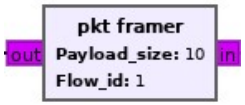
GNU RADIO BLOCK CREATION

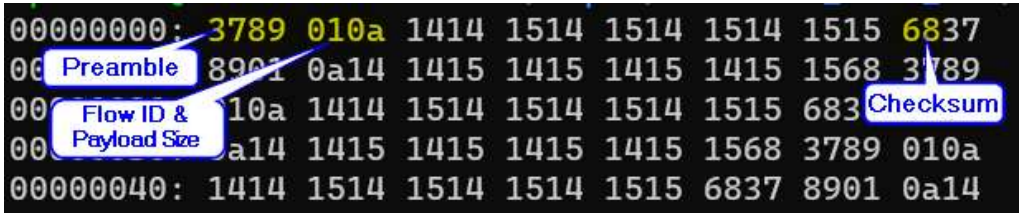
Igor Povarich

GOALS

- Create 2 GNURadio blocks
  - Transmitter block
    - Preamble (fixed)
    - Flow ID (user configurable)
    - User Data (user configurable)
    - Checksum (calculation)
  - Receiver block
    - Catch the preamble to identify the start of the data stream
    - Filter based on the flow ID
    - Check for errors using the checksum
    - Pass user data, if validated

TRANSMITTER BLOCK





Packet Framer File Sink output

```
// Simple 8bit Checksum
// Modulo 152 is used b/c the max value encoded into char is 152
int checksum = (b01 + b23 + b4 + b5 + sum) % 152; // calc
*out = checksum;
out++;
```

- Transmitter block that writes a packed byte stream
- Encodes Preamble identification of start of stream (0x37 and 0x89)
- Encodes Flow ID for filtering and payload size for parsing the user data
- Calculates and encodes a simple 8-bit checksum for error identification
  - The value 152 was used for the modulo calculation to avoid overflows because that seemed to be the max value a C++ char could encode

RECEIVER BLOCK

- Performs the functions of the transmitter block in reverse, which is more involved
- Speculatively builds 2 8-bit bytes (16 bits) to look for the preamble
  - Checked as matched strings
  - If the assembled bytes don't match the preamble, it steps back 15 steps (+1 from starting point)
- Other than preamble, data is reconstructed one bit a time, then packed into a bitset for processing

```
// Extract four bytes of data to look for preamble
for(int j = 0; j < 4; j++){
    int x0(*in); in++;
    int x1(*in); in++;
    int x2(*in); in++;
    int x3(*in); in++;
    pream_chk[j] = to_string(x0)+to_string(x1)+to_string(x2)+to_string(x3);
}
```

```
} else {
    in = in - 15; // go back to just after the first index
}
```

```
// extract flow_id
in = in + 4; // skip blank bytes
int x0(*in); in++;
int x1(*in); in++;
int x2(*in); in++;
int x3(*in); in++;
bitset<4> fid_data(to_string(x0)+to_string(x1)+to_string(x2)+to_string(x3));
fid_num = fid_data.to_ulong();
cout << "Flow ID: " << fid_num << " ";
```

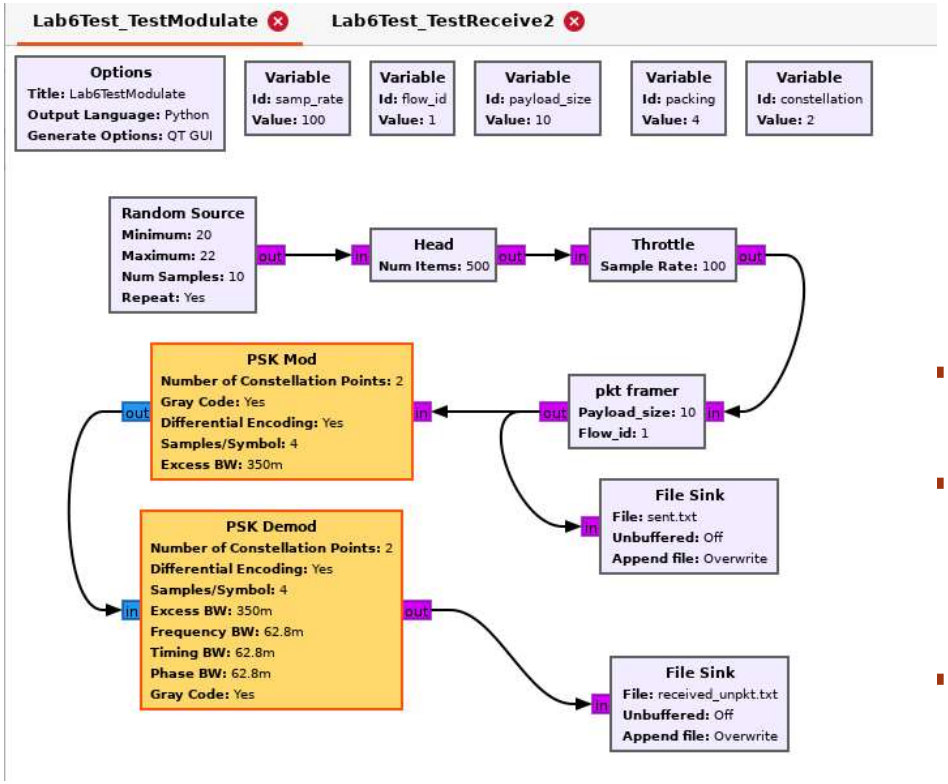
RECEIVER BLOCK

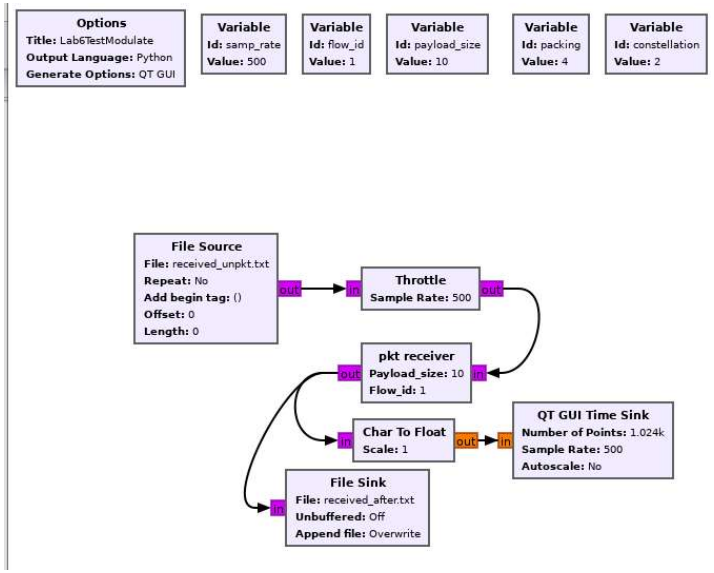
```
if (pream_chk == pream_valid){
    cout << endl;
    cout << "***************************" << endl;
    cout << "Index: " << i << endl;
    cout << "Preamble Matched!! <<<<---" << endl;
    cout << "***************************" << endl;


  // compare flow id and output if correct
  if(fid_num==_flow_id){
      cout << "Flow ID Matched!" << endl;


      pkt_cnt++; // increment packet count
      valid_stream = true; // only true if both
```

```
// checksum calculation
// preamble+flow_id+payload_size+user_data
int chksum_calc = (192 + fid_num + pyld_data.to_ulong() + user_sum) % 152;
cout << "checksum_calc: " << chksum_calc << endl;
if(chksum_data == chksum_calc){
    cout << "Checksums Matched!!" << endl;
} else {
    cout << "Warning: Checksums NOT Matched!!" << endl;
}
```

- If preamble AND flow id are matched, the valid_stream bool is set to true, which allows rest of the code to run
  - If flow id is not validated, it returns to trying to find the next preamble
- Next, payload_size, user data, and checksum are extracted
- The checksum is calculated from the extracted data and matched against the extracted checksum
  - If they don't match a warning is sent to the console
- For a valid stream, all relevant data is passed through to the output

TEST FLOWGRAPHS





- Two main GNURadio Companion graphs are used for testing
- The first passes the pkt_framer output through a PSK Mod/Demod block combo to a file sink "received_unpkt.txt"
- The 2nd receives the data from the file sink, processes it, and passes on valid data to another file sink

TESTING

Main console output from the pkt_receiver block includes notifications about matching and user data

```
****************************
Index: 164
Preamble Matched!! <<<<---
****************************
Flow ID: 1 Flow ID Matched!
User Data0: 20
User Data1: 20
User Data2: 21
User Data3: 20
User Data4: 21
User Data5: 20
User Data6: 21
User Data7: 20
User Data8: 21
User Data9: 21
checksum_data: 104
checksum_calc: 104
Checksums Matched!!
```

If the flow_id is mismatched, everything after the preamble match is skipped

```
****************************
Flow ID: 1
****************************
Index: 1036
Preamble Matched!! <<<<---
****************************
Flow ID: 1
****************************
Index: 1133
Preamble Matched!! <<<<---
****************************
Flow ID: 1
****************************
Index: 1230
Preamble Matched!! <<<<---
****************************
```

CHALLENGES AND LEARNING

- Had a lot of difficulty with the exit conditions for the blocks in GRC (behavior when attempting to stop the simulation)
  - Made testing more difficult and time-consuming
  - Reason for choosing to have two separate graphs with a file sink in-between
- Learned a lot about working with low-level binary data in GNURadio OOT blocks, especially how packing affects the way a data stream is processed or encoded
- Re-learned some of the data type and pointer operations in C++
- Like everyone else, I was confused by the single-bit output from the modulator blocks