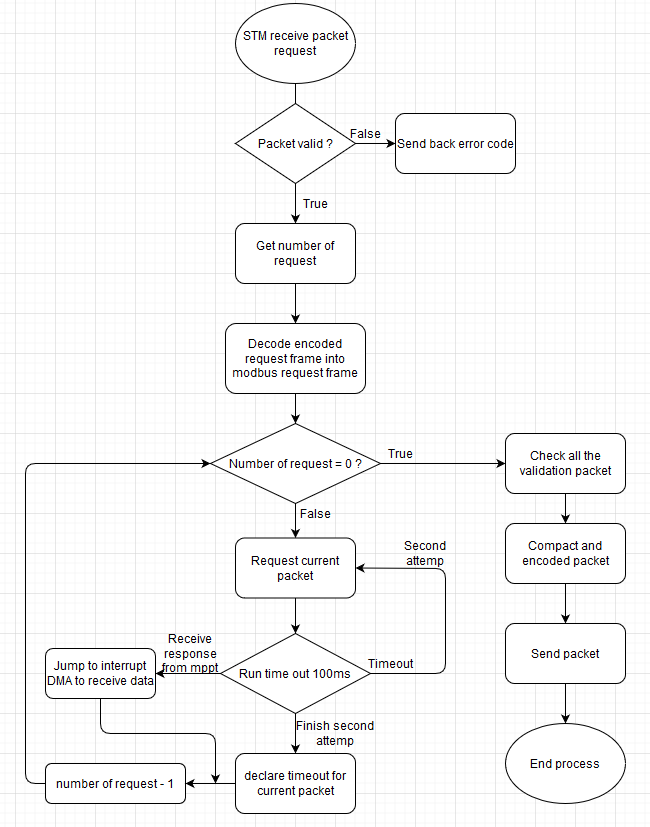
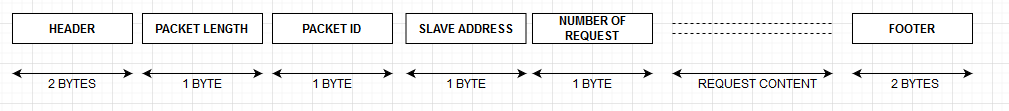
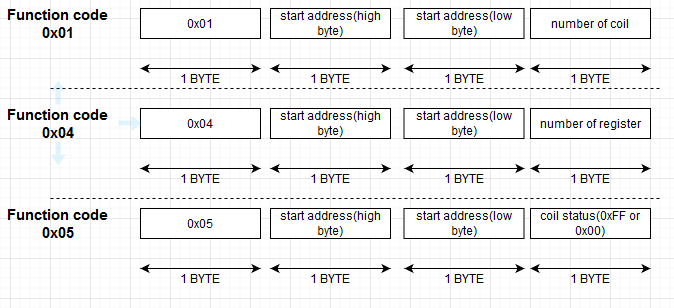
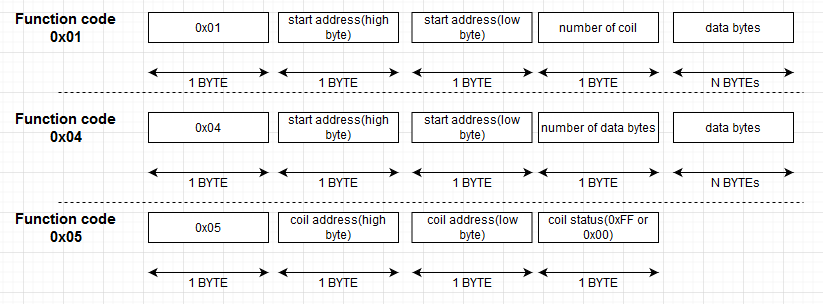
* Ngôn ngữ lập trình chủ yếu ở phần nhúng ta dùng C++. Với đặc trung thế mạnh OOP của C++ ta có thể giải quyết các vấn đề một cách có hệ thống hơn
* Ta viết nên các thư viện để handle những chuỗi request và response từ Modbus cũng như encode và decode các chuỗi byte theo định nghĩa của người lập trình. Từ đó rút ngắn chiều dài packet phải gửi không dây, tăng độ tin cậy của đường truyền hơn
* Sau đây là một sơ đồ giải thuật khi gửi một hệ các request xuống node phụ
* 
* Với ý tưởng thực hiện model trên được diễn tả như sau. Các data trên các thanh địa chỉ của Modbus thường nằm liên tục, tuy nhiên khi giám sát các thông số, yêu cầu chỉ cần lấy những thông số cần thiết để giám sát. Mà các địa chỉ thanh ghi các thông số này không liên tục nhau
* Từ đây ta có 2 hướng giải quyết
* Đọc hết tất cả những thanh ghi không cần thiết để có tính liên tục rồi gửi tất cả chúng về node chính để lọc => **lượng data dư nhiều, gây hao phí đường truyền**
* Đọc những thanh ghi cần thiết bằng cách request nhiều lần với mỗi lần ở những địa chỉ khác nhau => **có thể thực hiện nhiều function code trong một lần gửi, chiều dài data gửi về node chính giảm đi**
* Chính vì lý do này ta viết một thư viện để có thể xử lí tự động nhiều request. Đó cũng chính là nội dung của flowchart ở trên
* Để có thể tạo ra một frame truyền nhiều request gửi xuống node phụ, ta không thể ghép các frame Modbus của mỗi request vào với nhau rồi gửi. Như vậy node phụ cũng khó có thể tách các request đó ra và chiều dài chuỗi byte truyền trở nên bị dư. Chính vì thế ta phải encode nhiều frame request theo định dạng sau



* **Header, Footer** và **packet length** sẽ là những yếu tố chung để lọc ra một frame data hoàn chỉnh
* **Packet ID** là con số sẽ xác định chức năng của packet gửi tới. Ở đây chức năng của node phụ sẽ chỉ giao tiếp với MPPT nên sẽ có riêng một packet ID là 38. Ngoài ra còn có các packet ID cho các Error handler. Trong tương lai nếu mở rộng sẽ có các packetID cho các chức năng bật tắt IO riêng biệt. Với mỗi packet ID các data theo sau nó sẽ có các format khác nhau
* **Slave address** chính là địa chỉ của slave Modbus mà node phụ giao tiếp tới
* **Number of request** là số request tổng cộng
* **Request content** bao gồm nhiều frame request được encode với format được nêu như sau



* Hình trên là ví dụ cho ba function code 0x01, 0x04, 0x05
* Khi node phụ nhận được giả sử một chuỗi byte gồm ba request này, với từng frame request trên node phụ sẽ construct lên một object class. Từng object class tương ứng với một request sẽ tự tạo một frame request Modbus đầy đủ với CRC và sẽ hoạt động như flow chart ở mục trên
* Ví dụ như một frame: 0x04 0x31 0x00 0x0A thì node phụ sẽ decode thành chuỗi request đầy đủ 0x01 0x04 0x31 0x00 0x00 0x0A 0x7E 0xF1 và gửi xuống cho MPPT
* Như thấy ở trên thay vì gửi một chuỗi đầy đủ với chuỗi encoded ta đã tiết kiệm đi được 4 byte
* Và tương tự như vậy với chuỗi data response đóng gói để gửi về node chính ta cũng sẽ có kiểu encode như sau
* 
* Ví dụ cho ba function code 0x01, 0x04, 0x05
* Với kiểu trên, cho từng request ta sẽ tiết kiệm được 1 byte
* Phía application giám sát ta sẽ viết một giao diện trên nên tảng web và server
* Ngôn ngữ chính ta viết cho web server ở đây là Nodejs, giao diện hiển thị ta dùng EJS engine đi kèm với bootstrap css
* Mô hình giao tiếp giữa web server và node chính được mô tả tổng quát ở hình sau