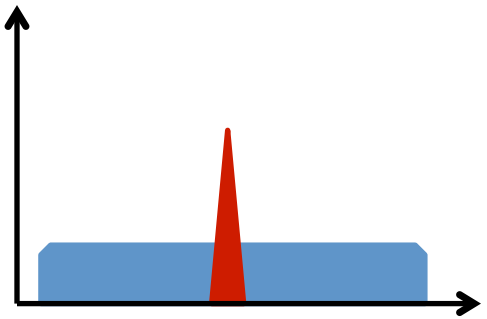
GNU Radio

THE FREE & OPEN SOFTWARE RADIO ECOSYSTEM



Spread Spectrum Techniques in GNU Radio /

Paul David
Advisor: Dr. Robert McGwier
Hume Center – Virginia Tech
GNU Radio Conference '14

Goal: Develop a generic toolkit for Spread Spectrum

- Add to the available blocks for PSK, FSK, and others by adding spread spectrum

- Create an example of a spread spectrum communication system in GNU Radio

An Overview of GNU Radio Features

GNU Radio provides several handy features along with the scheduler itself

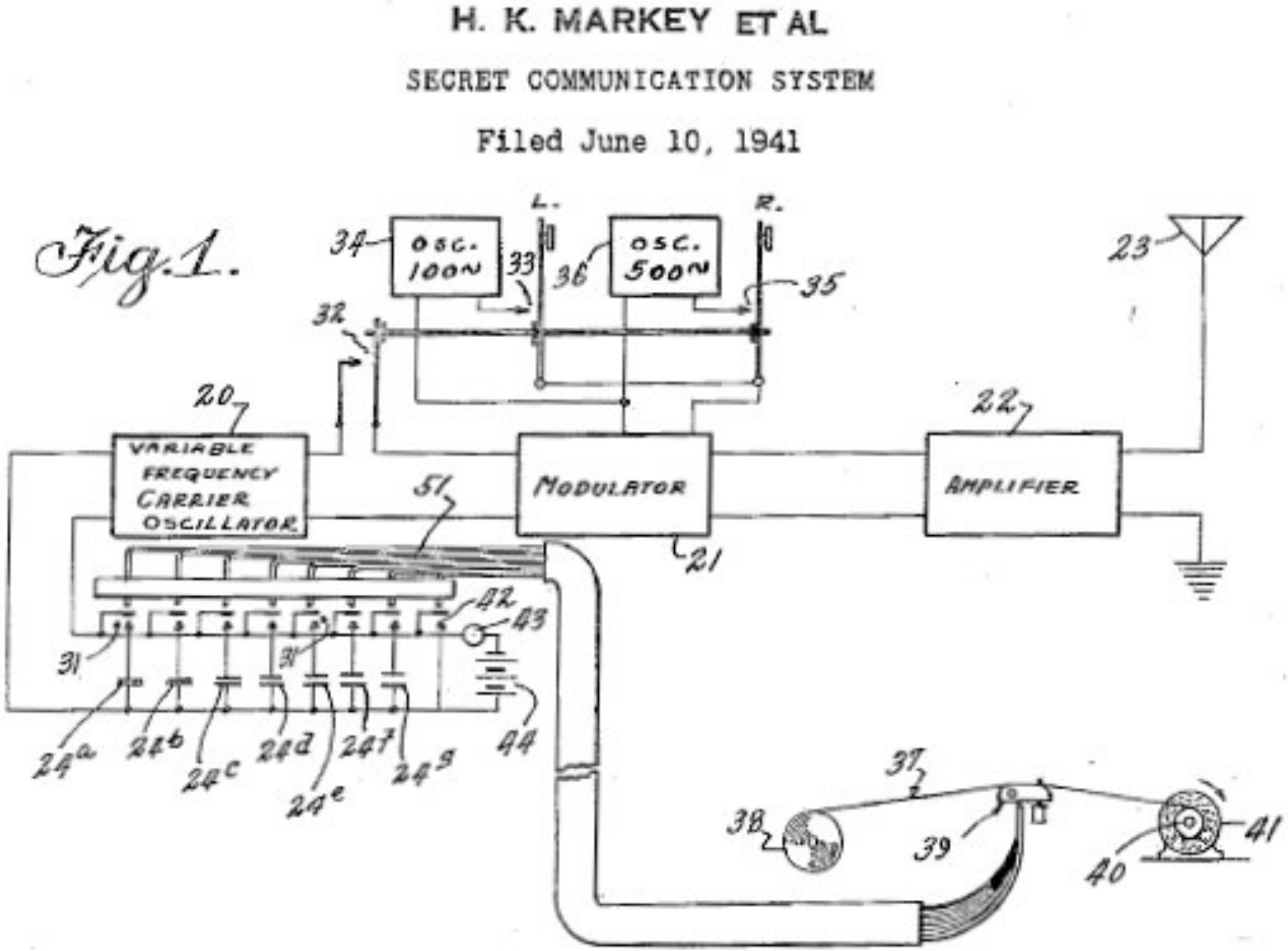
- Different types of blocks (decimation, interpolation, sync, etc.)
- Stream tags for endowing the stream with meta data
- Message passing API and polymorphic data types (PMTs) for conveniently wrapping most kinds of data
- Filter API for generating and implementing

A Little Bit of Spread Spectrum History…

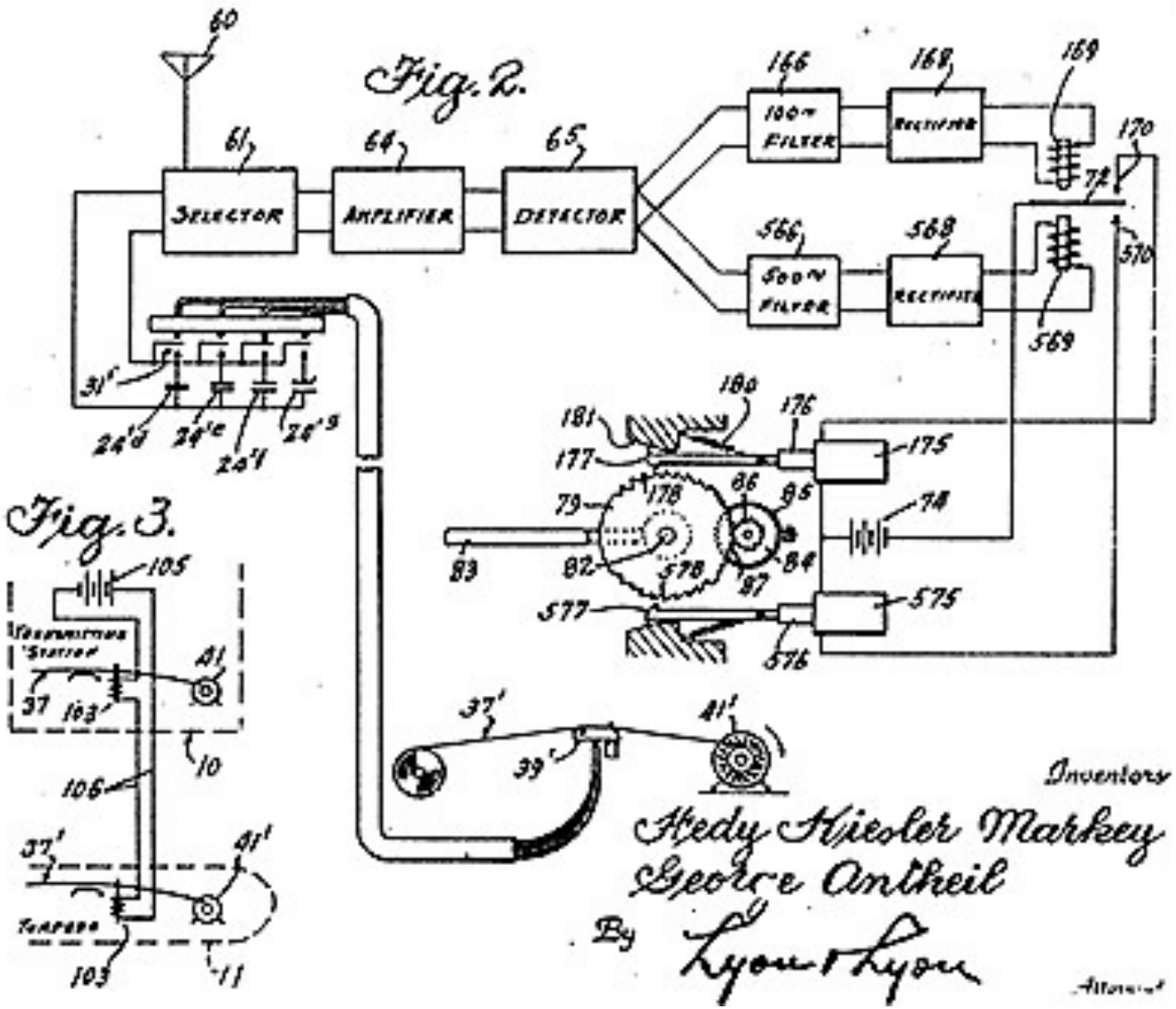
- Spread spectrum techniques have existed since the 1940s
- Initial idea: Covert communications: Signal can be hidden or hard to spot
- Main idea: resistance to interference, both hostile and incidental
- Actress Hedy Lamarr and composer George Antheil used Frequency Hopping for radio guided Torpedoes

  Patent No. 2,292,387

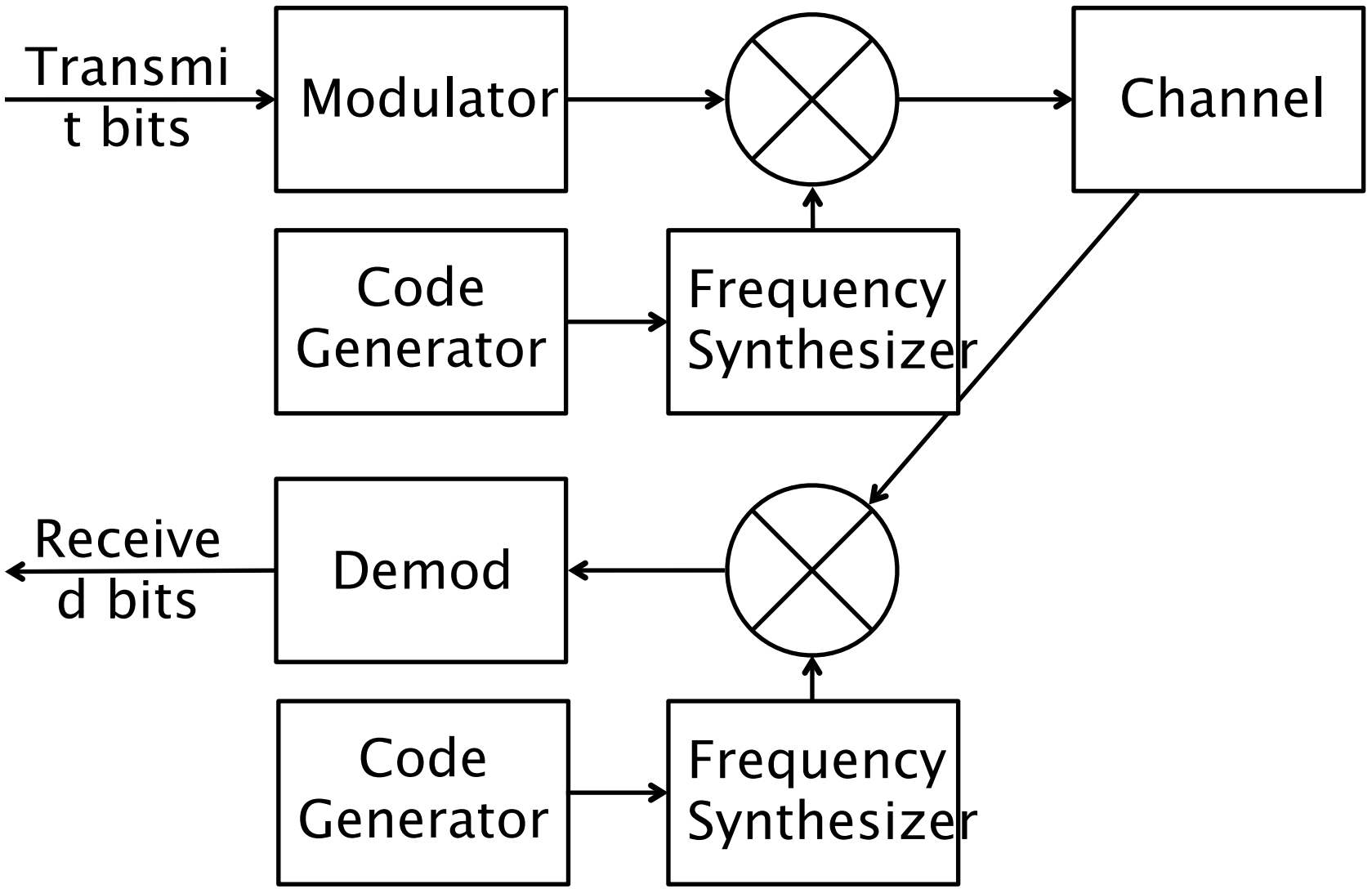
Patent No. 2,292,387



Patent No. 2,292,387



Frequency-Hopping Spread Spectrum



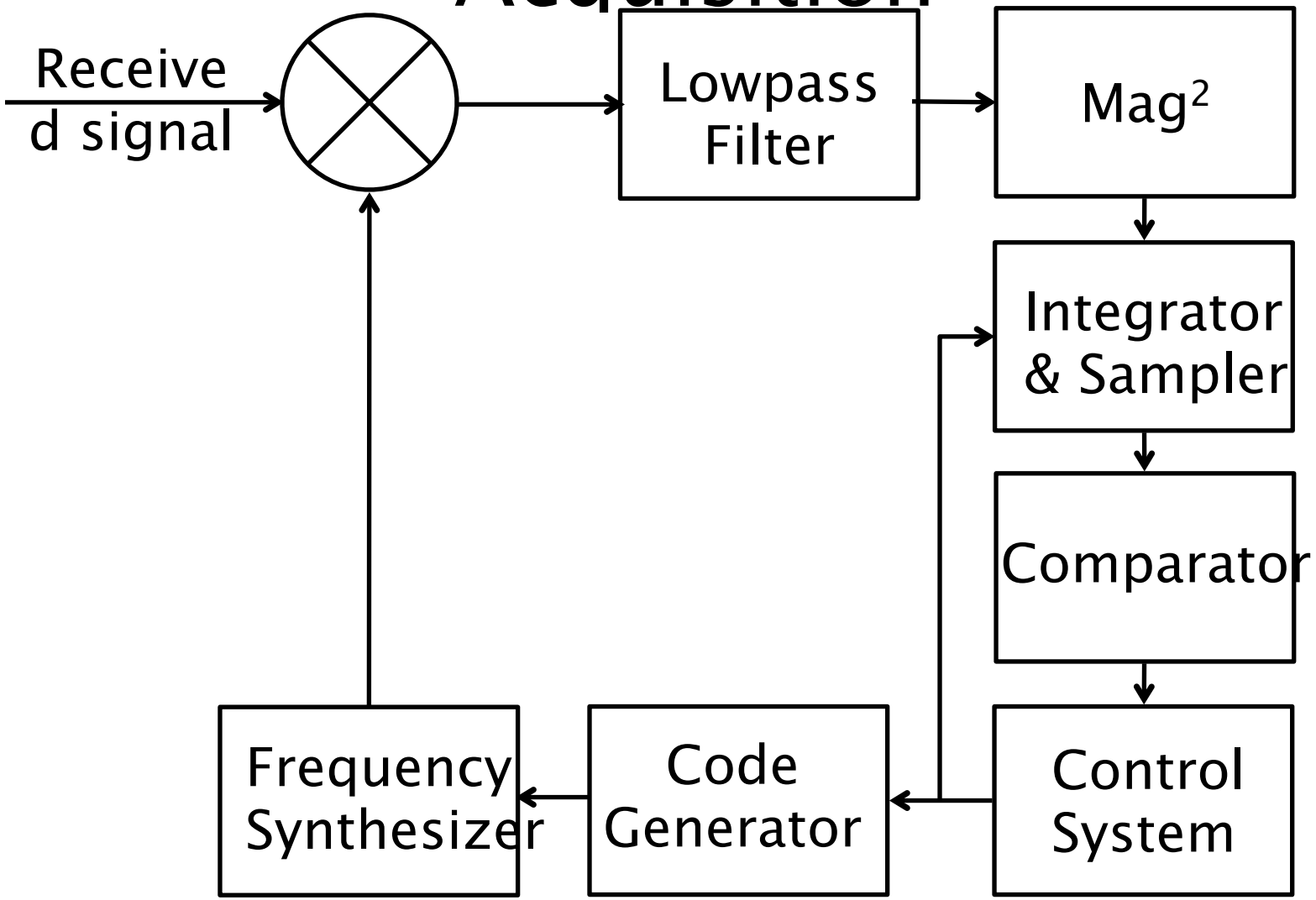
Frequency-Hopping Spread Spectrum

The previous picture was over simplified…

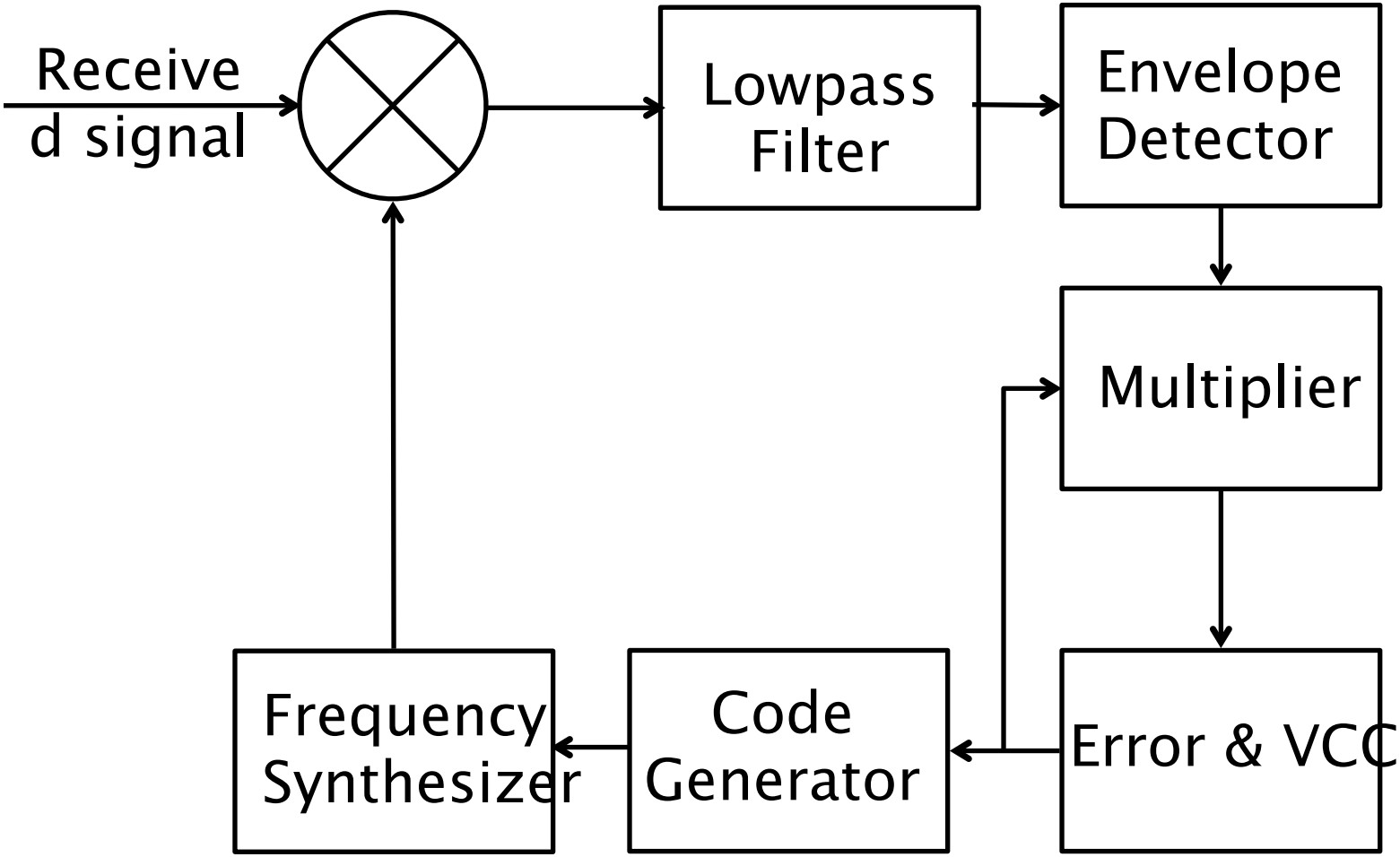
Still needs

- Acquisition system
- Tracking loop of some sort
- And more synchronization…

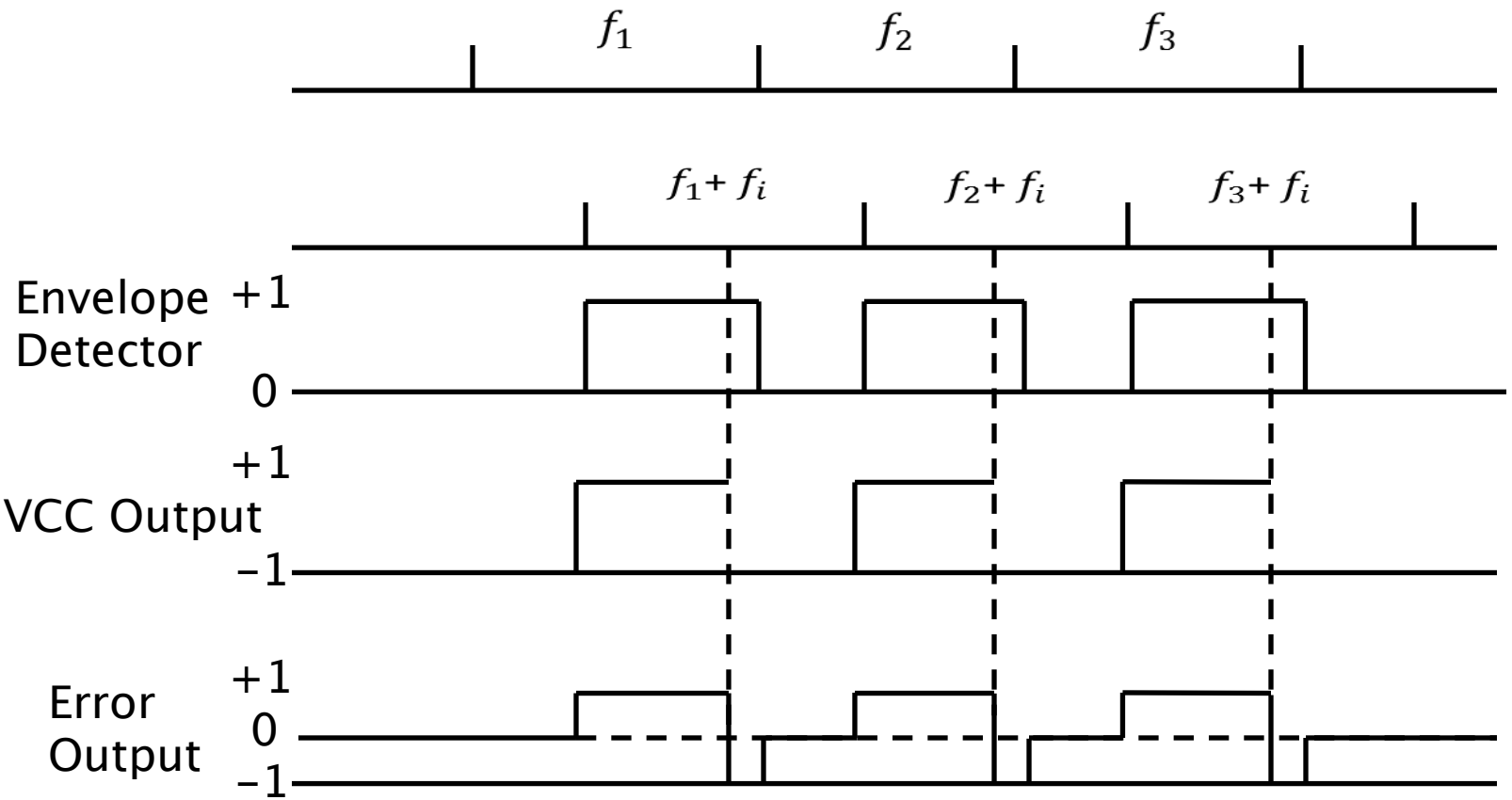
Frequency-Hopping Serial Acquisition



Frequency-Hopping Tracking

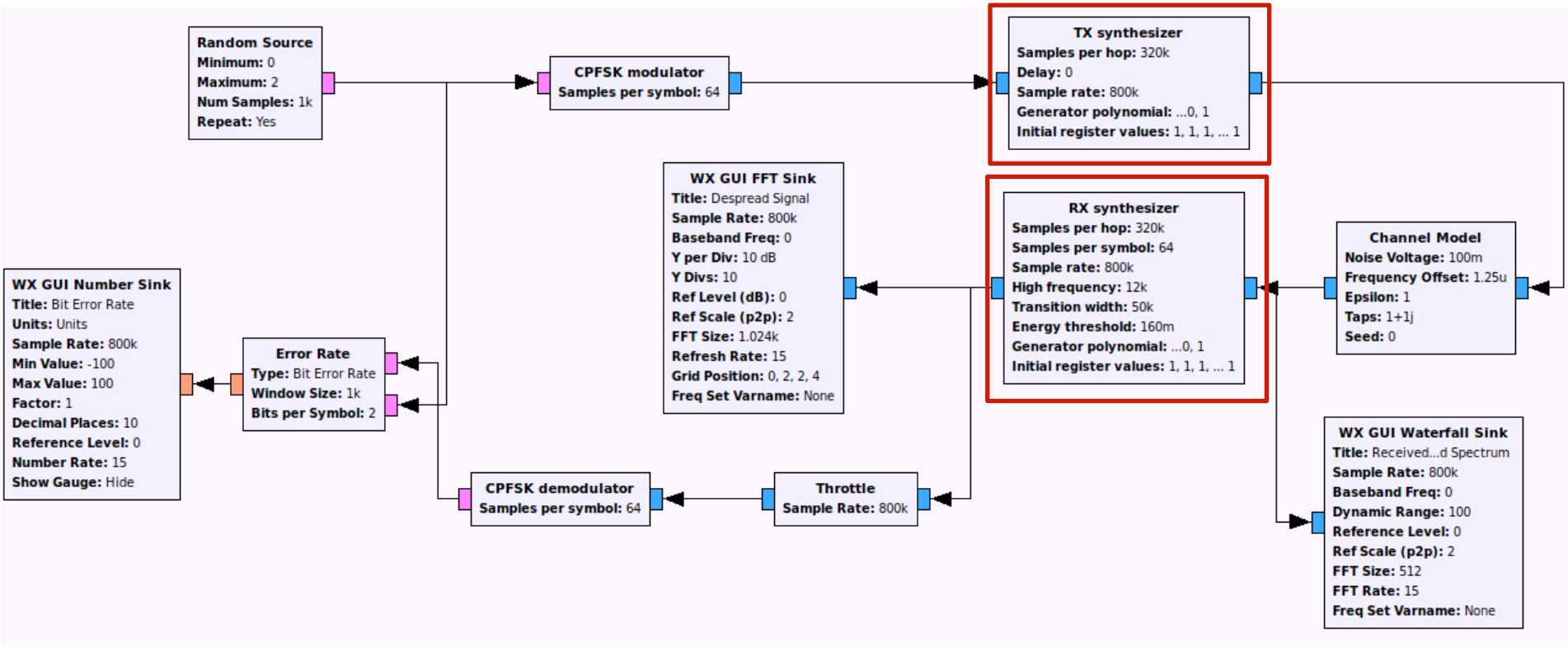


Frequency-Hopping Tracking



FHSS GNU Radio

Frequency hopping flowgraph example…

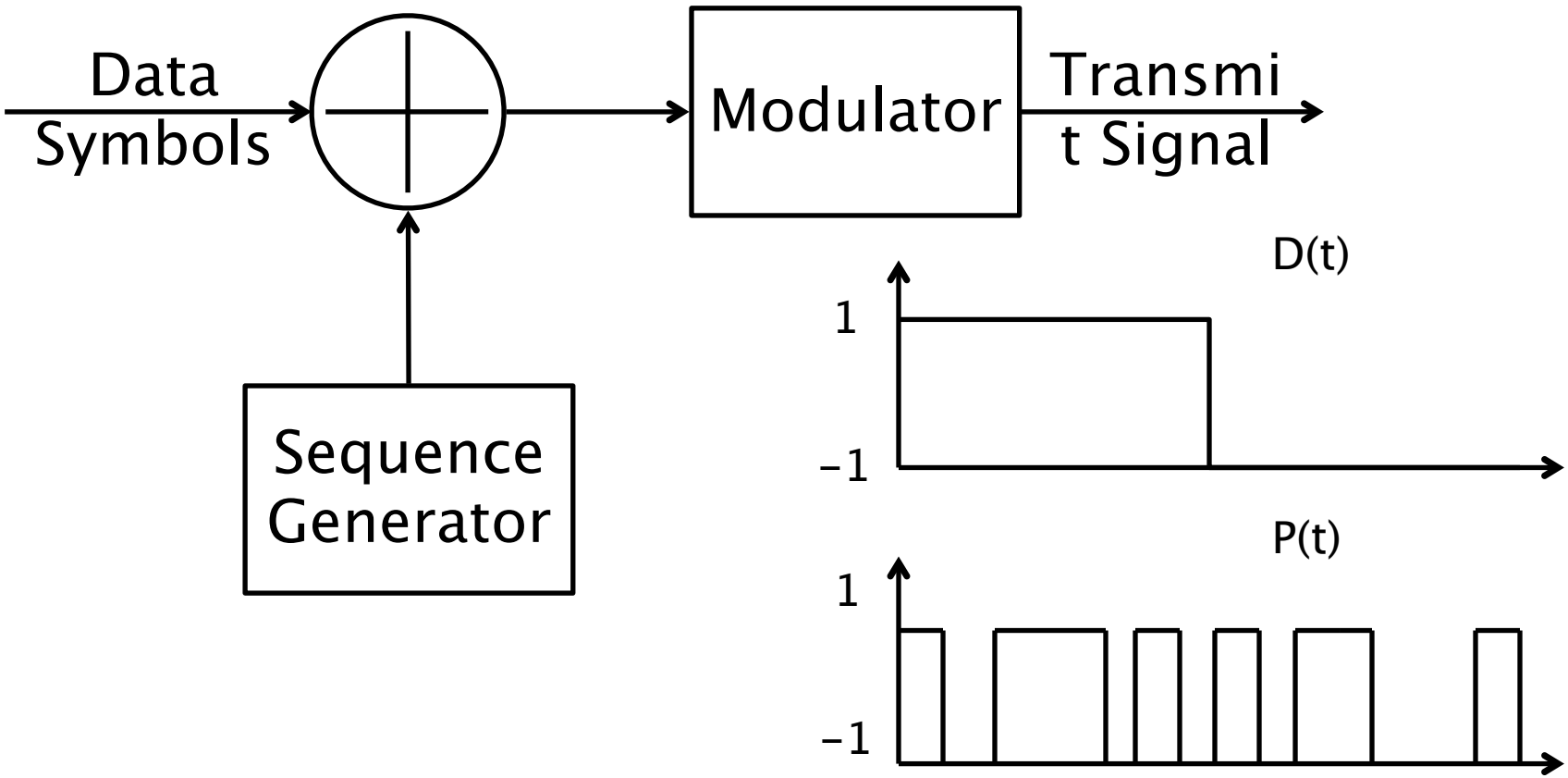


FHSS GNU Radio

Frequency hopping in action...

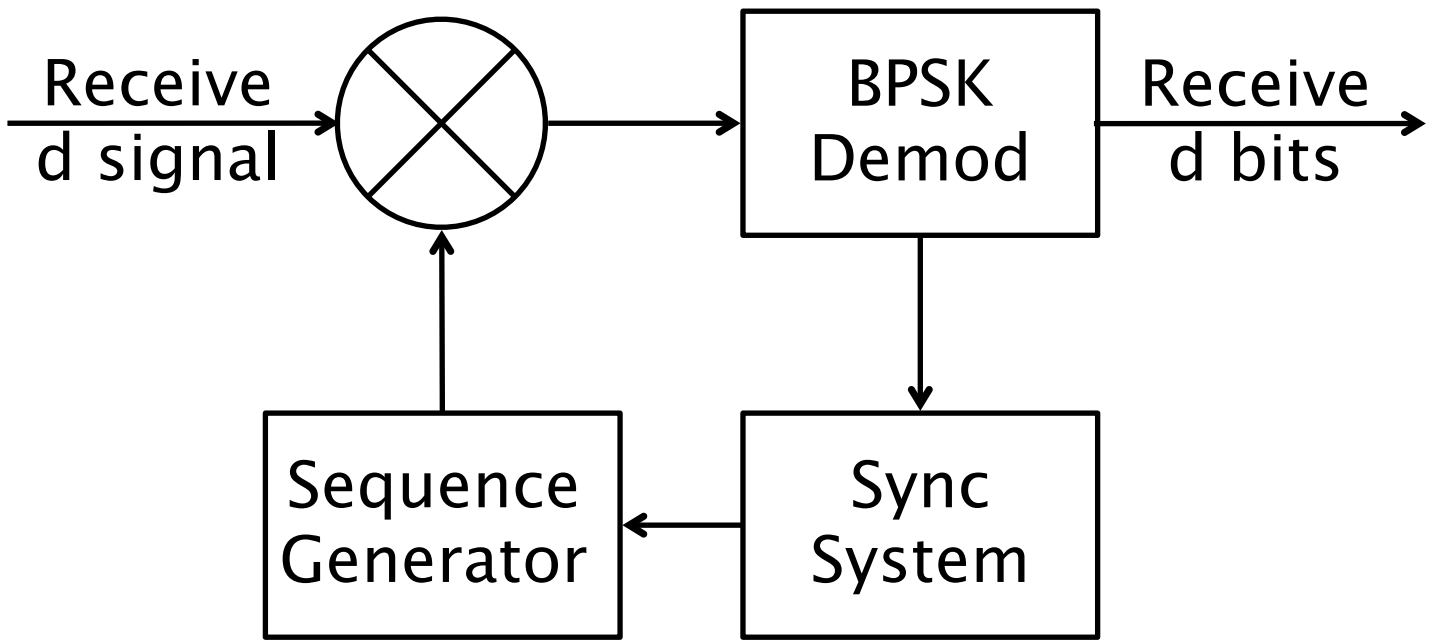
Direct-Sequence Spread

BPSK or DBPSK transmitter



Direct-Sequence Spread

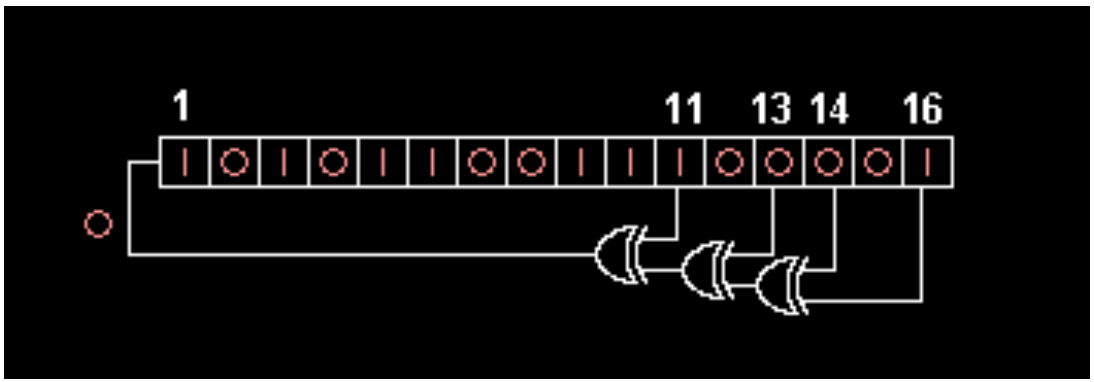
BPSK or DBPSK receiver



Direct-Sequence Spread

How are the codes generated?

Linear Feedback Shift Registers - LFSR

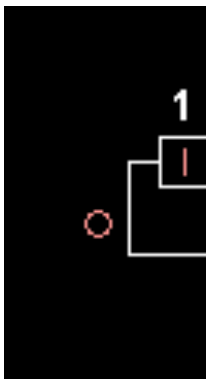



```
LFSR::LFSR(polynomial gen, polynomial init) : nreg(gen.size() - 1),
                                              generator(gen),
                                              _init(init)
{
    if (nreg != init.size())
        throw std::runtime_error("Initalized wrong number of registers...");

    // Set the initial state
    for (polynomial::iterator it = init.begin(); it != init.end(); ++it) {
        state.push_back(*it);
    }
}
```

Direct-Sequence Spread

How are th

Linear Feed - LFSR

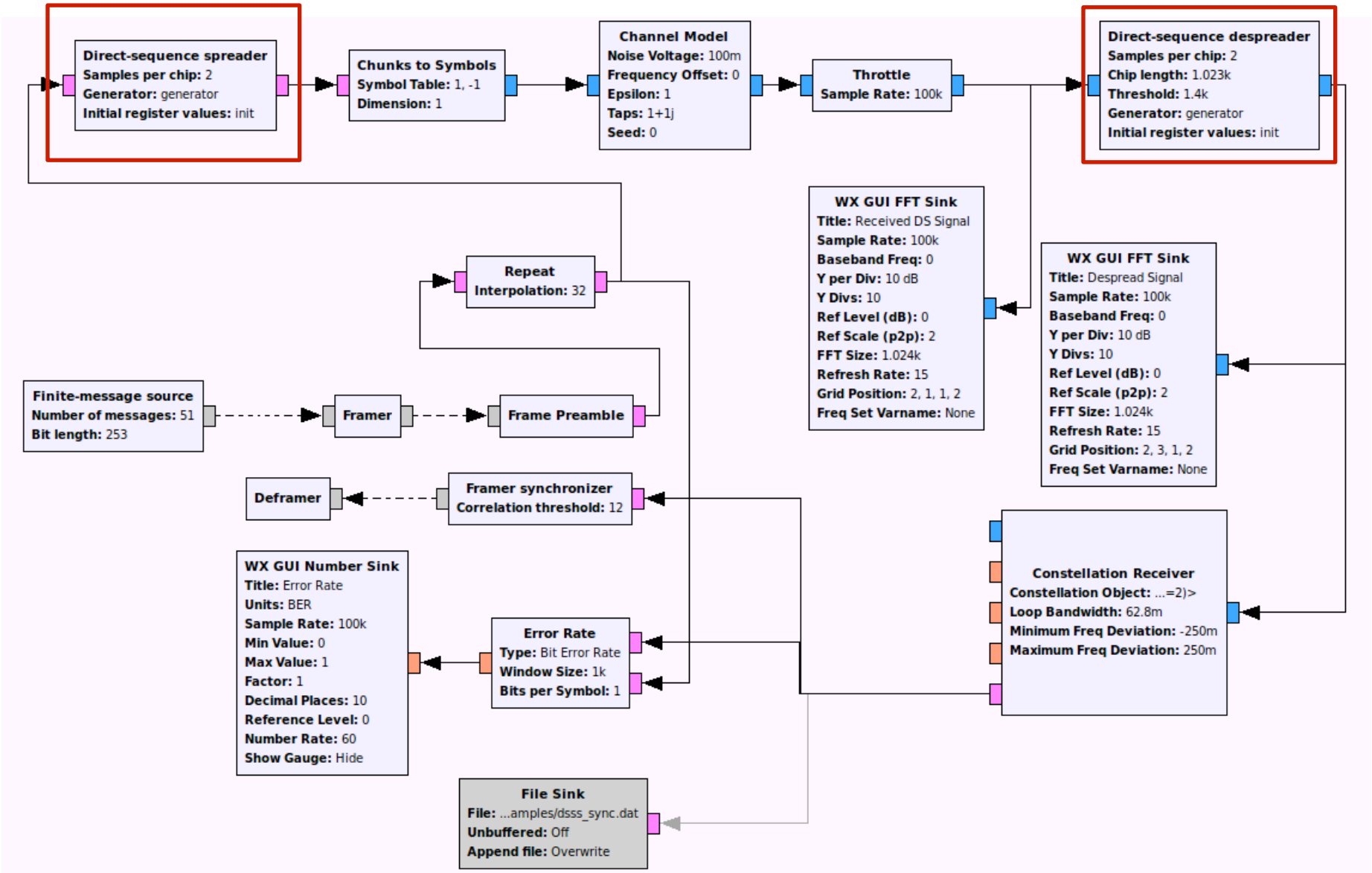


Bits	Feedback polynomial	Period
n		$2^n - 1$
2	$x^2 + x + 1$	3
3	$x^3 + x^2 + 1$	7
4	$x^4 + x^3 + 1$	15
5	$x^5 + x^3 + 1$	31
6	$x^6 + x^5 + 1$	63
7	$x^7 + x^6 + 1$	127
8	$x^8 + x^6 + x^5 + x^4 + 1$	255
9	$x^9 + x^5 + 1$	511
10	$x^{10} + x^7 + 1$	1023
11	$x^{11} + x^9 + 1$	2047
12	$x^{12} + x^{11} + x^{10} + x^4 + 1$	4095
13	$x^{13} + x^{12} + x^{11} + x^8 + 1$	8191
14	$x^{14} + x^{13} + x^{12} + x^2 + 1$	16383
15	$x^{15} + x^{14} + 1$	32767
16	$x^{16} + x^{14} + x^{13} + x^{11} + 1$	65535
17	$x^{17} + x^{14} + 1$	131071
18	$x^{18} + x^{11} + 1$	262143
19	$x^{19} + x^{18} + x^{17} + x^{14} + 1$	524287

```
LFSR::LFSR(polyno

{
    if (nreg != i
        throw std

    // Set the in
    for (polynomi
        state.pus
    }
}
```

```
ers...");

++it) {
```

DSSS GNU Radio

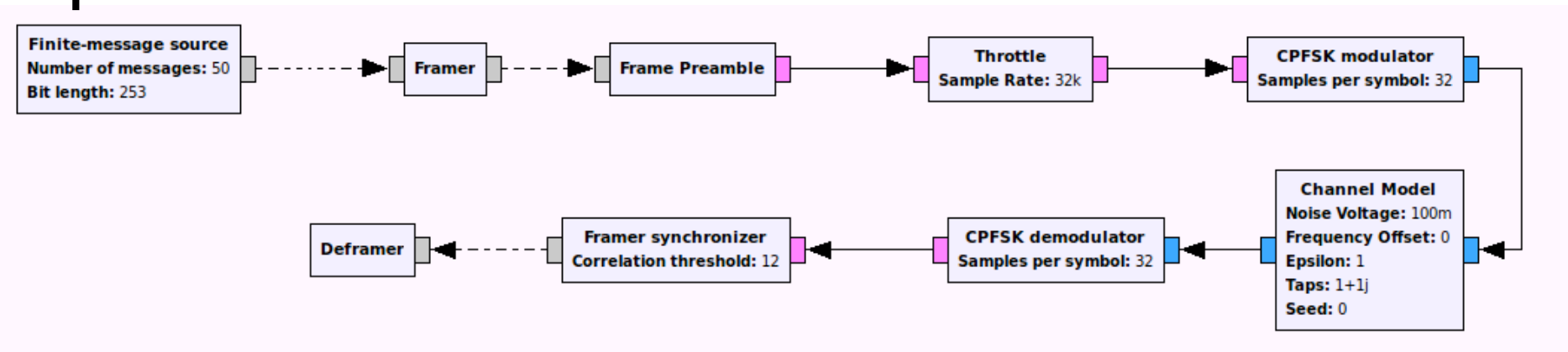


DSSS GNU Radio

DSSS in action…

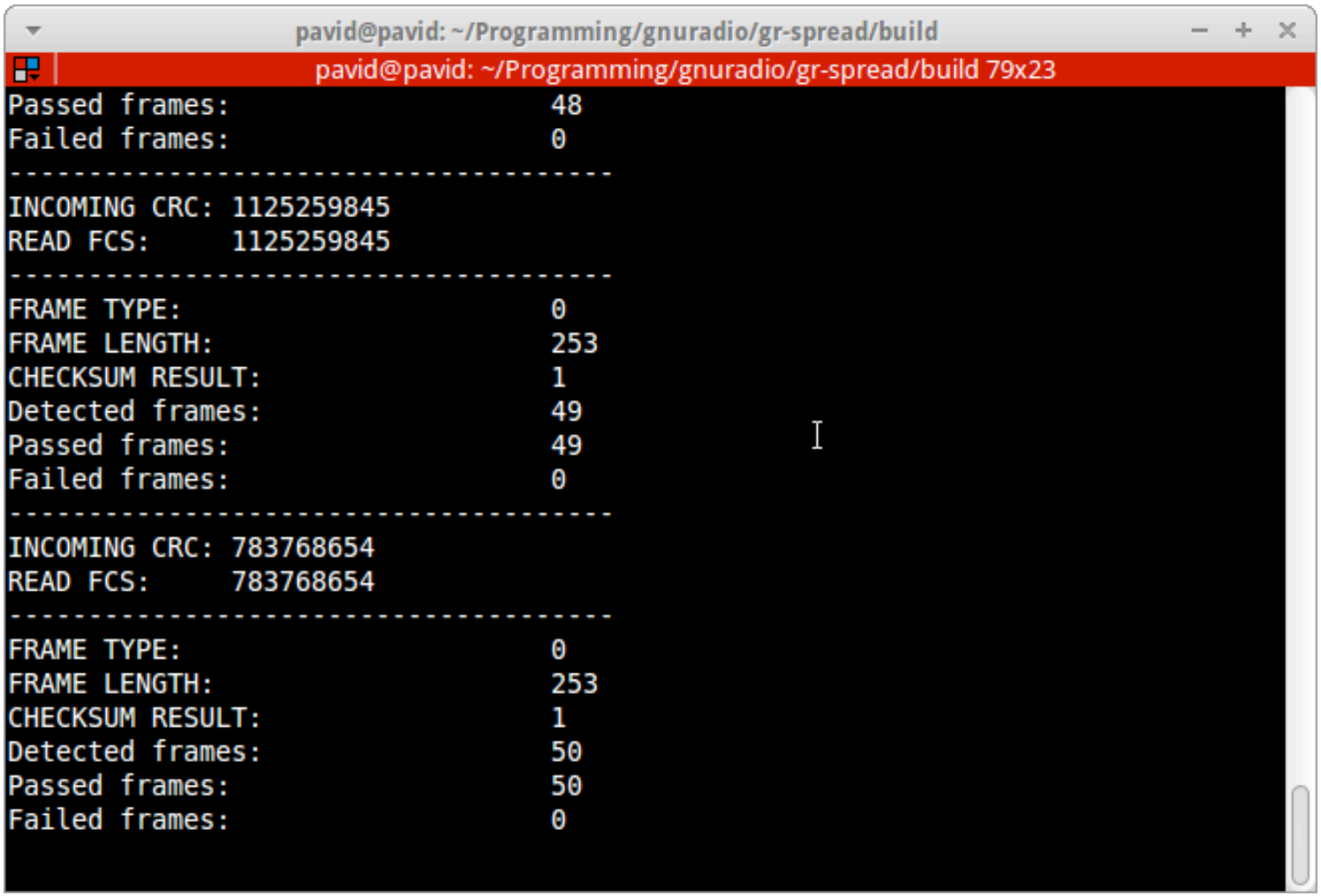
GNU Radio Tips & Tricks

Message passing can be used to construct and transport PDUs between packet-oriented blocks



GNU Radio Tips & Tricks

Framing / deframing



GNU Radio Tips & Tricks

Framing / deframing

```
struct frame_header
{
    uint8_t type;
    uint8_t dst;
    uint8_t src;
    uint8_t len;
} __attribute__((packed));
```

GNU Radio Tips & Tricks

Framing / deframing

```
void framer_impl::create_frame(const char *pdu_data, const int pdu_len)
{
    // Create the frame header
    // TODO: Create some higher MAC layer for multiple radios

    frame_header header;

    header.type = 0x00;
    header.dst = 0x00;
    header.src = 0x00;

    header.len = pdu_len;

    // Copy the data into the buffer
    frame_len = 4 + pdu_len + 4; // header + data + fcs
    frame_data = new char[frame_len];

    std::memcpy(frame_data, &header, sizeof(frame_header));
    std::memcpy(frame_data + 4, pdu_data, pdu_len);

    // Create the checksum
    boost::crc_32_type data;
    data.process_bytes(frame_data, 4 + pdu_len);

    uint32_t fcs = data.checksum();
    /*
    std::cout << "====================================" << std::endl;
    std::cout << "FRAMER OUT FCS:    " << fcs << std::endl;
    std::cout << "FRAMER OUT LENGTH: " << pdu_len << std::endl;
    */

    std::memcpy(frame_data + 4 + pdu_len, &fcs, 4);
}
```

```
struct frame_header
{
    uint8_t type;
    uint8_t dst;
    uint8_t src;
    uint8_t len;
} __attribute__((packed));
```

GNU Radio Tips & Tricks

Framing / deframing

```
void framer_impl::create_frame(const char *pdu_data, const int pdu_len)
{
    // Create the frame header
    // TODO: Create some higher MAC layer for multiple radios

    frame_header heade

    header.type = 0x00
    header.dst = 0x00;
    header.src = 0x00;

    header.len = pdu_l

    // Copy the data i
    frame_len = 4 + pdu_len + 4; // header + data + fcs
    frame_data = new char[frame_len];

    std::memcpy(frame_data, &header, sizeof(frame_header));
    std::memcpy(frame_data + 4, pdu_data, pdu_len);

    // Create the checksum
    boost::crc_32_type data;
    data.process_bytes(frame_data, 4 + pdu_len);

    uint32_t fcs = data.checksum();
    /*
    std::cout << "====================================" << std::endl;
    std::cout << "FRAMER OUT FCS:    " << fcs << std::endl;
    std::cout << "FRAMER OUT LENGTH: " << pdu_len << std::endl;
    */

    std::memcpy(frame_data + 4 + pdu_len, &fcs, 4);
}
```

```
// Create a frame
create_frame(pdu, msg_len);
pmt::pmt_t blob = pmt::make_blob(frame_data, frame_len);

// Publish the message
message_port_pub(pmt::mp("out"), blob);

delete[] frame_data;
frame_data = NULL;
```

```
struct frame_header
{
    uint8_t type;
    _t dst;
    _t src;
    _t len;
bute__((packed));
```

GNU Radio Tips & Tricks

Framing / deframing

```
void framer_impl::create_frame(const char *pdu_data, const int pdu_len)
```

```
struct frame_header
```

```
packed));
```

```
// Insert the synchronization bits
if (msg_queue.size() > 0 && d_offset == 0) {

    pmt::pmt_t msg = msg_queue.front();

    msg_queue.pop();

    if (pmt::is_blob(msg)) {
        int blob_size = pmt::blob_length(msg);
        const char *blob = static_cast<const char *>(pmt::blob_data(msg));

        // Create buffer to hold samples
        buffer_length = sizeof(zero_pad) + sizeof(sync_bits) + 8 * blob_size;
        buffer = new char[buffer_length];

        // Copy the frame into the buffer
        std::memcpy(buffer, zero_pad, sizeof(zero_pad));
        std::memcpy(buffer + sizeof(zero_pad), sync_bits, sizeof(sync_bits));

        // Unpack the blob bytes to the buffer
        unpack_bytes(buffer + sizeof(zero_pad) + sizeof(sync_bits), blob, blob_size);
    }

}
```

```
     */

    std::memcpy(frame_data + 4 + pdu_len, &fcs, 4);
}
```

GNU Radio Tips & Tricks

Framing / deframing

```
void framer_impl::create_frame(const char *pdu_data, const int pdu_len)
```

```
struct frame_header
```

```
packed));
```

```
// Insert the synchronization bits
if (msg_queue.size() > 0 && d_offset == 0) {

    pmt::pmt_t msg = msg_queue.front();

    msg_queue.pop();

    if (pmt::is_blob(msg)) {
        int blob_size = pmt::blob_length(msg);
        const char *blob = static_cast<const char *>(pmt::blob_data(msg));

        // Create buffer to hold samples
        buffer_length = sizeof(zero_pad) + sizeof(sync_bits) + 8 * blob_size;
        buffer = new char[buffer_length];

        // Copy the frame into the buffer
        std::memcpy(buffer, zero_pad, sizeof(zero_pad));
        std::memcpy(buffer + sizeof(zero_pad), sync_bits, sizeof(sync_bits));

        // Unpack the blob bytes to the buffer
        unpack_bytes(buffer + sizeof(zero_pad) + sizeof(sync_bits), blob, blob_size);
    }

}
```

```
    */

    std::memcpy(frame_data + 4 + pdu_len, &fcs, 4);
}
```

GNU Radio Tips & Tricks

Framing / deframing

```
void framer_impl::create_frame(const char *pdu_data, const int pdu_len)
```

```
struct frame_header
```

```
// Insert the synchronization bits
if (msg_queue.size() > 0 && d_offset == 0) {

    pmt::pmt_t msg = msg_queue.front();

    msg_queue.pop();

    if (pmt::is_blob(msg)) {
        int blob_size = pmt::blob_length(msg);
        const char *blob = static_cast<const char *>(pmt

        // Create buffer to hold samples
        buffer_length = sizeof(zero_pad) + sizeof(sync_b
        buffer = new char[buffer_length];

        // Copy the frame into the buffer
        std::memcpy(buffer, zero_pad, sizeof(zero_pad));
        std::memcpy(buffer + sizeof(zero_pad), sync_bits

        // Unpack the blob bytes to the buffer
        unpack_bytes(buffer + sizeof(zero_pad) + sizeof(
    }

}
```

```
for (int i = 0; i < noutput_items; i++) {
    // The incoming bytes are unpacked
    // Pack them into a header object
    if (d_offset < 32) {
        header_bytes[d_offset / 8] |= (in[i] << (d_offset % 8));
    }

    // We're done with the header
    // and now know the length
    if (d_offset == 31) {
        std::memcpy(&header, header_bytes, 4);

        frame_data = new char[4 + int(header.len) + 4]; // header + data + fcs
        std::memcpy(frame_data, &header, 4);

    } else if (d_offset >= 32 && d_offset < (64 + 8 * int(header.len))) {

        frame_data[d_offset / 8] |= (in[i] << (d_offset % 8));

    }

    if (d_offset == (63 + 8 * int(header.len))) {

        // We completed the PDU - pack it into a blob
        pmt::pmt_t blob = pmt::make_blob(frame_data, 4 + int(header.len) + 4);
        message_port_pub(pmt::mp("out"), blob);

        frame_found = false;
        return i + 1;
    }

    d_offset++;
```

```
    */

    std::memcpy(frame_data + 4 + pdu_len, &fcs, 4);
}
```

GNU Radio Tips & Tricks

Framing / deframing

```
void framer_impl::create_frame(const char *pdu_data, const int pdu_len)
```

```
struct frame_header
```

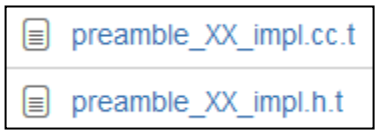
```
// Insert the synchronization bits
if (msg_queue.size() > 0 && d_offset == 0) {

    pmt::pmt_t msg = msg_queue.front();

    msg_queue.pop();

    if (pmt::is_blob(msg)) {
        int blob_size = pmt::blob_length(msg);
        const char *blob = static_cast<const char *>(pmt

        // Create buffer to hold samples
        buffer_length = sizeof(zero_pad) + sizeof(sync_b
        buffer = new char[buffer_length];

        // Copy the frame into the buffer
        std::memcpy(buffer, zero_pad, sizeof(zero_pad));
        std::memcpy(buffer + sizeof(zero_pad), sync_bits

        // Unpack the blob bytes to the buffer
        unpack_bytes(buffer + sizeof(zero_pad) + sizeof(
    }

}
```

```
for (int i = 0; i < noutput_items; i++) {
    // The incoming bytes are unpacked
    // Pack them into a header object
    if (d_offset < 32) {
        header_bytes[d_offset / 8] |= (in[i] << (d_offset % 8));
    }

    // We're done with the header
    // and now know the length
    if (d_offset == 31) {
        std::memcpy(&header, header_bytes, 4);

        frame_data = new char[4 + int(header.len) + 4]; // header + data + fcs
        std::memcpy(frame_data, &header, 4);

    } else if (d_offset >= 32 && d_offset < (64 + 8 * int(header.len))) {

        frame_data[d_offset / 8] |= (in[i] << (d_offset % 8));

    }

    if (d_offset == (63 + 8 * int(header.len))) {

        // We completed the PDU - pack it into a blob
        pmt::pmt_t blob = pmt::make_blob(frame_data, 4 + int(header.len) + 4);
        message_port_pub(pmt::mp("out"), blob);

        frame_found = false;
        return i + 1;
    }

    d_offset++;
```

```
     */

    std::memcpy(frame_data + 4 + pdu_len, &fcs, 4);
}
```

GNU Radio Tips & Tricks

Templates generated by cmake can be used to make blocks for multiple types




```
/*
 * The private constructor
 */
@NAME_IMPL@::@NAME_IMPL@(const int symbols_per_chip,
                         const int chips,
                         std::vector<gr_complex> pattern)
  : gr::block("preamble",
          gr::io_signature::make(1, 1, sizeof(@TYPE@)),
          gr::io_signature::make(1, 1, sizeof(@TYPE@))),
    _sym_chip(symbols_per_chip * chips),
    _pattern(pattern),
    d_symbols_left(0),
    d_offset(0)
{}

/*
 * Our virtual destructor.
 */
@NAME_IMPL@::~@NAME_IMPL@()
{
}
```

```
macro(expand_cc_h_impl root)
  #make a list of all the generated files
  unset(expanded_files_cc_impl)
  unset(expanded_files_h_impl)
  foreach(sig ${ARGN})
    string(REGEX REPLACE "X+" ${sig} name ${root})
    list(APPEND expanded_files_cc_impl ${CMAKE_CURRENT_BINARY_DIR}/${name}_impl.cc)
    list(APPEND expanded_files_h_impl ${CMAKE_CURRENT_BINARY_DIR}/${name}_impl.h)
    list(APPEND expanded_files_h ${CMAKE_CURRENT_BINARY_DIR}/../include/${name}.h)
  endforeach(sig)

  #create a command to generate the _impl.cc files
  add_custom_command(
    OUTPUT ${expanded_files_cc_impl}
    DEPENDS ${CMAKE_CURRENT_SOURCE_DIR}/${root}_impl.cc.t
    COMMAND ${PYTHON_EXECUTABLE} ${PYTHON_DASH_B}
    ${CMAKE_CURRENT_BINARY_DIR}/generate_helper.py
    ${root} ${root}_impl.cc.t ${ARGN}
  )

  #create a command to generate the _impl.h files
  add_custom_command(
    OUTPUT ${expanded_files_h_impl}
    DEPENDS ${CMAKE_CURRENT_SOURCE_DIR}/${root}_impl.h.t
    COMMAND ${PYTHON_EXECUTABLE} ${PYTHON_DASH_B}
    ${CMAKE_CURRENT_BINARY_DIR}/generate_helper.py
    ${root} ${root}_impl.h.t ${ARGN}
  )

  #make _impl.cc source files depend on headers to force generation
  set_source_files_properties(${expanded_files_cc_impl}
    PROPERTIES OBJECT_DEPENDS "${expanded_files_h_impl}"
  )

  #make _impl.h source files depend on headers to force generation
  set_source_files_properties(${expanded_files_h_impl}
    PROPERTIES OBJECT_DEPENDS "${expanded_files_h}"
  )

  #install rules for the generated cc files
  list(APPEND generated_sources ${expanded_files_cc_impl})
endmacro(expand_cc_h_impl)
```

Spread Spectrum toolkit

Created blocks	Still in Development
Finite Message Source	Improved FHSS Acquisition
Framer	Working tracking for DSSS
Deframer	Support more than CPFSK/BPSK
CPFSK Modulator	
CPFSK Demodulator	
Frame Preamble Inserter	
Frame Synchronizer	
FHSS Spreader	
FHSS Despreader	
DSSS Spreader	
DSSS Despreader	

End of talk!

Thanks having me!

Any questions?