Ссылка на статью:
// Ученые записки УлГУ. Сер. Математика и информационные технологии. УлГУ. Электрон. журн. 2020, № 2, с. 55-63.

Поступила: 02.12.2020
Окончательный вариант: 02.12.2020

УДК 004.451.7:004.7

Декодирование осциллограммы сигнала в линии с интерфейсом ARINC 429

Рубцов М.А.

mihanr1995@mail.ru
УлГУ, Ульяновск, Россия

В статье рассматривается метод расшифровки сигнала в линии с интерфейсом ARINC 429 с помощью языка программирования Python и последующей обработки полученной информации для наглядного представления и эффективного анализа.

***Ключевые слова**: ARINC-429, авионика, декодирование сигнала, интерфейс информационного взаимодействия, РТМ 1495-75, язык программирования Python.*

Введение

Для выявления причин неправильной работы различных блоков бортового оборудования необходимо проанализировать согласованность и исправность их взаимодействия согласно интерфейсам, принятым в авионике. Самым распространённым интерфейсом информационного обмена в авионике являются интерфейс, описываемый стандартом ARINC 429 (Российский аналог РТМ 1495-75 изм.3(кодовая линия связи (КЛС)). Несмотря на наличия тестеров для данного интерфейса, они не позволяют наиболее эффективно проанализировать информацию с линий связи и быстро выявить программные и аппаратные ошибки каких-либо блоков, участвующих в информационном обмене. Для более эффективной и быстрой обработки данных КЛС можно проанализировать осциллограмму, снятую с линии связи, с помощью любого языка программирования. В данной статье будем использовать язык программирования Python.

1. Передача информации по интерфейсу ARINC 429

Рассмотрим принципы передачи информации согласно интерфейсу ARINC 429 (РТМ 1495-75). Интерфейс строится на основе одной двухпроводной линии передачи. Резерви-

рование второй линией не предусмотрено. Но в авионике надёжность играет главную роль, поэтому для ответственных бортовых систем или блоков линии резервируются. Выделяют два типа технических средств, подключаемых к шине: передатчики и приемники, причем их количество должно быть не более 1 и 20 штук соответственно, и это определяет одностороннюю передачу данных в сети (симплекс). Для организации двухсторонней передачи данных стандарт допускает присутствие двух и более разделенных шин. Физически соединения согласно ARINC 429 представляет собой витую пару с экраном. Данные по шине передаются со скоростями 12.5, 50, 100 кбит/с [3, 5]. За счёт низких скоростей и простоты аппаратной реализации данный интерфейс устойчив к помехам. Главной причиной ошибок при передаче данных по данному интерфейсу являются программные ошибки в ПО блоков.

Рассмотрим структуру передачи информации по стандарту Arinc 429. Согласно стандарту ни одно устройство не является контроллером канала. Данные передаются в основном асинхронно, т.е. передатчик передаёт параметры все время, а приёмник считывает данные тогда, когда необходимо, в соответствии с программой. Кроме того, существует передача информации «по запросу» и «по готовности», в этом случае выделяется ещё одна линия, для того чтобы система могла проинформировать о том, что она готова передавать данные, либо запросить данные от другой системы. [5] Данные кодируются согласно RZ-коду (рис.1) [1, 5]. Биты передаются в виде прямоугольных импульсов с периодом T и скважностью 2. Импульсу с напряжением 10±1В соответствует логическое значение бита «1», а с напряжением -10±1 В соответствует значение «0» [3,5]. Отсутствие напряжения (пауза между словами и битами) составляет 0±0.5 В. Период выдачи бита в пределах одного информационного слова зависит от скорости передачи. Для скоростей 12.5, 50 и 100 кбит/сек период T выдачи бит сигнала равен соответственно 80, 20 и 10 мкс. соответственно [4, 5]. Логическое состояние каждого бита определяется разностью сигнала первого выхода (а – неинвертирующего) и второго (б - инвертирующего). Инвертирующий и неинвертирующий выходы формируют сигналы Ua и Uб соответственно, а напряжение Uаб равняется разности сигналов Ua и Uб (рис.1) [1, 4]. Такой способ передачи сигнала позволяет погасить помехи, так как на каждом проводе будут синхронные помехи и при вычитании сигналов они обнуляются. Передаваемая информация согласно интерфейсу подразделяется на слова по 32 бита и ориентирована на передачу двоичного кода (ДК), двоично-десятичного кода (ДДК), трёхформатного кода, разовых команд (РК). Иногда информация может передаваться и в виде файла по согласованию разработчиков систем [5]. Для того, чтобы система различала слова, между словами должна быть пауза, равная 4T, где T – период выдачи бит [5]. Каждое слово имеет метку (адрес) по которому получатель идентифицирует вид информации. Адрес расположен с 1 по 8 бит, поэтому всего разновидностей слов, которые может принимать система по одному каналу, может быть 256 [5]. Стандарт рекомендует разработчикам систем назначать для определённых типовых видов информации (абсолютная высота, угол атаки, воздушная скорость и т.п.) одинаковые адреса. 9-й и 10-й бит идентифицируют источник, в том случае, если идентичных систем несколько, то

система-приёмник понимает, от какого именно источника пришла информация. Но стандарт допускает по согласованию разработчиков 9-10 бит добавлять к адресу для увеличения количества видов передаваемой информации [5]. На 30-31 бит обычно назначается матрица состояния, по которой система-приёмник определяет состояние системы (исправность системы, отсутствие информации, либо состояние тест-контроля). Бит 32 является битом чётности, который имеет такое значение, чтобы в сумме было нечётное число бит в слове. Бит чётности и матрица состояния позволяют оценить достоверность данных и исправность источника. Остальные биты кодируют передаваемые параметры [4, 5]. Стандарт интерфейса рекомендует использовать определённую структуру слова в пределах бит данных, в зависимости от характера данных, но в общем случае структура определяется в протоколе взаимодействия систем, особенно если тип данных не описан в стандарте.

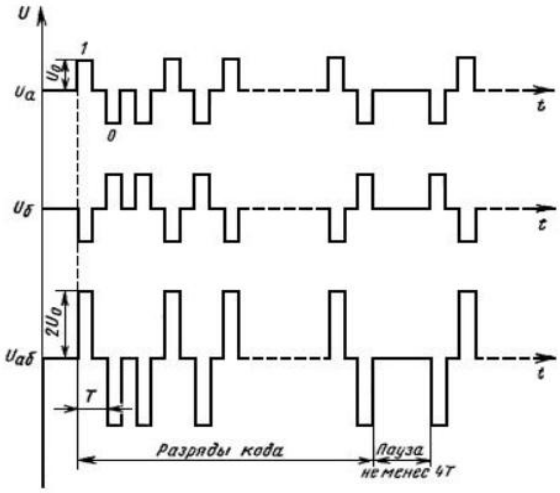

Рис. 1. График осциллограммы сигнала в линиях с интерфейсом ARINC-429

2. Способы анализа сигнала ARINC 429

Предпочтительным способом анализа сигнала является анализ во временной области с помощью цифрового осциллографа, и записи осциллограммы в файл для дальнейшего его анализа [4]. Непосредственно на осциллографе наблюдать сигнал и проверять правильность битовых последовательностей не представляется возможным, из-за неудобства и непериодичности RZ-кода. Для анализа кода можно записать значения напряжении на линии в каждый момент времени (период дискретизация сигнала должен быть равен 1 мкс - 2мкс для лучшего представления сигнала и распознавания фронтов импульсов). При измерении сигнала мы получаем все биты, передаваемые за определённый промежуток времени, который нужен. Анализ осциллограммы в виде файла (содержащего значения напряжения сигнала в каждый момент времени) можно легко автоматизировать, используя

любой язык программирования. Оценка формы, фронтов и частоты импульсов позволит оценить исправность аппаратной части передающей системы. Анализ битовых данных в словах даст представление о правильности функционирования программной части и её согласованность с протоколом взаимодействия.

Так как важной чертой рассматриваемых интерфейсов является чёткая установка интервалов напряжений и типов кодирования, то анализ сигналов не представляется сложным и возможна автоматизация анализа сигнала. С помощью программы можно определить начало каждого слова, распознать паузу длительностью 4T, распознать каждый бит в слове, в зависимости от полярности импульса, а в дальнейшем расшифровать и оценить правильность передаваемых данных и их согласованности с протоколом взаимодействия и стандартом. Благодаря автоматизированному анализу можно отслеживать ошибки по определённым критериям или в строго определённых словах, которые необходимы.

3. Создание алгоритма для анализа данных сигнала ARINC 429 на языке программирования Python

Для анализа данных нам понадобятся два бинарных файла с массивом данных отсчётов времени (период дискретизации в нашем случае равен 1 мкс) и с массивом значений напряжений для каждого отсчета. В совокупности эти два файла представляют зависимость функции напряжения от времени в табличном представлении. Данные файлы имеют расширение «dbl» (массив значений 8 байтных чисел с плавающей точкой) и получены с использованием специального цифрового осциллографа. Для написания алгоритма и программы используем официальный интерпретатор языка Python версии «3.8.3» и среду разработки «Pycharm Community».

```
import struct

def ReadDouble(FileName):
    fread = open(FileName, 'rb')
    BinaryData = fread.read()
    fread.close()
    Precision = 8
    NumData = int(len(BinaryData) / Precision)
    Data = []
    for i in range(NumData):
        Data.append(float(struct.unpack('d', BinaryData[i *
        Precision:(Precision * i) +
        Precision])[0]))
    return Data
```

Рис. 2. Функция считывания данных типа double из файла

```
Data = ReadDouble("C:/Напряжение.dbl")
Time = ReadDouble("C:/Время.dbl")
```

Рис. 3. Считывание данных

Создадим функцию «ReadDouble», в которой реализуем считывания данных из файлов с расширением “dbl”. Данная функция будет принимать в качестве аргумента строку «FileName», с помощью которой мы будем передавать полное имя файла «dbl». В качестве результата данная функция будет возвращать массив (список) чисел типа “float”. Реализация функции «ReadDouble» приведена на рис.2. В данной функции мы используем стандартный модуль «struct» и его функцию “unpack” для перевода бинарных данных, полученных после функции «open», в список с типом “float”. После реализации функции «ReadDouble» используем её (рис.3). Для графического вывода осциллограммы на основе считанных данных воспользуемся модулем «matplotlib»[2]. Код для вывода осциллограммы представлен на рис.4. После запуска программы в окне графического вывода получим график U(t) (рис.5).

```
fig, ax = plt.subplots()
ax.plot(Time,Data, color="blue", label="U(t)")
ax.set_xlabel("Время, с")
ax.set_ylabel("Напряжение, В")
ax.legend()
ax.set_xlim(0.0005, 0.0013)
plt.show()
```

Рис. 4. Код отрисовки участка осциллограммы

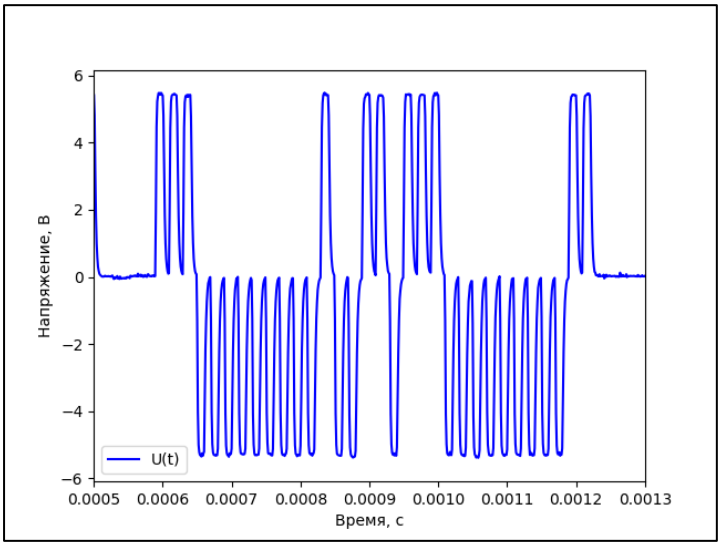

Рис. 5. График U(t) в пределах одного слова данных

Теперь напишем алгоритм для чтения бит с осциллограммы, для этого нам нужно распознать место начала слова и начала каждого импульса, после чего мы на основании его полярности определяем логическое состояние бита. Создадим переменные, необходимые для дальнейшего написания алгоритма декодирования (рис.6).

```
CountWords = 80000
word = 0
Words = [[0] * 32 for Words in range(CountWords)]
WordAdress = [0 for i in range(CountWords)]
timeStartWords = [0 for c in range(CountWords)]
changeFlag = True
deltaT = Time[1]-Time[0]
bitrate = 50000
WaitWord = 4/bitrate
CountWaitWord = WaitWord/deltaT
col = 0
i = 0
```

Рис. 6. Переменные для записи слов

Укажем назначения данных переменных: «CountWords» - максимальное количество считанных слов; “word” – порядковый номер текущего слова; «Words» - двумерный массив для хранения значений бит каждого слова по ARINC-429; “WordAdress” – массив для хранения адресов слов; “timeStartWords” – массив для хранения времени начала слова в соответствии с осциллограммой, “changeFlag”- флаг для обнаружения изменения информации в слове с определённым адресом; «deltaT» - шаг дискретизации по времени; “bitrate” - скорость передачи данных в линии ARINC-429 (бит/с); «WaitWord» - теоретическая длительность паузы между словами (равна 4T, где T период выдачи бита); «CountWaitWord» - количество отсчётов времени, приходящуюся на паузу между словами; «col» - количество подряд идущих отсчётов с нулевым напряжением; «i» – общий счётчик для перебора всех значений осциллограммы.

Для начала нам нужно найти начало слова. Для этого ищем паузу 4T в осциллограмме, которая находится перед словом. Пауза 4T найдена если число подряд идущих значений с нулевым напряжением равно: CountWaitWord * 0.7. Алгоритм поиска паузы показан на рис. 7.

```
while i < len(Data)-1:
    while i < len(Data)-1 and col <int(CountWaitWord * 0.7):
        col = 0
        while i < len(Data)-1 and Data[i] < 2 and Data[i] > -2:
            col += 1
            i += 1
        i += 1
    col = 0
```

Рис. 7. Цикл поиска начала слова

После того как нашли начало слова, записываем значение i-го отсчёта времени, на котором мы остановились, в массив “timeStartWords” (рис.8).

```
if i < len(Data)-1:
    timeStartWords[word] = Time[i] * 1000
```

Рис. 8. Запись времени начала слова в массив

Затем можем считать отдельные биты, для этого просматриваем подряд идущие отсчёты с положительным и отрицательным напряжением. После считывания бит записываем в массив «WordAdress» адрес для слова «word» на основании 8 первых бит. Итоговый алгоритм распознавания слов приведён на рис.9.

```
while i < len(Data)-1:
    while i < len(Data)-1 and col <int(CountWaitWord * 0.7):
        col = 0
        while i < len(Data)-1 and Data[i] < 2 and Data[i] > -2:
            col += 1
            i += 1
        i += 1
    col = 0
    if i < len(Data)-1:
        timeStartWords[word] = Time[i] * 1000

    for j in range(32):
        if i < len(Data)-1 and Data[i] > 2 and Data[i] < 6:
            Words[word][j] = 1
        if i < len(Data)-1 and Data[i] < -2 and Data[i] > -6:
            Words[word][j] = 0
        while i < (len(Data)-1) and ((Data[i] < 6 and Data[i] > 2) or
                                     (Data[i] > -6 and Data[i] < -2)):
            i += 1
        while i < len(Data)-1 and j<31 and  (Data[i] < 2 and Data[i] > -2):
            i += 1

    WordAdress[word] = ((Words[word][0]*2+Words[word][1]*1)*100+
                        (Words[word][2]*4+Words[word][3]*2+Words[word][4])*10+
                        (Words[word][5] * 4 + Words[word][6] * 2 + Words[word][7]))
    word += 1
```

Рис. 9. Алгоритм декодирования слов осциллограммы

После получения побитовых значений слов можем вывести их в консоль (или в файл) для дальнейшего анализа, либо выполнить проверку на соответствие протоколу. Для примера выведем список слов со значением бит, адресом, временем начала слова, матрицей состояния. Алгоритм для вывода в консоль представлен на рис. 10. В результате работы программы получим в консоли наглядный список слов (рис. 11).

Аналогичным образом можем интерпретировать любую информацию по битам слова и проанализировать её на соответствие требованиям. В зависимости от протокола взаимодействия в слове может содержаться информация различного характера. Например, значения параметров от датчиков (воздушная скорость), или логические параметры состояния.

```
i=0
while i<word-1:

    Value = 0;

    if changeFlag == True:
        print("Word[",i,"] =",sep = "",end = " ")
        for j in range(32):
            print(Words[i][j],end="")
        print(" ---АДРЕС = ", WordAdress[i],end=" ")
        print(" ---ВРЕМЯ НАЧАЛА СЛОВА = ", timeStartWords[i], "мс")

        if Words[i][29] == 0 and Words[i][30] == 0:
            print("Матрица состояния слова = НОРМ")
        if Words[i][29] == 0 and Words[i][30] == 1:
            print("Матрица состояния слова = ДАННЫЕ НЕ ГОТОВЫ")
        if Words[i][29] == 1 and Words[i][30] == 0:
            print("Матрица состояния слова = КОНТРОЛЬ")
        if Words[i][29] == 1 and Words[i][30] == 1:
            print("Матрица состояния слова = ОТКАЗ")
        print('')
    changeFlag = 0
    for j in range(32):
        if Words[i][j] != Words[i+1][j]:
            changeFlag = True
    i+=1
```

Рис. 10. Алгоритм вывода слов в консоль

```
Word[0] = 11100000000010011011100000000011 ---АДРЕС =  340  ---ВРЕМЯ НАЧАЛА СЛОВА =  0.5920000000000079 мс
Матрица состояния слова = ДАННЫЕ НЕ ГОТОВЫ

Word[2570] = 11100000000000010001110010000100 ---АДРЕС =  340  ---ВРЕМЯ НАЧАЛА СЛОВА =  1850.9889999379102 мс
Матрица состояния слова = КОНТРОЛЬ

Word[2605] = 11100000000110010001110010000100 ---АДРЕС =  340  ---ВРЕМЯ НАЧАЛА СЛОВА =  1876.188999935837 мс
Матрица состояния слова = КОНТРОЛЬ

Word[2629] = 11100000000000010001110010000100 ---АДРЕС =  340  ---ВРЕМЯ НАЧАЛА СЛОВА =  1893.4689999344155 мс
Матрица состояния слова = КОНТРОЛЬ

Word[5121] = 11100000000000010001110010000010 ---АДРЕС =  340  ---ВРЕМЯ НАЧАЛА СЛОВА =  3687.7070001615557 мс
Матрица состояния слова = ДАННЫЕ НЕ ГОТОВЫ

Word[7052] = 11100000000011001111100000000001 ---АДРЕС =  340  ---ВРЕМЯ НАЧАЛА СЛОВА =  5078.025000355891 мс
```

Рис. 11. Консольный вывод в результате работы программы

Заключение

В рамках данной статьи был проведен обзор авиационного интерфейса информационного обмена ARINC 429 (РТМ 1495-75). Были рассмотрены аппаратные и архитектурные особенности данного интерфейса. Был проведён анализ осциллограммы сигнала, снятого с линии связи, и интерпретация битовой информации для эффективной проверки на ошибки с помощью алгоритма на языке программирования Python.

Список литературы

1. ГОСТ 18977-79. *Комплексы бортового оборудования самолетов и вертолетов. Типы функциональных связей и уровни электрических сигналов*. М.: Изд-во стандартов, 1979 г.
2. Доля П.Г. *Введение в научный Python*. Харьков.: Харьковский Национальный Университет, 2016.
3. Ермошин, Н. Комплексный подход к освоению интерфейсов ARINC-429 и МКИО / Н. Ермошин, А. Власов, В. Ануфриев // *Компоненты и технологии*. 2015, № 9, с. 95-98.
4. Лемешко, Н. Анализ сигналов стандартов MIL-1553 и Arinc-429 с использованием осциллографов компании Rohde&Schwarz / Н. Лемешко, П. Струнин // *Компоненты и технологии*. 2018, № 9, с. 114-122.
5. Руководящий технический материал авиационной техники РТМ 1495-75. *Обмен информацией двухполярным кодом в оборудовании летательных аппаратов*. Дата введения в действие 01.07.1975 г.