МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КОПМП'ЮТЕРНИХ НАУК ТА ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ СИСТЕМ ТА МЕРЕЖ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач випускової кафедри

_____Ігор ЖУКОВ.

« ___ » _____ 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ "МАГІСТР"
Спеціальність 123 "Комп'ютерна інженерія"

Тема: «Підсистема управління динамічними об'єками »

Виконавець: Денис БУБЛИК

Керівник: Володимир АНРЄЄВ

Нормоконтролер: Василь МАЛЯРЧУК

Засвідчую, що у кваліфікаційній роботі немає
запозичень праць інших авторів
без відповідних посилань

Студент_____ Бублик. Д.С.

Київ 2023

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет комп'ютерних наук та технологій
Кафедра комп'ютерних систем та мереж
Напрям (спеціальність) 123 «Комп'ютерна інженерія»
(шифр, найменування)

ЗАТВЕРДЖУЮ
Завідувач кафедри
комп'ютерних систем та мереж

_____Ігор ЖУКОВ

« ___ » _____ 2023 р.

З А В Д А Н Н Я
на виконання дипломного проєкту

Бублика Дениса Сергійовича
(прізвище, ім'я, по батькові випускника в родовому відмінку)

1. Тема проєкту (роботи): Підсистема управління динамічними об'єками

затверджена наказом ректора від "29" серпня 2023 року №1521/ст

2.Термін виконання проєкту (роботи): 02.10.2023 р.- 31.12.2023 р.

3.Вихідні дані до проєкту (роботи): розробка концепції пристрою шифрування з полегшеною інтеграцією для модифікації *FPV* та інструментарій для легкої взаємодії для його налаштування

4.Зміст пояснювальної записки: титульна сторінка, список термінів та скорочень, огляд особливостой проектування сучасних систем управління динамічними обєками, огляд особливостей сучасних систем та підбір елементної та алгоритмічної бази, реалізація елементів пристрою апаратного шифрування на *FPGA*

5.Перелік обов'язкового графічного матеріалу:
Презентація *Power Point*.

6. Календарний план

№ з/п	Етапи виконання дипломного проєкту	Термін Виконання етапів	Підпис Керівника
1	Ознайомитись з завданням на виконання кваліфікаційної роботи, та літературою.	03.10.23-05.10.23	
2	Розглянути класифікацію динамічних об’єктів.	06.10.23-13.10.23	
3	Розглянути основні проблеми каналу зв’язку.	14.10.23-25.10.23	
4	Підбір алгоритмічної бази	26.10.23-13.11.23	
5	Підбір апаратної бази	14.11.23-23.11.23	
6	Реалізація елементів пристрою шифрування на *FPGA*	24.11.23-02.12.23	
7	Оформити пояснювальну записку та графічний матеріал	03.12.23-10.12.23	
8	Представити кваліфікаційну роботу кафедрі	11.12.23-18.12.23	

7. Дата отримання завдання « 02 » 10 2023 р.

Керівник кваліфікаційної роботи Володимир АНДРЄЄВ

(підпис)

Завдання прийняв до виконання Денис БУБЛИК

(підпис студента)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи «Підсистема управління динамічними об’єками» 90с., 20 таблиць, 68 рисунків, 28 літературних джерела.

КОМП’ЮТЕРНЕ УПРАВЛІННЯ ДИНОМІЧНИМ ОБ’ЄКТОМ, АПАРАТНА СИСТЕМА ШИФРУВАННЯ

Об’єкт дослідження: сучасні системи комп’ютерного управління динамічними об’єктами.

Мета кваліфікаційної роботи: розробка концепції пристрою шифрування з полегшеною інтеграцією для модифікації *FPV* та інструментарій для легкої взаємодії для його налаштування.

Методи дослідження: аналіз існуючих систем комп’ютерного управління динамічними об’єктами, та алгоритмів підвищення надійності каналу зв’язку.

Запропонована архітектура може використовуватись як методика для побудування пристроїв систем комп’ютерного управління динамічними об’єктами.

Пристрій розроблений на основі оглянутої моделі можна використовувати для підвищення надійності та стійкості каналу керування.

Запропонована модель надає наступні можливості при використанні зменшення ризику перехоплення управління динамічним об’єктом.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

ПЛІС - Програмована Логічна Інтегральна Схема

FPGA - Field-Programmable Gate Array (Програмована користувачем вентильна матриця)

PLA - Programmable Logic Array (Програмована Логічна Матриця)

IOB - Input/Output Block (Блок Вводу/Виводу)

CLB - *Configure Logical Block* (Конфігуровний Логічний Блок)

LUT - Look-Up Table (Пошукова таблиця)

ASIC - Application-Specific Integrated Circuit (Інтегральна Схема Спеціального Призначення)

HDL - hardware description language (мова апаратного опису системи)

USB - Universal Serial Bus

CDC - Communication Device Class (пристрій обміну даними)

FPV - First Person View

БПЛА - Безпілотний літальний апарат

ДКЛА - дистанційно керований літальний апарат

MAVLink - Micro Air Vehicle Link

UART - Universal Asynchronous Receiver/Transmitter(Універсальний асинхронний передатчик/приймач)

FHSS - Frequency Hopping Spectrum Spreading

САПР - истема втоматизованого роектування і розрахунку

PWM - pulse-width modulation (Широнно-Імпульсна модуляція)

FIFO - first in first out

АЛП - Арифметико-логічний пристрій

ICO - International Organization for Standardization (Міжнародної організації зі стандартизації)

ВСТУП

Системи управління динамічними об’єктами отримали велике поширення за останнє десятиліття. На даний момент сфери використання системи комп’ютерного управління динамічними об’єктами охоплюють багато сфер людської діяльності починаючи від вивчення морського дна до вивчення космосу. Данна система на сьогоденній день вбудована майже в всі динамічні об’єкти.

Використання системи управління динамічними об’єктами забезпечує спрощення управління за допомогою зворотних зв’язки та реалізації систем якими людина фізично не зможе керувати (наприклад система АБС в сучасних машинах, система стабілізації дронів та ін.), повна автоматизація об’єктів які виконують дії за закладеною в них програмою (автоматичні електричні підемники, ракети та ін.), та системи удавленого пілотованого та керованого управління (дрони, БПЛА та інші).

Основині переваги використання комп’ютерних систем управління динамічними об’єктами:

- зменшення впливи людського фактору;
- автоматизоване управління;
- полегшення керування динамічним об’єктом;
- можливість віддаленого керування об’єктом;
- зменшення ризиків для пілоту.

На сьогоднішній дана система є актуальною оскільки вона широко використовується в військовому роду діяльності як в цілях розвідки так і в знищенні цілей.

Завдання дослідження. Розробити модульну, гнучку архітектуру комп’ютерної системи управління динамічними об’єктами, що забезпечує максимальну універсальність системи для використанні в динамічних об’єктах.

Об’єктом дослідження є системи управління динамічними об’єктами з гнучкою архітектурою архітектурою.

Предметом дослідження є процес розробки модульної архітектури комп’ютерної системи управління динамічним об’єктом.

Практичне значення одержаних результатів. Розроблена архітектура дозволяє модифікувати існуючі громадянські *FPV* для збільшення захищеності каналу зв’язку та реалізувати динамічні об’єкти з нуля оскільки модель легко модифікується під різні типи динамічних об’єктів.

РОЗДІЛ 1

ОГЛЯД ОСОБЛИВОСТЕЙ ПРОЄКТУВАННЯ СУЧАСНИХ СИСТЕМ УПРАВЛІННЯ ДИНАМІЧНИМИ ОБЄКТАМИ

Комп’ютерні системи управління динамічними об’єктами – це вбудовані системи управляння в об’єкти для керування самими об’єктами. В якості об’єкта управління може виступати будь який об’єкт який має виконавчі елементи системи які саме приводять його в дію та (необов’язково) елементами органів зворотного зв’язку.

Основна задача даних систем це забезпечення керування певним об’єктом. Під керуванням мається регулювання керованості об’єктом та спрощення його управління за рахунок взаємозв’язків багатьох підсистем.

Оскільки кожен конкретний об’єкт може суттєво відрізнятися за конструкцією від інших в даному проекті розгонено лише найпоширеніші об’єкти за для систематизації та часткового спрощення моделей систем управління динамічними об’єктами для створення універсальної моделі. Для того щоб зрозуміти структуру систем КСУДО розглянемо спрощену класифікацію динамічних об’єктів.

1.1. Класифікація динамічних об’єктів

За оточенням використання об’єктів динамічні об’єкти можна класифікувати на:

- наземні;
- літальні;
- надводні;
- підводні;
- аерокосмічні.

За типом управління:

- безпілотний автоматичний динамічний об’єкт;
- дистанційно пілотований динамічний об’єкт;
- дистанційно керований динамічний об’єкт.

Безпілотний автоматичний динамічний об’єкт - безпілотний апарат що виконує своє призначення за спеціальним алгоритмом який закладається в нього лише перед запуском. Після запуску виконую закладену задачу в автоматичному режимі. Як правило містить складну систему зворотного зв’язку як для корегування місця положення так і місця положення у просторі[2-3].

Дистанційно пілотований динамічний об’єкт – безпілотний динамічний об’єкт з прямим управлінням яке здійснюється з пункту управління. Управління відбувається з початку з початку руху та до кінця поставленої задачі. Основна перевага над системою автоматичного керування це точність прийняття рішень оскільки керування на протязі всього часу відбувається безпосередньо людиною. Може як містити вбудовані системи зворотного зв’язку так і ні. Зворотній зв'язок може забезпечуватися завдяки пілоту. Якщо зв'язок присутній то основна його задача це полегшити керування об’єктом та підтримувати задані параметри. З недоліків при втраті зв’язку буде продовжувати очікувати команд[2-3].

Дистанційно пілотований динамічний об’єкт - безпілотний апарат що виконує своє призначення за спеціальним алгоритмом який закладається в нього не лише перед запуском а й під час функціонування для зміни цілі. Після запуску виконую закладену задачу в автоматичному режимі. Іноді (наприклад в розвідувальних ДКЛА) також реалізується система дистанційно пілотованих динамічних об’єктів. Як правило містить складну систему зворотного зв’язку як для корегування місця положення так і місця положення у просторі[2-3].

Залежно від того, чи використовуються для вироблення керуючих впливів інформація про стан керованого об’єкта, системи управління поділяються на

- розімкнені;
- замкнуті.

Замкнуті. В замкнутих системах є два інформаційні потоки. Перший потік це управління шаговими двигунами другий негативний зворотній зв'язок який відслідковує реальне положення виконавчих органів, швидкості і т.д. та корегування розбіжностей с заданою програмою. Датчики негативного зворотного зв’язку можуть бути як цифровими так і аналоговими.

Розімкнуті системи відрізняються від замкнутих наявністю лише одного інформаційного потоку від системи управління до виконавчих органів.

1.2. Структурна схема систем числового програмного управління динамічними об’єктами.

В залежності від типу управління відрізняється й сама структурна схема управління.

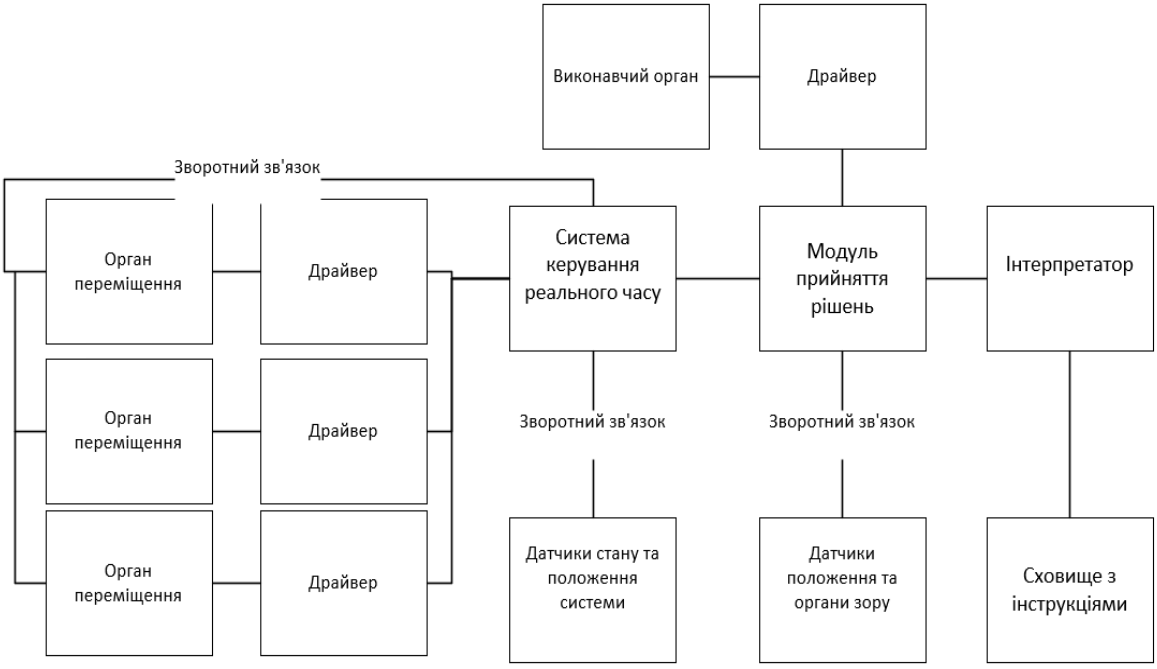

Рис.1.1. Структурна схема управління безпілотного автоматичного динамічного об’єкта

Системи управління безпілотного автоматичного динамічного об’єкта(рис.1.1).Основні елементи системи.

Органи переміщення. Основна задача даного елементу системи це забезпечення переміщення та керованості динамічним об’єктом. Оснащуються датчиками зворотного зв’язку(оптичні інкодер. датчики хола і тд.) для точного контролю виконання роботи.

Драйвер. Основна задача драйвера це на основі отриманих сигналів генерувати керуючий сигнал для управління в виконавчим органом або органом переміщення. В багатьох драйверах вже реалізована система корекції на основі зворотного зв’язку. В даній моделі обробку та корекцією роботи бере на себе система керування реального часу оскільки не всі драйвери мають можливість робити це самостійно.

Система керування реального часу. Даний модуль включає в себе декілька під задач а саме: управління драйверами, корегування роботи органів переміщення на основі зворотного зв’язку, робота алгоритмічно математичного пристрою ,надання алгоритму корекційних даних від датчиків положення системи (за наявності).

Модуль прийняття рішень. Основне завдання даного елементу системи це вводити корекції маршруту на основі даних з датчиків положення та зору (*GPS*, ультразвукові датчики та інші). Інша задача це управління виконавчими органами (управління маніпулятором, тощо). Виконує завдання згідно наданих даних. Всі завдання які зв’язані з переміщенням та стабілізації об’єкту в просторі передається системі керування реального часу.

Інтерпретатор. Основна задача даного елементу це зчитування команд зі сховища та перетворювати одержані команди у зручний для модуля прийняття рішень формат.

Ролі елементів системи такі самі як в системі управління безпілотного автоматичного динамічного об’єкта за виключенням модуля прийняття рішень. Інтерпретатор вже вбудований в нього.

Трансивер. Основна його завдання це передача та отримання фреймів інформації та відсічення всіх пакетів які не призначені даному об’єкту та перевірка

цілісності отриманого пакету. Як правило фільтрація пакетів відбувається за апаратним номером пульта управління. Іноді функція фільтра відсутня.

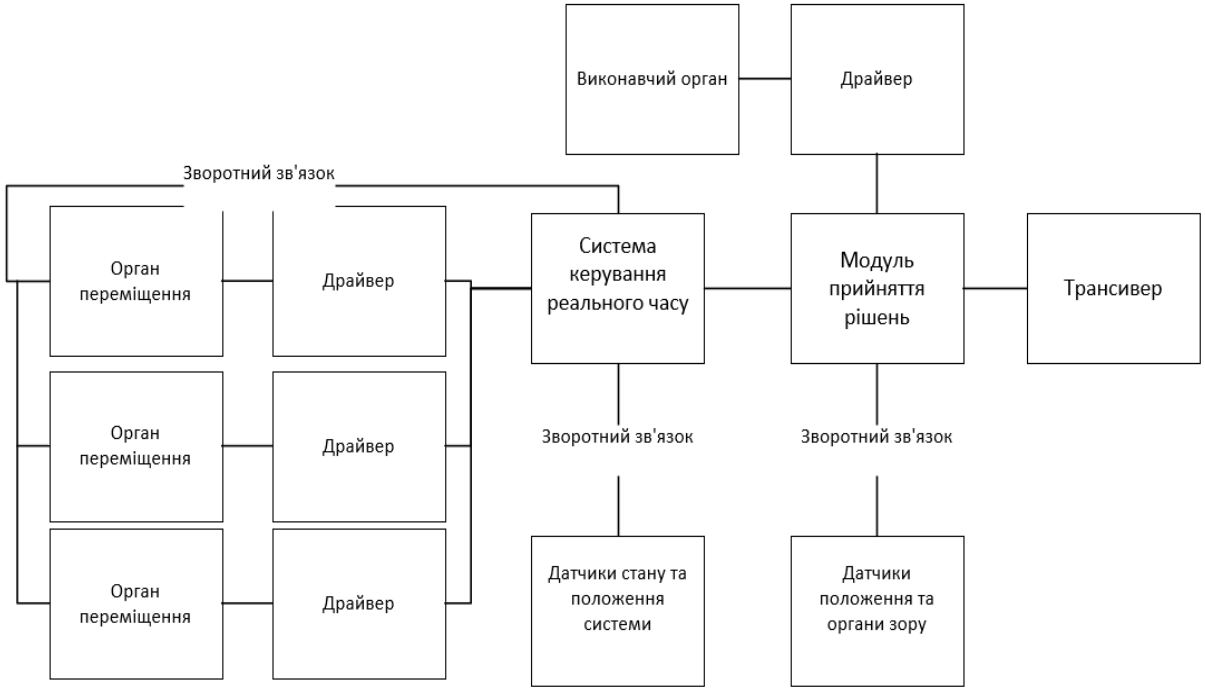

Рис.1.2.Структура системи управління дистанційно пілотованого динамічного об’єкта

Системи управління безпілотного дистанційно пілотованого динамічного об’єкта (рис.1.2).Основні елементи системи.

Трансивер. Основна його завдання це передача та отримання фреймів інформації та відсічення всіх пакетів які не призначені даному об’єкту та перевірка цілісності отриманого пакету. Як правило фільтрація пакетів відбувається за апаратним номером пульта управління. Іноді функція фільтра відсутня.

Останні ролі елементів системи такі самі як в системі управління безпілотного автоматичного динамічного об’єкта за виключенням модуля прийняття рішень та інтерпретатора.

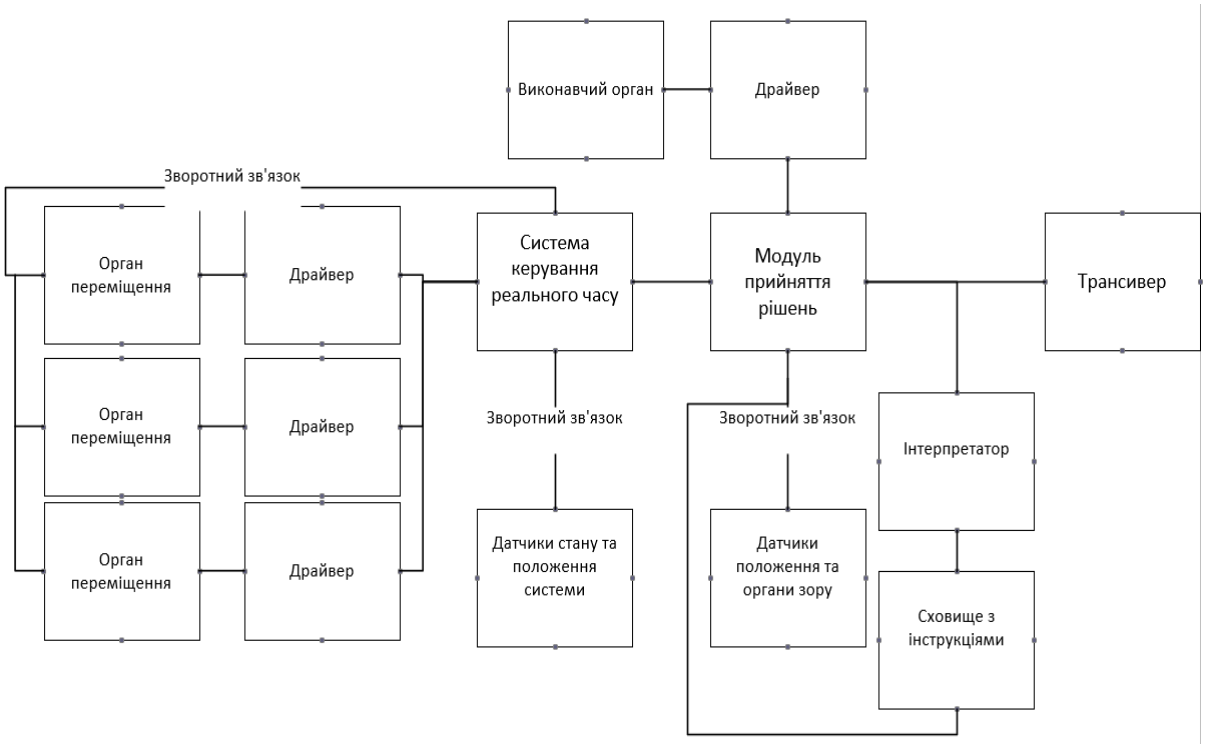

Рис.1.3. Структура системи управління дистанційно керованими динамічними об’єктами

Системи управління дистанційно керованими динамічними об’єктами (рис.1.3).Останні ролі елементів системи такі самі як в системі управління безпілотного автоматичного динамічного об’єкта за виключенням модуля прийняття рішень та інтерпретатора. Окрім вже описаних задач модуль прийняття рішень може напряму записувати всі данні після перевірки цілісності пакетів та їх нумерацію завантажувати до сховища та всі пакети. В разі якщо нумерація порушена запросити відсутні через трансивер повторно.

1.3. Потокове шифрування каналів зв’язку.

Оскільки в звичайних системах управління динамічними об’єктами як провело є лише фізичний адрес пульта управління який завжди знаходиться в заголовку перехоплення управління таким об’єктом не є складною задачою оскільки самі

команди є стандартизованими для всіх подібних пристроїв(один виробник, одна серія).

Наступна проблема це звичайне прослуховування каналу і навіть якщо сам пакет зашифрований одним ключем його можна записати та подати повторно з більш потужного передавача. Таким чином можна заставити об’єкт виконати дію яка не планувалася користувачем.

Для вирішення цих проблеми використовується шифрування пакетів за допомогою спеціальних алгоритмів або підраховуванням кількості команд.

Метод підрахування команд означає що кожна команда шифрується одним і тим самим ключем та пронумерована. Таким чином при передачі такого ж самого сигналу він не виконається оскільки він вже був. Недолік такої моделі в тому що при втраті одного з пакету потрібно запрошувати його знову в разі коли пакет був битий. В разі пропуску його всі наступні команди будуть прийняті як хибні.

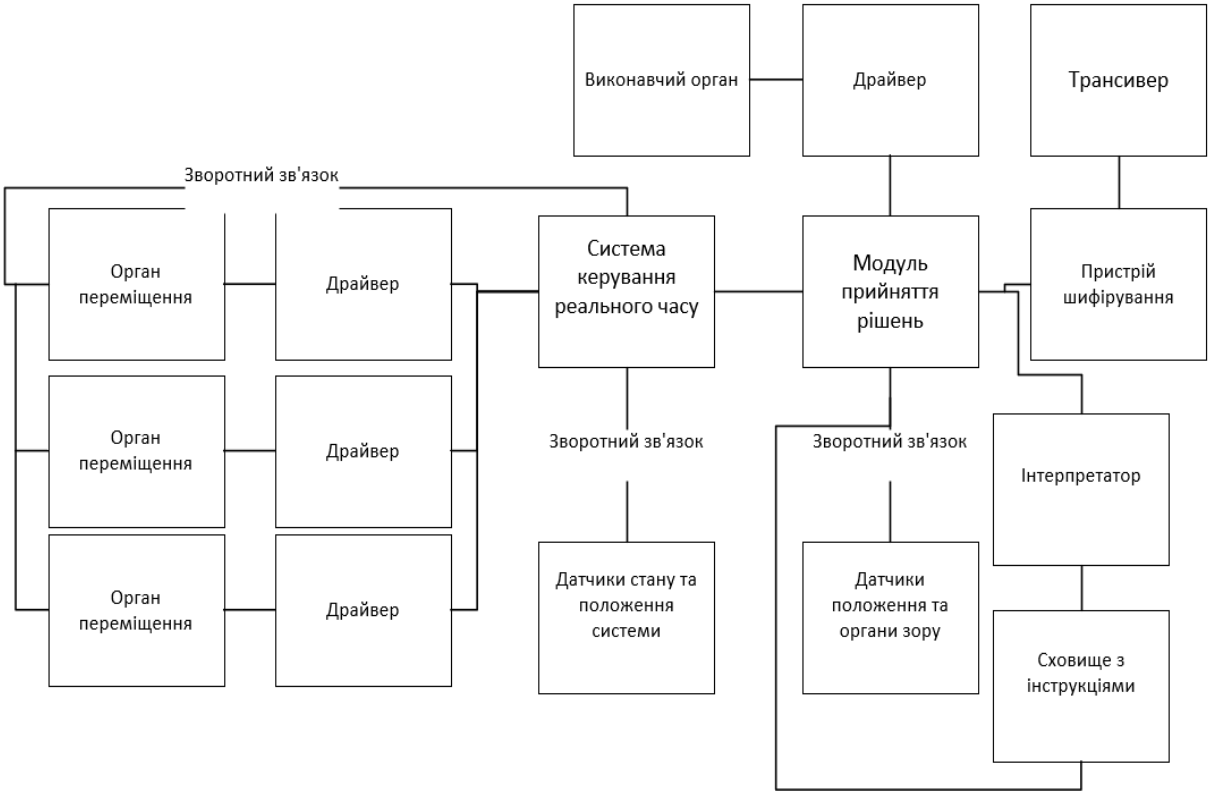

Рис.1.4. Структура системи управління дистанційно керованими динамічними об’єктами з вбудовуванням елементу шифрування даних

Для того щоб недопустимий цього використовуються часова синхронізація пакетів. Але і в такому разі оскільки адрес заліза відомий та не шифрується все що залишається це підібрати ключ. Для запобігання цього використовуються алгоритми з генерацією ключів подібних до алгоритму шифрування *DES* з розширеним ключем.

Наступною проблемою є швидкодія алгоритму оскільки під час дистанційного управління в залежності від об’єкту швидкодія є дуже важливим параметром. Тому для пришвидшення процесу шифрування (якщо це необхідно) використовують апаратну реалізацію алгоритмів шифрування. Як провело вони реалізуються або як просто вузькоспеціалізована мікросхема або в ПІЛС. Реалізація в ПЛІС має безліч переваг навіть до повної заміни алгоритму під час роботи пристрою оскільки її можна перепрограмувати.

Як правило пристрій шифрування вже має вбудовану систему перетворення послідовного коду в паралельний та паралельний в послідовний (залежать від протоколу обміну даних) . Тому вся система працює у нормальному режимі без змін.

В системі управління вбудовується наступним чином(рис.1.4. рис.1.5).

Рис.1.5. Структурна схема блока управління

1.4. Програмована логічна інтегральна схема

Програмовані інтегральні логічні схеми це вид інтегральних схеми в основі яких лежить принцип програмованих логічних блоків та блоків які відповідають за комутації логічних матриць. Конфігурування даних пристроїв дозволяє вирішувати специфічні завдання з нахилами в швидкодію. В залежності від можливості

багаторазового перепрограмування ПЛІС їх поділяють на одноразово програмовані та багаторазово програмовані.

Пліс на відмінно від комп'ютерних систем на основі Фон Наймана не виконують задачу покрокова за набором команд з використанням АЛП(арифметично логічний пристрій) а за допомогою ПЛМ(програмована логічна матриця, *PLA, Programmable Logic Array*).

Всі обчислювальні системи на основі АЛП виконують обробку даних занесених в акумулятор згідно програми покроково. В системах на основі ПЛМ після завнтаження програми відбувається одноразове налаштування автомату в

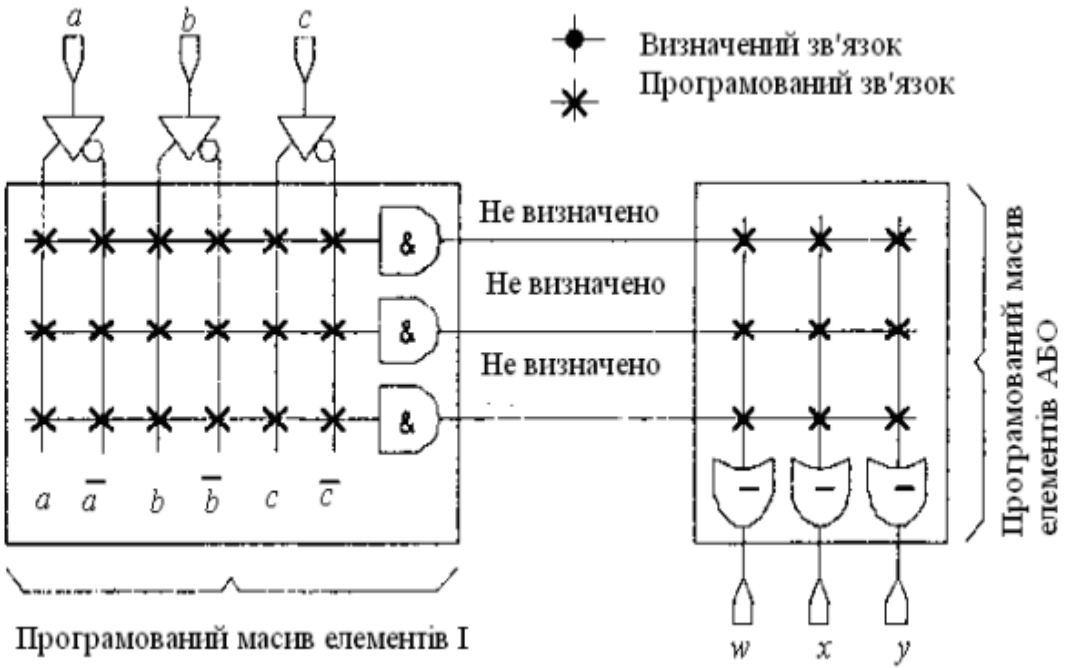

Рис.1.6.Структурна схема ПЛМ до програмування

комірках ПЛМ яка описується в прогамі. Процес налаштування зображенно на рис1.6,рис.1.7.

Таким чином результатом роботи першого може бути лише модифікована програми то в другому випадку результатом роботи програми є логічна схема.

Основноою перавоагою такої системи є швидкість роботи ПЛІС при певних умовах буде обмажена лише параметрами транзисторів таким як ємність затвору, швидкість наростання напруги на стоці і тд.

Розглянемо докладно топологію ПЛІС на основі ППІС (*FPGA*) серії *Virtex* фірми *Xilinx* (рис. 1.8). На площі кристала ПЛІС розміщені матриця конфігурованих логічних блоків КЛБ (CLB - *Configure Logical Block*), матриця відрізків міжз'єднань, покритих матрицями з польових транзисторів - перемичок МП (*IB - Interconnect Block*), блоки настроюваних ОЗУ (*RAM*); блоки введення/виведення сигналів (*IOB - Input / Output Block*) і периферійний канал міжз'єднань з мінімальною затримкою (*VR - VersaRing*), розміщені по периметру кристала. Для забезпечення достатніх можливостей маршрутизації міжз'єднань і мінімальної затримки при передачі сигналів використовується до дев'яти шарів металізації[10].

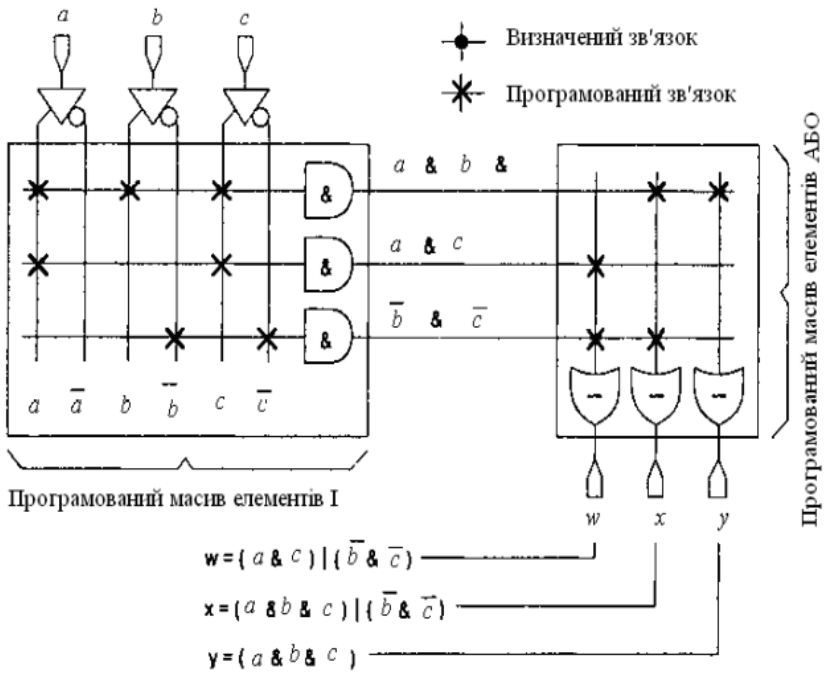

Рис.1.7. Структурна схема ПЛМ після програмування

Базовий осередок в архітектурі *FPGA* набуває вигляду КЛБ (*CLB - Configurable Logical Block* рис1.9.), основним логічним елементом якого є логічна таблиця ЛТ (*LUT - Look-Up Table*), що становить однобітний ОЗУ, призначена для зберігання логічної функції від n-аргументів. Фактично в пам'ять безпосередньо

заноситься таблиця істиності булевої функції, причому набір значень аргументів використовується як адреса блоку пам'яті, за яким на вихід надходить значення функції. Для завдання таблиці істинності логічної функції від n аргументів потрібно однобітний блок пам'яті, який має 2*n* осередків. Позначення чотиривходового блоку ЛТ (*LUT*) для завдання функції «І» від чотирьох аргументів[10].

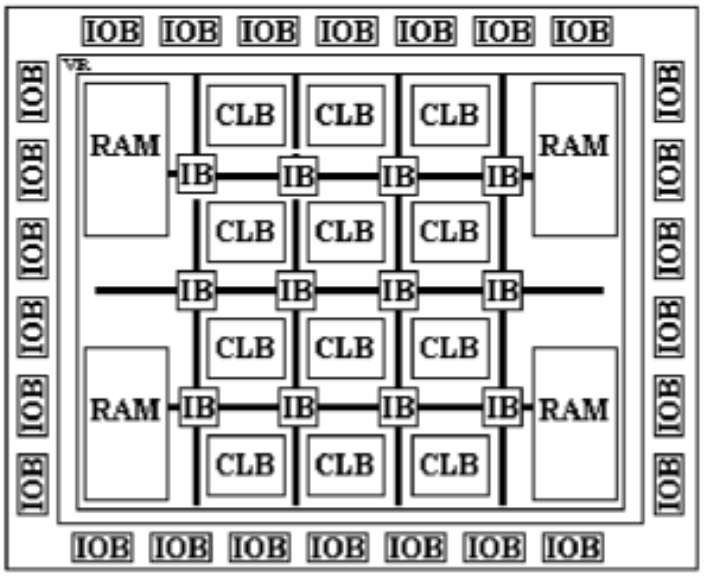

Рис.1.8. Топологія кристала ПЛІС серії *Virtex* фірми *Xilinx*

КЛБ(рис.1.9.) характеризуються такими властивостями:

–функціональність (*functionality*) визначає наскільки великі логічні можливості одного КЛБ;

–зернистість (*granularity*) визначає наскільки функціонально і схемотехнічно простими будуть елементи, складові КЛБ. Прикладом дрібнозернистого КЛБ може служити ЛБ фірми *Crosspoint Solutions*, що містить ланцюжки транзисторів; прикладом великозернистого КЛБ можуть служити ЛБ, у мікросхемах *Xilinx XC*4000, що містять 2 ЛТ на 4 аргумента, 1 ЛТ на 3 аргумента, 2 *D* - тригери, пов'язані через кілька мультиплексорів, а також логічні схеми вентильного рівня;

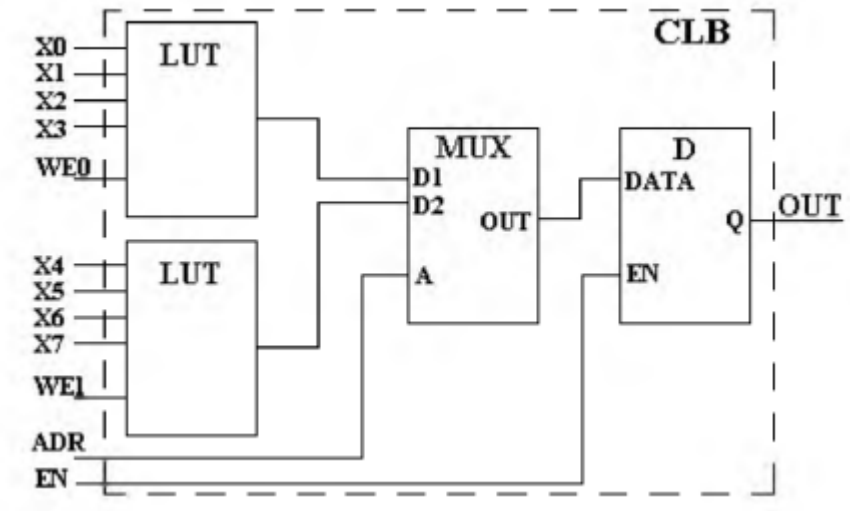

Рис.1.9. Структурна схема КЛБ (*CLB*)

–універсальність і зручні налаштування КЛБ вимагають накладних витрат і не кращим чином позначаються на швидкодії пристроїв на *FPGA*. Підвищення технічних характеристик у цих пристроях забезпечується зануренням у кристал готових функціональних блоків (блоків пам'яті, помножувачів, процесорних ядер), спроектованих і оптимізованих на рівні схемотехніки.

1.5. Засоби автоматизованого проектування цифрових пристроїв на ПЛІС

В сучасних розробках набула широкого використання практика використання програмованих логічних інтегральних схем (ПЛІС). Проектування складних пристроїв на основі ПЛІС як правило відбувається за допомогою спеціалізованих систем автоматизованого проектування(САПР). На рис.1.10. зображені всі основні етапи проектування, перетворюється до алгоритму автоматизованого проектування ПЛІС.

На концептуальному рівні проектування визначаються функції пристрою, безлічі вхідних і вихідних сигналів, можливості розбиття проекту на частини. Цей рівень проектування, практично, не пов'язаний з автоматизацією і його реалізація цілком покладається на розробника.

Рис.1.10. Етапи системного проектування цифрових пристроїв

Результати концептуального синтезу вводяться в САПР, в якій виконується компіляція проекту - синтез пристрою в базисі бібліотеки моделей. Компіляція розбивається на ряд послідовних етапів: формування бази даних проекту, контроль з'єднань, логічна мінімізація проекту і т.д. Результатом компіляції є файл, який містить конфігураційну інформацію для заданої ПЛІС. Скомпільований проект вимагає ретельної перевірки, тому за етапом синтезу йде етап аналізу за допомогою моделювання та теоретичної верифікації. Моделювання здійснюється на декількох рівнях, що характеризуються різним ступенем відображення властивостей реального об'єкта. Так, за допомогою функціонального моделювання можна перевірити правильність логічної структури пристрою. Часове моделювання дозволяє проводити тестування роботи пристрою з урахуванням затримок сигналів у компонентах без урахування остаточної топології трасування. В результаті моделювання можуть виявитися помилки, які вимагають виправлень, що надає процесу проектування ітеративний характер з поверненнями до попередніх етапів і введення в проект потрібних змін[9].

Застосування САПР для розробки пристроїв на ПЛІС вимагає ефективних, наочних, керованих і контрольованих засобів опису проекту. На сучасному етапі

найбільш поширеними універсальними способами опису проектів є графічний і текстовий.

Графічний спосіб заснований на поданні схеми проекту в базисі допустимих для обраної САПР бібліотечних елементів, наприклад, у базисі елементів стандартної серії ТТЛ (Ш). Основні переваги графічного способу - традиційність і наочність, пов'язані зі звичністю розробників до сприйняття зображень схем. Однак у ході проектування пристроїв на ПЛІС, складність яких оцінюється десятками і сотнями тисяч еквівалентних вентилів, різко втрачається наочність проекту навіть під час коректного застосування системного підходу. Тому в останнє десятиріччя набули істотного розвитку кошти текстового опису цифрових пристроїв на ПЛІС, реалізовані у вигляді мов опису апаратури (*Hardware Description Languages - HDL*)[9].

Сучасні мови опису апаратури допускають опис проектованого пристрою як з точки зору його поведінки (виконуваних функцій), так і з точки зору його структури. Ці можливості дозволяють подавати проект у формі текстового опису алгоритмів функціонування окремих компонентів пристрою в поєднанні з описом між компонентних з'єднань. Перевагами текстового способу опису проекту за допомогою мов опису апаратури є компактність, автоматизація більшості перетворень, можливість перенесення проекту з однієї апаратної платформи на іншу, простота документування.

1.6. Мови опису апаратури

Сьогодні засоби *HDL* розділяються на два основні різновиди - мови низького і високого рівнів. Мови опису апаратури низького рівня за своїми командами безпосередньо орієнтовані на роботу з апаратними засобами і мають потенційні можливості для створення проектів з оптимізованими параметрами. Дані *HDL*, як правило, жорстко прив'язані до певної апаратури. За допомогою мов низького рівня, що враховують спеціалізовані особливості архітектури ПЛІС, істотно полегшується створення проектів з найкращими часовими характеристиками. Прикладами таких

мов можуть служити *AHDL* (*Altera HDL*) і *ABEL* (*Xilinx*). Мови опису апаратури високого рівня менш пов'язані з апаратними платформами і володіють більшою універсальністю. Найбільш поширеними мовами цієї групи є *VHDL* і *Verilog*. Ці мови, як і інші мови програмування високого рівня, дозволяють описати будь-який алгоритм у послідовній формі - за допомогою послідовності операторів присвоювання і прийняття рішень. Особливістю даних мов є наявність засобів відображення паралельно виконуваних апаратних дій, які подаються окремими паралельно виконуваними процесами із загальним ініціалізованим впливом. Найбільш поширеною мовою цього класу, специфікованою міжнародними стандартами, є мова опису апаратури *VHDL*[10].

Висновок за розділом

В даному розділі було розглянута класифікація систем комп’ютерного управління динамічними системи, елементи різних систем комп’ютерного управління та шляхи підвищення надійності каналів зв’язку. Також розглянуті методи втручання в канали зв’язку та запропоновані методи вирішення проблем даних систем за рахунок апаратного шифрування даних. Для реалізації даних алгоритмів доцільно використовувати ПЛІС. Реалізацію алгоритмів шифрування на ПЛІС доцільно робити на мов *VHDL*.

РОЗДІЛ 2

РОЗГЛЯД ОСОБЛИВОСТЕЙ СУЧАСНИХ СИСТЕМ ТА ПІДБІР ЕЛЕМЕНТНОЇ ТА АЛГОРИТМІЧНОЇ БАЗИ.

За основу даного проекту буде розглядатись бюджетні FPV дорни. Вони мають безліч конфігурацій та прошивок польотних контролерів. Більшість з них мають різну систему передачі управляючого сигналу від системи управління до польотного контрейлера. Тому слід розібрати апаратні складові контролера та методи передачі керуючого сигналу.

2.1. Огляд апаратних блоків сучасних FPV БПЛА

Існує велика кількість конфігурацій дронів. Вони значно відрізняються один від одного в залежності від типу БПЛА та задач під які він був спроектований. Основнними компоменетами які присутні в будь якій конфігурації:

- польотні контролер;
- гіроскоп;
- акселерометр;
- бараметричний сенсор;
- *GPS* модуль;
- трансивер;
- драйвери.

Польтний контролер це основний елемет управління БПЛА. Основна його задача це керування БПЛА на основі даних які він отримує з датчиків та з пункту управління. Забезпечує взаємодію всіх вузлів системи упраління та на основі закладених алгоритмів видає керуючі сигнали на драйвери, що відповідають як за органи переміщення так і на виконавчі органи. Також стабільність та полегшення екрування дрона на основі закладених алгоритмів[2].

Алгоритми містять велику кількість зворотни звязків на основі багаторівневих *PID*(рис. 2.1.) регуляторів з актентом на диференційну частину регулятора. Тому швидкодія та реагування на змінии досить повільне(рис.2.2.). Для виправлення даної ситуації в нього закладуються спеціальні таблиці. З них беруться приблизні значення уравління при зміні команди управління і лище потім на основі зворотніх звязків відбувається точна корекція положення.

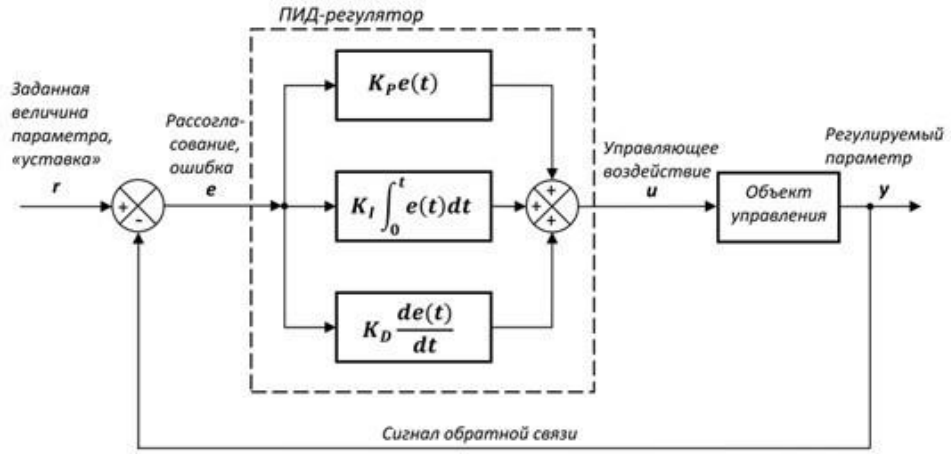

Рис.2.1. Структурна схема *PID* регулятора

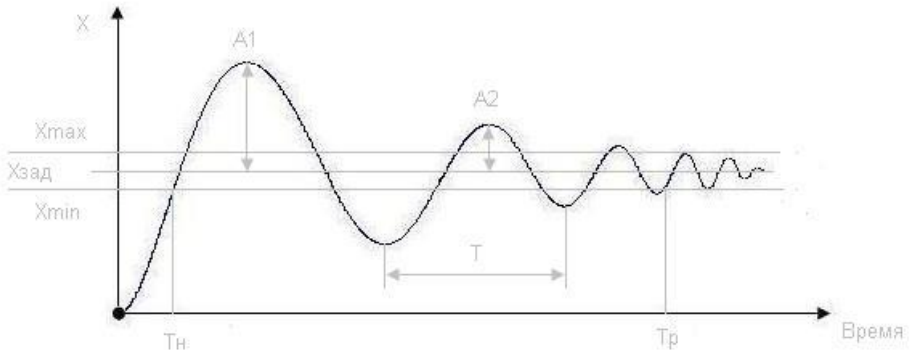

Рис.2.2. Графік перехідних процесів *PID* регулятора

Гіроскоп це дачик який що реагує на зміни кутів орієнтації тіла, на якому воно встановлене, в трьох площинах. Це забезпечує можливість БПЛА корегувати кут положення у просторі не лише по таблицям а й за допомогою негативного зворотного зв’язку.

Акселерометр це датчик що заміряє лінійне та кутове прискорення об’єку. Інує декілька типів датчиків. Найросповсюджений це ємкісний. В основі його принципу лежить зміна відстаней між обкладенками конденцатора в наслідок чого відбувається зміна його емності. Структура ємкісного акселерометра зображена на рис. 2.3. Я кправило його реалізація відбувається на кристалі рис.2.4. тому їх розміри досить маленькі.

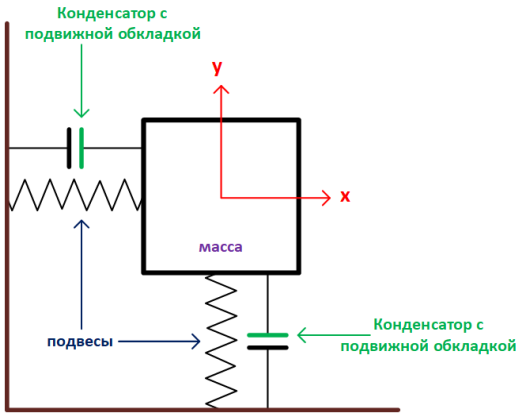

Рис.2.3. Прицинова схема робота ємкісного аксилераметра

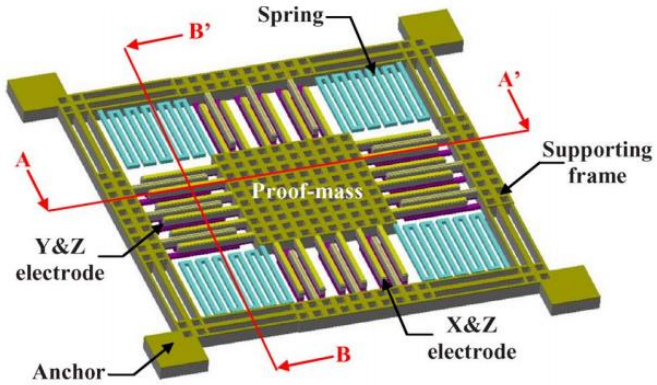

Рис.2.4. Принип реалізації акселерометра на кресталі кремнію

Барометричний сенсор. Основна задача це замір тиску. Оскільки тиск на пряму пов'язаний з плотністю повітря система управління враховує і даний показник для стабільного утримання висоти в розрідженому повітрі. Як правило задля зменшення навантаження на обчислення з плавуючою запятою використовують таблиці.

Трансивер. Основна задача трасиверу це забезпечення звязку між пунктом управління та дроном. Транисвар має два протоколи. Перший описує метод передавання даних між трансиверами. Основні протоколи які використовуються:

- *D8, D16, LR12 (Frsky);*
- *DSM, DSM2,DSMX (Spektrum)*;
- *Flysky (Flysky);*
- *A-FHSS (Hitec)*;
- *FASST (Futaba)*;
- *Hi-Sky (Deviation)*.

Для даного проеку оскільки основна задача це збільшення надійності каналу від можливості перехоплетнення та збільшення стійкості для електромагнітних перешкод може бути будь який протокол розширення спектру на основі Frequency *Hopping Spread Spectrum – fhss*.

Другий протокол відповідає за взаємодію між трансивером і польотним контролером. Як правило використовуються шини переферії. Їх можно поділити на дві категорії. Перша це всі протоколи в яких використовується *Puls Modulation*(*PWM,PPM,PCM*)(рис.2.5.). Протоколи передачі даних з послідовною передачою даних. Оскільки до недавнього часу популярної стандартизації не будл кожна компанія мала свій протокол формування пакетів(SBUS, XBUS,IBUS,SUMD,SUMH) даних та роботу з налаштуванням переферії. За основу взяті протоколи вбудованої переферії такі як *UART*, *TWI*(*I2C*), *SPI*.

На сьогодення більшість виробників перейшли на уніфікований проток *MAVlink*(детально пункт). Оскільки даний проткол описує прикладний рівень в моделі *OSI* це дозволяє використовувати будь які системи передачі покету. В сучасних *FPV* дронах використовують шину *UART*(рис.2.6).

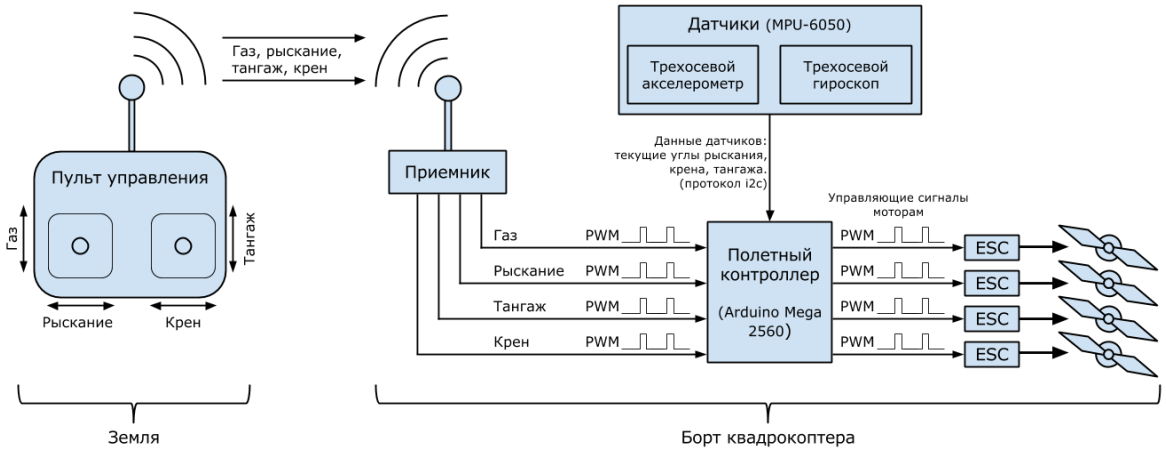

Рис.2.5. Структура БПЛА на основі *Pulse Modulation*

FPV дрони містять трансивер та відео передатчик. Тому для передачі відео сигнал та управляючого використовують два розведених за частотою широкочастотного спектра. На рис.2.7 Зображена передача даних каналу управління на основі *FHSS* на частоті 905МГц з шириною спектра 4МГц. Для передачі відеосигналу використовується інший широкополосний канал на частоті 2.2ГГц(рис.2.8.).

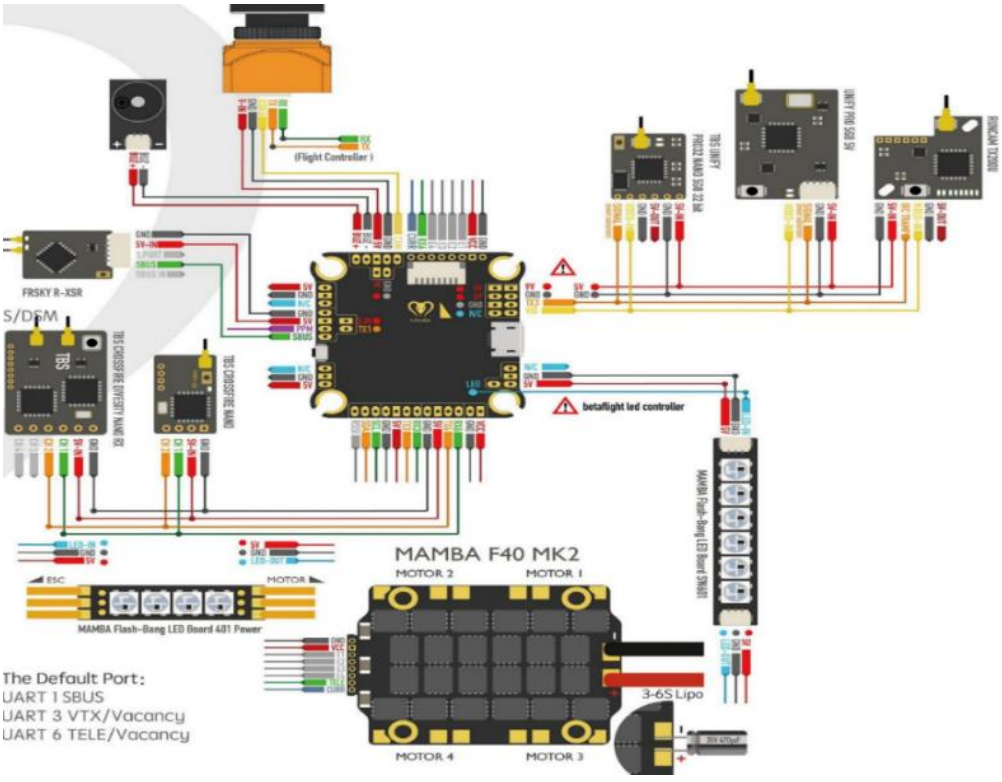

Рис.2.6. Структура БПЛА на основі шин послідовної передачі данних

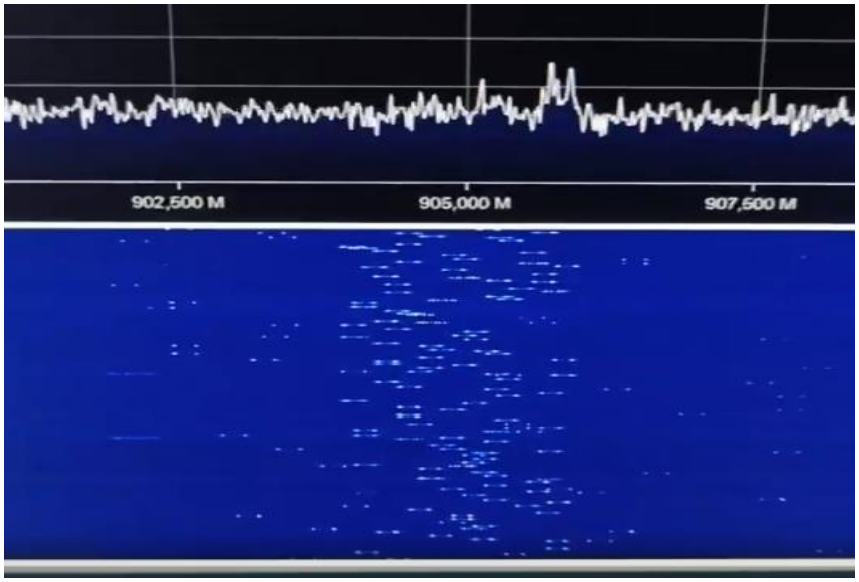

Рис.2.7. Спектр передачі даних на основі протоколу *FHSS*

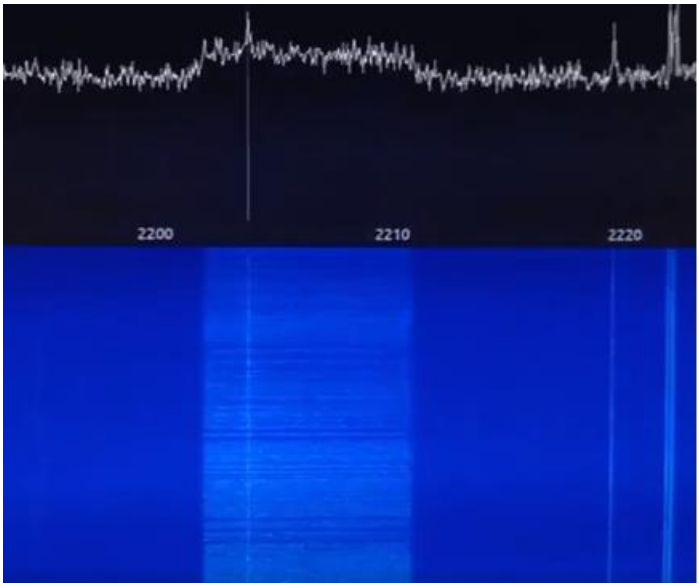

Рис.2.8. Спектр передачі відеоканала на частоті 2.2ГГц

Всі цивільні *FPV* дорни використовують композитний канал передачі відео. Це означає що сигнал від камери до передатчика аналоговий тому без втручання в модуль рис. неможливе шифрування каналу відеозв’язку. Оскільки при невдалому втручанні можна нашкодити ВЧ монтажу та пошкодити електромагнітну ізоляцію корпусу.

Рис.2.9. Типовий передатчик відео та аудіо сигналу на *FPV* підвищеної потужності

2.2. Огляд протоколу зв'язку *UART*

UART це універсальний асинхронний протокол зв'язку між пристроями. *UART* був розроблений в США для передачі символів таблиці *ASCII*. Саме після виходу даної таблиці розмірність даних стала 8 байт.

Будь який *UART* інтерфейс використовує дві лінії зв'язку *RXD*(*Resaved data*) та *TXD*(*Transmitted data*)(рис. 2.10).

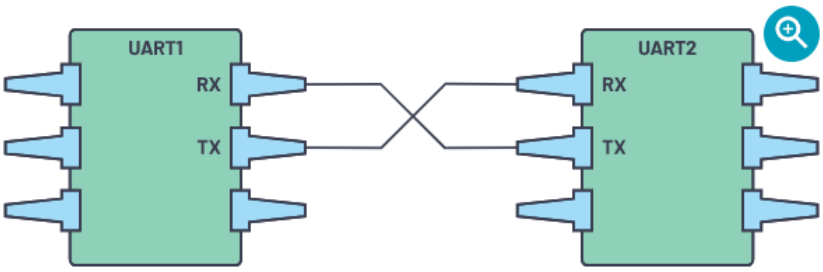

Рис.2.10. Схема під'єднання модулів із інтерфейсом *UART*

Оскільки кожна з двох шин є однононаправленою та асинхронною проблему синхронізації лежить на канальному рівні. Вданій шині це реалізовано за допомогою двох рішень. Передше це синхронізація по передньому фронту пакета. Друге

частота двох пристроїв повинна бути однакова. На рис. 2.13. зображений сигнал передачі одного біта даних. Кількість біт в одному кадрі може бути значно більша але це призводить до збільшення кількості помилок оскільки неможливо отримати абсолютно однакову частоту синхронізації. Іншою проблемою стала дальність передачі. За стандартом вона не повинна перевищувати 15м. Але при певних модифікаціях пов’язані з використання диференційних пар та зменшення напруги сигналу до 3.3В вдалось значно збільшити як відстань так і частоту. Основні частоти що використовуються це 9600 12800 В таблиці 2.1 наведена залежність кількості помилок від швидкості передачі.

Як правило зміна частоти роботи шини закладена на апаратному рівні за допомогою додаткових прискейлерів. Типова схема апаратної реалізації системи трактування зображена н рис.2.11. Діаграма трактування для даної структури зображена на рис.2.12.

Таблиця 2.1

Залежність кількості помилок від швидкості передачі

Baud Rate	*Divisor Value*	*Actual Bound Rate*	*Error*
2400	3906	2400.154	0.01
4800	1953	4800.372	0.01
9600	977	9595.701	0.004
19200	488	19211.066	0.06
38400	244	38422.131	0.06
56000	167	56137.725	0.25
128000	73	129807.7	0.33

На рис. 2.13. Зображений фрейм передачі даних. Він складається з наступних частин: Стартовий байт, біт інформації, байт перевірки на парність, стоп байт

В вільному стані шина підтримується в стані одиниці. Початок ініціалізації передачі відбувається по спадаючому передньому фронту START. Також через асинхронність шини команда *START* використовується як синхронізація.

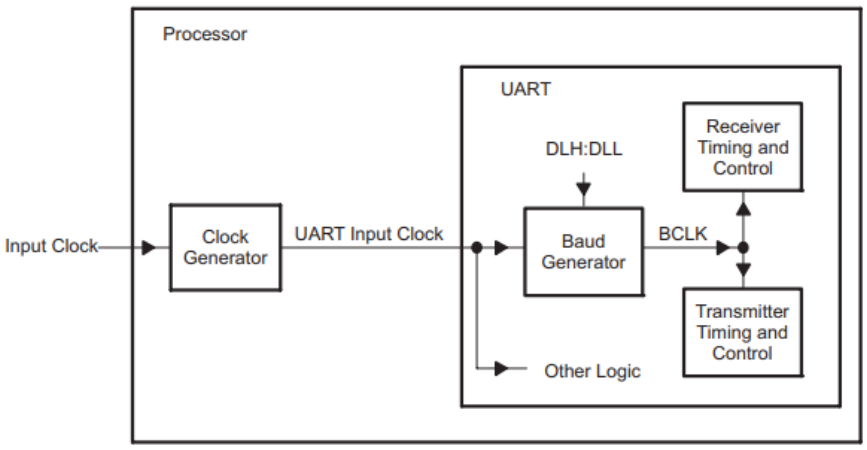

Рис.2.11. Структурна схема система трактування *UART* інтерфейсу

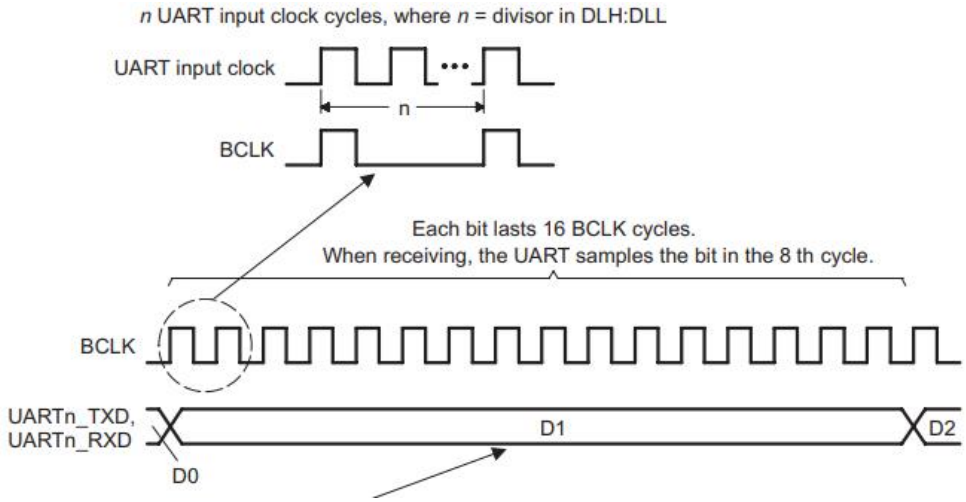

Рис.2.12. Діаграма трактування шини *UART*

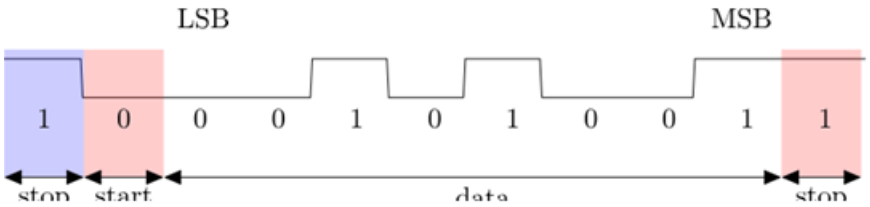

Рис.2.13. Фрейм передачі даних по шині UART.

Після ініціалізації передачі по шині передається певна кількість байт даних згідно заданої частоти. В залежності від реалізації один кадр може містити значно більше байт ніж вісім. Довжина кадра в апаратних реалізаціях як правило не

змінюється. Оскільки частота синхронізації не ідеальна існують апаратні обмеження дожени кадра.

MSB байт який відповідає за перевірку цілісності кадру. Реалізовується за рахунок підрахунку парності байту. Після передачі всіх необхідних біт формується сигнал завершення передачі даних. Формується він за допомогою одиниці. Таким чином після передачі фрейму лінія завжди залишається в позиції одинці.

Алгоритми роботи для *RX* та *TX* частини відрізняються між собою. Основна відмінність це часові здвиги для роботи кожної лінії.

Після встанови стартового біту відбувається синхронізація і початок відліку для обох частин каналу. Довжина стартового біта визначається частотою передачі каналу. Час передачі сигналу *TX* не змінюється і відповідає початковій. Час прийому *RX* відповідає часту яку встановлено на *TX* але з зміщенням в половину часу необхідного на передачу одного біту(рис.2.14). З цього виходить що частота трактування *RX* частини каналу повинна бути мінімум в двічі важча за *TX*.

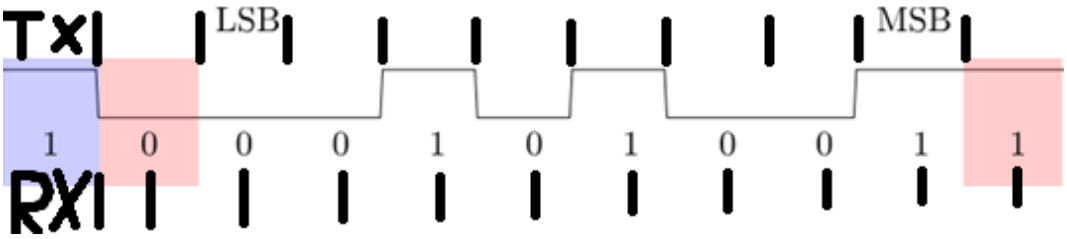

Рис.2.14. Ключові точки активності *TX* та *RX* частини

Сучасні мікроконтролери містять вбудовану апаратну реалізації шини *UART*. Структурна схема зображена на рис.2.15.

Основні компоненти даної системи:

- банк регістрів управління;
- система логіки управління перериваннями;
- система контролю черги(*FIFO*)(рис.2.16);
- буфер інформаційної шини;
- здвигові регістри передачі/прийому.

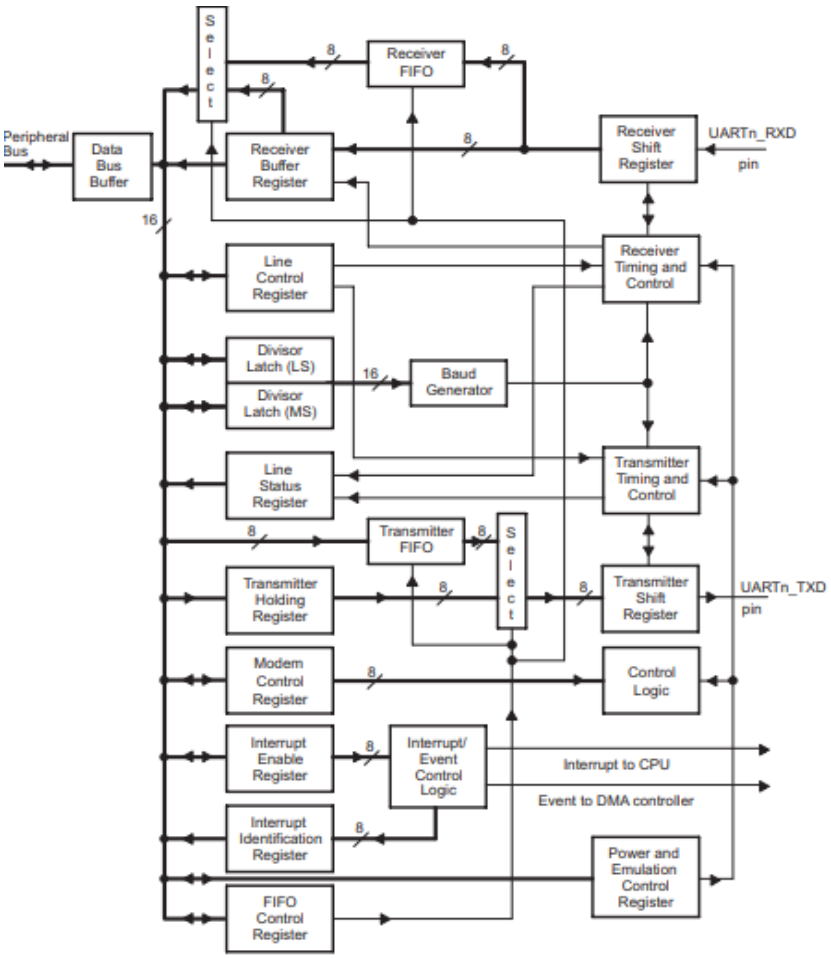

Рис.2.15. Схема блоків апаратної реалізації *UART* в мікроконтролерах

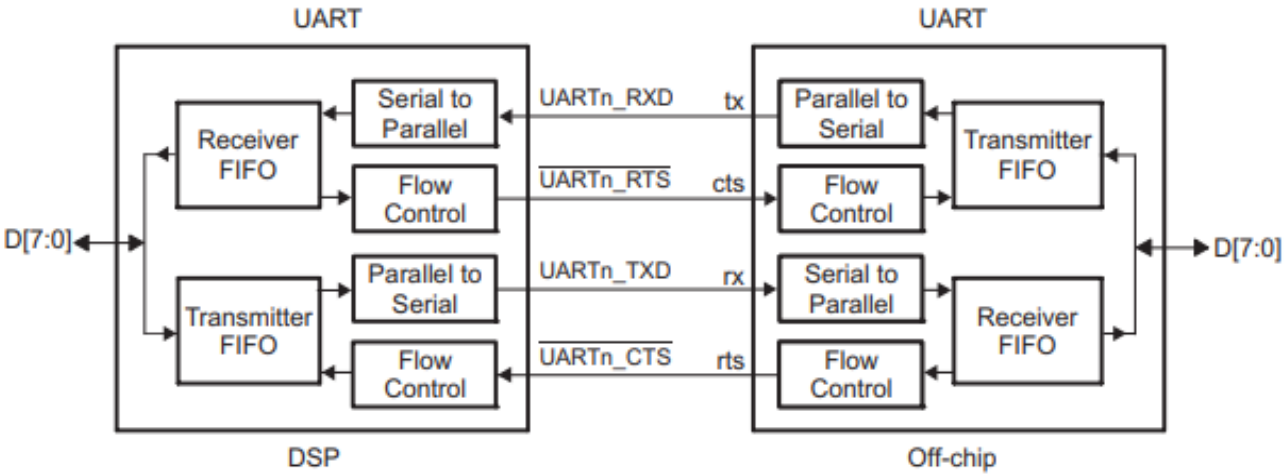

Рис.2.16. Спрощена схема *UART* з передаючою інформацією за чергою

2.3.Огляд протоколу зв'язку *MAVLINK*

Протокол *MAVLink* визначає механізм структури повідомлень і спосіб їх серіалізації на прикладному рівні. Потім ці повідомлення пересилаються на нижчі рівні (тобто транспортний рівень, фізичний рівень) для передачі в мережу.

Перевагою протоколу *MAVLink* є те, що він підтримує різні типи транспортних рівнів і середовищ завдяки своїй легкій структурі. Він може передаватися через *WiFi*, *Ethernet* (тобто мережі *TCP/IP*), послідовні шини(*MAVLink-over-serial*)(рис.2.17.) послідовні телеметричні канали низької смуги пропускання, що працюють на частотах нижче ГГц, а саме 433 МГц, 868 МГц або 915 МГц. Частоти ГГц дозволяють досягти великих діапазонів зв'язку для дистанційного керування безпілотною системою[5].

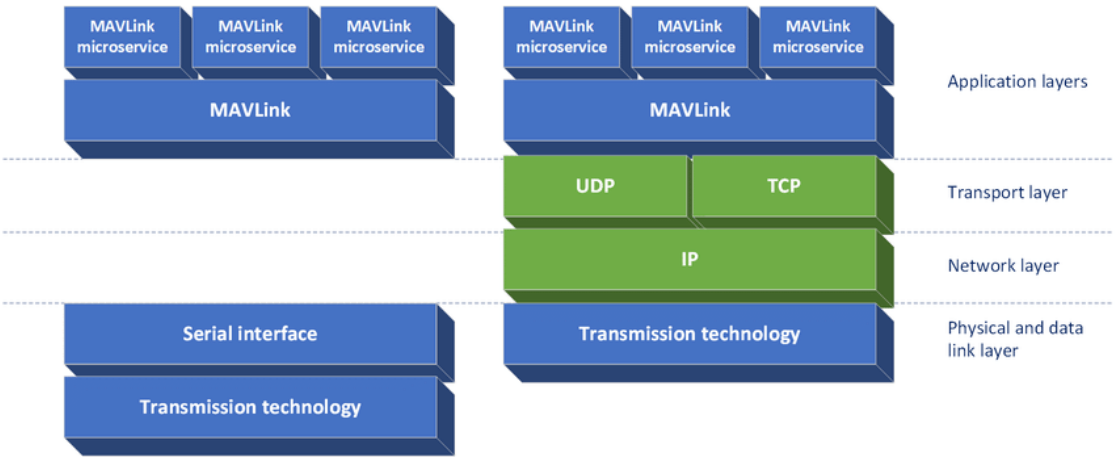

Рис.2.17. Стек *OSI MAVLink* через послідовні протоколи та через IP

Максимальна швидкість передачі даних може досягати 250 Кбіт/с, а максимальний радіус дії зазвичай становить 500 м, але це сильно залежить від навколишнього середовища, рівня шуму та налаштування антени.

Друга альтернатива полягає у використанні мережевого інтерфейсу, яким зазвичай є *WiFi* або *Ethernet*, і передачі повідомлень *MAVLink* через *IP*-мережі. У цьому випадку автопілот, що використовує протокол *MAVLink*, зазвичай підтримує як *UDP*, так і *TCP*-з'єднання на транспортному рівні між наземною станцією та

безпілотником, залежно від рівня надійності, якого вимагає додаток. Звичайно, загальновідомо, що *UDP* — це протокол, який не потребує з'єднання між клієнтом і сервером і не має механізму, який гарантував би надійну доставку повідомлень, але надає швидку легшу альтернативу для роботи в режимі реального часу та стійку до втрат. потокове повідомлення. З іншого боку, *TCP* є надійним протоколом, орієнтованим на з'єднання, який забезпечує кращу надійність передачі завдяки своєму механізму підтвердження, але може піддаватися перевантаженню та інтенсивному управлінню з'єднанням. Вибір транспортного протоколу залишається за користувачем залежно від вимог, необхідних йому для обміну повідомленнями між безпілотною системою та наземною станцією[5].

Типи та структури повідомлень Безпілотна система спілкується з наземною станцією через обмін повідомленнями *MAVLink*, які є двійковими повідомленнями. Двійкова серіалізація означає, що вміст повідомлення перетворюється на послідовність байтів для передачі через мережу. Одержувач запакованого повідомлення виконує його розпаковку в зворотному порядку, щоб відновити оригінальне надіслане повідомлення. Ця властивість двійкової серіалізації має значну перевагу, оскільки максимально зменшує розмір переданого повідомлення порівняно з іншими типами серіалізації, такими як *XML* або навіть легший *JSON*.

Кожне повідомлення *MAVLink* містить заголовок, доданий до корисного навантаження повідомлення. Заголовок містить інформацію про повідомлення, тоді як корисне навантаження містить дані, які транспортуються повідомленням..

Заголовок протоколу *MAVLink* 1.0: Як показано на рис. 2.18, є вісім важливих полів. Перше поле – це STX і відноситься до символу, який представляє початок кадру *MAVLink.* У *MAVLink* 1.0 *STX* дорівнює спеціальному символу *0XFE*. Другий байт (*LEN*) представляє довжину повідомлення в байтах і кодується в 1 байт. Третій байт (*SEQ*) позначає порядковий номер повідомлення. Він кодується в 1 байт і приймає значення від 0 до 255. Коли він досягає 255, порядковий номер знову скидається на 0 і збільшується в кожному згенерованому повідомленні. Порядковий номер повідомлення, увімкненого для виявлення втрат повідомлень у приймачі[5].

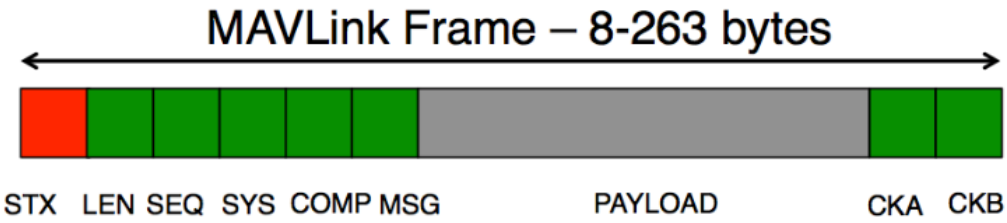

Рис.2.18. Структура кадру протоколу *MAVLink*

Четвертий байт *SYS* представляє ідентифікатор системи. Кожна безпілотна система повинна мати свій *System ID*, зокрема, якщо ними керує одна наземна станція. Ідентифікатор системи 255 зазвичай призначається наземним станціям. Одне з обмежень *MAVLink* 1.0 полягає в тому, що він обмежує кількість дронів, якими керує одна наземна станція, до 254, оскільки ідентифікатор системи кодується в 1 байт.

П’ятий байт — це ідентифікатор компонента, він визначає компонент системи, який надсилає повідомлення. У *MAVLink* 1.0 є 27 типів обладнання (тобто компонентів). Якщо немає підсистеми або компонента, то він не використовується. Шостий байт представляє ідентифікатор повідомлення (*MSGID*), який відноситься до типу повідомлення, вбудованого в корисне навантаження. Наприклад, ідентифікатор повідомлення, що дорівнює 0, відноситься до повідомлення типу *HEARTBEAT*, яке вказує на те, що система працює, і надсилається кожну секунду. Ще один приклад з ідентифікатором повідомлення, рівним 33, яке відноситься до повідомлення, яке містить *GPS*-координати безпілотної системи. Ідентифікатор повідомлення — це важлива інформація, яка дозволяє проаналізувати корисне навантаження та витягти з нього інформацію на основі типу повідомлення. Кожне повідомлення містить визначену кількість полів в двійковому форматі в певному порядку відповідно до стандартної специфікації.

Корисне навантаження розташовується відразу після ідентифікатора повідомлення і може займати максимум 255 байт. Нарешті, останні два байти для контрольної суми. *CKA* та *CKB* представляють перевірку циклічної надлишковості (*CRC*), розраховану за початковими значеннями *A* та *B* відповідно.

CRC гарантує, що повідомлення не було пошкоджено чи модифіковано під час його передачі, і що відправник і одержувач мають те саме повідомлення. Він обчислюється за допомогою хешу *ITU X.25/SAE AS-4* байтів повідомлення, за винятком поля *STX* (хеш застосовується до 6+*n*+1 байтів, а додаткове значення є початковим значенням).

Початковий код додається в кінці повідомлення під час обчислення *CRC*.

Мінімальна довжина повідомлення *MAVLink* 1.0 становить 8 байт для пакетів підтвердження без корисного навантаження. З іншого боку, максимальна довжина повідомлення *MAVLink* 1.0 становить 263 байти для повного корисного навантаження. Підсумок і пояснення кожного поля заголовка *MAVLink* 1.0 представлені в таблиці II. 2) Заголовок протоколу *MAVLink* 2.0: Заголовок протоколу *MAVLink* 2.0 було випущено на початку 2017 року та є поточною рекомендованою версією. Він зворотно сумісний із версією *MAVLink* 1.0 і містить кілька покращень порівняно з версією *MAVLink* 1.0. Спочатку ми починаємо з представлення заголовка протоколу *MAVLink* 2.0; потім ми висвітлюємо основні відмінності між двома версіями. На рис. 2 показана структура заголовка *MAVLink* 2.0. Заголовок *MAVLink* 2.0 поділяє всі поля з заголовком *MAVLink* 1.0 і додає нові поля на додаток до зміни розміру деяких існуючих полів. Перший байт — це маркер початку тексту, а його конкретне значення — *0xFD* для *MAVLink* 2.0 (на відміну від *0xFE* для *MAVLink 1.0*). Таким чином, синтаксичний аналізатор повинен спочатку розпізнати цей символ, перш ніж зможе проаналізувати інші поля повідомлення *MAVLink 2.0*. Довжина корисного навантаження є наступним полем і є таким самим, як і в попередньому протоколі. *MAVLink* вводить два нові прапорці перед порядковим номером (*SEQ*) повідомлення. Перші прапорці – це прапорці несумісності, які впливають на структуру повідомлення. Прапори вказують, чи містить пакет деякі функції, які необхідно враховувати під час аналізу пакета. Наприклад, прапор несумісності, що дорівнює 0x01, означає, що пакет підписаний і що підпис додається в кінці пакета[5].

Другий прапор — це прапори сумісності, які не впливають на структуру повідомлення. Він вказує на прапори, які можна проігнорувати, якщо вони не

зрозумілі, і це не заважає аналізатору обробити повідомлення, навіть якщо прапор не можна інтерпретувати. Наприклад, це може стосуватися прапорів, які вказують на пріоритет пакета (наприклад, високий пріоритет), оскільки це не впливає на структуру пакета. Порядковий номер (*SEQ*), ідентифікатор системи (*SYSID*) та ідентифікатор компонента (*COMPID*) такі самі, як і в заголовку протоколу *MAVLink* 1.0. Однак ідентифікатор повідомлення (*MSGID*) закодований у 24 біти замість 8 бітів у попередній версії, що дозволяє значно більшу кількість типів повідомлень у *MAVLink* 2.0, досягаючи до 16777215 можливих типів. Незрозуміло, що є причиною проектування такого величезного простору можливих типів повідомлень, оскільки кількість можливостей надто велика. Поле *Payload* може займати до 255 байт даних, що залежить від типу повідомлення. Контрольна сума подібна до однорангової в *MAVLink* 1.0. Нарешті, *MAVLink* 2.0 використовує необов'язкове поле підпису розміром 13 байт, щоб переконатися, що посилання захищене від несанкціонованого доступу. Ці функції значно покращують аспекти безпеки *MAVLink* 1.0, оскільки дозволяють автентифікувати повідомлення та перевіряти, що воно походить із надійного джерела. Підпис повідомлення додається, якщо прапорці несумісності встановлені на 0x01[5].

2.4. Огляд алгоритм шифрування DES

DES є блоковий шифр, він шифрує дані 64-бітовими блоками. З одного кінця алгоритму вводиться 64-бітовий блок відкритого тексту, з другого кінця виходить 64-бітовий блок шифротекста. *DES* є симетричним алгоритмом: для шифрування та дешифрування використовуються однакові алгоритм та ключ (за винятком невеликих відмінностей у використанні ключа).

Довжина ключа дорівнює 56 біт. (Ключ зазвичай представляється 64-бітовим числом, але кожен восьмий біт використовується для перевірки парності і ігнорується. Біти парності є найменшими бітами байтів ключа.) Ключ, який може бути будь-яким 56-бітовим числом, можна змінити в будь-який момент часу. Ряд

чисел вважаються слабкими ключами, але можна легко уникнути. Безпека повністю визначається ключем.

На найпростішому рівні алгоритм не становить нічого більшого, ніж комбінація двох основних методів шифрування: зміщення та дифузії. Фундаментальним будівельним блоком *DES* є застосування до тексту одиничної комбінації цих методів (підстановка, а за нею – перестановка), яка залежить від ключа. Такий блок називається етапом. *DES* складається з 16 етапів, однакова комбінація методів застосовується до відкритого тексту 16 разів (рис. 2.19)[7].

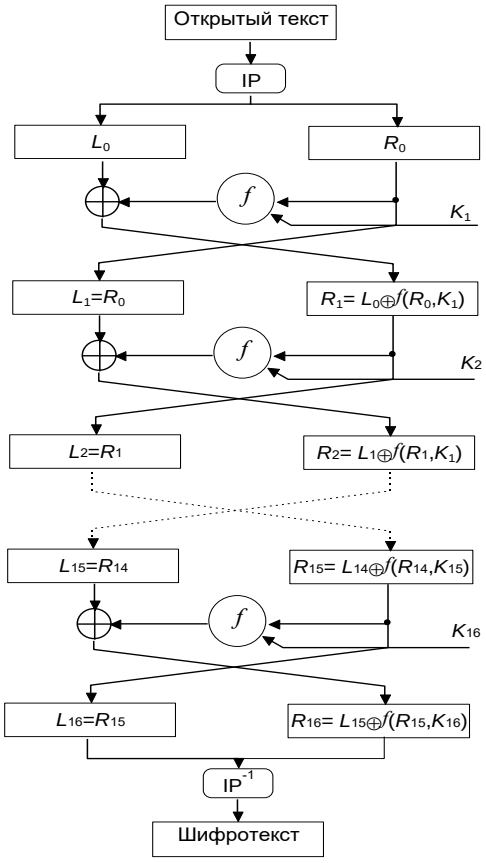

Рис.2.19. Структурна схема тракту шифротексту

Алгоритм використовує лише стандартну арифметику 64-бітових чисел та логічні операції, тому він легко реалізовувався в апаратурі другої половини 70-х. Достаток повторень в алгоритмі робить його ідеальним для реалізації у спеціалізованій мікросхемі ПЛІС. Початкові програмні реалізації були досить незграбні, але сьогоднішні програми набагато кращі.

DES працює з 64-бітовим блоком відкритого тексту. Після початкової перестановки блок розбивається на праву та ліву половини довжиною по 32 біти. Потім виконується 16 етапів однакових дій, які називають функцією *f*, в яких дані об'єднуються з ключем. Після шістнадцятого етапу права і ліва половини об'єднуються і алгоритм завершується заключною перестановкою (зворотною по відношенню до початкової)[7].

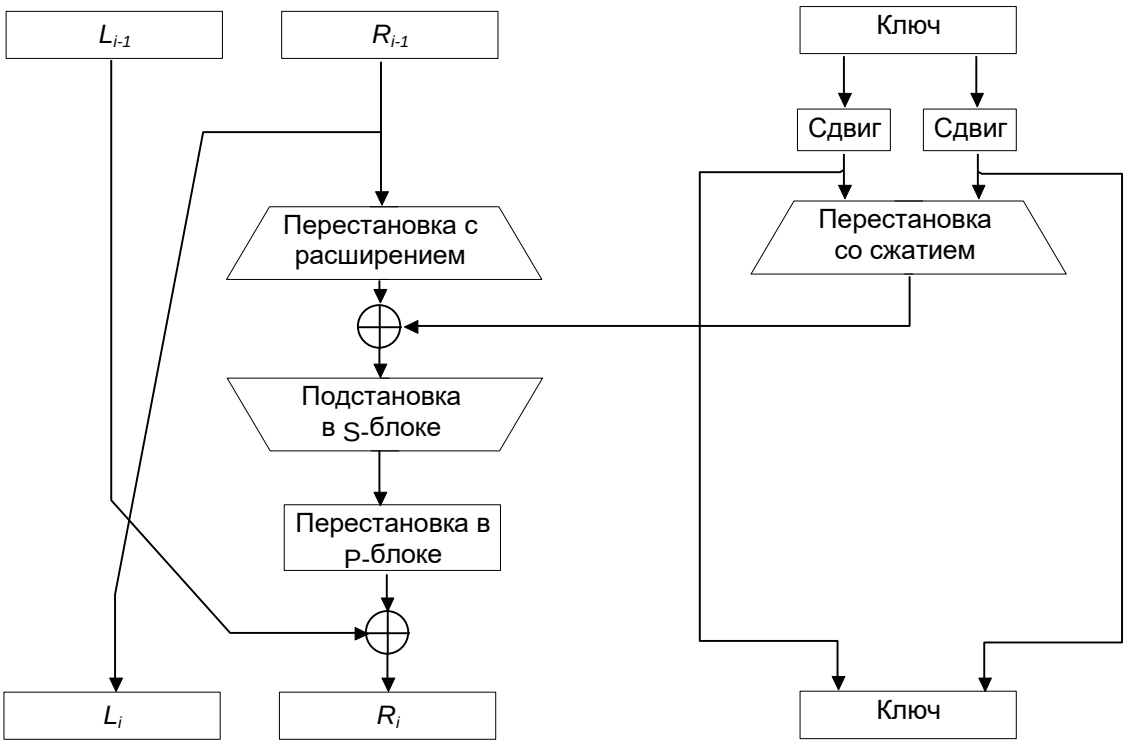

Рис.2.20. Структурна схема одного з 16 блоків шифрування

На кожному етапі (рис.2.20.) біти ключа зсуваються, і потім з 56 бітів ключа вибираються 48 бітів. Права половина даних збільшується до 48 бітів за допомогою перестановки з розширенням, об'єднується за допомогою *XOR* з 48 бітами зміщеного та переставленого ключа, проходить через 8 *S*-блоків, утворюючи 32 нових біти, і переставляється знову.

Ці чотири операції виконуються функцією *f*. Потім результат функції *f* поєднується з лівою половиною за допомогою іншого *XOR*. Через війну цих процесів утворюється нова права половина, а стара права половина стає нової лівої. Ці дії повторюються 16 разів, утворюючи 16 етапів *DES*.

Початкова перестановка виконується ще до етапу 1, при цьому вхідний блок переставляється, як показано в таблиці 2.2. Цю та всі інші таблиці цього розділу треба читати зліва направо та зверху вниз. Наприклад, початкова перестановка переміщає біт 58 в бітову позицію 1, біт 50 - бітову позицію 2, біт 42 - бітову позицію 3, і так далі[7].

Таблиця 2.2

Початковий блок перестановки шифротоексту

I	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	58	50	42	34	26	18	10	2	60	52	44	36	28	20	12	4
1	62	54	46	38	30	22	14	6	64	56	48	40	32	24	16	8
2	57	49	41	33	25	17	9	1	59	51	43	35	27	19	11	3
3	61	53	45	37	29	21	13	5	63	55	47	39	31	23	15	7

Початкова перестановка та відповідна заключна перестановка не впливають на безпеку *DES*. (Як можна легко помітити, ця перестановка в першу чергу служить для полегшення побайтного завантаження даних відкритого тексту та шифротексту в мікросхему *DES*. Не забувайте, що *DES* з'явився раніше 16- та 32-бітових мікропроцесорних шин.) Оскільки програмна реалізація цієї багатобітової перестановки нелегка (на відміну від тривіальної апаратної), у багатьох програмних реалізаціях *DES* початкова та заключні перестановки не використовуються. Хоча такий новий алгоритм не менш безпечний, ніж *DES*, він не відповідає стандарту DES і тому не може називатися *DES*[7].

Спочатку 64-бітовий ключ *DES* зменшується до 56-бітового ключа відкиданням кожного восьмого біта, як показано в таблиці 2.3. Ці біти використовуються лише для контролю парності, дозволяючи перевіряти правильність ключа. Після вилучення 56-бітового ключа для кожного з 16 етапів *DES* генерується новий 48-бітовий підключення. Ці підключи, *Ki*, визначаються в такий спосіб.

По-перше, 56-бітовий ключ ділиться на дві 28-бітові половинки. Потім половинки циклічно зрушуються ліворуч на один або два біти в залежності від етапу. Цей зсув показаний в таблиці 2.4.

Таблиця 2.3

Блок зменшення ключа

I	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	57	49	41	33	25	17	9	1	58	50	42	34	26	18
1	10	2	59	51	43	35	27	19	11	3	60	52	44	36
2	63	55	47	39	31	23	15	7	62	54	46	38	30	22
3	14	6	61	53,	45	37	29	21	13	5	28	20	12	4

Таблиця 2.4

Блок здвигів ключа

I	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Після зсуву вибирається 48 із 56 бітів. Так як при цьому не тільки вибирається підмножина бітів, а й змінюється їхній порядок, ця операція називається перестановкою зі стисненням. Її результатом є набір із 48 бітів. Перестановка зі стиском (також звана переставленим вибором) визначено в таблиці 2.5. Наприклад, біт зрушеного ключа позиції 33 переміщається в позицію 35 результату, а 18-й біт зрушеного ключа відкидається.

Через зсув для кожного підключення використовується відмінне підмножина бітів ключа. Кожен біт використовується приблизно в 14 з 16 підключень, хоча не всі біти використовуються точно однакове число разів.

Таблиця 2.5

Блок мапінгу ключа

I	0	1	2	3	4	5	6	7	8	9	10	11
0	14	17	11	2	1	5	3	28	15	6	21	10
1	23	19	11	4	26	8	16	7	27	20	13	2
2	41	52	31	37	47	55	30	40	51	45	33	48
3	44	49	39	56	34	53	46	42	50	36	29	32

Перестановка із розширенням. Ця операція розширює праву половину даних, *Ri*, від 32 до 48 бітів. Так як при цьому не просто повторюються певні біти, а й змінюється їхній порядок, ця операція називається перестановкою з розширенням. Вона має два завдання: привести розмір правої половини у відповідність з ключем для операції *XOR* і отримати більш довгий результат, який можна буде стиснути в ході операції підстановки. Однак головний криптографічний зміст зовсім інший. За рахунок впливу одного біта на дві підстановки швидше зростає залежність бітів результату від вихідних бітів даних. Це називається лавинним ефектом. *DES* спроектований так, щоб якнайшвидше домогтися залежності кожного біта шифротексту від кожного біта відкритого тексту та кожного біта ключа[7].

Перестановка з розширенням показана на рис.2.21. Іноді він називається *E*-блоком (від *expansion*). Для кожного 4-бітового вхідного блоку перший і четвертий біт є два біти вихідного блоку, а другий і третій біти - один біт вихідного блоку. У таблиці 2.6 показано, які позиції результату відповідають яким позиціям вихідних даних. Наприклад, біт вхідного блоку позиції 3 переміститься в позицію 4 вихідного блоку, а біт вхідного блоку позиції 21 - в позиції 30 і 32 вихідного блоку.

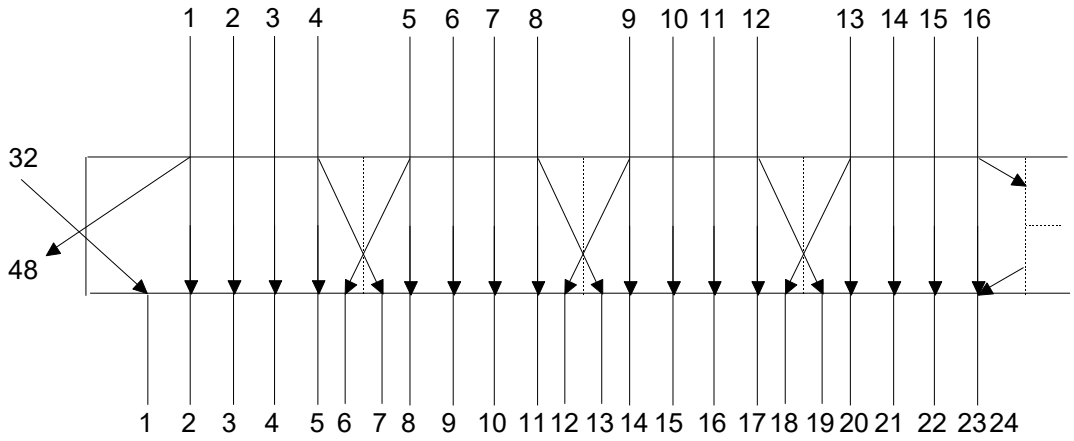

Рис.2.21. Блок перестановки розширенням

Таблиця 2.6

Блок перестановки розширенням

I	0	1	2	3	4	5	6	7	8	9	10	11
0	32	1	2	3	4	5	4	5	6	7	8	9
1	8	9	10	11	12	13	12	13	14	15	16	17
2	16	17	18	19	20	21	20	21	22	23	24	25
3	24	25	26	27	28	29	28	29	30	31	32	1

Підстановка за допомогою *S*-блоків Після об'єднання стисненого блоку з розширеним блоком за допомогою *XOR* над 48-бітовим результатом виконується операція підстановки. Підстановки виконуються у восьми блоках підстановки або *S*-блоках (від *substitution*). У кожного *S*-блоку 6-бітовий вхід і 4-бітовий вихід, всього використовується вісім різних *S*-блоків. (Для восьми *S*-блоків *DES* потрібно 256 байтів пам'яті.) 48 бітів діляться на вісім 6-бітових підблоків. Кожен окремий підблок обробляється окремим *S*-блоком: перший підблок – *S*-блоком 1, другий – *S*-блоком 2 тощо. (рис.2.22)[7].

Кожен *S*-блок є таблицею з 2 рядків і 16 стовпців. Кожен елемент у блоці є 4-бітовим числом. По 6 вхідних біт *S*-блоку визначається, під якими номерами стовпців і рядків шукати вихідне значення. Усі вісім *S*-блоків показані в Табл. 2.7-2.14

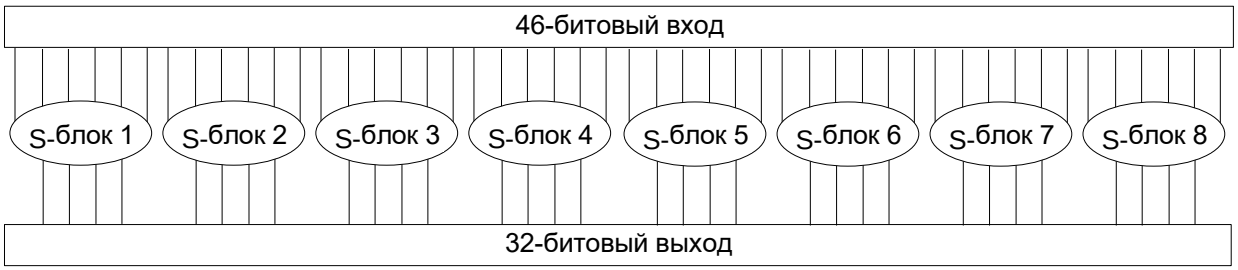

Рис.2.22. Структура *S* блоку

Таблиця 2.7

Заміна в *S*-блоці 1

I	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

Таблиця 2.8

Заміна в *S*-блоці 2

I	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
2	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
3	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

Таблиця 2.9

Заміна в *S*-блоці 3

I	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
1	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
2	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
3	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

Таблиця 2.10

Заміна в *S*-блоці 4

I	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
1	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
2	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

Таблиця 2.11

Заміна в *S*-блоці 5

I	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
1	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
2	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
3	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

Таблиця 2.12

Заміна в *S*-блоці 6

I	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
1	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
2	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
3	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

Таблиця 2.13

Заміна в *S*-блоці 7

I	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
1	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
2	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
3	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

Таблиця 2.14

Заміна в *S*-блоці 7

I	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
2	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
3	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

Вхідні біти особливо визначають елемент *S*-блоку. Розглянемо 6-бітовий вхід *S*-блоку: *b1*, *b2*, *b3*, *b4*, *b5* та b6. Біти b1 і b6 об'єднуються, утворюючи 2-бітове число від 0 до 3, що відповідає рядку таблиці. Середні 4 біти, *b2* по *b5*, об'єднуються, утворюючи 4-бітове число від 0 до 15, відповідне стовпцю таблиці. Наприклад, нехай на вхід шостого *S*-блоку (тобто, біти функції *XOR* з 31 по 36) потрапляє 110011. Перший і останній біт, об'єднуючись, утворюють 11, що відповідає рядку

шостого 3 *S*-блоку. Середні 4 біти утворюють 1001, що відповідає стовпцю 9 того ж *S*-блоку. Елемент *S*-блоку 6, що знаходиться на перетині рядка 3 та стовпця 9, - це 14. (Не забувайте, що рядки та стовпці нумеруються з 0, а не з 1.) Замість 110011 підставляється 1110.

Для ефективної реалізації даного елементу потрібно використати МДНФ або МДКНФ. Таким чином всі *S*-блоки будуть побудовані на основі логічних елементів будуть працювати значно швидше але через кількість вузлів ПЛМ які будуть використані буде великим. Тому для апаратної реалізації даного елементу потрібна значно більша матриця ПЛМ що означає використання значно дорожчої ПЛІС. Тому для реалізації даного елементу буде використовуватись тіло *process* та перехід від типу даних *std_logic_vector* до *integer* та навпаки. Це дозволить набагато легше реалізувати *S*-блоки програмно у вигляді масивів з 64 елементами. Для цього потрібно переупорядкувати елементи, що не є складним завданням. (Змінити індекси, не змінюючи порядок елементів, недостатньо. *S*-блоки спроектовані дуже ретельно.) Однак такий спосіб опису *S*-блоків допомагає зрозуміти, як вони працюють. Кожен *S*-блок можна розглядати як функцію підстановки 4-бітового елемента: *b*2 по *b*5 є входом, а деяке 4-бітове число – результатом. Біти *b*1 і *b*6 визначаються сусідніми блоками, визначають одну з чотирьох функцій підстановки, можливих в даному *S*-блоці.

Підстановка за допомогою *S*-блоків є основним етапом *DES*. Інші дії алгоритму лінійні та легко піддаються аналізу. *S*-блоки нелінійні, і саме вони більшою мірою, ніж решта, забезпечують безпеку *DES*. В результаті цього етапу підстановки виходять вісім 4-бітових блоків, які об'єднуються в єдиний 32-бітовий блок. Цей блок надходить на вхід наступного етапу – перестановки за допомогою P-блоків[7].

Перестановка за допомогою *P*-блоків. 32-бітовий вихід підстановки за допомогою *S*-блоків перетасовуються відповідно до *P*-блоку. Ця перестановка переміщає кожен вхідний біт до іншої позиції, жоден біт не використовується двічі, і жоден біт не ігнорується. Цей процес називається прямою перестановкою або

просто перестановкою. Позиції, у яких переміщуються біти, показано в таблиці. 2.15. Наприклад, біт 21 переміщається в позицію 4, а біт 4 - позицію 31.

Таблиця 2.15

Таблиця перестановки *P* блока

I	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	16	7	20	21	29	12	28	17	1	15	23	26	5	18	31	10
1	2	8	24	14	32	27	3	9	19	13	30	6	22	11	4	25

Нарешті результат перестановки за допомогою *P*-блоку об'єднується за допомогою *XOR* з лівою половиною початкового 64-бітового блоку. Потім ліва та права половини змінюються місцями, і починається наступний етап.

Заключна перестановка Заключна перестановка є зворотною по відношенню до початкової перестановки та описана в таблиці 2.16. Зверніть увагу, що ліва та права половини не змінюються місцями після останнього етапу *DES*, натомість об'єднаний блок *R16L16* використовується як вхід заключної перестановки. У цьому немає нічого особливого, перестановка половинок з наступним циклічним зрушенням призвела б до такого самого результату. Це зроблено у тому, щоб алгоритм можна було використовувати як шифрування, так дешифрування.

Таблиця 2.16

Таблиця кінцевої перестановки

I	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	40	8	48	16	56	24	64	32	39	7	47	15	55	23	63	31
1	38	6	46	14	54	22	62	30	37	5	45	13	53	21	61	29
2	36	4	44	12	52	20	60	28	35	3	43	11	51	19	59	27
3	34	2	42	10	50	18	58	26	33	1	41	9	49	17	57	25

Після всіх підстановок, перестановок, операцій *XOR* і циклічних зрушень можна подумати, що алгоритм дешифрування, різко відрізняючись від алгоритму шифрування, так само заплутаний. Навпаки, різні компоненти *DES* були підібрані так, щоб виконувалося дуже корисна властивість: для шифрування та дешифрування використовується той самий алгоритм. *DES* дозволяє використовувати для шифрування або дешифрування блоку одну й ту саму функцію. Єдина відмінність полягає в тому, що ключі повинні використовуватись у зворотному порядку. Тобто якщо на етапах шифрування використовувалися ключі $K1$, $K2$, $K3$, ..., $K16$, то ключами дешифрування будуть $K16$, $K15$, $K14$, ..., $K1$. Алгоритм, який створює ключ кожного етапу, також циклічний. Ключ зсувається праворуч, а число позицій зсуву дорівнює 0, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1[7].

2.5. Концепт пристрою шифрування

Основним технічним завданням поставлено створення пристрою потокового шифрування даних який дозволить модифікувати цивільні *FPV* дорни без суттєвих втручання в схемотехніку та програмне забезпечення системи управління. Це дозволить підвищити надійність каналу управління та запобігти втручання в його роботу. Постановка даних пріоритетів при створення даного пристрою дозволить легке інтегрування в вже існуючі БПЛА без складних технічних процесів. Спрощена схема блоків та між блочних протоколів зображено на рис.2.23.

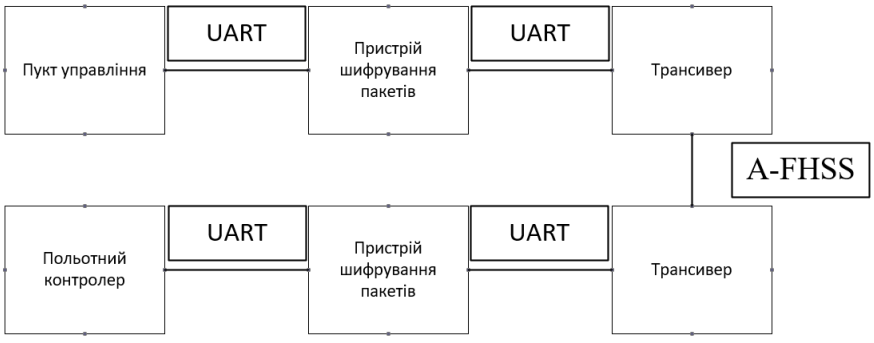

Рис.2.23. Спрощена схема блоків та між блочних протоколів тракт керуючого сигналу

Спочатку відбувається ініціалізація і налаштування периферії. Під час цього пристрій шифрування повинен напряму пропускати всі сигнали не оброблюючи їх (рис.2.24). Час ініціалізації периферії може бути різним. Але через можливість зміни внутрішніх процесів блоків монет включення пристрою шифрування буде визначатися системою затримки або вручну за допомогою кнопки.

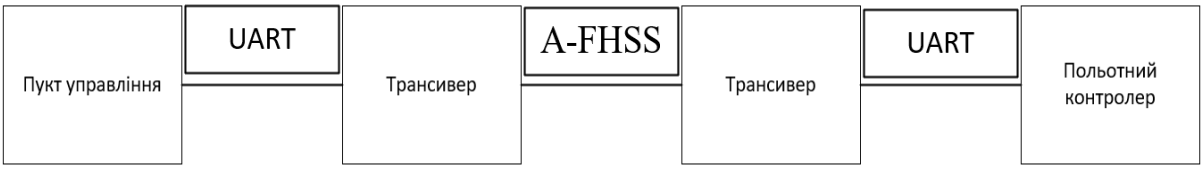

Рис.2.24. Тракт керуючого сигналу під час ініціалізації

Після ініціалізації периферії робота шини перетворюється на скрізну. В даний момент вигляд шини можна представити наступним чином (рис.2.25) оскільки робота трансиверів більше не впливає на дані які передаються по шині. Єдиа відмінність це присутність затримку передачі пакету через відстань.

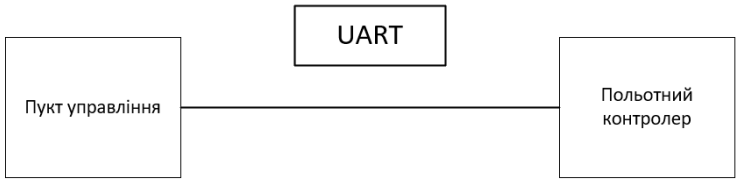

Рис.2.25. Тракт керуючого сигналу після ініціалізації

Після проходження певного часу підключається пристрій шифрування даних (рис.2.26). Оскільки *UART* має можливість перевірки цілісності кадрів кадр який був перервано буде або перезаписаний або проігнорований в залежності від налаштувань. *MAVLink* теж має свою систему перевірки цілісності пакету. Він теж може працювати як в режимі *TCP* з підтвердженням отримання пакети так і в *UDP*.

Ключ пристрою шифрування встановлений з початку, але може бути змінений з зовні. Оскільки зовнішній *EPROM* модуль пам’яті відсутній після від’єднання живлення встановлений сторонній ключ буде скинутий.

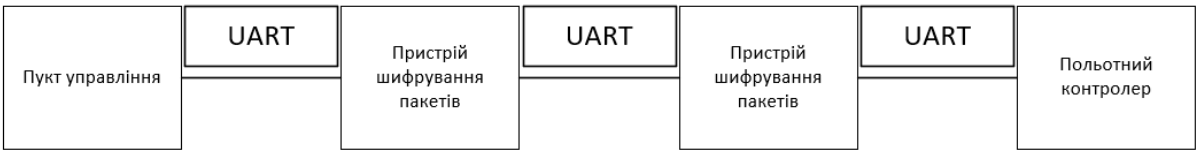

Рис.2.26. Тракт керуючого сигналу після ініціалізації з шифруванням каналу

Для зручної роботи з даним пристроєм без зміни в польотному контролері знадобиться додатковий пристрій. Цей інструмент дозволить встановлювати, синхронізувати ключи як в портативному режимі так стаціонарному при використанні з *API* через шину *USB* в режимі Virtual *COM*.

Алгоритм шифрування який буде використовуватися в даній реалізації це *DES* оскільки він є перевіреним часом та має в основі симетричний алгоритм. Недоліками даного вибору є розбиття пакету по 64 біти. *MAVLink V*1 дозволяє ігнорування *SYS ID*. Таким чином його довжина зменшується з 263 до 255. Що дозволяє вмістити пакет в 4 кадри шифрування. Але це не буде працювати з *MAVLink V*2 оскільки основна відмінність це наявність додаткових 13 біт цифрового підпису.

2.6.Специфікація шини *USB*

USB — це опитувана шина прийому та передачі даних. Хост-контролер ініціює всі передачі даних. Більшість транзакцій шини включають передачу до трьох пакетів. Кожна транзакція починається, коли Контролер хосту за розкладом надсилає *USB*-пакет із описом типу та напрямку транзакції, адреса *USB*-пристрою та номер кінцевої точки. Цей пакет називається «пакет маркерів»(*token packet*). *USB* Пристрій, який адресується, вибирає себе шляхом декодування відповідних полів

адреси. У певній транзакції дані передається або з хоста на пристрій, або з пристрою на хост.

Напрямок передачі даних є визначеними у пакеті маркеру. Потім джерело транзакції надсилає пакет даних або вказує, що у нього немає даних для передачі. Адресат, як правило, відповідає пакетом рукостискання, вказуючи, чи є передача пройшла успішно. Деякі транзакції шини між хост-контролерами та концентраторами передбачають передачу чотирьох пакетів.

Ці Типи транзакцій використовуються для керування передачею даних між хостом і повно-/низькошвидкісними пристроями. Модель передачі даних *USB* між джерелом або пунктом призначення на хості та кінцевою точкою на пристрої є називають каналом(*pipe*).

Існує два типи каналів: потік і повідомлення. Потокові дані не визначаються *USB* структура, тоді як дані повідомлення мають. Крім того, канали мають асоціації пропускної здатності даних, передачі тип служби та характеристики кінцевої точки, такі як спрямованість і розмір буфера. Більшість труб входять існування, коли пристрій *USB* налаштовано. Один канал повідомлень, контрольний канал за замовчуванням, завжди існує коли пристрій увімкнено, щоб надати доступ до конфігурації, стану та керування пристроєм інформації[8].

Розклад транзакцій дозволяє керувати потоком для деяких каналів потоку. На апаратному рівні це запобігає буферизується від ситуацій недовантаження або перевищення за допомогою рукостискання *NAK* для дроселювання швидкості передачі даних. Коли *NAKed*, транзакція повторюється, коли доступний час автобуса. Механізм керування потоком дозволяє побудова гнучких графіків, які враховують одночасне обслуговування неоднорідної суміші потоків труби. Таким чином, кілька потокових труб можна обслуговувати з різними інтервалами та пакетами різного розміру[8].

USB надає послуги зв’язку між хостом і приєднаними *USB*-пристроями. Однак простий вигляд, який бачить кінцевий користувач підключення одного або кількох *USB*-пристроїв до хосту(рис.2.27.), насправді трохи складніший у реалізації, ніж показано на рисунку. Щоб пояснити конкретні вимоги *USB* з точки зору різних

розробників, необхідні різні погляди на систему. Необхідно підтримувати кілька важливих концепцій і функцій, щоб забезпечити кінцевому користувачеві надійну роботу, яка вимагається від сучасних персональних комп'ютерів. *USB* представлено багаторівневим способом, щоб полегшити пояснення та дозволити розробникам конкретних продуктів *USB* зосередитися на деталях, пов'язаних із їхнім продуктом.

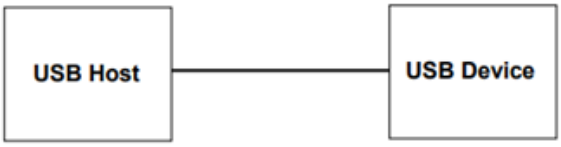

Рис.2.27. Взаємодія *Host*/*Device USB*

На рис.2.28 більш повна структура *USB*, розділяючи різні рівні системи, які будуть описані більш детально в решті специфікації. Зокрема, існує чотири напрямки реалізації.

1. Фізичний *USB*-пристрій: частина апаратного забезпечення на кінці кабелю USB, яка виконує деякі корисні для кінцевого користувача функції.

2. Клієнтське програмне забезпечення: програмне забезпечення, яке виконується на хості, що відповідає *USB*-пристрою. Це клієнтське програмне забезпечення зазвичай постачається з операційною системою або разом із пристроєм *USB*.

3. Системне програмне забезпечення *USB*: програмне забезпечення, яке підтримує *USB* у певній операційній системі. Системне програмне забезпечення *USB* зазвичай постачається разом з операційною системою, незалежно від конкретних пристроїв *USB* або клієнтського програмного забезпечення.

4. Хост-контролер *USB* (інтерфейс шини на стороні хоста): апаратне та програмне забезпечення, яке дозволяє приєднувати *USB*-пристрої до хосту. Існують спільні права та обов'язки між чотирма компонентами системи *USB*. Решта цієї специфікації описує деталі, необхідні для підтримки стійких, надійних потоків зв'язку між функцією та її клієнтом[8].

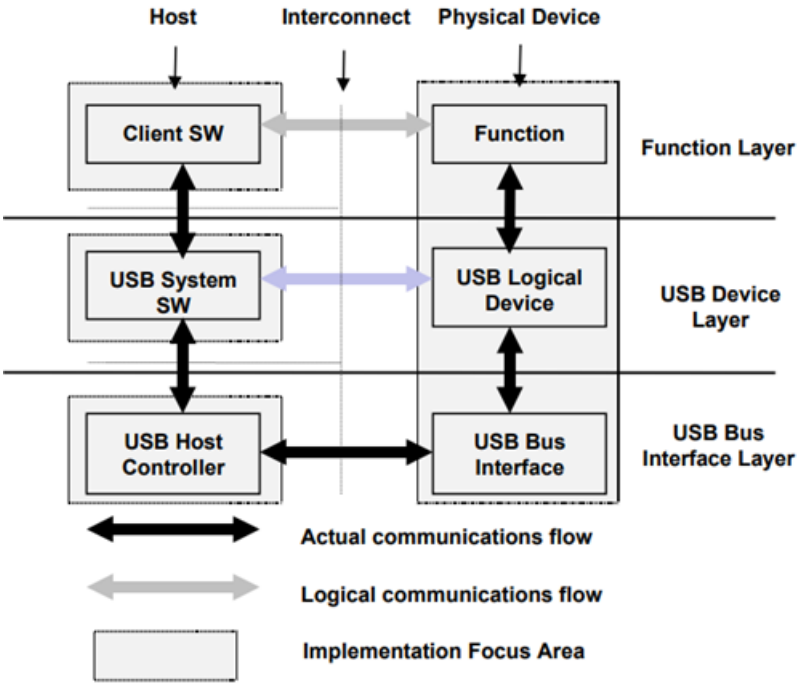

Рис.2.28. Рівні реалізації *USB*

Як показано (див. рис. 2.28) підключення хоста до пристрою вимагає взаємодії між кількома рівнями та об’єктами.

Рівень інтерфейсу шини *USB* забезпечує фізичне/сигнальне/пакетне з’єднання між хостом і пристроєм.

Рівень *USB*-пристроїв — це подання системного програмного забезпечення USB для виконання загальних операцій *USB* із пристроєм.

Функціональний рівень надає хосту додаткові можливості через відповідний відповідний рівень клієнтського програмного забезпечення. Рівні *USB*-пристрою та функції мають уявлення про логічний зв’язок у межах свого рівня, який фактично використовує рівень інтерфейсу шини *USB* для здійснення передачі даних. Фізичний вигляд зв’язку через *USB* пов’язаний із виглядом логічного зв’язку Для опису та керування *USB*-зв’язком важливі наступні поняття:

– топологія шини;

– моделі потоку зв’язку як відбувається зв’язок між хостом і пристроями через *USB*, і визначають чотири типи передачі через *USB*;

– керування доступом до шини: як керується доступом до шини в хості для підтримки широкого діапазону комунікаційних потоків за допомогою пристроїв *USB.*

2.7.Надійність системи

Через особливості умов експлуатації даної системи необхідно забезпечити стабільність її роботи. Відказостійкість системи залежить від сукупності програмного та апаратного забезпечення. Оскільки за умовами концепту системи відсутня можливість модифікації прошивки льотного контролера для перевірки критичних помилок можна лише опрататися на *HEARTBIT* протоколу *MAVLnk*.

Основною перешкодою для роботи даної системи будуть електромагнітні наводки. Існує два методи боротьби з ними. Основний находиться в площині апаратного рішення. Це екранування важливих елементів. Окрім перешкод для керуючого електромагнітні шуми можуть впливати на роботу контролера. А саме за допомогою індуктивних струмів впливати на систему живлення та їх трактування. Ще однією частою проблемою в відказ стійких системах це можливість в випадку некоректної роботи програми автоматичне перезавантаження. Оскільки в такому випадку немає жорсткого прив’язування виводу *RSCLR* до плюса живлення. Лише через резистор.

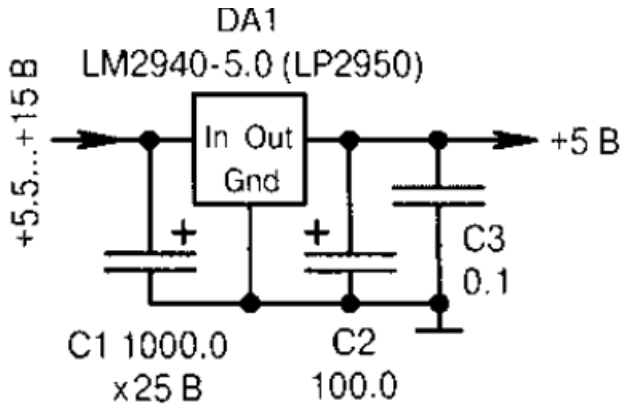

Рис.2.29. Схема стабілізації живлення мікроконтролера

Проблеми та їх апаратні рішення які будуть задіяні мають наступний вигляд:

Перша проблема це стабілізація живлення для ПЛІС та відсіювання ВЧ шумів оскільки вони через паразитну ємність можуть впливати на роботу систем трактування. Схема стабілізації живлення зображена на рис.2.29.

В основі даної системи стабілізації лежить *LM*. Основними недоліками використання такого рішення це відсутність захисту від короткого замикання. Таке може відбутися в випадку пробиттям статикою ПЛІС. Як правило при виході з ладу через теплове перенавантаження *LM* перетворюється в перемичку через це повне живлення в разі якщо воно завищене попаде на землю що може призвести до повного знеструмлення дрону скільки більшість полілітійіоних акумуляторів мають систему захисту від короткого замикання. Ще з недоліків можна віднести відсутність можливості підвищення напруги в разі її довгострокового падіння. Від корок строкових просадок напруги дана схема має конденсатори *C1* та *C2*. Конденсатор *C3* виконує дві функції. Перша це фільтрація високочастотних шумів. Друга це зменшення індуктивної складової конденсатора *C2* що скасується на протидію високочастотним шумам. Ще одним плюсом даної схеми можна відзначити реалізації системи зворотного зв’язку як можна ближче до системи управління. Це також зменьшує вплих шумів на стабілізацію невідмінно від систем з зовнішнім зворотнім зв’язком на основі мікросхем серії *TL*. Оскільки кожен мм некранового кабелю ловить наводки і впливає на роботу схеми.

Друга проблема полягає в протидії шумам та перенапругам по керуючим пін мікропроцесорної системи. Розгляномо схему виводів які основані на розповсюдженій технології *PUSH/PULL*(рис.2.30.).

При ініціалізація вбудованій периферії мікропроцесорних систем як правило *GPIO* ініціалізується в третьому стані або в режимі порта вводу. Оскільки при ініціалізації в режимі вводу пін перетворюється в з’єднання через ще один каскад push pull в регістр *ODR* або *BRSS* (залежить від архітектури *GPIO*).

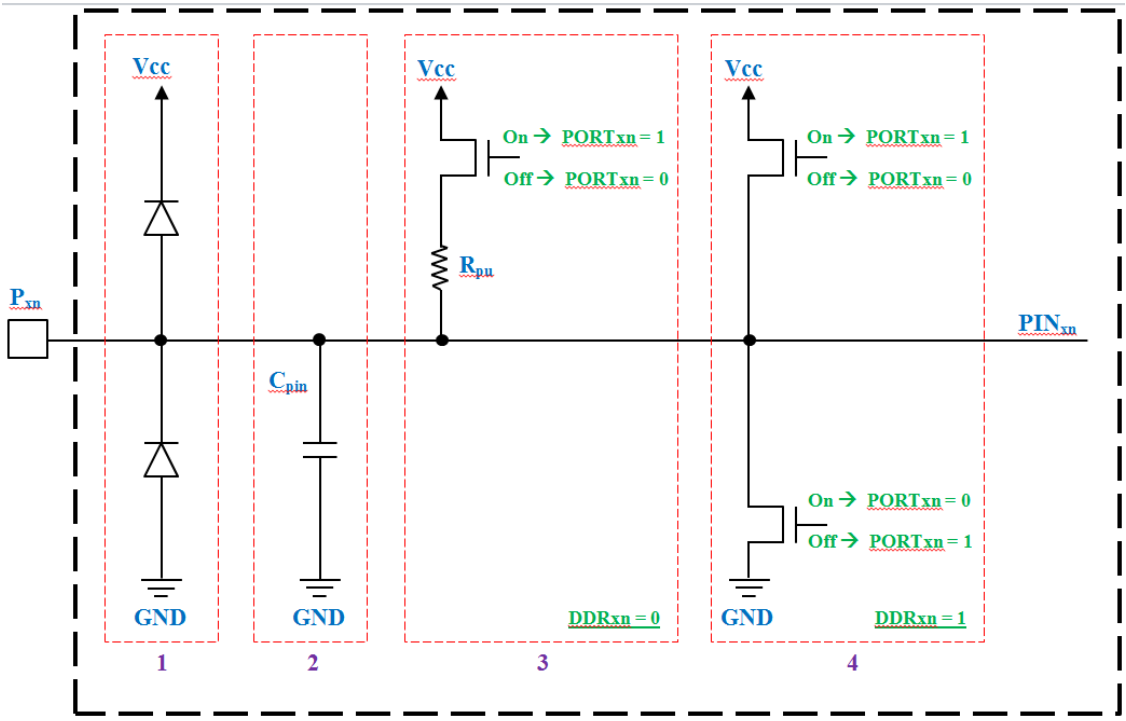

Рис.2.30. Спрощена схема пристрою *GPIO* з каскадом *PUSH/PULL*

В разі попадання на даний пін високого потенціалу може відбутися в кращому випадку виводиться з ладу периферія *GPIO* в гіршому через шину даних чи матрицю вибиває мікропроцесор. Задля запобігання цьому в пристрої *GPIO*(див.рис. 2.31) використовується схема з захисними діодами. При напрузі більше напруги живлення лишній потенціал заземляється через плюс живлення. В випадку коли потенціал відносно землі менше за неї потенціал заземляється об землю. Даний механізм захищає систему від статики. Струм доволі таки малий. Тому для запобігання таких ситуацій всі піни що не використовуються садять на землю. Це можна виконати як програмно встановивши логічний нуль так і апаратно шляхом заземлення через резистор. Використовують обидва варіант.

Системи підвищеної надійності повинні вміти витримати перепади напруги в 600В. Тому всі інформаційні піни чи піни управління незалежно від типу мікропроцесорної системи захищають. Для захисту пінив що використовуються від перепадів напруги за допомогою наступних схем.

Варіант захисту портів вводу/виводу за допомого супресорів на каналі. Доволі таки проста та повиринав схема захисту. Схема зображена на рис.2.31. Як правило в промислових схемах використовують не звичайні стабілітрони а *TVS* діоди.

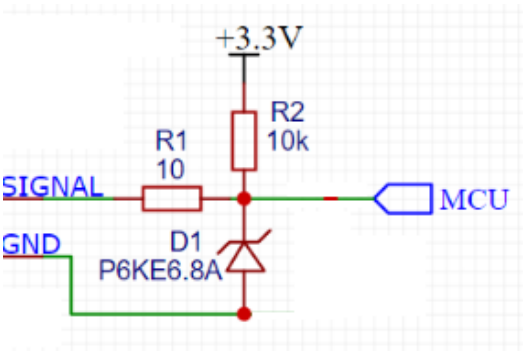

Рис.2.31. Схема захисту портів мікропроцесорних систем на основі супресорів

В разі проникнення високої напруги на каналі при перевищенні наруги в 6.8В струм піде через резистор *R1* та діод *D*1. Як правило *GPIO* що розраховано на 5В залишається робочим. Не може бути використаним для мікропроцесорних систем з напругою 3.3В оскільки це призведе як мінімум до відмови функціонування порту.

Перевагами даної схеми можна зазначити простоту реалізації, Автоматичною підтяжкою до плюса живлення(в разі обрива лінії не буде хибних спрацьовувань.)

До недоліків можна віднести певні вимоги для пристрою що подає сигнал на порту *MCU* оскільки струм витоку *TVS* досить великий. Тому резистор *R*1 необхідно брати мінімально можливий для системи.

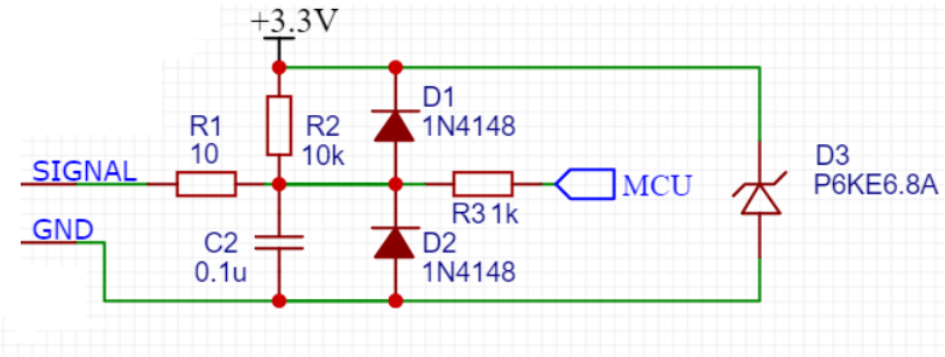

Рис.2.32. Схема захисту портів мікропроцесорних систем на основі супресора та захисних дублюючих діодів

Наступна схема яка широко використається для захисту мікропроцесорних систем в основі має принцип дублювання захисних діодів(рис.2.32) які вже як правило містяться в периферії мікропроцесорних систем *GPIO*.

Основним критеріями до діодів *D*1 та *D*2 це швидкодія та мінімальна напруга падіння на переходах. Оскільки в разі виникнення аварійної ситуації напруга на піні периферії буде рівна сумі наруги живлення та напруги падіння на діоді для перешкоди позитивного потенціалу відносно землі та тільки напруги падіння на переході для перешкод негативного потенціалу відносно землі.

Як і попередній схемі під час обриву лінії лінія не буде “висіти в повітрі” а лінія буде притягнута до плюса живлення. Резистор *R*3 вираховується залежно від частоти оскільки вхід периферії мікропроцесорної системи *GPIO* має ємність. Тому утворюється *RC* фільтр. Розрахунки відбувається на основі частоти каналу.

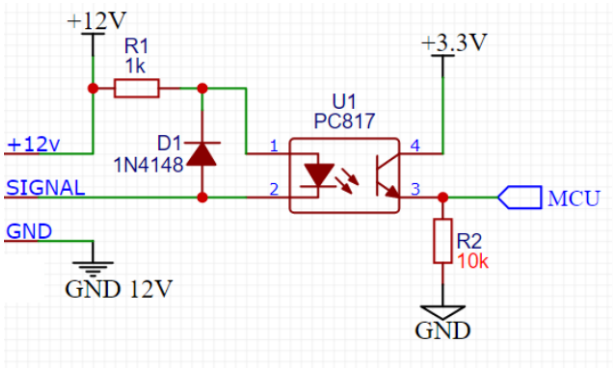

Рис.2.33. Схема захисту портів мікропроцесорних систем з гальванічною розв’язкою на основі оптрона

Для захисту мікропроцесорної системи використовують схеми з гальванічним розв’язками(рис.2.33). Використання даної схеми дозволяє вирішить декілька проблем. Основні це можливість узгодження напруги логіки та гальванічна розвязка частин схеми. Таке рішення дозволяє нам використовувати будь яку напругу сигналу. В основі даної схеми лежить оптрон. Необхідна напруга для відкриття формується за допомогою резистора *R*1. Струми для робти аптрону зі сторону діода великий. *Я*кщо брати поширений *PC*817 струм становить 20мА. Стандартний

максимальний струм зовнішньої переферерії для більшості периферії *GPIO* повинен бути 20мА. Допуск складає 8мА.

Одеж недоліками даної система є одно направленість сигналу в каналі де застосовується дана схема та неможливість використання для більшості мікропроцесорних систем у яких максимальний струм дорівнює 20мА без використання додаткового каскаду. Додавання додаткового каскаду призводить до збільшення кількості перехідних процесів що при високих швидкостях роботи шини може впливати на кількість помилок при передачі.

2.8. Підбір елементної бази

В основі даного проекту буде використовуватись ПЛІС на основі *FPGA*. В подальшому для масового виробництва на основі проектах можливим є переведення на *ASIC*. Оскільки пристрій шифрування буде встановлено на автономну систему важливим критерієм є енергоефективність.

Таблиця 2.17

Основні параметри ПЛІС на основі *EP3C10*

Кількість логічних елементів	10 320 одиниць
Кількість M9K блоків	46 одиниць
RAM	423 936 біт
Блоки множення 18x18	23 одиниці
Затримка скрізного проходження сигналу від *IOB* до *IOB*	4.7 нс
Кількість портів вводу/виводу	182
Підтримка конфігураційних схем	*Active Serial(AS) Active Parallel(AP), Fast Passive Parallel(FPP), Join Test Action Group(JTAG)*
Підтримка напруги вводу/виводу	1.2В, 1.8В, 2.5В, 3.3В.

При переведенні проекту на *ASIC* що дозволить в значній степені знизити енергозатрати та зменшити вартість однієї одиниці. Для відладки проекту буде використовуватися ПЛІС *EP3C10*. Характеристик даної ПЛІС наведені у таблиці 2.17.

Згідно заданої концепції необхідне зовнішній інструмент для швидкої роботи та відгадки пристрою шифрування. За основу такого інструменту буде використаний мікроконтролер S*TM32C8T6*. Параметри даного мікроконтролера надані у таблиці 2.18. На рис.2.34 наведена архітектура периферійних блоків та між блочних шин. *STM32C8T6* має вбудований кварцовий резонатор що сприяє надійності системи в боротьбі з електромагнітними шумами оскільки відсутні провідники та конденсатори для підключення зовнішнього кварцового резонатора.

Таблиця. 2.18

Характеристики мікроконтролера *STM32C8T6*

Основні характеристики бллоків	Параметри
EEPROM	1 Кб
Аналогові входи	10 каналів 12 розрядів
Вхідна напруга (рекомендована)	3.3В
ОЗП	8 кб
Постійний струм для виводу 3.3 В	50мА
Постійний струм через вхід/вихід	40мА
Робоча напруга	3.3 В
Частота CPU	72*MHz*
Тактова частота	8*MHz*
Флеш-пам’ять	64 Кб *Flash* 20Кб *SRAM*
Корпу	*LQFP* 48

Використання даного контролеру є доцільним оскільки він дешевий та містить велику кількість вбудованої периферії що дозволить легко модифікувати його під необхідні потреби.

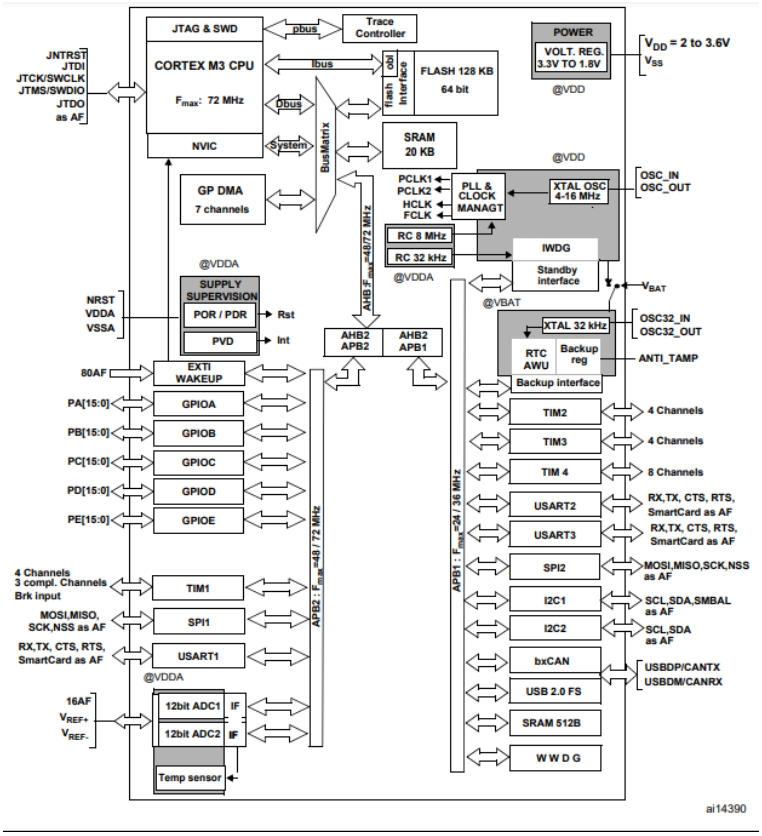

Рис.2.34. Внутрішня архітектура *STM32F103C8T6*

Реалізація системи захисту портів від електромагнітних наведень буде реалізована за допомогою дублювання захисних діодів за допомогою діодів Шотки. Через виклику швидкодію та малу напругу падіння на переході знайшли широке використання в системах захисту мікропроцесорних систем. Оскільки пристрій повинен витримувати короткочасні перепади напруги до 600 В вирішено обрати в якості захисних діодів . Характеристики подані в таблиці 2.19. Оскільки пристрій буде зібрана на *SMD* деталях краще обрати корпус *SMA*. Даний вибір обґрунтований зворотнім струмом. Оскільки в такому режимі шина завжди буде під навантаженням швидкість розсіювання статики збільшується.

Таблиця. 2.19

Характеристики діода *US1M*

Параметр	Одиниці вимірювання	Значення
Максимальна робоча напруга.	В	1000
Максимальний прямий струм при 60Гц.	А	1.0
Максимальна напруга падіння на переході при прямому включенні.	В	1.3
Максимальний постійний струм при зворотному включенні.	А	0,0025
Час відновлення при перемиканні між прямим та зворотнім включенням	нС	50
Ємність при 1МГц та напрузі 4В.	пФ	15

2.9. Інструментарій для розробки

Для роботи з FPGA використовуються спеціальні системи автоматизованого проектування. Для програмування *FPGA* використовуються мови опису апаратури(*HDL*). Існує багато середовищ розробки оскільки кожен виробник має своє середовище для розробки під їх архітектуру. Наприклад компанія *Intel*(*Altera*) має свою мову для опису апаратури *A-HDL*(*Altera Hardware Description Language*). Розробка буде вестись на більш універсальній мові *VHDL*.

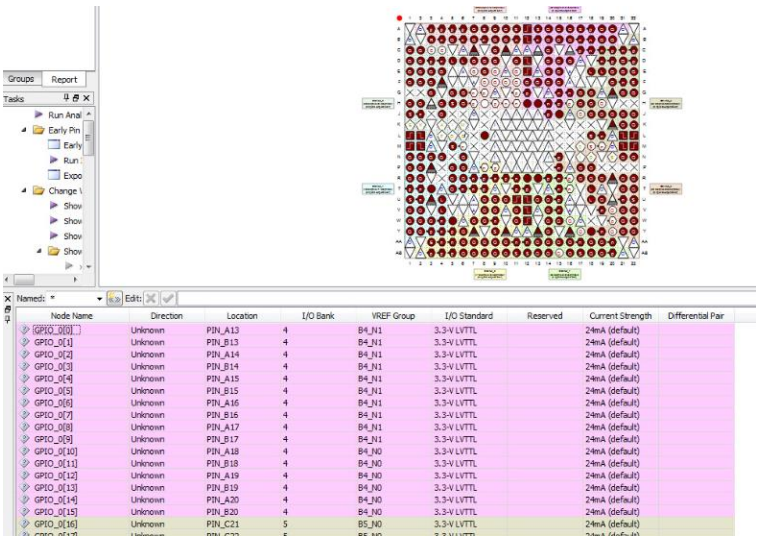

Рис.2.35. *Quartus* Конфігурація *FPGA*

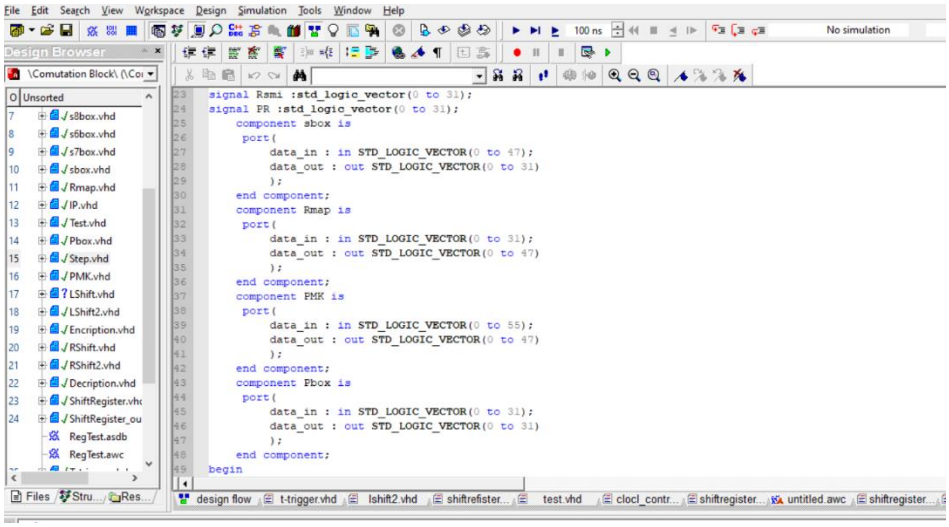

Рис.2.36. Інтерфейс прогами *AHDL*

Для роботи з *FPGA* Cyclone буде використовуватись *Quartus* оскільки це програмне забезпечення *Altera*. В даному проекті за допомогою нього буде описана лише конфігурація входів виходів(рис.2.35). Вся інша структура буде виконуватись на САПР *AHDL* оскільки *Quartus* має багато додаткових перевірок та довгу компіляцію файлу. Також *Quartus* буде використовуватись для перевірок обмежень архітектури *Cyclone* оскільки проект вже враховує її особливості. Симулятор створений на основі пакету *Simulink.*

В *AHDL*(рис.2.36) є певні переваги при розробці. Але основний недолік неможливість опису *I/O* блоків. Тому описати конфігурацію пінів для *Cyclone* є

неможливим. Але він має свої переваги такі як зручність візуального інтересу, швидка компіляція та відсутність необхідності при симуляції вказувати *Top Level Design* оскільки при симуляції в автоматичному режимі підтягує всі описані компоненти.

Для розробки інструменту встановлення ключів буде використовуватись плата *Blue Pill* з мікроконтролером *STM32F103C8T6* в основі якого лежить мікропроцесор *CORTEX-M* на основі *ARM* архітектури. При розробці буде використано дві програми *STM32CubeMX* та *STM32CubeIDE*.

STM32CubeMX(рис.2.37) використовується для спрощеної конфігурації проекту. До конфігурації відносяться рівень програмування (бібліотеки *HAL*, *CMSIS*, *LL*), налаштування системи трактування внутрішньої периферії та налаштування периферії. В основі створеного проекту лежить *CMAKE*. Це дозволяє легко портувати проект під будь-який *IDE*.

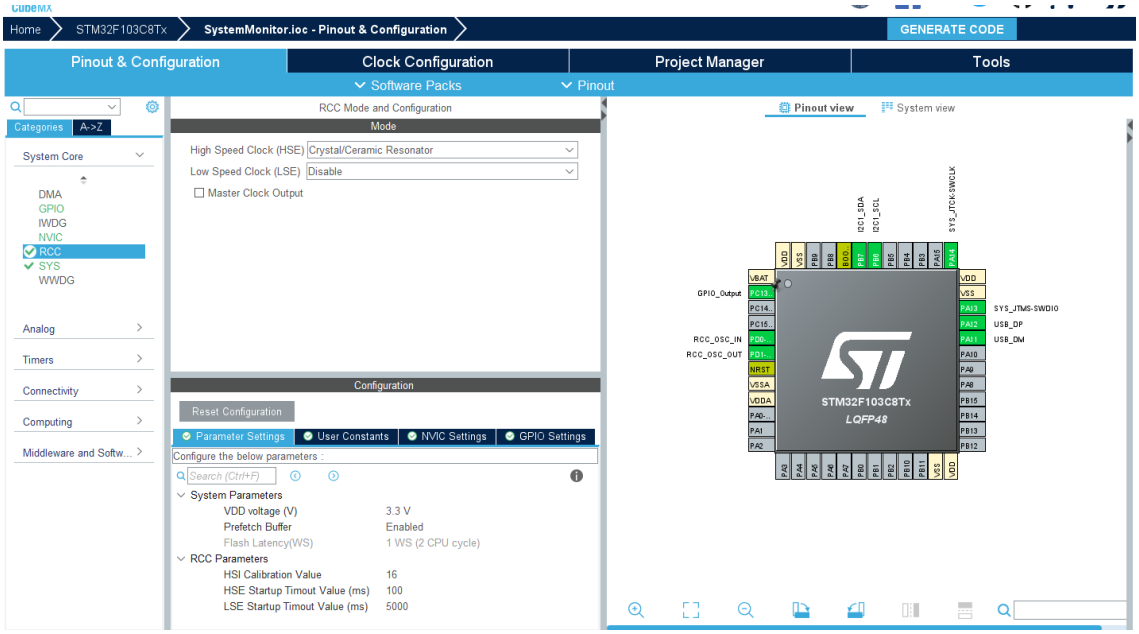

Рис.2.37. Інтерфейс програми *STM32CubeMX*

STM32CubeIDE(рис.2.38) використовується для розробки програмної частини проекту. В основі має *IDE Eclipse* з вбудованими бібліотеками *MDK ARM*. Для роботи з проектом використовує *CMAKE*. Має можливість прямого відгадки програми на робочому залізі. Для цього використовує *GDB* сервер.

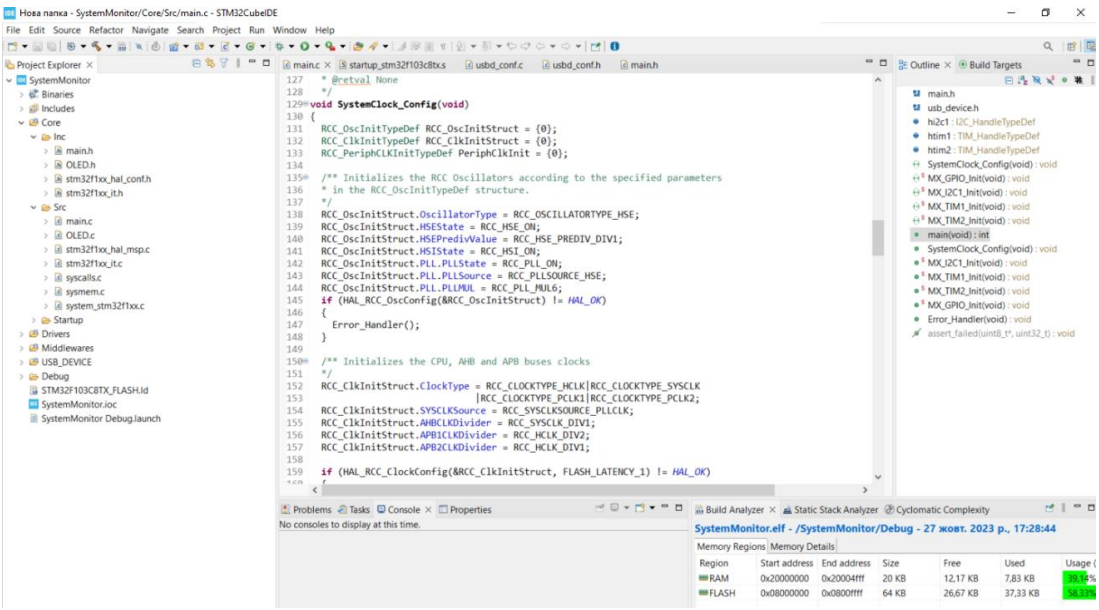

Рис.2.38. Інтерфейс програми *STM32CubeIDE*

Висновок за розділом

В даному розділі визначено концепт пристрою шифрування . Це пристрій оснований на *FPGA Cyclone*. Для полегшення взаємодії та швидкого налаштування необхідно також розробити спеціальний інструмент. Він буде базуватися на мікроконтролері *STM32F103C8T6*. В якості програмного забезпечення для розробки буде використано наступні програми: для розробки для FPGA частини *Quarts* та *AHDL*, для *STM32 STM32CubeMX*, *STM32CubeIDE*. В якості алгоритму шифрування буде виступати алгоритм *DES*. В якості захисту пнінв вводу виводу буде ви користуватись методика дублювання захисних діодів.

РОЗДІЛ 3

РЕАЛІЗАЦІЯ ПРИСТРОЮ АПАРАТНОГО ШИФРУВАННЯ ДАНИХ НА ОСНОВІ *FPGA*

На основі попередніх розділів пристрій буде представляти з себе *UART* to *UART* пристрій шифрування на основі алгоритму *DES*. Загальна схема підключення блоків один до одного зображена на рис. 3.1.

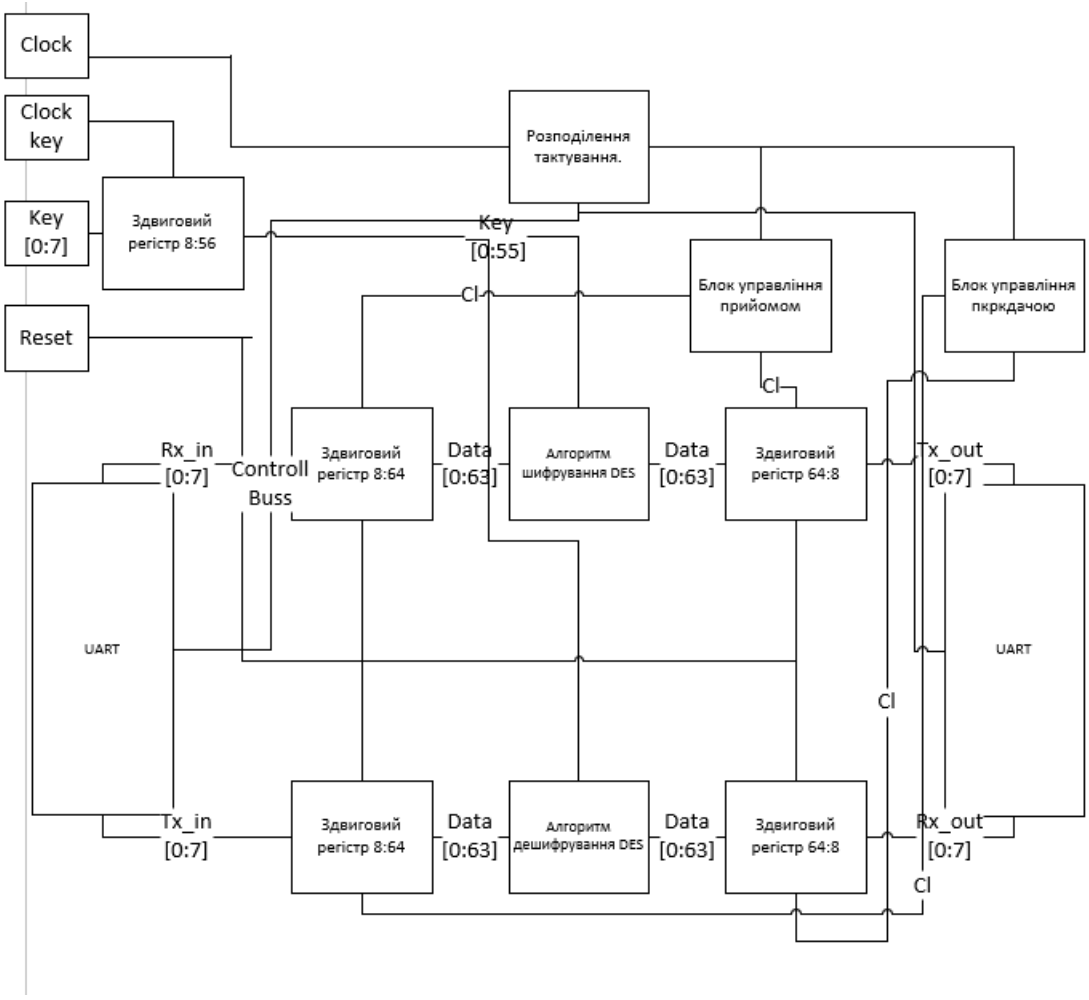

Рис.3.1. Загальна схема з’єднання блоків та між блочних зв’язків

Систему можна умовно розбити на дві частини. Перша це тракт проходження шифротексу а друга це система управління автоматом

Розглянемо детально тракт проходження шифротексу(рис.3.2). Дана система складвється з двогх однакових частин. Це тракт прийому та передачі. Вони складаються з вхідного *UART* пристрою, вихідного *UART* пристрою, пристрїв шифрування та дишифрування за алгоритмом *DES* та додатковою переферією для розбиття 64 бітового слова на слова по 8 біт. Цю ролі відіграють спеціальні здвигові регістри. Вхідні мають розмірність 8:64 а вихідні навпаки 64:8.

Наступною частиною цього тракту є пристрій задання ключа. Для цього теж використовується здвиговий регістр подібних до описаних вище. Єдина відміність це розмірність 8:56. Його керування відбувається незалежно від внутрішньої системи курування автоматом. Для зручної взаємодії з ним необхідно маси спеціальний інструмент описаний в другому розділі.

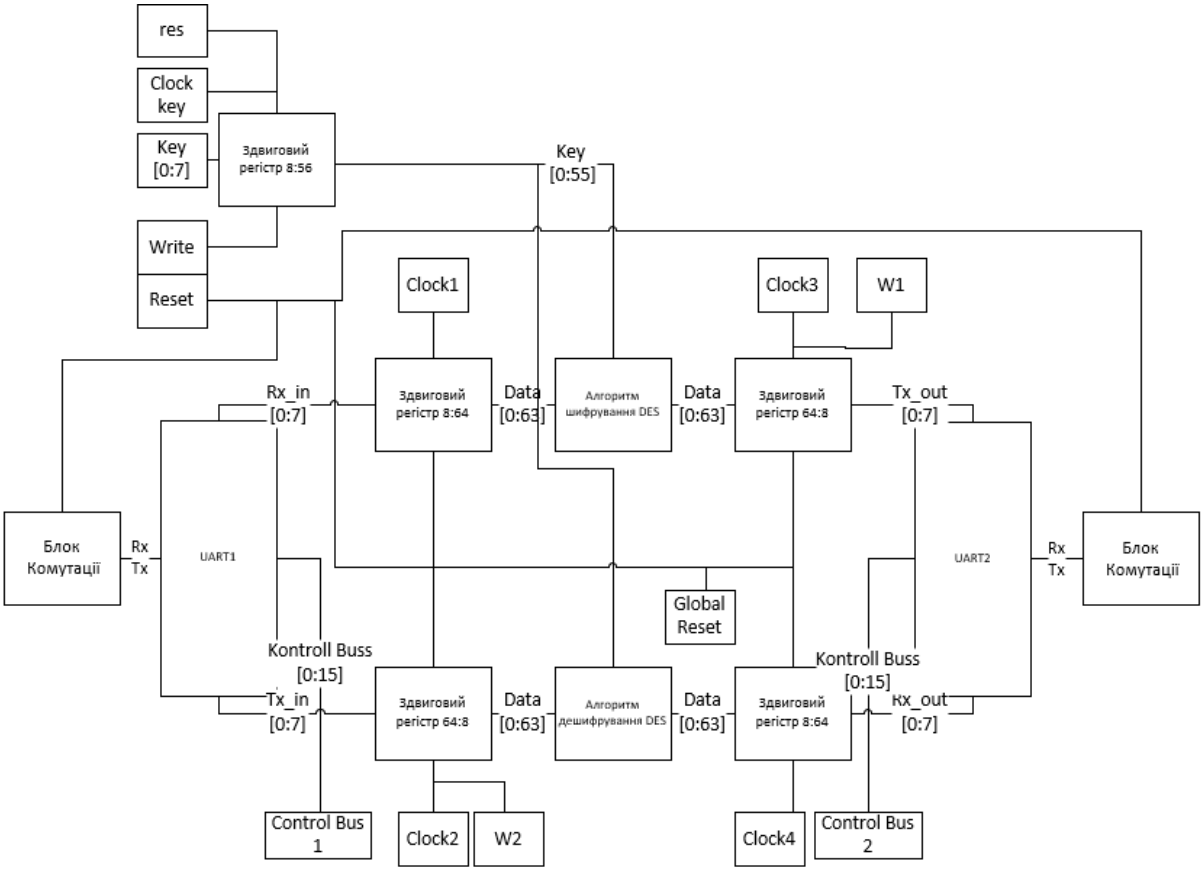

Рис.3.2. Схема тракту шифротексту

Розглянемо блоки управління(рис. 3.3). Основне завдання системи управління це керування вхідними та вихідними здвиговими регістрами на основі отриманих керуючих сигналів від апаратного пристрою *UART*. Розглядати систему управління необхідно як дві симетричні підсистеми. Кожна з них однаково керує лише своїм напрямом. Основним елементом керування є блоки управління прийому/передачі.

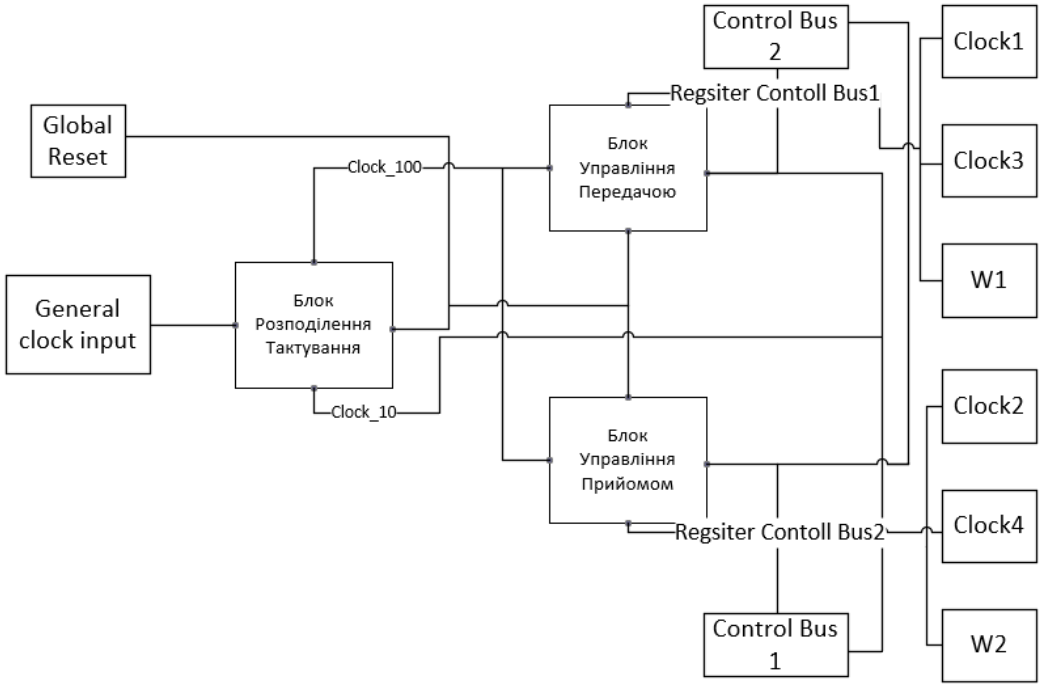

Рис.3.3. Схема управління

3.1.Апаратний пристрій UART

Блок апаратної реалізації *UART* необхідно розглядати як дві однакові окремі тракти. Спрощений вигляд даного тракту зображено на рис.3.5. В основі лежить принцип передачі 8-ми бітнго слова по послідовній шині. Часова діаграма роботи одго блоку *UART* зображена на рис.3.4.

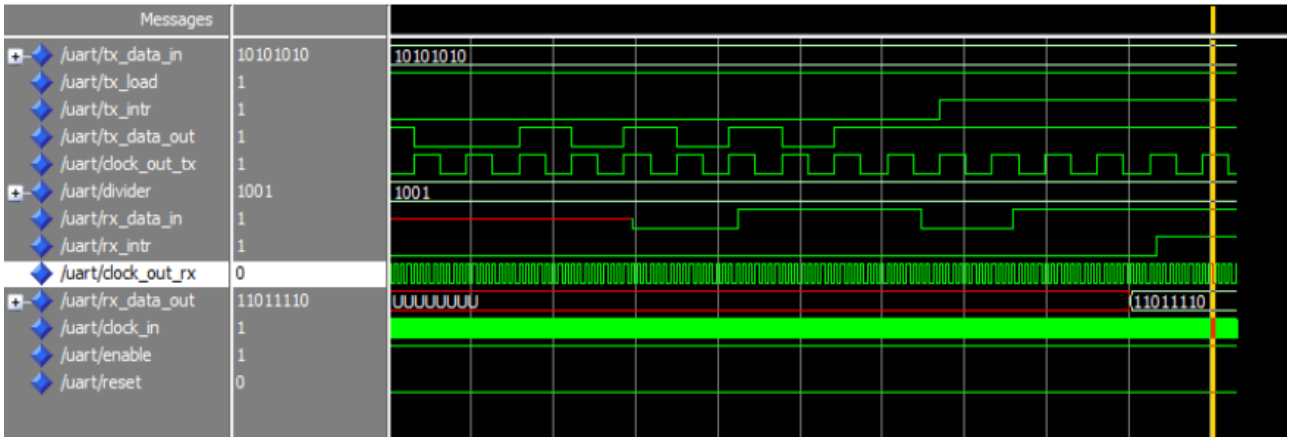

Рис.3.4. Часова діаграма роботи пристрою *UART*

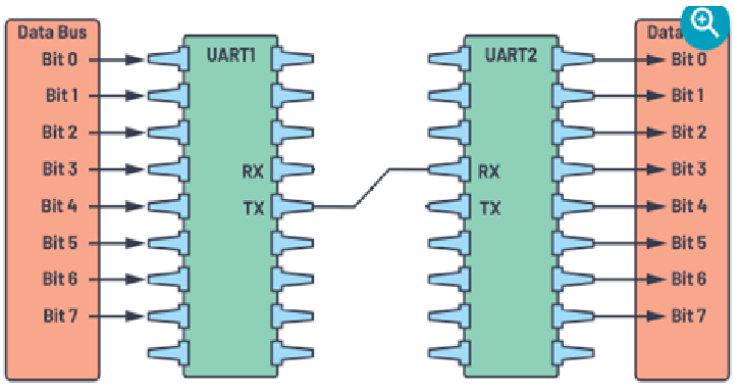

Рис.3.5. Спрощена структура апаратного бристрою *UART* без системи керування

Основні сигнальні виводи блоку *UART*:

Tx_load – сигнальний вхід. При встановленні в значення логічного нуля ініціалізує передачу даних за протоколом *UART* з 8-мибітної шини *tx_data_in*.

Reset вхід що при встановлення на ньому логічного нуля скидає всі регістри модуля до початкового стану.

Enable сигнальний вхід що при встановлення логічної одиниці перемикає всі сигнальні входи в низькоомний режим що дозволяє звертатися до них. Використовується при адресній структурі пристрою що дозволяє по одному адресу звертатись до різної периферії.

Clock_in це сигнальний вхід для трактування пристрою. На основі даного сигналу відбувається синхронізація сигналу. Допуск відхилень для тактозадаючого пристрою +/- 4%. Стандартна частота для даної бібліотеки 100*Mhz.*

Divader сигнальна шинна розмірністю 4 біта. Використовується для встановлення *Baud rate*(таблиця).

Tx_data_in вхідна шина даних розбірністю 8 біт. Фіксація даних відсутня тому необхідно підтримувати необхідні значення доки передача не завершиться.

Tx_data_out сигнальний вихід. Використовується для передачі кадру згідно протоколу передачі *UART*.

Rx_data_in сигнальний вхід. Використовується для передачі кадру згідно протоколу передачі *UART*.

Rx_dara_out вихідна внутрішня шина даних розбірністю 8 біт. Дозволяє зчитати дані які прийшли на пристрій. Має буфер тому є можливісті зчитати дані до того часу як не буде повністю отримане нове повідомлення.

Tx_intr Внутрішній сигнальний вивід який слугує для приривання що свідчить про передане повідомлення. Відслідковувати слі лише по зростаючому фронту. При повторному запуску системми передачі даних за допомогою *Tx_load* автоматично встановлює сигнальний вивід *Tx_intr* в стан логічного нуля. Таким чином робота при реалізації подальшого пристрою буде вимагати тіла process.

Clock_out_tx сигнальний вивід внутрішньої системи трактування оснований на внутрішньому частотному дільнику. Таким чином при встановлені будь якої частоти передачі за допомогою вхідної шини *Divader* моживим буде синхронізація за частотою передачі слова.

Rx_intr Внутрішній сигнальний вивід який слугує для приривання що свідчить про передане повідомлення. Відслідковувати слід не лише по зростаючому фронту а й по спадаючому. Кожен такий перехід означає переривання. Таким чином робота при реалізації подальшого пристрою буде вимагати тіла process.

Розплодження сигнальних виводів з шині *Control Buss* пристрою *UART* зображена в таблиці 3.1.

Таблиця 3.1

Нумерація та послідовність положення керуючих виводів в шині *Control Bus* 1 та *Control Bus*

0	1	2	3	4	5	6	7	8	9	10
res	*load*	*en*	*cl_in*	*B_r[3]*	*B_r[2]*	*B_r[1]*	*B_r[0]*	*Tx_i*	*Rx_i*	*Rx_Cl*

Таблиця. 3.2

Таблиця встановлення бод рейту за допомогою *Baud rate.*

Baud rate vector input	*Baud rate*
0000	600
0001	1200
0010	2400
0011	4800
0100	9600
0101	19200
0110	38400
0111	57600
1000	115200

3.2. Блок розподілення трактування

Основна задача даного блоку це ділення частоти трактування оскільки вбудований кварцовий резонатор складає 500*MHz*. Для нормального функціонування Апаратного блоку *UART* необхідна частота 100*MHz*. Але оскільки в середині пристрою є теж дільник частоти ще підходящими частотами для швидкості 9600 є 50*Mhz* при *Baud rate vector input* з значенням 0101 та 25MHz з *Baud rate vector input* зі значенням 0110.

Даний блок представляє собою здвиговий регістр з послідовно з’єднаним *T* тригером. Запис початкового імпульсу в здвиговий регістр буде відбуватися після

Reset. Класична реалізація здвигового регістра на основі *D* тригерів(рис.3.6.) на основі *VHDL* є неможливою. Оскільки реалізація самих *D* тригерів на основі простої логіка *NAND* призводить до нестабільного функціонування та неможливості повноцінної симуляції оскільки реалізація на простій логіці веде до наявності в схемі логічного циклу на основі вибору поточного стану на основі попереднього. В такому випадку оскільки вихід ще не визначено (в симуляції позначається Unitarize) відбувається помилка. Реалізація даного блоку буде задіяти алгоретмічніні можливості паралельних операторів процеса.

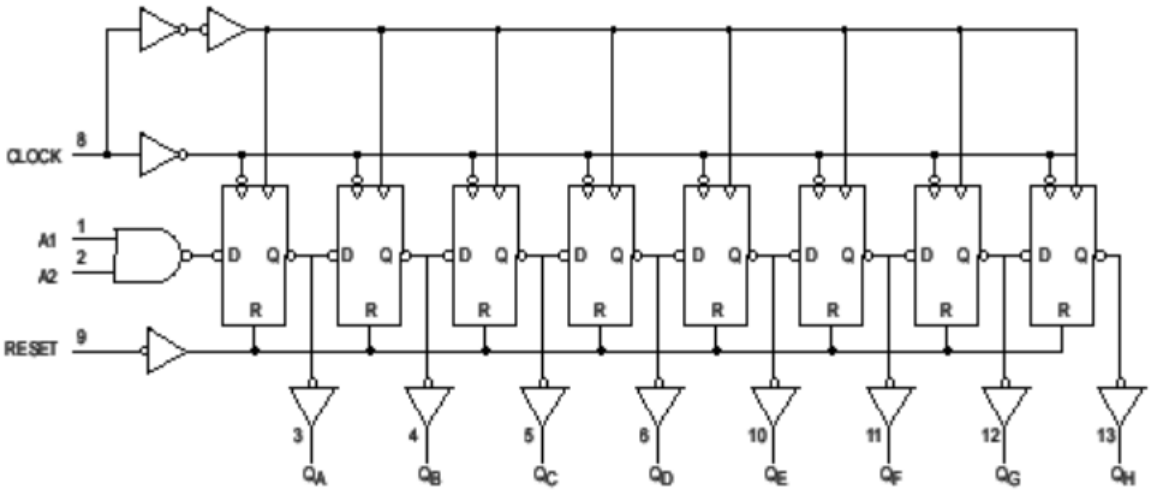

Рис.3.6. Реалізація здвигового регістру на основі *D* тригерів

Принцип роботи дільника частоти буде наступний. В основі лежить сдвиговий регістр та тригер. В здвиговому регістру по сигналу синхроімпульсу постійно здвигається логічна одиниця(рис.. *buffer*(0 *to* 3)+*f*). В якості тригера для перемкнення вихідного сигналу обраний Т тригер оскільки йому необхідний здвиговий регістр в половину менший аніж *RS* чи *JK* тригеру (для роботи в необхідному режимі їм знадобиться два сигнали на різних входах з однаковими часовими інтервалами)(рис.3.7).

Оскільки частота ділення в такій системі повинна бути парною буде використаний дільник на 10 для токування блоку *UART* і дільник на 100 для трактування блоків управління. Зниження частоти для блоку управління необхідно

оскільки трактування здвигових регістрів маперів буде від них. Часова діаграма даного блоку зображена на рис.3.8.

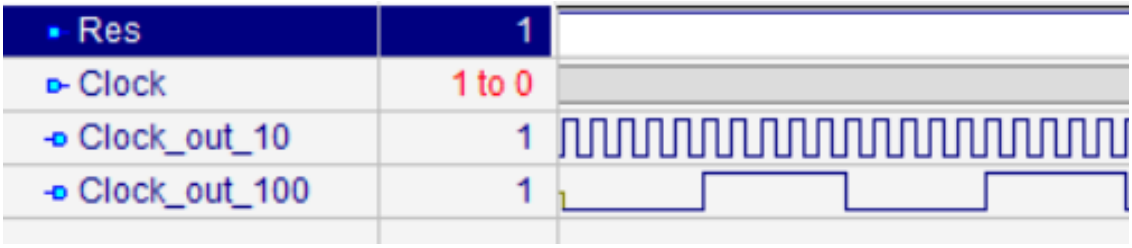

Рис.3.7. Часова діаграма роботи блоку розподілення тактових імпульсів

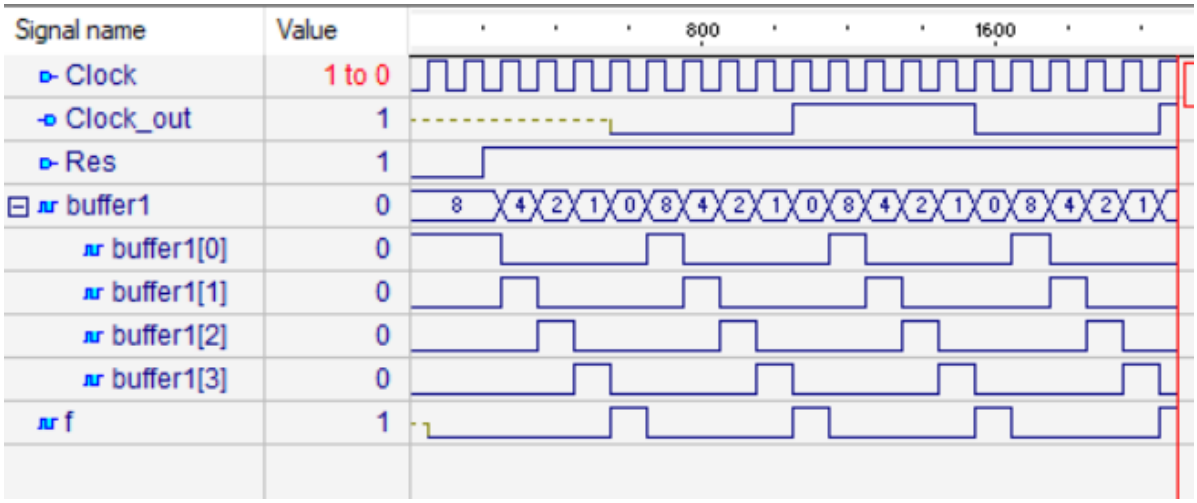

Рис.3.8. Часова діаграма роботи блоку розподілення тактування

3.3.Здвигові регістри

В даній системі використовуються блок для мапінгу пакету даних. Оскільки алгоритм *DES* потребує слово розміром 64 біти використовуються. Протокол *UART* передає лише 8 біт тому необхідно мати буфер який складає 8 слів по 8 біт у 64-ох бітне слово. Вихід з алгоритму теж складає 64 біти тому необхідний блок який виконає розкладання навпаки з 64 біт 8слів по 8біт.

В даній реалізації подібних блоків п’ять. Двоє регістри для формування 64-бітного слова, два для розбору 64-бітного слова і ще один для формування 56-бітного ключа. Хоч за структурою схожі але мають певні особливості.

Вхідні регістри мають розмірність мапінгу з 8 біт до 64біт. Оскільки сигнал який вони отримують від апаратного пристрою *UART rx_intr* d є спільним для блоків керування та вхідними здвиговими регістрами зсув повинен відбуватися як по зростаючому фронту та і по спадаючому оскільки саме таким чином передається переривання що до прийому 8-ми бітного слова. Напрям запису від старших бітів до молодших(рис.3.9).

Наступний це вихідний здвиговий регістр. Він має займається роздрібненням 64-ох бітного слова на слова по 8 біт. Управління ним відбувається з блоку керування за допомогою виводів *write, clock res*. Фіксація даних в 64-бітному буфері відбувається не автоматично оскільки між подачею даних на блок шифрування і отриманням зашифрованих даних необхідний невеликий час.

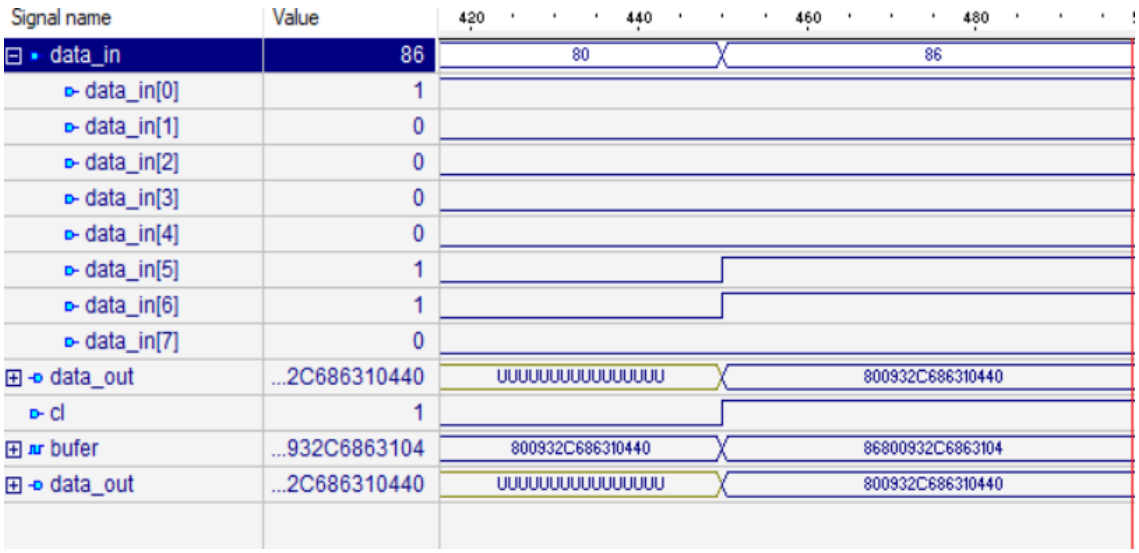

Рис.3.9. Часова діаграма вхідного здвигового регістру 8:64

Оскільки вхідний здвиговий регістр має напрям від старших до молодших щоб не змінити порядок отриманих слів вихідний здвиговий регістр повинен теж видавати дані від старших розрядів до молодших. Часова діаграмма даного регістру зображена на рис.3.10.

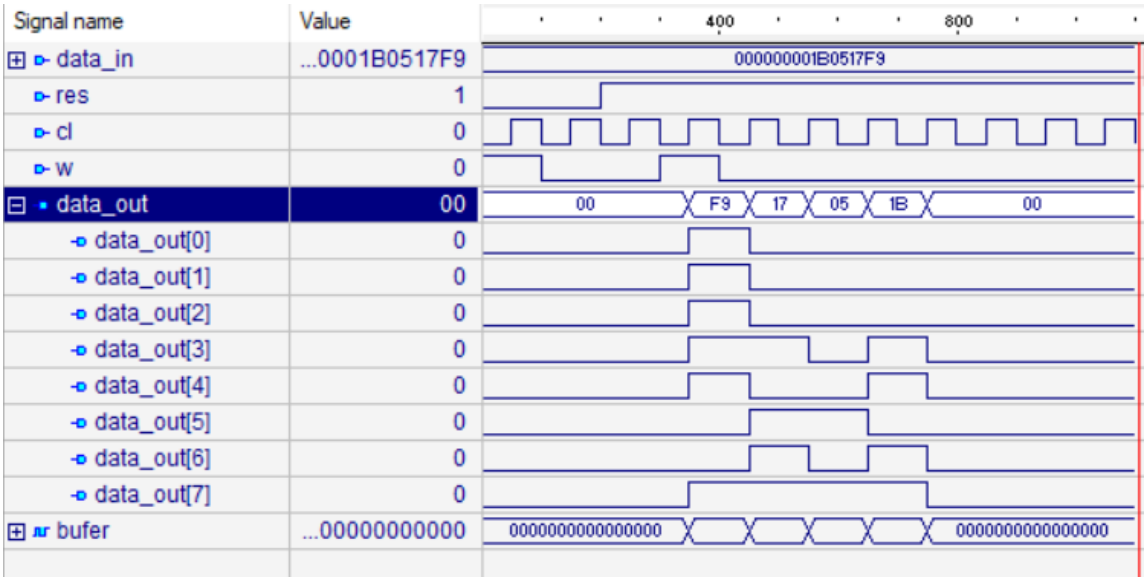

Рис.3.10. Часова діаграма роботи вихідного здвигового регістру

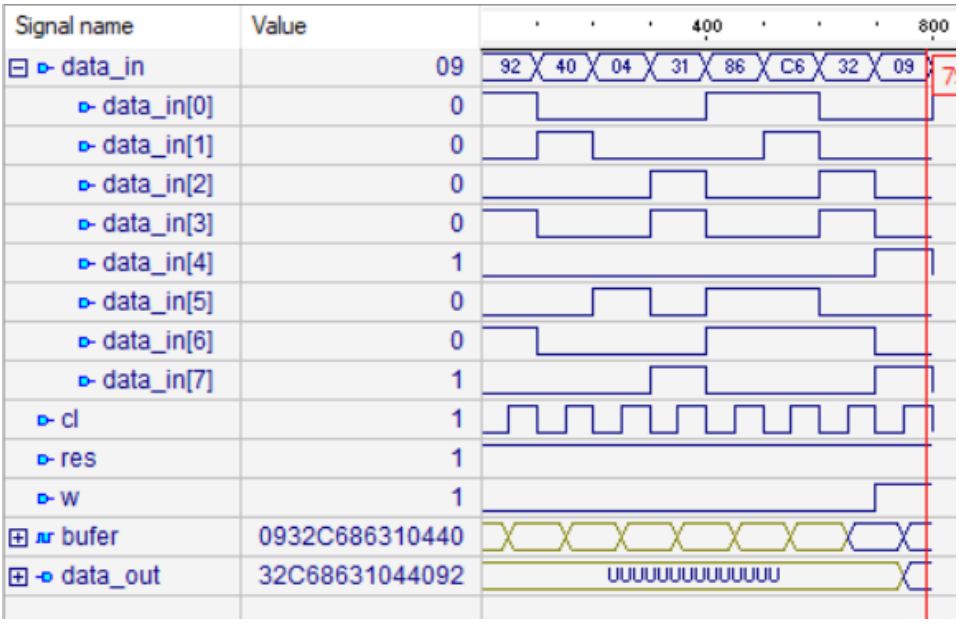

Рис.3.11. Часова діаграма роботи здвигового регістру для запису ключа

Здвиговий регістр для встановлення ключа. За своїм функціоналом схожий на вхідний здвиговий регістр блоку *UART*. Головною конструктивними відмінностями є менша кількість розрядів буфера та відсутні автоматичного видачі даних на шину після заповнення. З блокових особливостей є повна незалежність від системи управління оскільки всі функціональні входи необхідні для роботи спеціального

інструмента для встановлення ключів. Часова діаграмма даного регістру зображена на рис.3.11.

3.4. Алгоритм шифрування *DES*

Алгоритм шифрування *DES* в дані реалізації складається з двох блоків. Блок шифрування та блок дешифрування. Оскільки дані блоки є асинхронними єдине що потрібно регулювати це час з який відбувається принцес з даними. Цей час задається за дорогою вхідного та вихідного регістра кожного блоку під керуванням блоку управління. Час який буде виділено напряму залежить від встановленої швидкості передачі даних блоком *UART*.

В даній реалізації блоку шифрування є недолік а саме *S*-блок. Його можна реалізувати двома методами. Побудувати повністю на основі простих логічних елементів за допомогою МДНФ та МДКНФ. Але оскільки дані методики потребували дуже веку кількість матриць разом з реалізацію внутрішнього автомати блоку *UART* не вмістились. Тому було прийняте рішення реалізовувати на основі оператора паралельного процеса з використанням *LUT* блоків. Тому в разі необхідності можна ще розігнати данини блок та збільшити енергоефективність. Останній критерій дуже важливий для пристою шифрування оскільки він використовується в портативному пристрою. Час автономічного роботи пристрою теж важлива частина розробки. Тому для використання в промислових масштабах доведеться переводити на *ASIC*.

Для демонстрації роботи блоку шифрування та дешифрування *DES* доцільно з’єднати дані блоки наступним чином(рис.3.12). Результат роботи блоків зєднаних за схемою(рис.3.12) зображено на рис.3.13.

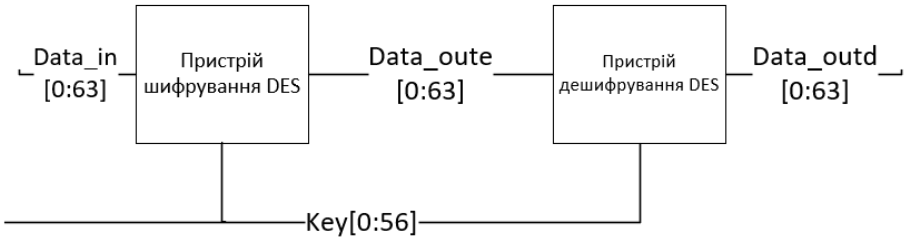

Рис.3.12. Схема включення блоків для тестування

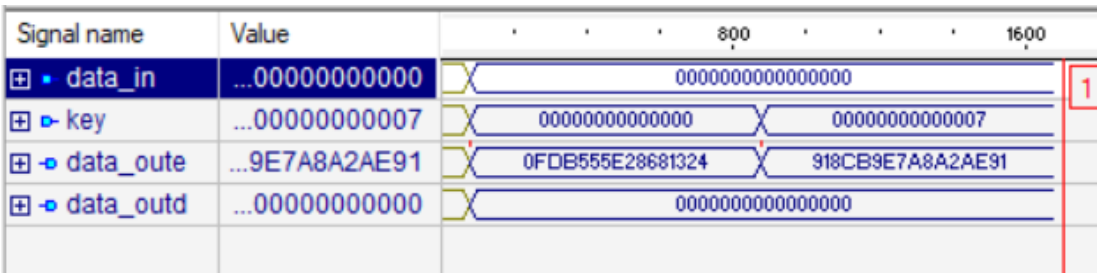

Рис.3.13. Часова діаграма роботи алгоритму шифрування та дешифрування *DES*

3.5. Блоки управління.

Основна задача блоків управління це на основі отриманих даних з блоків апаратної реалізації *UART* керувати вхідними та вихідними регістрами маперами та створювати затримку в 7 тактів швидкості передачі шини *UART* для для фіксації вихідного регістру. Також даний блок займається загальним тактуванням 4-ох здвигових регістрів які необхідні для роботи алгоритму шифрування *DES*.

В разі виникнення помилки при передачі даних між блоками апаратний пристрій *UART* має сигнальний вивід *Error*. В разі виявлення даного сигналу необхідно вивести всі останні блоки так щоб їх загальна кількість складала 32 слова по 8 біт. Таким чинов в разі виявлення помилки не зіб’ється кількісна послідовність в зашифрованих блоках щоб частина попереднього пакету не змішалася з наступним.

3.6. Блоки комутації.

Оскільки для ініціалізації периферії необхідний прями канал *UART* необхідно мати механізм який дозволяє сигналу без змін дістатися до кінцевого пристрою. Тому для цього існує блок прямого включення. Під час зажатої кнопки *reset* відбувається пряма передача даних без змін в обхід пристрою шифрування.

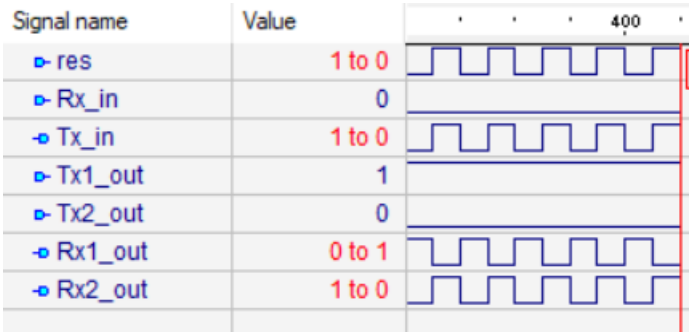

Рис.3.14. Часова діаграма комутаційного блоку

Обидва блоки є однаковими. При подачі на сигнальний вхід *res* логічної одиниці комутація з'єднує канали *Rx_in->Rx1_out* та Tx1*_out->Tx_in*. При встановленні логічного нуля використовується друга пара *Rx2_out* та *Tx2_out*. Результат роботи даного блока зображений на рис.3.14.

3.7. Розробка інструменту

Обмін даними між інструментам та комп'ютером відбувається за дорогою *USB*. В логічній структурі роботи *USB* за фізичний та канальний рівень відповідає вбудований периферійний пристрій. За функціональний рівень відповідає бібліотека *HALL*. *USB* працює в режимі *Devise*(*Slave*) як *VCM*(*Virtual Com Port*) на основі *Communication Device Class*. Така реалізація дозволяє використовувати будь який бітрейт при налаштуванні. Також це дозволить легко написати інтерфейсу частину яка буде запущена на комп'ютері оскільки більшість зовнішнішньої периферії реалізовується зо допомогою *USB-TTL* перехідників.

Реалізація переривання по прийому через *USB* в бібліотеці *HALL* відбувається в тілі функції *static int8_t CDC_Receive_FS(uint8_t* Buf, uint32_t *Len*)(рис.3.15). На основі неї будується *Callback* та перезапис до зовнішнього буферу.

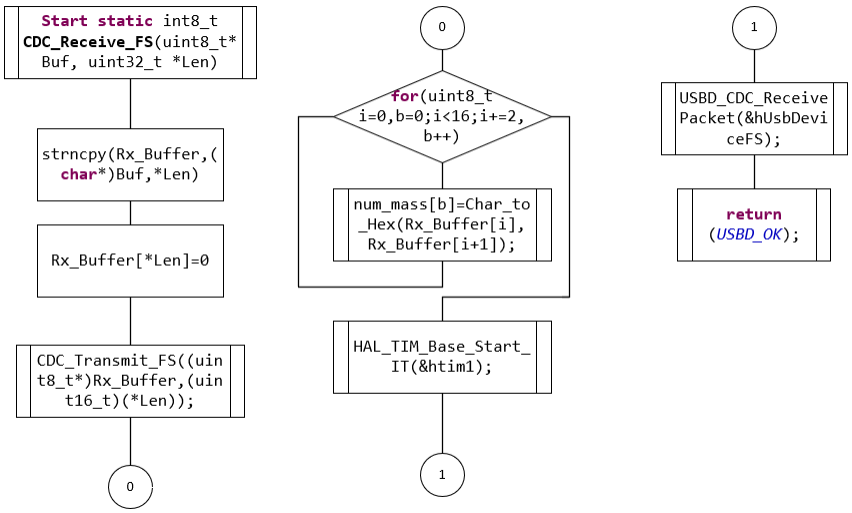

Рис.3.15. Блок схема функції *t CDC_Receive_FS*

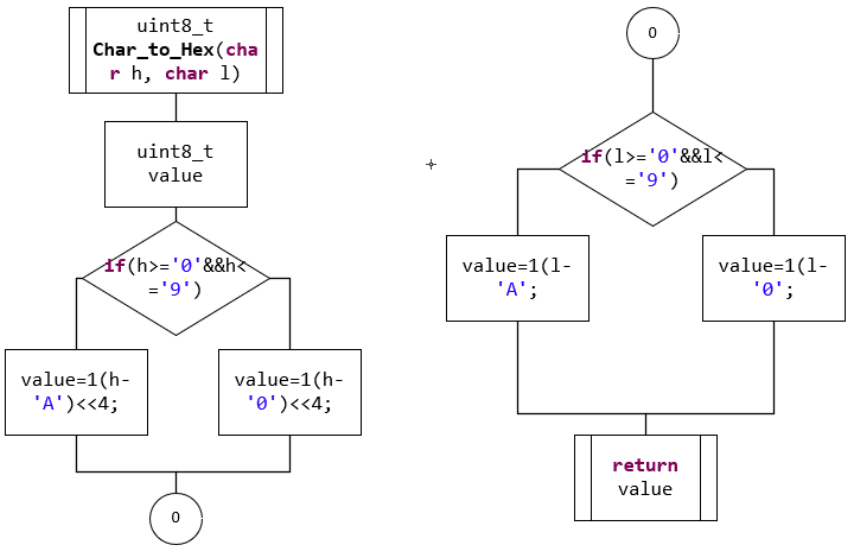

Рис.3.16. Блок схема функції *Char_to_Hex*

В свою чергу дана функція викликає *HAL_TIM_Base_Start_IT(&htim1)* що запускає таймер. Як результат запускається й тіло переривання таймеру. Буфер *Rx_Buffer* отримує значення з з прийому в форматі *Char*. Оскільки ключ

встановлюється за допомогою *HEX* необхідно перевести *Char* до *Hex*. Для даної задачі використовується функція *Char_to_Hex(char h,char l)*(рис.3.16).

Таким чином отримуємо масив *num_mass* з цілими числами які в подальшому й будуть передані. Функція *CDC_Receive_FS* викликає функцію

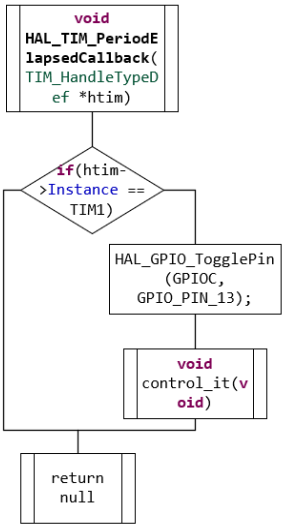

Рис.3.17. Блок схема функції *HAL_TIM_PeriodElapsedCallback*

Для регулювання швидкості передачі задіяно апаратний таймер який підраховає час інтервалів. За допомогою нього можна встановлювати швидкість передачі даних для встановлення ключа. Реалізовується дана частина програми за допомогою тіла переривання *void HAL_TIM_PeriodElapsedCallback (TIM_HandleTypeDef *htim)*(рис.3.17). В свою черга дана функція при кожному перериванні передає управління функції *void control_it(void)*(рис.3.18). Саме дана функція роздає неохідні сигнали згідно масиву на пристрій *GPIO* а саме молодші 8 біт *GPIOA*. Функція не блокує потік команд. Така реалізація функції дозволить функції робити асинхронно від основних функцій які в подальшому можна реалізувати в залежності віднеобхідностей.

Функціональний блок *OLED* використовується для зовнішньої індикація встановлення ключа що полегшує роботу в разі якихось нештатних ситуацій. Для цієї ж цілі використовується *Callback*.(рис.3.19) За допомогою нього інтерфейс може діагностувати повноцінну роботу прийому даних.

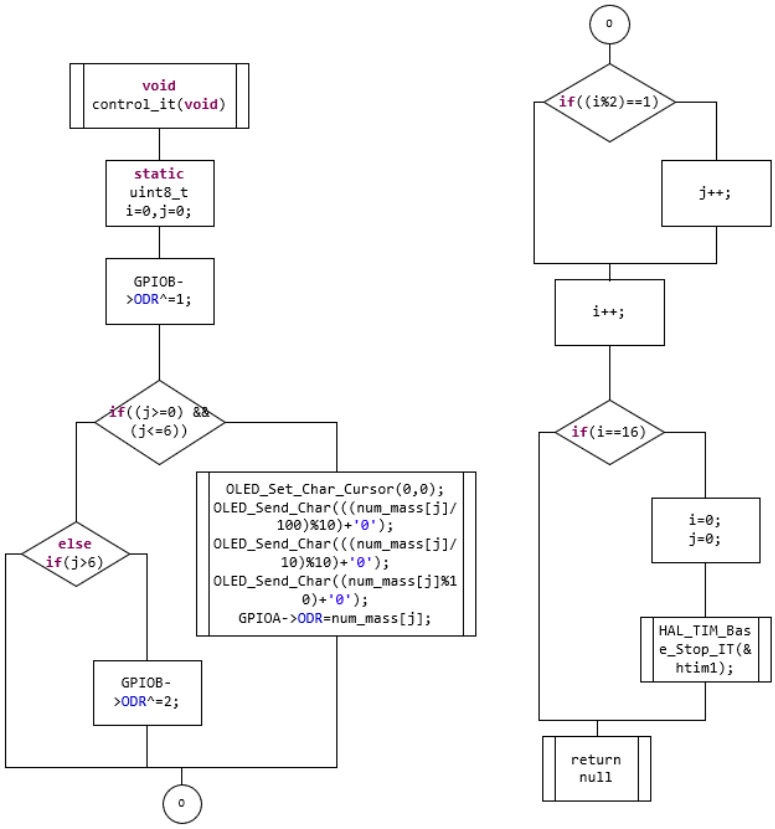

Рис.3.18. Блок схема функції *control_it*

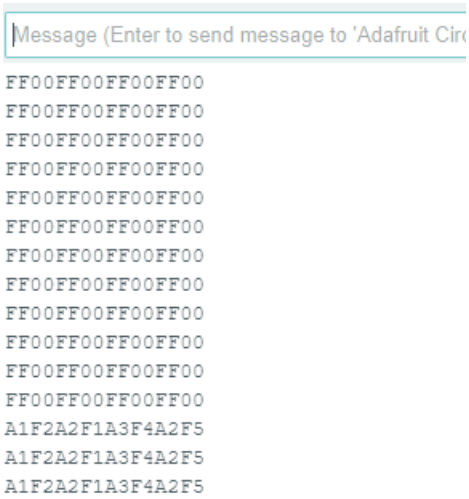

Рис.3.19. Робота *Callback*

Результат роботи пристрою встановлення ключа зображено на рис.3.20. Заміри відбувалися за рахунок 8-ти канального *logic analyzer*. Тому всі канали не вмістились оскільки їх 11.

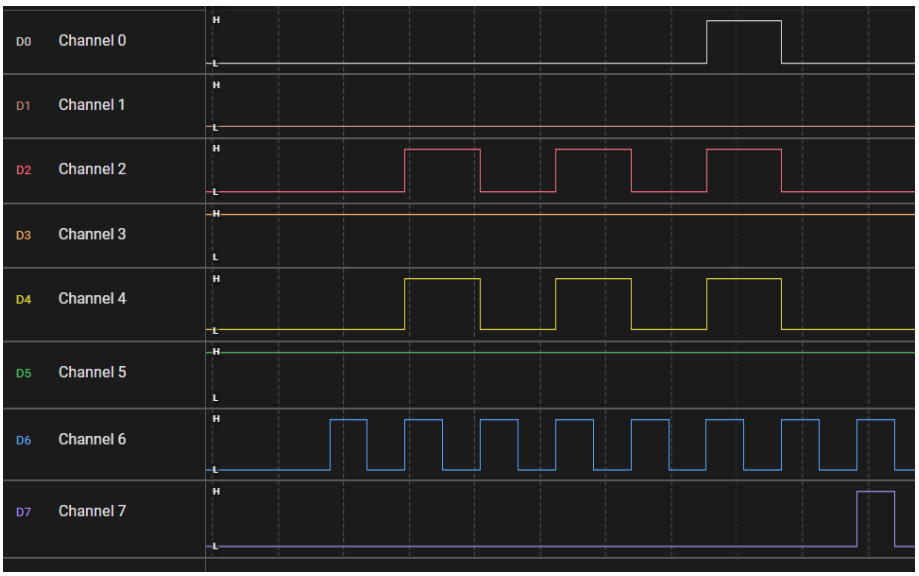

Рис.3.20. Часова діаграма роботи при встановленні ключа *A1F2A2F1A3F4A2F5*

Висновок за розділом

Реалізація пристрою шифрування на ПЛІС значно пришвидшує її швидкодію відносно реалізації всіх блоків на апаратній периферії звичайного контролера. Дана реалізація системи є не найкращою оскільки вона використовує багато блоків *LUT* що в свою чергу сильно впливає на енергоефективність та автономність системи. Ще недоліком такої реалізації оскільки частота трактування блоку *UART* була значно зменшена це менший вибір швидкостей для передачі. Тепер вона обмежена наступними бітрейтами 4800, 9600, 18200.

Висновки

За мету поставлена задача підвищення надійності каналу звязку для розповсюджених цивільних *FPV*. Для цього розроблено пристрій апаратного шифрування даних який вклинюється в шину *UART* між польотним контролером та трансивером. В основі формування пакетів даних дорнів лежить протокол *MAVLINK*. Для використання даної реалізації пристрою потокового апаратного шифрування необхідна можливість встановлення в дронів параметрів ігнорування *SysID* та сталого розміру пакетів. Дана реалізація пристрою має декілька рішень по протидії сильним електромагнітним шумам що дозволить після проходження через них залишити пристрій у робочому стані. Дане рішення має декілька проблем і найбільша з них це споживання енергії. Дану проблему можна частково вирішити якщо перевести даний пристрій в подальшому на *ASIC* оскільки там залишаться лише блоки які використовуються. Найсучасніші дорни використовують протокол *MAVLink V2*. В ньму на відмінно від старого використовується ще цифровий підпис але через його розмір пакету використання даного пристрою з ним є неможливим. Тому даний проект є компромісним рішенням для *MAVLink V*1 чи будь якого протоколу розмір пакета якого є кратний 64-ом.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Безпілотні літальні апарати радіаційної розвідки і сільськогосподарського призначення : монографія / В. Я. Канченко, Р. В. Карнаушенко, О. О. Ключников, О. П. Мариношенко, М. Л. Чепур ; НАН України, Ін-т проблем безпеки АЕС. – Чорнобиль (Київ. обл.) : Ін-т проблем безпеки АЕС, 2015. - 180 с.

2. Кутовий, О.П. Тенденції розвитку безпілотних літальних апаратів / О.П. Кутовий // Наука і озброєння – 2014. – № 4. – С. 39–47.

3. Тимочко О.І. Класифікація безпілотних літальних апаратів / О.І. Тимочко, Д.Ю. Голубничий, В.Ф. Третяк, І.В. Рубан // Системи озброєння і військова техніка. – 2007. – Вип. 1(9) – С. 61.

4. О.І. Тимочко, Д.Ю. Голубничий, В.Ф. Третяк, І.В. Рубан Харківський університет Повітряних Сил ім. Івана Кожедуба, Харків Класифікація Безпілотних літальних апаратів.,2012. – 61с

5. *Anis Koubaa, Azza Allouch, Maram Alajlan, Yasir Javed, Abdelfettah Belghith, and Mohamed Khalgu i* – Micro Air Vehicle Link (MAVLink) in a Nutshell: A Survey 2015. – 28с.

6. Андрєєв В.І., Жуков І.А Проектування електронної апаратури з використанням інтегральних схем: Методичні вказівки по курсовому та дипломному проектуванню – К.НАУ, 2001. – 48 с.

7. Bruce Schneier. Applied cryptography protocols algorithms and source code in C. – 1993. – 610с.

8. John Garney, Ken Stufflebeam,Universal, David Wooten, Matt Nieberger, John Howard – Serial Buss Specification 2000. – 650с

9. Френк Бруно. – Програмування FPGA для початківців. 2022. – 320с.

10. ПЛМ [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: *https://studfile.net/preview/5171135/page:50/*

11. Технології розширеного спектру [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: *https://studfile.net/preview/9533197/page:7/*

12. Ловыгин А. А. Современный станок с ЧПУ и *CAD/CAM*-системы / А.А. Ловыгин, А.В. Васильев, С.Ю. Кривцов. – М. : «Эльф ИПР», 2006. – 286 с.

13. Цифрові системи управління й обробки інформації. Конспект лекцій. Розділ 1: Організація й програмування систем ЧПУ. (для студентів спеціальності 151 “Автоматизація та комп‘ютерно-інтегровані технології”) / Укл. О. О. Сердюк. - Краматорськ: ДДМА, 2018. - 126 с.

14. Про схвалення Стратегії розвитку вітчизняної авіаційної промисловості на період до 2020 року : розповсюдження Кабінету Міністрів України від 27 грудня 2008 р. № 1656-р [Електронний ресурс]. – Режим доступу : http://zakon3.rada.gov.ua.

15. Андрєєв В.І., Андрєєв О.В., Комп’ютерна електроніка: Навчальний посібник. – К.: НАУ, 2010. – 320 с.

16. Завченко *Ф.А.* Радіокеровані пристрої, 2005. - 300с.

17. Цифрові системи управління й обробки інформації. Конспект лекцій. Розділ 1: Організація й програмування систем ЧПУ. (для студентів спеціальності 151 “Автоматизація та комп‘ютерно-інтегровані технології”) / Укл. О. О. Сердюк. - Краматорськ: ДДМА, 2018. - 126 с.

18. Мікропроцесорні та мікроконтролерні системи : підручник. У 2 ч. Ч. 1. Мікропроцесорні системи [Електронний ресурс] / А. О. Новацький. – Електронні текстові дані (1 файл: 16,7 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, Вид-во «Політехніка», 2020. – 361с.

19. Michael Simpson, Building the KRMx01 CNC: The Illustrated Guideto Building a High Precision CNC. Kronos Robotics, 2012 г. *ISBN* 978-1938687105

20. Новацький А. О. Комп’ютерна електроніка-3. Мікропроцесорні системи. Апаратні засоби мікропроцесорних систем : навч. посіб. / А. О. Новацький. – Київ : НТУУ «КПІ», 2015. – 470 с.

21. Комп’ютерна електроніка: Мікропроцесорні системи: Програмування мікропроцесорних систем : навч. посіб. для студ. напряму підготов. 6.050201 «Системна інженерія» кафедри Автоматики та управління у технічних системах / А.О. Новацький. – Київ : НТУУ «КПІ», 2014. – 307 с

22. Проектування мікропроцесорних систем: Проектування мікропроцесорних систем на базі мікроконтролерів сімейства MCS-51: Периферійні модулі мікроконтролерів сімейства MCS-51 : навч. посіб. для студ. напряму підготовки 6.050201 «Системна інженерія» кафедри Автоматики та управління у технічних системах / А. О. Новацький. – Київ : НТУУ «КПІ», 2016. – 399 с.

23. Fedasyuk D. Method of Developing the Structural-Automaton Models of Fault-Tolerant Systems [Text] / Dmytro Fedasyk, Serhiy Volochiy // Proceedings 14th International Conference “The Experience of Designing and Application of CAD Systems in Microelectronics (CADSM)”, 21–25 February 2017, Polyana, Ukraine. 2004. – 220 с.