УДК 621.317.18:621.317.619

ОБРАБОТКА ТЕЛЕМЕТРИЧЕСКОЙ ИНФОРМАЦИИ НА БАЗЕ AVR СИСТЕМ

Жигарев М.Ю., магистрант; Паслён В.В., зав. каф., к.т.н., доц.
(ГОУ ВПО «Донецкий национальный технический университет», г. Донецк ДНР)

Введение.

Профессиональные квадрокоптеры благодаря своим характеристикам обладают широким спектром применения. Однако недостатком профессиональных беспилотных летательных аппаратов (БПЛА) такого типа является высокая стоимость. Проектирование квадрокоптеров с помощью составных частей существенно снижает стоимость летательного аппарата, а также позволяет подобрать необходимые технические характеристики и функционал [1].

Рассмотрим систему стабилизации, как одну из систем квадрокоптера. Система стабилизации производит вычисление углов отклонения квадрокоптера. Компенсацию углов отклонения осуществляется путем изменения длительности импульсов управляющих ШИМ-сигналов управляющих электронными регуляторами хода. Электронные регуляторы хода управляют частотой вращения бесколлекторных двигателей [1].

При проектировании и отладке системы стабилизации квадрокоптера, необходимо производить измерение, обработку и передачу телеметрической информации в реальном времени. Проблема передачи телеметрической информации состоит в том, что данные имеют разное представление (положительные и отрицательные значения, значения с фиксированной точкой, плавающей точкой, массивы данных). Существует несколько подходов к передаче данных имеющих разное представление. Первый способ заключается в том, что для каждого представления данных используются разные алгоритмы их обработки и передачи. Второй способ предусматривает разработки алгоритма, способного обрабатывать и передавать данные разных типов.

Целью работы является разработка алгоритма для передачи данных, имеющих разное представление и написание программного модуля, которая реализует разработанный алгоритм на языке программирования Си.

Основная часть.

Разрабатываемый программный модуль будет оперировать объектами. Каждый объект представляет из себя структуру, которая содержит три поля: идентификатор, функцию и тип данных. Идентификатор – уникальное числовое значение, которое характеризует передаваемые данные. Поле «функция» содержит функцию обратного вызова (callback) – функцию, которая будет вызвана, перед передачей данных и которая позволяет получить данные, которые необходимо передать. Тип данных содержит информацию о представлении данных, которая возвращает функция обратного вызова. На рисунке 1 показано объявление структуры, которая соответствует объекту программного модуля, и доступные типы данных.

Определим требования к разрабатываемому программному модулю. Разрабатываемый программный модуль должен:

1. передавать данные разных типов, инкапсулируя процессы преобразования данных к единому виду перед передачей и восстановление данных после приема;
2. быть независимым от протокола передачи данных;
3. быть кроссплатформенным.

Рассмотрим подробнее процесс передачи данных с разным представлением. В языке программирования Си существует особый тип указателей – указатели типа $void$, которые также называют пустыми указателями. Эти указатели используются в том случае, когда тип переменной не известен. При инициализации значения возвращаемые функциями обратного вызова приводятся к указателю типа $void$. Такой подход позволяет избежать конфликта

типов. При передаче данных также передается информация о типе данных. Это позволяет восстановить принятые данные [2].

```
// Pointer to a callback function
typedef void* (*getter)(void);

// Telemetry items structure
typedef struct {
    // Identifier of data
    u8 id;

    // Callback functions that used to get data
    getter func;

    // Variable type which return callback functions
    u8 type;
} telemetry_item;

// Types of a data
#define ONE_BYTE      1
#define TWO_BYTE      2
#define FOUR_BYTE     4
#define ARRAY         5
#define FLOAT         6
```

Рисунок 1 – Объявление структуры-объекта программного модуля

Для того, чтобы обеспечить независимость от протокола передачи данных, в программном модуле определены собственные функции приема и передачи информации. Для реализации данного функционала используется препроцессор языка программирования Си. Для корректной работы модуля необходимо лишь указать две базовые функции, которые будут использоваться для передачи и приема информации. Базовые функции осуществляют передачу и прием одного байта информации. Таким образом программный модуль «не знает» каким образом происходит передача и прием данных. Функции передачи/приема нескольких байт информации, чисел с плавающей точкой и числовых массивов программный модуль реализует самостоятельно. На рисунке 2 показана реализация независимости от протокола передачи данных и кроссплатформенности.

```
#if PLATFORM == ATMEGA
    #define Telemetry_transmitData(x)       (USART_Transmit(x))
    #define Telemetry_receiveData()         (USART_Receive())
// For use on orangePi (rapberryPi)
#elif PLATFORM == ORANGE
    // UART settings
    #define BAUD                            9600
    #define INTERFACE                       "/dev/ttyS0"
    // File descriptor
    static int fd;
    // Initialisation serial interface for orangePi
    #define Telemetry_init()                (fd = serialOpen(INTERFACE, BAUD))
    #define Telemetry_transmitData(data)    (serialPutchar(fd, data))
    #define Telemetry_receiveData()         (serialGetchar(fd))
#endif
```

Рисунок 2 – Независимость от протокола передачи данных и кроссплатформенность

Кроссплатформенность программного модуля также достигается за счет препроцессора языка программирования Си. Перед компиляцией есть возможность указать используемую платформу, для которой будет сконфигурирован программный модуль. Также есть возможность добавления собственных платформ.

Рассмотрим пример использования программного модуля. Для использования модуля необходимо два устройства: передатчик и приемник. В качестве передатчика выступает восьмибитный контроллер семейства AVR. В качестве приемника, в данном случае, выступает одноплатный компьютер OrangePI Win Plus.

Для использования модуля в исходном коде передатчика, необходимо сначала проинициализировать программный модуль. Инициализация модуля происходит с помощью функции $Telemetry\_getItems$, которая принимает в качестве параметров количество объектов, массив идентификаторов, массив функций обратного вызова и массив, который содержит типы данных. Функция $Telemetry\_getItems$ возвращает массив объектов. На рисунке 3 показан процесс инициализации программного модуля.

```
getter functions[] = {(getter)BMP180_getTemp, (getter)get_array};
u8 ids[] = {TEMP, HEIGHT};
u8 types[] = {FOUR_BYTE, ARRAY};

telemetry_item* items = Telemetry_getItems(2, ids, functions, types);
```

Рисунок 3 – Инициализация программного модуля

После инициализации необходимо использовать функцию $Telemetry\_streamData$, которая в качестве аргументов принимает массив объектов-структур и количество объектов. Данная функция ожидает получение запроса, и как, передает необходимые данные как только запрос был получен. В качестве запроса выступает идентификатор объекта-структуры. На рисунке 4 показан исходный код функции $Telemetry\_streamData$.

```
void Telemetry_streamData(telemetry_item* items, u8 count) {
    // // Receiving data identifier
    u8 id = Telemetry_receiveData();

    for (u8 i = 0; i < count; i++) {
        // Find telemetry item with right identifier
        if (items[i].id == id) {
            // Getting data to transmit
            void* ptr = items[i].func();

            // Transmitting the data
            Telemetry_dataTransmit(items[i].type, &data);
            break;
        }
    }
}
```

Рисунок 4 – Исходный код функции $Telemetry\_streamData$

Функция $Telemetry\_dataTransmit$, в качестве аргументов принимает тип данных и данные, имеющие неопределенный тип ($void$), осуществляет преобразование данных к нужному типу и передает их приемнику. Исходный код функции $Telemetry\_dataTransmit$ представлен на рисунке 5.

```
void Telemetry_dataTransmit(u8 type, void* data) {
    // Transmitting "start" identifier
    _Telemetry_transmitRawData(START, TWO_BYTE);

    // Transmitting data type identifier
    Telemetry_transmitData(type);

    // Check data type and use a special function if a type is "float" or "array"
    switch (type) {
        case FLOAT:
            // Transmitting data that having "float" type and return from the function
            float* d = (float *)data;
            Telemetry_transmitFloat(*d);
            return;

        case ARRAY:
            // Transmitting data that having "array" type and return from the function
            s32* arr = (s32 *)data;
            Telemetry_transmitArray(arr);
            return;
    }

    // Transmitting data
    Telemetry_nthBytesTransmit(*data);
}
```

Рисунок 5 – Исходный код функции Telemetry_dataTransmit

Для получения данных в исходном коде приемника следует использовать функцию *Telemetry_getData*, которая в качестве аргументов принимает идентификатор объекта-структуры, а возвращает полученные данные. Исходный код функции *Telemetry_getData* показан на рисунке 6.

```
s32 Telemetry_getData(u8 id) {
    // Transmitting data identifier
    Telemetry_transmitData(id);

    // If the identifier "start" is not received - do nothing
    if (_Telemetry_receiveRawData(TWO_BYTE) != START) return NULL;

    // Receiving data type identifier
    u8 type = Telemetry_receiveData();

    switch (type) {
        case ARRAY:
            s32* data = Telemetry_receiveArray();
            break;

        case FLOAT:
            float data = Telemetry_receiveFloat();
            break;

        default:
            s32 data = Telemetry_nthBytesReceive(type);
    }

    // Receiving data
    return data;
}
```

Рисунок 6 – Исходный код функции Telemetry_getData

Вывод. Был разработан алгоритм для передачи данных, имеющих разное представление и написан программный модуль, которая реализует разработанный алгоритм на языке программирования Си.

Перечень ссылок

1. Жигарев, М. Ю. Управление бесколлекторными системами на базе AVR систем / М. Ю. Жигарев / Инновационные перспективы Донбасса. – 2016.

2. Дейтел, П. Как программировать на C : учебник для вузов / Пол Дейтел, Харви Дейтел. – Москва : Бином, 2017. – 1008 с.