CS-330 Project
A complete BPSK,QPSK telecommunication system
Phase A

December 10, 2023

General Information

The goal of the project is to construct a fully functional telecommunication system using PSK modulation and supporting variable length frames. The framing used, follows the IEEE 802.15.4 standard which is a very popular and established IoT protocol.

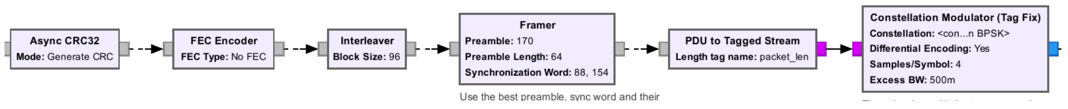
The provided flowgraphs supports burst tagging and can directly be used with real SDR hardware.

The project is divided into multiple phases, each one implementing a processing task of the transceiver. The project has been designed in gradually increasing difficulty level. Except the framing creation and the framing synchronization blocks, all others do not affect the functionality of the project, only the performance of the transceiver. In other words, the project will be still able to operate even if you do not implement the interleaving and/or FEC processing tasks. Of course the RF performance will degrade significantly.

All of the required blocks have been already created for you, as well as the YAML files required for the GNU Radio Companion. All you have to do is just write the necessary code. The blocks as well as the transceiver flowgraph itself contain several comments. Please read them and make sure that you understand them!
The Out-Of-Tree module of GNU Radio that you will implement is in the Github repository here.
Getting started: clone locally and make sure you can install it on your machine.

May the code be with you!

1. Transmitter


Async CRC32
FEC Encoder
Interleaver
Framer
PDU to Tagged Stream
Constellation Modulator (Tag Fix)
Use the best preamble, sync word and their

Mode: Generate CRC
FEC Type: No FEC
Block Size: 96
Preamble: 170
Preamble Length: 64
Synchronization Word: 88, 154
Length tag name: packet_len
Constellation: <con...n BPSK>
Differential Encoding: Yes
Samples/Symbol: 4
Excess BW: 500m

Figure 1: Transmitter processing chain

The transmitter chain depicted in Figure 1 starts with the CRC32 block that appends a 4 byte CRC at the end of each PDU (Protocol Data Unit). This block is already implemented and you do not have to do any special for it. It will be used to check the validity of a received frame at the receiver chain.

The next block performs FEC (Forward Error Correction). The basic implementation that you will develop in later phases should support Hamming (3,1). Until you implement the code, use the *No FEC* option, so the incoming PDUs can be forwarded unaltered to the next block.

After the FEC, the TX chain continues with the block interleaver. The supported block sizes are 96, 192 and 384 bits. The default implementation of this block copies the input PDU to the output, so even if you do not implement this block the transmitter will be able to operate. Again, in this phase you do not have to implement the interleaver.

The Framer block that follows is responsible to add a preamble, a synchronization word, as well as the frame length for each received PDU. The resulting frames are then converted from the message passing interface to a stream using the *PDU to Tagged Stream* block. The *Constellation Modulator* block converts the bit stream into a stream of complex constellation points. You can switch the modulation used, by properly altering the **constellation** variable. Occupied bandwidth can also be altered by adjusting the interpolation factor **sps**. Apart from these variables, you do not need to change anything else. The flowgraph will adapt auto-magically!

1.1 Phase 1: Framing 10%

As mentioned before the framer is responsible for inserting a repeated preamble, a synchronization word and the frame length in front of every PDU. The *Framer* block accepts three parameters:

1. **Preamble:** A byte pattern that will be repeated multiple times. The byte pattern should be something alternating. Common patterns are 0xAA (0b101010) or 0x33 (0b00110011)

2. **Preamble Length:** How many times to repeat the preamble. Longer preambles increase sensitivity but reduce the effective data rate

3. **Synchronization Word:** A byte array with an arbitrary synchronization word. Again longer synchronization words increase sensitivity, but reduce the effective data rate. Items of the synchronization word should have low correlation for best performance

The repeated preamble allows the receiver to identify that possibly a frame is about to be received. The need for a repeated pattern is to give the necessary time to several receiver components, like AGC (automatic gain control) or the frequency offset tracker, to settle. So even if the receiver looses portion of the preamble due to the settling time, it will still be able to understand that a frame transmission is active. However, due to its repeated nature it is very difficult for a receiver to recover the exact start of the frame payload and identify possible phase shifts. For this reason, most systems use a preamble followed by a synchronization word. A known synchronization word will allow the receiver to retrieve the start of the payload, as well as identify possible rotation of the PSK constellation points. Moreover, in order for the receiver to retrieve the size of the received frame, the frame contains a two byte field with the frame length in bytes. This field should be in Network Byte Order (Big endian). Figure 2 shows the resulting structure of the frame.

Provide the implementation of the aforementioned framing scheme at the *lib/framer_impl.cc* file. For simplicity you can assume that **unencoded** PDUs may be up to 2048 bytes long.

1.2 Testing

In the *examples* directory of the **gr-cs330_project2023** repository you can find the *psk_tranceiver.grc* flowgraph. This is the entire transceiver system. At this point, you can

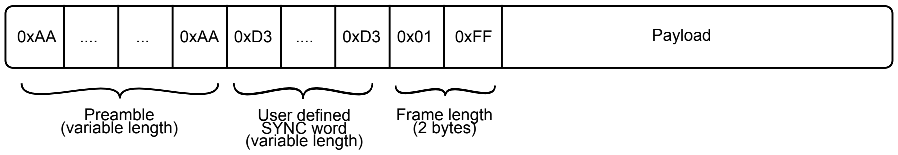
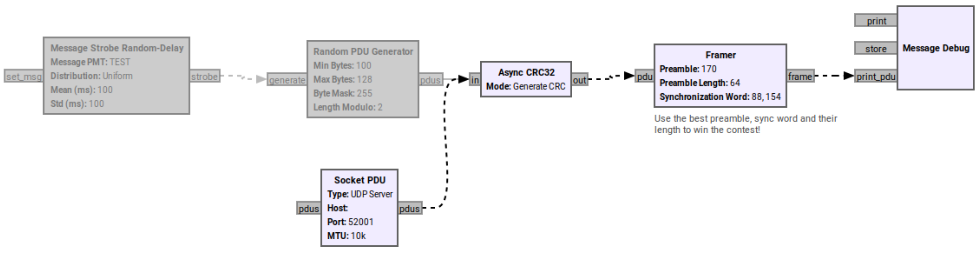

0xAA
....
...
0xAA
0xD3
....
0xD3
0x01
0xFF
Payload
Preamble (variable length)
User defined SYNC word (variable length)
Frame length (2 bytes)

Figure 2: System framing

use a simpler flowgraph to help you debug your block, the *test_framer.grc* flowgraph. Keep in mind that instead of random PDU messages you can send to your flowgraph your own messages using the *Socket PDU* message source block and the *netcat* Linux utility. An example of such a flowgraph can be found in Figure 3.


Message Strobe Random-Delay
Message PMT: TEST
Distribution: Uniform
Mean (ms): 100
Std (ms): 100

Random PDU Generator
Min Bytes: 100
Max Bytes: 128
Byte Mask: 255
Length Modulo: 2

Async CRC32
Mode: Generate CRC

Framer
Preamble: 170
Preamble Length: 64
Synchronization Word: 88, 154

Use the best preamble, sync word and their length to win the contest!

Socket PDU
Type: UDP Server
Host:
Port: 52001
MTU: 10k

Figure 3: Simplified transmitter