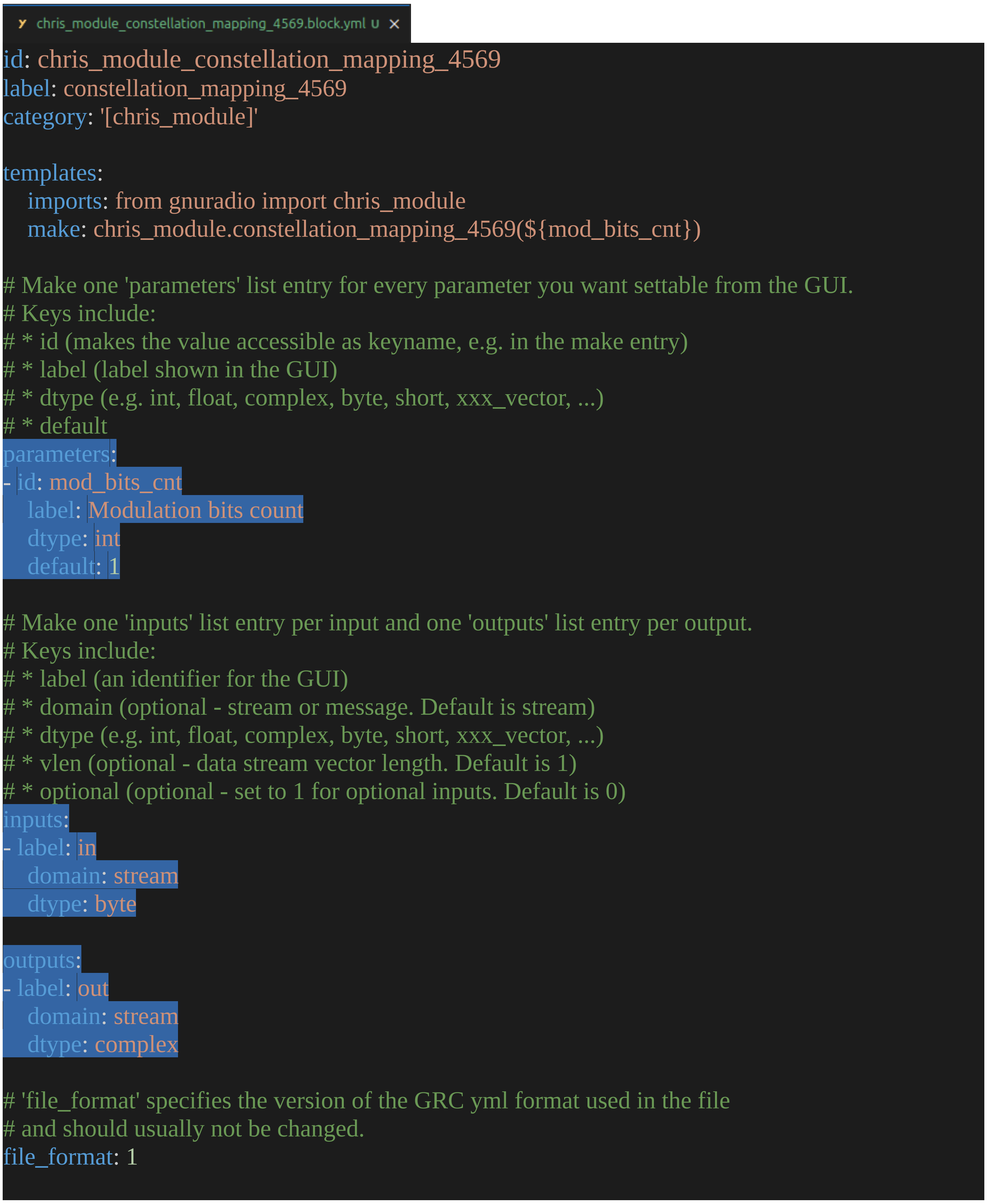


QPSK (4 bitsPB)



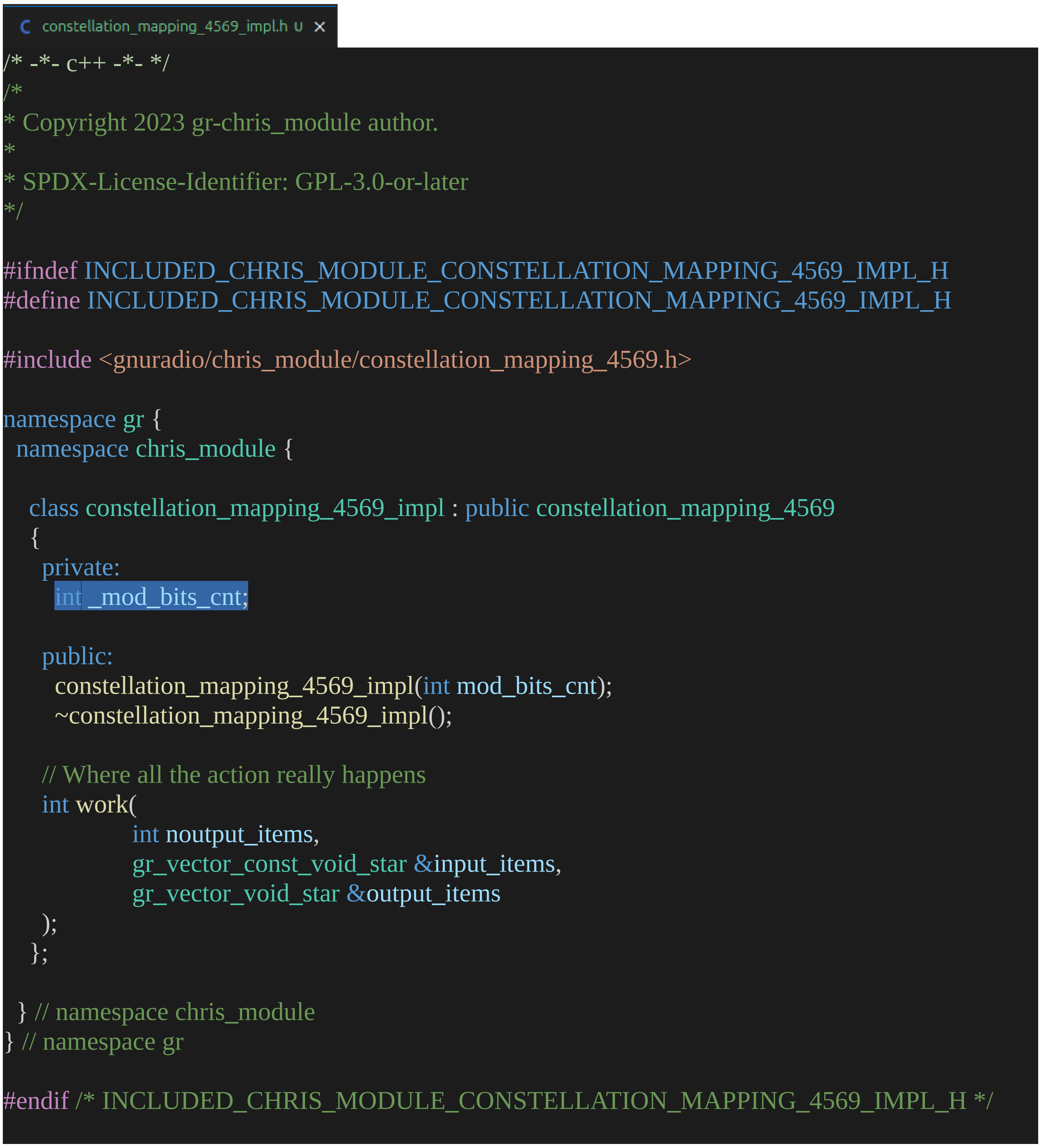
Error Case (3 bitsPB)

The block files are the following and the changes are highlighted:



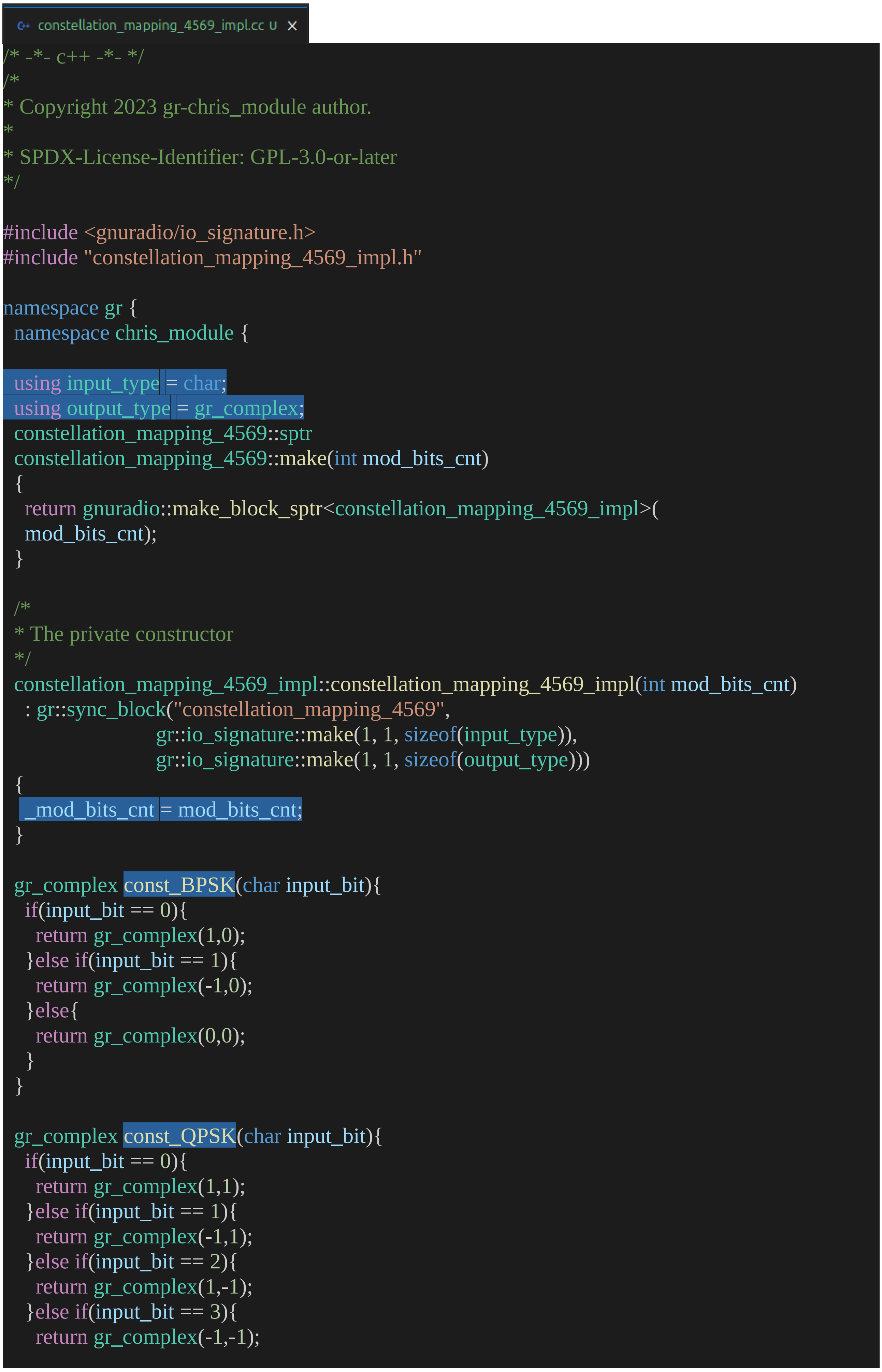
chris_module_constellation_mapping_4569.block.yml

```yaml
id: chris_module_constellation_mapping_4569
label: constellation_mapping_4569
category: '[chris_module]'

templates:
    imports: from gnuradio import chris_module
    make: chris_module.constellation_mapping_4569(${mod_bits_cnt})

# Make one 'parameters' list entry for every parameter you want settable from the GUI.
# Keys include:
# * id (makes the value accessible as keyname, e.g. in the make entry)
# * label (label shown in the GUI)
# * dtype (e.g. int, float, complex, byte, short, xxx_vector, ...)
# * default
parameters:
- id: mod_bits_cnt
    label: Modulation bits count
    dtype: int
    default: 1

# Make one 'inputs' list entry per input and one 'outputs' list entry per output.
# Keys include:
# * label (an identifier for the GUI)
# * domain (optional - stream or message. Default is stream)
# * dtype (e.g. int, float, complex, byte, short, xxx_vector, ...)
# * vlen (optional - data stream vector length. Default is 1)
# * optional (optional - set to 1 for optional inputs. Default is 0)
inputs:
- label: in
    domain: stream
    dtype: byte

outputs:
- label: out
    domain: stream
    dtype: complex

# 'file_format' specifies the version of the GRC yml format used in the file
# and should usually not be changed.
file_format: 1
```



constellation_mapping_4569_impl.h U ×

```
/* -*- c++ -*- */
/*
* Copyright 2023 gr-chris_module author.
*
* SPDX-License-Identifier: GPL-3.0-or-later
*/

#ifndef INCLUDED_CHRIS_MODULE_CONSTELLATION_MAPPING_4569_IMPL_H
#define INCLUDED_CHRIS_MODULE_CONSTELLATION_MAPPING_4569_IMPL_H

#include <gnuradio/chris_module/constellation_mapping_4569.h>

namespace gr {
  namespace chris_module {

    class constellation_mapping_4569_impl : public constellation_mapping_4569
    {
      private:
        int _mod_bits_cnt;

      public:
        constellation_mapping_4569_impl(int mod_bits_cnt);
        ~constellation_mapping_4569_impl();

      // Where all the action really happens
      int work(
              int noutput_items,
              gr_vector_const_void_star &input_items,
              gr_vector_void_star &output_items
      );
    };

  } // namespace chris_module
} // namespace gr

#endif /* INCLUDED_CHRIS_MODULE_CONSTELLATION_MAPPING_4569_IMPL_H */
```


constellation_mapping_4569_impl.cc U ×

```
/* -*- c++ -*- */
/*
* Copyright 2023 gr-chris_module author.
*
* SPDX-License-Identifier: GPL-3.0-or-later
*/

#include <gnuradio/io_signature.h>
#include "constellation_mapping_4569_impl.h"

namespace gr {
  namespace chris_module {

  using input_type = char;
  using output_type = gr_complex;
  constellation_mapping_4569::sptr
  constellation_mapping_4569::make(int mod_bits_cnt)
  {
    return gnuradio::make_block_sptr<constellation_mapping_4569_impl>(
    mod_bits_cnt);
  }

  /*
  * The private constructor
  */
  constellation_mapping_4569_impl::constellation_mapping_4569_impl(int mod_bits_cnt)
    : gr::sync_block("constellation_mapping_4569",
                          gr::io_signature::make(1, 1, sizeof(input_type)),
                          gr::io_signature::make(1, 1, sizeof(output_type)))
  {
    _mod_bits_cnt = mod_bits_cnt;
  }

  gr_complex const_BPSK(char input_bit){
    if(input_bit == 0){
      return gr_complex(1,0);
    }else if(input_bit == 1){
      return gr_complex(-1,0);
    }else{
      return gr_complex(0,0);
    }
  }

  gr_complex const_QPSK(char input_bit){
    if(input_bit == 0){
      return gr_complex(1,1);
    }else if(input_bit == 1){
      return gr_complex(-1,1);
    }else if(input_bit == 2){
      return gr_complex(1,-1);
    }else if(input_bit == 3){
      return gr_complex(-1,-1);
    }else{
      return gr_complex(0,0);
    }
  }

  gr_complex const_16QAM(char input_bit){
    if(input_bit == 0){
      return gr_complex(3,3);
    }else if(input_bit == 1){
      return gr_complex(1,3);
    }else if(input_bit == 2){
      return gr_complex(-1,3);
    }else if(input_bit == 3){
      return gr_complex(-3,3);
    }else if(input_bit == 4){
      return gr_complex(3,1);
    }else if(input_bit == 5){
      return gr_complex(1,1);
    }else if(input_bit == 6){
      return gr_complex(-1,1);
    }else if(input_bit == 7){
      return gr_complex(-3,1);
    }else if(input_bit == 8){
      return gr_complex(3,-1);
    }else if(input_bit == 9){
      return gr_complex(1,-1);
    }else if(input_bit == 10){
      return gr_complex(-1,-1);
    }else if(input_bit == 11){
      return gr_complex(-3,-1);
    }else if(input_bit == 12){
      return gr_complex(3,-3);
    }else if(input_bit == 13){
      return gr_complex(1,-3);
    }else if(input_bit == 14){
      return gr_complex(-1,-3);
    }else if(input_bit == 15){
      return gr_complex(-3,-3);
    }else{
      return gr_complex(0,0);
    }
  }

  /*
  * Our virtual destructor.
  */
  constellation_mapping_4569_impl::~constellation_mapping_4569_impl()
  {
  }

  int
  constellation_mapping_4569_impl::work(int noutput_items,
                  gr_vector_const_void_star &input_items,
                  gr_vector_void_star &output_items)
      {
       auto in = static_cast<const input_type*>(input_items[0]);
       auto out = static_cast<output_type*>(output_items[0]);

       // constellation function dispatcher
       gr_complex (*constellation_func[])(char) = {NULL, const_BPSK, const_QPSK, NULL,
  const_16QAM};
       float normal_factor[] = {0, 1, 1/sqrt(2), 0, 1/sqrt(10)};

       for(int i = 0; i < noutput_items; i++){
         if(_mod_bits_cnt == 1 || _mod_bits_cnt == 2 || _mod_bits_cnt == 4){
           out[i] = normal_factor[_mod_bits_cnt] * constellation_func[_mod_bits_cnt](in[i]);
         }else{
           out[i] = gr_complex(0,0);
         }
       }

         return noutput_items;
      }

    } /* namespace chris_module */
  } /* namespace gr */
```