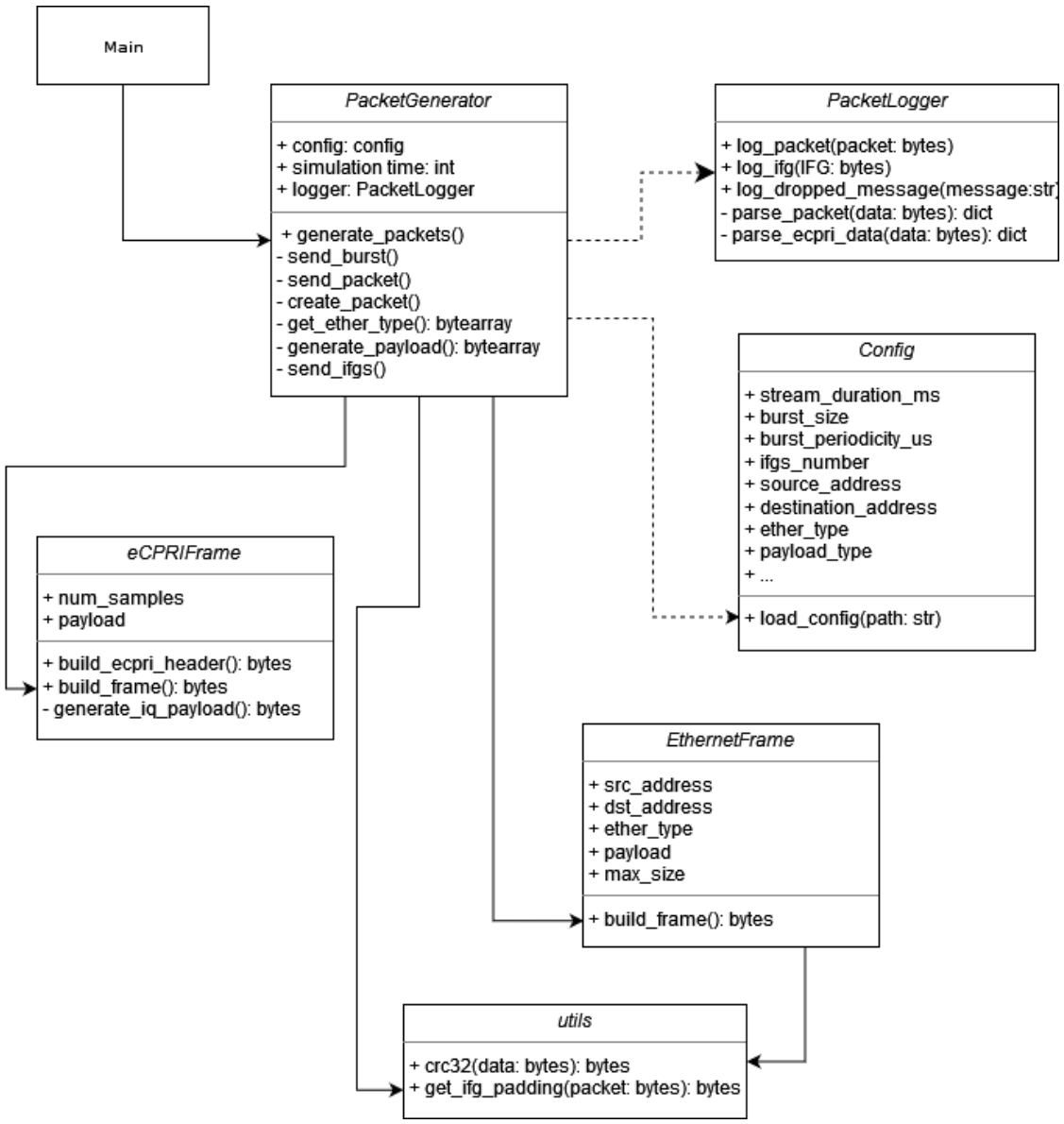
Design and Implementation Overview

Project Overview

This project is designed to simulate the generation of **Ethernet packets** and **eCPRI IQ Message Type 0 packets**. The program allows for the configuration of various parameters, such as packet size, burst periodicity, and the number of IQ samples, while ensuring proper 4-byte alignment of the generated packets. The project also generates packets in bursts, adhering to time and alignment constraints, and allows output of the generated packets in JSON format.

Class structure diagram



Key Components

1. `main.py`

This is the **entry point** of the program. It:

- Loads the configuration from config.txt.
- Instantiates the PacketGenerator class.
- Starts the packet generation process.

2. `config.py`

The Config class handles reading and parsing the configuration file (config.txt). It is responsible for:

- Loading parameters such as STREAM_DURATION_MS, BURST_SIZE, BURST_PERIODICITY_US, IFGs_NUMBER, SOURCE_ADDRESS, and others.
- Providing the configuration values to the PacketGenerator.

3. `packet.py`

This is the core of the packet generation logic. It contains the PacketGenerator class, which:

- **Generates Ethernet and eCPRI packets** based on the configuration.
- **Handles burst generation**, ensuring each burst sends a specified number of packets (BURST_SIZE) and respects the burst periodicity (BURST_PERIODICITY_US).
- **Manages 4-byte alignment** of packets by adding appropriate padding with IFGs (inter-frame gaps).
- **Writes generated packets** to a JSON file (packets.json), including their structure and contents.
- **Simulates time** during packet generation to match the streaming duration (STREAM_DURATION_MS).

  Key Methods:

  - **generate_packets()**: Generates packets in bursts and logs them in the packets.json file, simulating the time for each burst and handles time limit for currently generating packets.
  - **create_packet()**: Generates a single Ethernet packet containing payload either random bytes or an eCPRI frame.
  - **send_ifgs()**: Sends IFGs after each burst to ensure proper spacing and alignment.

4. ethernet_frame.py

This module defines the EthernetFrame class, which:

- Constructs Ethernet frames with fields like **source MAC address**, **destination MAC address**, **EtherType**, **payload**, and **CRC**.
- **Preamble and SOF** are also added as part of the Ethernet frame structure.

  Key Methods:

  - **build_frame()**: Builds the complete Ethernet frame, including the header data and payload with padding if needed for minimum packet length.

5. ecpri_frame.py

This module defines the eCPRIFrame class for constructing **eCPRI IQ Message Type 0 packets**. It:

- **Encapsulates the eCPRI frame** within an Ethernet frame.
- **Builds the eCPRI header** and appends the **IQ payload** (which can be random or fixed, as per the configuration).

  Key Methods:

  - **build_frame()**: Constructs the complete eCPRI frame with header and payload.
  - **build_ecpri_header()**: Builds the 4-byte eCPRI header, including message type and payload length as per the eCPRI configuration.

6. utils.py

This module contains utility functions such as:

- **CRC calculation**: Used to calculate the CRC value for Ethernet frames.
- **IFG padding calculation**: Used to calculate how many bytes of padding is needed to align each the packets to a multiple of 4.

7. config/config.txt

The configuration file is used to set parameters for packet generation. It includes:

- **STREAM_DURATION_MS**: Total streaming duration in milliseconds.
- **PACKET_TYPE**: regular ethernet packet or eCPRI packet.
- **BURST_SIZE**: Number of packets sent per burst.
- **BURST_PERIODICITY_US**: Periodicity between bursts in microseconds.
- **IFGs_NUMBER**: Number of IFG bytes inserted after CRC.
- **SOURCE_ADDRESS**, **DESTINATION_ADDRESS**, **ETHER_TYPE**: Addresses and EtherType for Ethernet frames.
- **PAYLOAD_TYPE**: Specifies if the payload is random or fixed.
- **MAX_PACKET_SIZE**: Maximum size of each packet.
- **IQ_SAMPLE_NUM**: Specifies the number of IQ samples in one eCPRI packet.

How It Works

1. Initialization

- The program starts by reading the configuration from config/config.txt.
- A PacketGenerator object is created using the loaded configuration.

2. Packet Generation

- The generate_packets() method is invoked, which begins a time simulation and generates packets according to the configuration.
- The program generates packets in **bursts**. Each burst contains a specified number of packets (BURST_SIZE), and bursts are sent at intervals defined by BURST_PERIODICITY_US.

3. Packet Structure

- Each Ethernet packet consists of:
    - **Preamble and SOF**: Standard Ethernet preamble (7 bytes) and start-of-frame delimiter (1 byte).
    - **Destination MAC Address**: 6 bytes.
    - **Source MAC Address**: 6 bytes.
    - **EtherType**: 2 bytes.
    - **Payload**: Random or fixed payload based on configuration with minimum length of 46 bytes (padding is added as necessary).
    - **CRC**: 4 bytes for error detection.
- For eCPRI packets, the Ethernet frame contains an **eCPRI header** and **IQ payload**.
- eCPRI IQ Message Type 0 Structure:
    - **Common Header (4 bytes)**:
        - **Protocol Revision and concatenation indicator** (1 byte): Indicates the version of the eCPRI protocol, and the C indicator which indicates if there is another eCPRI frame following this frame or not.
        - **Message Type** (1 byte): Indicates the message type. For **IQ Message Type 0**, this is **0**.
        - **Message Length** (2 bytes): Specifies the length of the payload in bytes, **excluding the header**.
    - **IQ Payload** (Variable length):
        - This is the actual IQ data. It contains a sequence of **I/Q samples**.
        - The payload size is specified by the **Message Length** field in the header.

4. Alignment and IFGs

- After generating each burst, the program ensures that packets are **4-byte aligned** by adding IFGs as necessary.
- The IFGs (30 bytes by default) are inserted between bursts to maintain proper spacing and alignment.

5. Simulation of Time

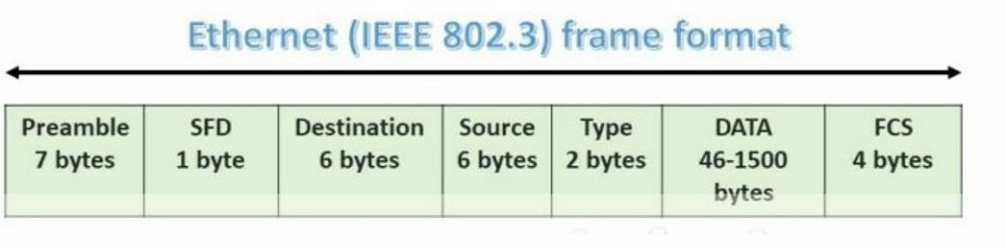
- The program simulates time progression, ensuring that the total generation duration matches the STREAM_DURATION_MS specified in the configuration.

6. Output

- The generated packets are written to a **JSON file (packets.json)**, which contains the full structure and content of each packet.

Sample output

- **Regular Ethernet packet**




```
{
        "preamble": "55555555555555",
        "SOF": "fd",
        "destination_adrs": "10:20:30:30:20:10",
        "source_adrs": "10:20:30:40:50:60",
        "ethertype/length": "0800",
        "data": "81e6106ccf20a9e511178470957586abeb0a5f45075d30b8bd92f4d507dbe091000000000000000000000000000000",
        "crc32": "064b54fa"
    }
```

- **eCPRI packet**

  eCPRI data encapsulated inside the Ethernet packet

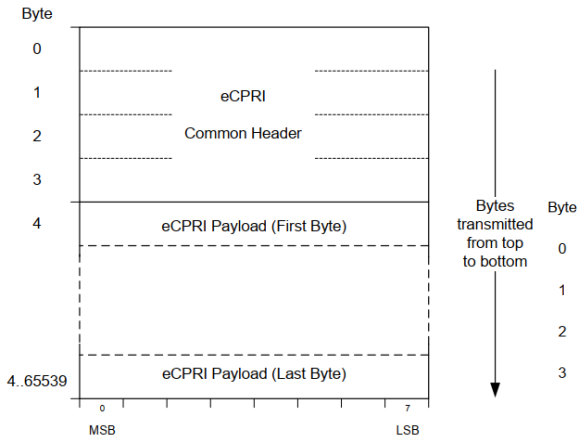


Figure 7: eCPRI message format

Figure 8: eCPRI Common Header format


```
{
        "preamble": "55555555555555",
        "SOF": "fd",
        "destination_adrs": "10:20:30:30:20:10",
        "source_adrs": "10:20:30:40:50:60",
        "ethertype/length": "aefe",
        "eCPRI data": {
            "version": 0,
            "message_type": 0,
            "message_length": 16,
            "data_payload": "d3077fec901c76d38a725fb9f0e56d7f",
            "padding": "00000000000000000000000000000000000000000000000000000000"
        },
        "crc32": "3c804c00"
    }
```