Лабораторная работа №6
Радио интерфейс, передача данных в диапазоне 433Мгц Arduino

На лабораторной работе мы решим задачу по передаче радиосигнала между двумя контроллерами Ардуино с помощью популярного приемопередатчика с частотой 433МГц.

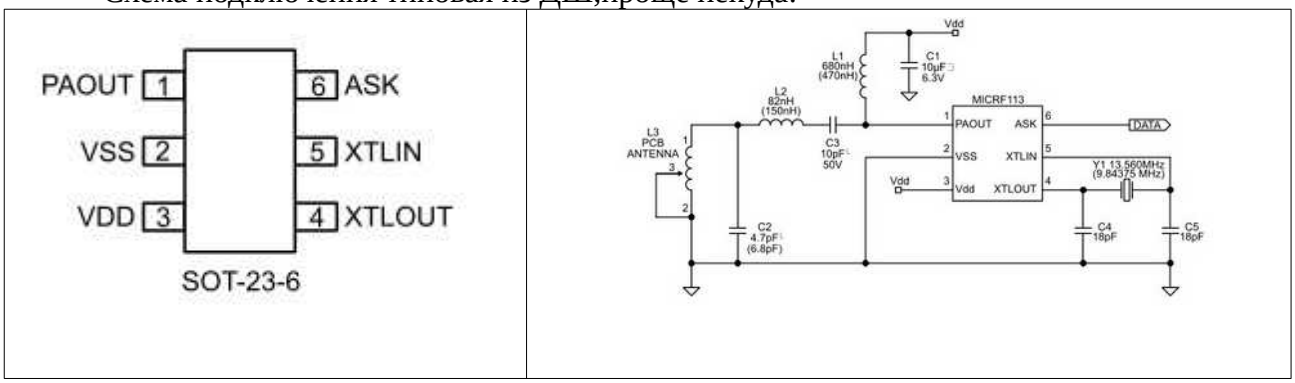
На самом деле, устройство по передаче данных состоит из двух модулей: приемника и передатчика. Данные можно передавать только в одном направлении. Это важно понимать при использовании этих модулей.

На этих модулях, можно сделать дистанционное управление любым электронным устройством, будь то мобильный робот или, например, телевизор. В этом случае данные будут передаваться от пульта управления к устройству. Другой вариант — передача сигналов с беспроводных датчиков на систему сбора данных. Здесь уже маршрут меняется, теперь передатчик стоит на стороне датчика, а приемник на стороне системы сбора.

Модули могут иметь разные названия: MX-05V, XD-RF-5V, XY-FST, XY-MK-5V, и т.п., но все они имеют примерно одинаковый внешний вид и нумерацию контактов. Также, распространены две частоты радиомодулей: 433 МГц и 315 МГц.

SYN113/SYN115 300-450MHz ASK Передатчик.

Схема подключения типовая из ДШ,проще некуда:



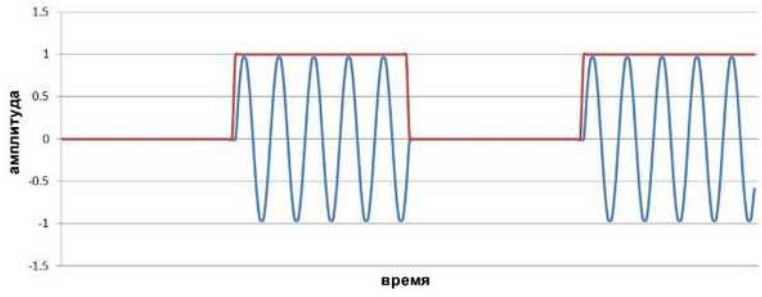
Передатчик SYN113/SYN115:
Частота передачи данных: 433 МГц
Напряжение питания: 1.8 – 3.6 В
Выходная мощность: <10 дБА
Скорость передачи данных: до 10 КБит
Модуляция: AM/ASK
Размеры платы: 15 × 12 × 3 мм.

Цифровая амплитудная модуляция

Это тип модуляции называется амплитудной манипуляцией (ASK, amplitude shift keying). Самый простой случай «включение-выключение» (OOK, on-off keying), и это почти соответствует математическим связям, обсуждаемым на странице, посвященной «аналоговой амплитудной модуляции»: если мы используем цифровой сигнал в качестве низкочастотного модулирующего сигнала, то перемножение модулирующего сигнала и несущей приводит к модулированному сигналу, который идет с нормальным уровнем при высоком логическом уровне и «выключен» при низком логическом уровне. Амплитуда логической единицы соответствует индексу модуляции.

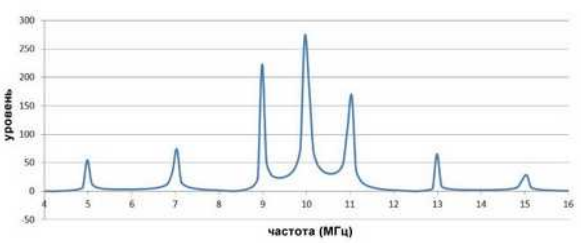
Временная область

Следующий график показывает сигнал амплитудной манипуляции «включено-выключено», сгенерированный с использованием несущей 10 МГц и цифрового тактового сигнала 1 МГц. Здесь мы работаем в математической области, поэтому амплитуда логической единицы (и амплитуда несущей) просто безразмерная «1», в реальной схеме вы можете иметь несущую с амплитудой 1 вольт и логический сигнал 3,3 вольта.

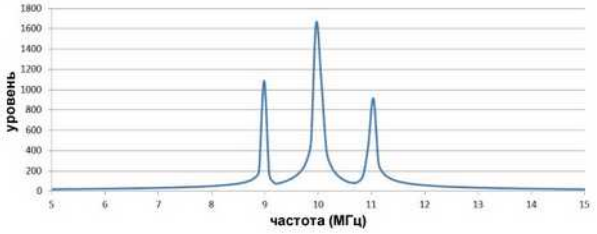


Частотная область

Ниже показан соответствующий спектр:



Спектр сигнала с амплитудной манипуляцией



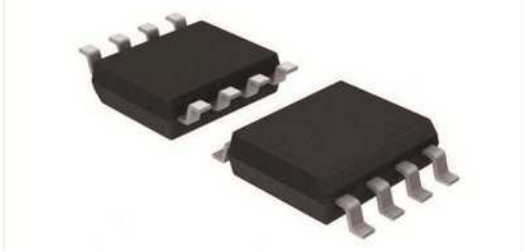
Спектр сигнала с амплитудной модуляцией сигналом синусоиды 1 МГц

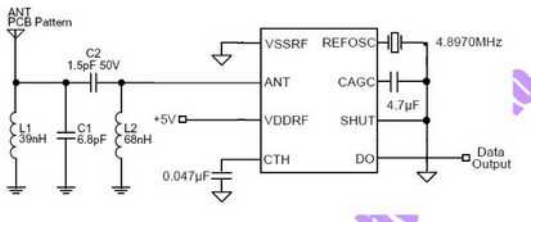
Бóльшая часть спектра одинакова: пик на несущей частоте (fнес), пик на fнес плюс частота модулирующего сигнала и пик на fнес минус частота модулирующего сигнала. Однако спектр амплитудной манипуляции (цифровой) имеет меньшие пики, соответствующие 3-й и 5-й гармоникам: основная частота (fF) равна 1 МГц, что означает, что 3-я гармоника (f3) равна 3 МГц, а 5-я гармоника (f5) равна 5 МГц. Таким образом, у нас есть пики при fнес плюс/минус fF, f3 и f5. И на самом деле, если бы вы расширили график, то увидели бы, что пики продолжают появляться в соответствии с этим примером.

Это имеет смысл. Преобразование Фурье прямоугольного сигнала состоит из синусоидальной волны на основной частоте и нечетных гармоник с понижающимися амплитудами, и эти гармонические составляющие являются тем, что мы видим на спектре, показанном выше.

Это обсуждение приводит нас к важной практической точке: резкие переходы, связанные с цифровыми методами модуляции, создают (нежелательные) высокочастотные составляющие. Мы должны помнить об этом, когда рассматриваем фактическую ширину полосы частот модулированного сигнала и наличие частот, которые могут мешать другим устройствам.

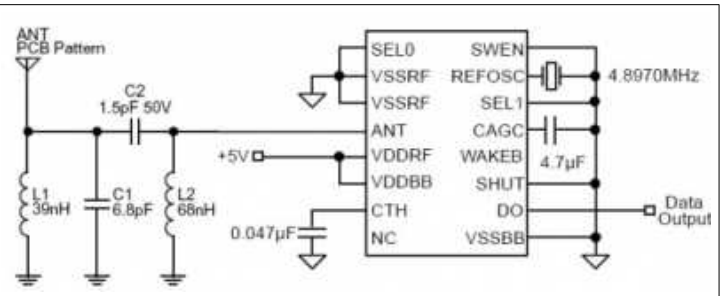
SYN470/SYN480 300-450MHz ASK Приемник





SYN470 обычный 16-ти пиновый SOP. SYN470 не так прост как 8-ми пиновый SYN480. В отличие от 480-го, который настроек не имеет, 470-й может работать в двух режимах и может выбрать полосу пропускания. За это отвечает комбинация подключения ног SWEN, SEL0 и SEL1.

SEL0	SEL1	Demodulator Bandwidth	
		Sweep Mode	Fixed Mode
1	1	5000 Hz	10000Hz
0	1	2500 Hz	5000Hz
1	0	1250 Hz	2500 Hz
0	0	625 Hz	1250 Hz



Номиналы деталей на разные частоты и формулы расчётов есть в ДШ.

Подключение:

Каждый из модулей подключается к контроллеру элементарно — питание через Vcc/Gnd и вывод Data подключается к свободному цифровому входу на микроконтроллере. Для повышения качества приёма/передачи рекомендуется дополнительно подключить антенну в виде провода размером 10-15 см. Кстати, дальность связи зависит ещё и от подающегося на модуль питания — если их запитать от 12В, то дальность и надёжность связи значительно возрастает.

Стандартная библиотека VirtualWire кодирует их специальным образом (каждая тетрада кодируется 6-ю битами, впереди добавляется синхронизирующий заголовок, и еще добавляется контрольная сумма для всего пакета) и на выходе превращает в более привычную последовательность байт. Но разбираться с ней уже приходится программисту самостоятельно.

Максимальная длина одного сообщения в VirtualWire равна 27 байт (см. документацию). Передача одного полного сообщения (оно автоматически дополняется сигнатурой 0xb38, значением длины сообщения и контрольной суммой) при выбранной мной скорости 1200 бит/с составляет 0,35 секунды.

Чем больше, кстати, выбранная скорость передачи, тем дальность передачи будет меньше. По опыту применения RS-232 известно, что при увеличении дальности допустимая скорость передачи экспоненциально падает: на скорости 19200 неэкранированная линия работает на 15 метров, на 9600 — 150 метров, а на скорости 1200 — более километра. Интересно было бы экспериментально выяснить характер этой зависимости для нашего случая, ведь очень много здесь зависит и от применяемой математики.

Инициализация передатчика в VirtualWire выглядит так:

```
. . . . .
#include <VirtualWire.h>
. . . . .
void setup() {
  vw_setup(1200); // Скорость соединения VirtualWire
  vw_set_tx_pin(10);   // Вывод передачи VirtualWire D10
. . . . .
}
```

Инициализация приемника в VirtualWire выглядит так:

```
#include <VirtualWire.h>
char str[5]; вспомогательная строка для преобразований ASCII в число
  uint8_t buf [VW_MAX_MESSAGE_LEN];  //буфер для хранения принятых данных
  uint8_t buflen = VW_MAX_MESSAGE_LEN;  // max длина принятых данных
```

```
. . . . .

void setup() {
  vw_set_rx_pin(2); //D2 Вывод приемника VirtualWire
  vw_setup(1200); // Скорость соединения VirtualWire
. . . . .
}
```

Пример передачи времени после сброса микроконтроллера:
Передатчик

```
#include  <VirtualWire.h>// Подключение библиотеки VirtualWire.h

const int led = 13; // Пин светодиода
const int transmit = 12; // Пин передатчика

void setup()
{
  vw_set_tx_pin(transmit); // Установка пина передачи данных
  vw_setup(2000);       // Скорость передачи (Бит в секунду)
  pinMode(led, OUTPUT);
}

void loop()
{

  digitalWrite(led, HIGH); // Зажечь светодиод перед передачей
  int seconds = (millis() / 1000); // Получение количества прошедших секунд
   String secondsresult = String(seconds); // Присвоить полученное значение строковой переменной
  char msg[14];
  secondsresult.toCharArray(msg, 14); //Скопировать символы строки в буфер.

  vw_send((uint8_t *)msg, strlen(msg)); // Отправить сообщение
  vw_wait_tx(); // Подождать до окончания отправки
  digitalWrite(led, LOW); // Сообщение отправлено - погасить светодиод
  delay(1000); // Задержка 1 секунда
}
```

Приемник

```
#include <VirtualWire.h> //Подключение библиотеки VirtualWire.h

byte message[VW_MAX_MESSAGE_LEN]; // Буфер хранения данных
byte messageLength = VW_MAX_MESSAGE_LEN; // Размер сообщения

const int receiver = 11; // Пин для приемника

void setup()
{
  Serial.begin(9600); // Для тестирования принятых сообщений
  Serial.println("Receiver is waiting for messages"); //Вывод сообщения
  vw_set_rx_pin(receiver); // Установка пина для приема

  vw_setup(2000); // Скорость передачи данных (бит в секунду)
```

```
  vw_rx_start(); // Инициализация приемника
}

void loop()
{
  messageLength = VW_MAX_MESSAGE_LEN; //Определение длины сообщения
  if (vw_get_message(message, &messageLength)) // Если данные получены
  {
    for (int i = 0; i < messageLength; i++) //В цикле
    {
      Serial.write(message[i]); // вывод их посимвольно в строку
    }
    Serial.write(" seconds from start"); //завершить сообщение
    Serial.println(); //Вывести набранную строку
  }
}
```

RadioHead Library — универсальная библиотека для беспроводных модулей

RadioHead — это библиотека, которая позволяет легко передавать данные между платами Arduino. Она настолько универсальна, что ее можно использовать для управления всеми видами устройств радиосвязи, включая наши модули на 433 МГц.

Библиотека RadioHead собирает наши данные, инкапсулирует их в пакет данных, который включает в себя CRC (проверку циклически избыточного кода), а затем отправляет его с необходимой преамбулой и заголовком на другую Arduino. Если данные получены правильно, принимающая плата Arduino проинформирует о наличии доступных данных и приступит к их декодированию и выполнению.

Пакет RadioHead формируется следующим образом: 36-битный поток из пар «1» и «0», называемый «обучающей преамбулой», отправляется в начале каждой передачи. Эти биты необходимы приемнику для регулировки его усиления до получения фактических данных. Затем следует 12-битный «Начальный символ», а затем фактические данные (полезная нагрузка). Последовательность проверки или CRC добавляется в конец пакета, который пересчитывается RadioHead на стороне приемника, и если проверка CRC верна, приемное устройство получает предупреждение. Если проверка CRC не пройдена, пакет отбрасывается. Весь пакет выглядит примерно так:

Описание методов класса RH_ASK

Для работы с данными радиомодулями будет использован класс RH_ASK. RH_ASK работает с рядом недорогих РЧ трансиверов ASK (амплитудная манипуляция), таких как RX-B1 (также известный как ST-RX04-ASK); Передатчик TX-C1 и приемопередатчик DR3100; Приемопередатчик FS1000A / XY-MK-5V; HopeRF RFM83C / RFM85. Поддерживает ASK (OOK).

RH_ASK ()

Конструктор. В настоящее время поддерживается только один экземпляр RH_ASK на скетч.

```
RH_ASK::RH_ASK (uint16_t        speed = 2000,
                uint8_t         rxPin = 11,
                uint8_t         txPin = 12,
                uint8_t         pttPin = 10,
                bool    pttInverted = false
```

Параметры:

speed — Желаемая скорость в битах в секунду

rxPin — Пин, который используется для получения данных от приемника

txPin — Пин, который используется для отправки данных на передатчик

pttPin — Пин, который подключен к EN передатчика. Будет установлено ВЫСОКОЕ состояние, чтобы включить передатчик (по умолчания pttInverted = true).

pttInverted — true, если вы хотите, чтобы pttin был инвертирован, чтобы НИСКОЕ состояние включило передатчик.

virtual bool init()

Инициализирует драйвер. Убедитесь, что драйвер настроен правильно перед вызовом init().

bool RH_ASK::init()

Возвращает: истина, если инициализация прошла успешно.

virtual bool available()

Проверяет, доступно ли новое сообщение из драйвера. Также переводит драйвер в режим RHModeRx до тех пор, пока сообщение не будет фактически получено транспортом, когда оно будет возвращено в RHModeIdle. Это может быть вызвано несколько раз в цикле ожидания.

bool RH_ASK::available()

Возвращает: true, если новое, полное, безошибочное несобранное сообщение доступно для извлечения с помощью recv().

virtual bool recv (uint8_t *buf, uint8_t *len)

Включает приемник, если он еще не включен. Если доступно допустимое сообщение, копирует его в buf и возвращает true, иначе возвращает false. Если сообщение копируется, *len устанавливается длина (Внимание, сообщения 0 длины разрешены). Вы должны вызывать эту функцию достаточно часто, чтобы не пропустить ни одного сообщения. Рекомендуется вызывать ее в основном цикле.

bool RH_ASK::recv(uint8_t * buf, uint8_t * len)

Параметры:

buf — место для копирования полученного сообщения

len — указатель на доступное пространство в буфере. Устанавливает фактическое количество скопированных октетов.

Возвращает: true, если действительное сообщение было скопировано в buf.

virtual bool send(const uint8_t *data, uint8_t len)

Ожидание завершения передачи любого предыдущего передаваемого пакета с помощью waitPacketSent(). Затем загружает сообщение в передатчик и запускает передатчик. Обратите внимание, что длина сообщения 0 НЕ допускается.

bool RH_ASK::send(const uint8_t * data, uint8_t len)

Параметры:

data — массив данных для отправки

len — Количество байтов данных для отправки (> 0)

Возвращает: истина, если длина сообщения была правильной, и оно была правильно поставлена в очередь для передачи

virtual uint8_t maxMessageLength()

Возвращает максимальную длину сообщения, доступную в этом драйвере.

uint8_t RH_ASK::maxMessageLength()

Возвращает: Максимальная допустимая длина сообщения

void setModeIdle()

Если текущий режим — Rx или Tx, он переключается в режим ожидания. Если передатчик или приемник работает, отключает их.

void INTERRUPT_ATTR RH_ASK::setModeIdle()

void setModeRx()

Если текущий режим Tx или Idle, изменяет его на Rx. Запускает приемник в RF69.

void RH_ASK::setModeRx()

void setModeTx()

Если текущий режим Rx или Idle, изменяет его на Rx. F Запускает передатчик в RF69.

void RH_ASK::setModeTx()

void handleTimerInterrupt()

не вызывайте этот метод, он используется обработчиком прерываний

uint16_t speed()

Возвращает текущую скорость в битах в секунду.

Пример скетча для приёмника

Приёмник будет получать числа от 0 до 255 по нарастающей. Если ожидаемое число получено не будет, тогда выводим в терминал ‘*’.

```
#include <RH_ASK.h>
#define SPEED        (uint16_t)1200
#define RX_PIN       (uint8_t)11
#define TX_PIN       (uint8_t)12
#define PTT_PIN      (uint8_t)10
#define PTT_INVERTED  false

/*
  Создаём экземпляр класса RH_ASK приёмника
*/
RH_ASK driver(SPEED, RX_PIN, TX_PIN, PTT_PIN, PTT_INVERTED);

void setup() {
  /*
    задаем скорость общения с компьютером
  */
  Serial.begin(115200);

  /*
    Инициализируем приёмник
  */
```

```
  if (! driver.init()) {
    Serial.println(F("RF init failed!"));
    while (true) {
      delay(1);
    }
  }
  /*
     Настройка встроенного светодиода
  */
  pinMode(LED_BUILTIN, OUTPUT);
}

void loop() {
  static uint8_t estdata;
  uint8_t data;
  uint8_t buflen = sizeof(data);
  /*
     Проверяем наличие новых данных
  */
  if (driver.recv((uint8_t*)&data, &buflen)) {
    /*
      Гасим светодиод
    */
    digitalWrite(LED_BUILTIN, LOW);
    /*
      Выводим в терминал '*' если полученные данные не совпадают с ожидаемыми
    */
    if (data != estdata) {
      Serial.print('*');
    }
    /*
      Выводим в терминал полученные данные
    */
    Serial.print("RX: ");
    Serial.println(data);
    /*
      Инкрементируем значение
    */
    estdata = data + 1;
    /*
      Включаем светодиод
    */
    digitalWrite(LED_BUILTIN, HIGH);
  }
}
```

Пример скетча для передатчика

Передатчик будет отправлять числа от 0 до 255 по нарастающей.

```
#include <RH_ASK.h>
#define SPEED        (uint16_t)1200
#define RX_PIN       (uint8_t)11
#define TX_PIN       (uint8_t)12
```

```
#define PTT_PIN       (uint8_t)10
#define PTT_INVERTED  false
/*
  Создаём экземпляр класса RH_ASK передатчика
*/
RH_ASK driver(SPEED, RX_PIN, TX_PIN, PTT_PIN, PTT_INVERTED);

void setup() {
  /*
    задаем скорость общения с компьютером
  */
  Serial.begin(115200);

  /*
    Инициализируем передатчик
  */
  if (!driver.init()) {
    Serial.println(F("RF init failed!"));
    while (true) {
      delay(1);
    }
  }
  /*
     Настройка встроенного светодиода
  */
  pinMode(LED_BUILTIN, OUTPUT);
}

void loop() {
  static uint8_t data = 0;

  /*
     Гасим светодиод
  */
  digitalWrite(LED_BUILTIN, LOW);
  /*
     Передаём данные
  */
  driver.send((uint8_t*)&data, sizeof(data));
  /*
    Ждем пока передача будет окончена
  */
  driver.waitPacketSent();
  /*
     Выводим в терминал отправленные данные
  */
  Serial.print("TX: ");
  Serial.println(data);
  /*
      Инкрементируем значение
  */
  ++data;
```

```
  /*
    Включаем светодиод
  */
  digitalWrite(LED_BUILTIN, HIGH);
  /*
    Ждём
  */
  delay(100);
}
```

Пример передачи данных рандомной длины

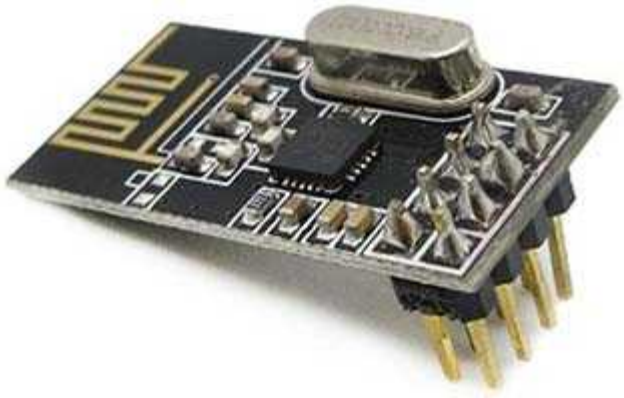
Для большего понимания принципа работы библиотеки, создал ещё один пример обмена данными между ардуинками. Передатчик будет отправлять массив данных uint8_t data[10] = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9'}; с интервалом в одну секунду. Количество отправленных байт будет неизвестно, потому что применяется функция random(1, 10). Принимающая сторона принимает эти данные и выводит их и их количество в терминал.

```
//Передатчик
#include <RH_ASK.h>
#define SPEED        (uint16_t)2000
#define RX_PIN       (uint8_t)11
#define TX_PIN       (uint8_t)12
#define PTT_PIN      (uint8_t)10
#define PTT_INVERTED  false
//  Создаём экземпляр класса RH_ASK передатчика
RH_ASK driver(SPEED, RX_PIN, TX_PIN, PTT_PIN, PTT_INVERTED);

void setup() {
//    задаем скорость общения с компьютером
Serial.begin(115200);
//    Инициализируем передатчик
  if (! driver.init()) {
    Serial.println(F("RF init failed!"));
    while (true) {
      delay(1);
    }
  }
//     Настройка встроенного светодиода
  pinMode(LED_BUILTIN, OUTPUT);
}

void loop() {
//    Буфер данных для отправки
uint8_t data[10] = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9'};
//     Гасим светодиод
  digitalWrite(LED_BUILTIN, LOW);
//    Передаём массив данных случайной длины (от 1 до 10)
  driver.send(data,  random(1, 10));
//    Ждем пока передача будет окончена
  driver.waitPacketSent();
//    Включаем светодиод
  digitalWrite(LED_BUILTIN, HIGH);
//    Ждём секунду
```

```
  delay(1000);
}

// Приемник
#include <RH_ASK.h>
 #define SPEED        (uint16_t)2000
#define RX_PIN       (uint8_t)11
#define TX_PIN       (uint8_t)12
#define PTT_PIN      (uint8_t)10
#define PTT_INVERTED  false
 //Создаём экземпляр класса RH_ASK приёмника
RH_ASK driver(SPEED, RX_PIN, TX_PIN, PTT_PIN, PTT_INVERTED);

void setup() {
//    задаем скорость общения с компьютером
  Serial.begin(115200);
//   Инициализируем передатчик
  if (! driver.init()) {
    Serial.println(F("RF init failed!"));
    while (true) {
      delay(1);
    }
  }
//    Настройка встроенного светодиода
pinMode(LED_BUILTIN, OUTPUT);
}

void loop() {
//     Буфер полученных данных
  uint8_t data[16];
//     Размер полученных данных
uint8_t buflen = sizeof(data);
uint8_t i;
//     Проверяем наличие новых данных
  if (driver.recv(data, &buflen)) {
//      Гасим светодиод
    digitalWrite(LED_BUILTIN, LOW);
//      Выводим в терминал полученные данные
    Serial.print("Size: ");
    Serial.print(buflen);
    Serial.println();
    Serial.print("Data: ");
    for (i = 0; i < buflen; i++) {
      Serial.print((char)data[i]);
      Serial.print(' ');
    }
    Serial.println();
//      Включаем светодиод
    digitalWrite(LED_BUILTIN, HIGH);
  }
}
```

Радиомодуль nRF24L01.

Существует очень большое разнообразие различных микросхем приёмопередатчиков и один из самых популярных это nRF24L01+. На столько высокую популярность они получили за счёт сочетания низкой цены, относительно небольшого энергопотребления, простоты работы с ними и высокой гибкостью при построении сетей различных топологий и сложности.



Так выглядит один из самых популярных вариантов модуля на nRF24L01:

Он хорошо подходит для первоначального ознакомления и использования в проектах, в некоторых модулях есть разъём для подключения с такой распиновкой и шагом контактов 2.54мм. Этим ассортимент модулей не ограничивается, есть множество других. Они бывают разных габаритов. С печатной антенной, керамической или внешней. С усилителем или без. На плате может быть стабилизатор питания на 3.3В, может и не быть.

GND - земля
VCC - плюс питания, от 1.9-3,9 Вольт. При превышении напряжения, например, при подключении к 5В, модуль может выйти из строя. Если на входных пинах (MOSI, SCK, CE, CSN) более 3.6 Вольт, напряжение питания должно находиться в пределах 2.7-3.3 Вольт
MOSI (Master Out Slave In) - выход ведущего, вход ведомого. Служит для передачи данных от ведущего устройства ведомому
MISO (Master In Slave Out) - вход ведущего, выход ведомого. Служит для передачи данных от ведомого устройства ведущему
SCK (Serial Clock) - входной, по нему происходит тактирование от ведущего устройства при передаче данных
CE (Chip Enable) - входной, служит для активации режима RX или TX
CSN (Chip Select Not) - входной, с помощью него происходит управление при передаче команд по SPI шине
IRQ - это вывод прерывания, через который мастеру сообщается о том, что что-то произошло (получен пакет, превышено количество попыток при отправке и т.д.)

Характеристики

- Частотный диапазон 2,4ГГц (2,4000-2,4835ГГц)
- GFSK модуляция
- 126 частотных канала
- Поддерживаемые битрейты: 250кбит, 1Мбит и 2Мбит
- Программируемая мощность передатчика и усиления приёмника

Существует ошибочное мнение, что 126 (с 0 по 125) частотных канала это максимальное количество модулей, между которыми можно передавать данные. Количество частотных каналов при некоторых условиях может влиять на количество модулей, но в

общем на прямую это никак не связано. Количество модулей зависит от ряда вещей - какая пропускная способность нужна, какая адресация реализована, топология сети и т.д. Даже если использовать аппаратную адресацию (поле адреса до пяти байт, т.е. более триллиона), столько nRF24L01 просто не существует.

Модули, настроенные на одинаковый канал, могут обмениваться данными. Если модули настроены на разные каналы, приёмник не может «услышать» данные, передаваемые другим модулем.

Частотные каналы хоть и можно использовать для псевдослучайной перестройки частоты, но в общем для этого скорей всего лучше окажется использование других приёмопередатчиков. А с nRF24l01 настройку частотного канала обычно используют, если на определённой частоте есть помехи от других устройств. Это могут быть WiFi, Bluetooth или ещё какие устройства. Для nRF24l01 так же могут быть использованы разные частоты, допустим, чтобы упростить код/алгоритм, протокол, настройку, максимальный битрейт тоже не бесконечный (разделение на подсети) и т.д. Два и более модуля на одну частоту, два и более модуля на другую частоту и т.д. - модули, работающие на разных частотах, не будут друг-другу мешать.

Внешний интерфейс:

Четырёхвыводной SPI

Максимальный битрейт 8Мбит

Трёхуровневые FIFO буферы по 32 байта для приёма и передачи

Выводы толерантны к 5В

Использование SPI интерфейса может оказаться особенно полезным, если понадобиться использовать программную реализацию интерфейса или к одной плате с управляющим микроконтроллером нужно подключить более одного модуля nRF24L01.

Аппаратный уровень обработки пакетов ShockBurst:

Сборка пакета при отправке

Обнаружение входящих пакетов и их проверка

Настраиваемые источники прерывания для активирования вывода IRQ

Автоматическая отправка обратного уведомления при получении пакета

Повторная отправка утерянных пакетов (если включено, можно настроить количество попыток)

Настраиваемое количество байт для поля данных от 0 до 32 байт в статическом режиме и автоматическое задание в динамическом режиме

Настраиваемая длина адреса от 3 до 5 байт

«MultiCeiver» для реализации топологии «1:6 звезда»

ShockBurst прост и гибок в настройке. Беря на себя выполнение части функций, позволяет упростить код и работу с модулем, а также немного разгружает микроконтроллер/микропроцессор управляющей платы и даёт ему возможность переходить в энергосберегающий режим в то время, когда модуль отправляет данные, ожидает пакет или обрабатывает его.

Формат пакета:

Преамбула, 1 байт

Адрес, 3-5 байт

Поле контроля пакета, 9 бит

Данные, 0-32 байт

Контрольная сумма, 1-2 байта

Данные в поле преамбулы устанавливаются модулем автоматически, в зависимости от значения первого бита адреса.

Структура поля контроля пакета:

Количество байт в поле данных, 6 бит. Это поле используется только если включен динамический режим и ему присваивается число от 0 до 33 (33 означает, что длина поля данных не имеет значения)

Идентификатор пакета, 2 бита. На стороне передатчика поле идентификатора пакета автоматически увеличивается при получении по SPI нового пакета. На стороне приёмника это поле используется для определения, этот пакет новый или был отправлен повторно

Флаг NO_ACK (сокращение от «No Acknowledgment», т.е. «нет подтверждения»), 1 бит. Этот флаг используется только если включен функционал автоматического подтверждения

Поле данных может содержать от 1 до 32 байт. Длина поля равная 0 может использоваться для проверки, доступен ли радиомодуль с определённым адресом.

Поле контрольной суммы используется для определения, получен повреждённый пакет или нет.

В режиме приёма, модуль сначала ожидает преамбулу, затем адрес. Приём следующих полей пакета продолжается только после успешной проверки, что преамбула соответствует первому биту адреса и что такой адрес удалось найти в регистрах RX_ADDR. После получения остальных полей пакета, происходит проверка контрольной суммы и в случае успеха, данные записываются в FIFO буфер приёмника, из которого их уже может считать управляющий контроллер.

При включенном функционале автоматического подтверждения, модуль в режиме приёма после получения пакета отправит обратно пакет подтверждения.

При включенном функционале повторной отправки, модуль будет пытаться отправлять пакет до тех пор, пока или не получит обратный пакет подтверждения или будет превышено количество попыток. В регистре «SETUP_RETR» можно настроить количество попыток отправки (до 15 или 0, если нужно отключить) и время ожидания пакета подтверждения (от 250мкс до 4000 мкс, с шагом 250 мкс).

Подключение nRF24L01 к плате Arduino

Подключение радиомодуля NRF24L01 к Arduino осуществляется по SPI-интерфейсу, что предполагает использование 5 проводов не считая выводов питания. Для разных линеек Arduino номера выводов, на которые завязан аппаратный SPI-интерфейс, могут отличаться. На рисунке показана карта подключения NRF24L01 к различным сериям Arduino.

NRF24L01	GND	VCC	CE *	CSN *	SCK	MOSI	MISO	IRQ
Arduino UNO	GND	3,3V	любой	любой	13	11	12	*
Arduino Nano	GND	3,3V	любой	любой	13	11	12	*
Arduino Pro Mini	GND	внешний ИП	любой	любой	13	11	12	*
Arduino Mega	GND	3,3V	любой	любой	52	51	50	*
Arduino Leonardo	GND	3,3V	любой	любой	ICSP–SCK	ICSP–MOSI	ICSP–MISO	*

Выводы CE и CSN могут быть соединены с любыми цифровыми пинами Arduino. Единственное что потребуется – указать их номера при написании скетча. Что касается программирования, то для взаимодействия с NRF24L01 существует несколько библиотек, но наиболее популярной и стабильной является библиотека RF24.

Как правило, большинство любительских проектов начального уровня предусматривают использование двух модулей NRF24L01, один из которых работает в режиме передатчика, а другой как приёмник на одинаковой частоте. Но что делать, когда на одном канале необходимо контролировать сразу несколько датчиков, например температуру в разных комнатах? В этом случае, функциональные возможности радиомодуля NRF24L01 предусматривают возможность организации мини-сети. А именно, на одной частоте или канале могут работать до 6 передатчиков и 1 приёмник. При этом каждому передатчику присваивается свой уникальный идентификатор («Pipe ID» или «Идентификатор трубы»), а

приёмнику необходимо присвоить все идентификаторы тех передатчиков, от которых он будет принимать данные.

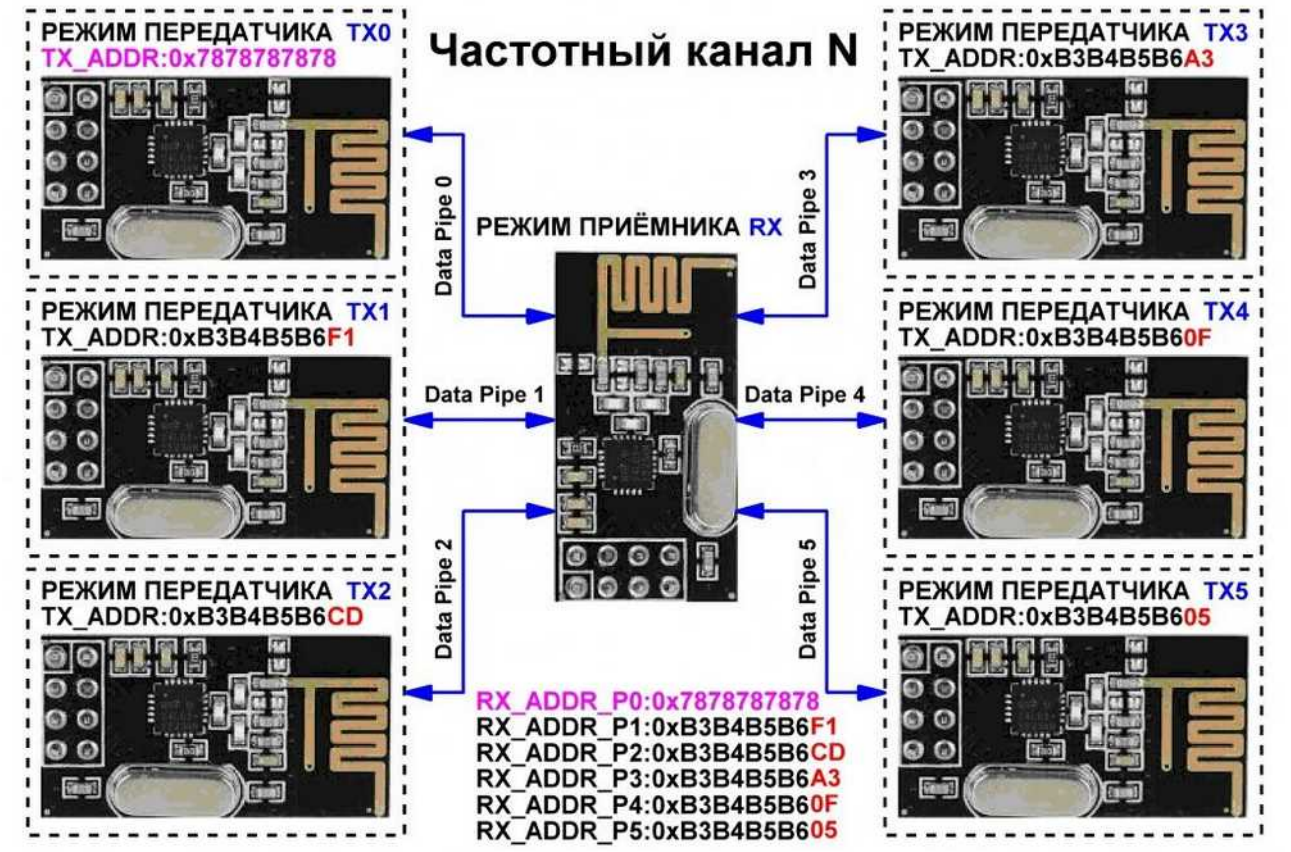
Каждый идентификатор представляет из себя произвольное число, состоящее из 5 байт, но он должен задаваться по определённым правилам, а именно:

На одном и том же канале идентификатор каждого передатчика должен быть обязательно уникальным;

Чтобы приёмник мог принимать данные от передатчиков, ему должны быть указаны их идентификаторы;

Идентификаторы труб Pipe0 и Pipe1 должны отличаться всеми пятью байтами, например Pipe0 = 0x7878787878, а Pipe1 = 0xB4B5B6B7F1;

Идентификаторы труб Pipe2 – Pipe5 должны отличаться от Pipe1 только последним байтом, например: Pipe1 = 0xB4B5B6B7F1; Pipe2 = 0xB4B5B6B7CD; Pipe3 = 0xB4B5B6B7A3; Pipe4 = 0xB4B5B6B70F; Pipe5 = 0xB4B5B6B705;



Циклический избыточный код (англ. Cyclic redundancy code, CRC)

При передаче сигнала через любой канал связи возможно возникновение ошибок, которые могут приводить к искажению переносимой информации. Существует много методов для исправления подобных ошибок, но прежде чем исправлять, необходимо эти ошибки обнаружить.

Для этого также существуют определенные методы, основанные на избыточности передаваемой информации, что позволяет не только выявлять наличие факта искажения информации, но и в ряде случаев устранять эти искажения. Наиболее известные из методов обнаружения ошибок передачи данных являются:

- Посимвольный контроль четности, называемый также поперечным, подразумевает передачу с каждым байтом дополнительного бита, принимающего единичное значение по четному или нечетному количеству единичных битов в контролируемом байте. Посимвольный контроль четности прост как в программной, так и в аппаратной реализации,

но его вряд ли можно назвать эффективным методом обнаружения ошибок, так как искажение более одного бита исходной последовательности резко снижает вероятность обнаружения ошибки передачи. Этот вид контроля обычно реализуется аппаратно в устройствах связи.

- Поблочный контроль четности, называемый продольным. Схема данного контроля подразумевает, что для источника и приемника информации заранее известно, какое число передаваемых символов будет рассматриваться ими как единый блок данных. В этой схеме контроля для каждой позиции разрядов в символах блока (поперек блока) рассчитываются свои биты четности, которые добавляются в виде обычного символа в конец блока. По сравнению с посимвольным контролем четности поблочный контроль четности обладает большими возможностями по обнаружению и даже корректировке ошибок передачи, но все равно ему не удается обнаруживать определенные типы ошибок.
- Вычисление контрольных сумм. В отличие от предыдущих методов для метода контрольных сумм нет четкого определения алгоритма. Каждый разработчик трактует понятие контрольной суммы по-своему. В простейшем виде контрольная сумма — это арифметическая сумма двоичных значений контролируемого блока символов. Но этот метод обладает практически теми же недостатками, что и предыдущие, самый главный из которых — нечувствительность контрольной суммы к четному числу ошибок в одной колонке и самому порядку следования символов в блоке.
- Контроль циклически избыточным кодом — CRC (Cyclical Redundancy Check). Это гораздо более мощный и широко используемый метод обнаружения ошибок передачи информации. Он обеспечивает обнаружение ошибок с высокой вероятностью. Кроме того, этот метод обладает рядом других полезных моментов, которые могут найти свое воплощение в практических задачах.

Циклический избыточный код (англ. Cyclic redundancy code, CRC) — алгоритм вычисления контрольной суммы, предназначенный для проверки целостности передаваемых данных. Алгоритм CRC обнаруживает все одиночные ошибки, двойные ошибки и ошибки в нечетном числе битов. Понятие циклических кодов достаточно широкое, однако на практике его обычно используют для обозначения только одной разновидности, использующей циклический контроль (проверку) избыточности. В связи с этим в англоязычной литературе CRC часто расшифровывается как Cyclic Redundancy Check.

CRC некоторой последовательности вычисляется на основании другой (исходной) битовой последовательности. Главная особенность (и практическая значимость) значения CRC состоит в том, что оно однозначно идентифицирует исходную битовую последовательность и поэтому используется в различных протоколах связи, а также для проверки целостности блоков данных, передаваемых различными устройствами. Благодаря относительной простоте алгоритм вычисления CRC часто реализуется на аппаратном уровне.

Основная идея вычисления CRC заключается в следующем. Исходная последовательность битов, которой могут быть и огромный файл, и текст размером несколько слов и даже символов, представляется единой последовательностью битов. Эта последовательность делится на некоторое фиксированное двоичное число (полином, CRC-полином, генераторный полином, англ. Generator polinomial). Интерес представляет остаток от этого деления, который и является значением CRC. Все, что теперь требуется, — это некоторым образом запомнить его и передать вместе с исходной последовательностью. Приемник данной информации всегда может таким же образом выполнить деление и сравнить его остаток с исходным значением CRC. Если они равны, то считается, что исходное сообщение не повреждено, и т. д.

Степенью CRC-полинома W называют позицию самого старшего единичного бита. Например, степенью полинома 10011 2 равна 4.

Для вычисления CRC используют специальную т.н. полиномиальную арифметику. Вместо представления делителя, делимого (сообщения), частного и остатка в виде положительных целых чисел, можно представить их в виде полиномов с двоичными

коэффициентами или в виде строки бит, каждый из которых является коэффициентом полинома.

Приведем пример: Предположим, что сообщение состоит из 2 байт (6,23). Их можно рассматривать, как двоичное число 0000 0110 0001 0111. Предположим, что ширина регистра контрольной суммы составляет 1 байт, в качестве делителя используется 1001, тогда сама контрольная сумма будет равна остатку от деления 0000 0110 0001 0111 на 1001. Хотя в данной ситуации деление может быть выполнено с использованием стандартных 32 битных регистров, в общем случае это неверно. Поэтому воспользуемся делением "в столбик" в двоичной системе счисления:

1001=9d делитель

0000011000010111 =0617h =1559d делимое

0000011000010111/1001=2d остаток

Хотя влияние каждого бита исходного сообщения на частное не столь существенно, однако 4 битный остаток во время вычислений может радикально измениться, и чем больше байтов имеется в исходном сообщении (в делимом), тем сильнее меняется каждый раз величина остатка. Вот почему деление оказывается применимым там, где обычное сложение работать отказывается.

В нашем случае передача сообщения вместе с 4 битной контрольной суммой выглядела бы (в шестнадцатеричном виде) следующим образом:06172,где 0617 – это само сообщение, а 2 – контрольная сумма. Приемник, получив сообщение, мог бы выполнить аналогичное деление и проверить, равен ли остаток переданному значению.

Полиномиальная арифметика

Все CRC алгоритмы основаны на полиномиальных вычислениях, и для любого алгоритма CRC можно указать, какой полином он использует. Что это значит?

Вместо представления делителя, делимого (сообщения), частного и остатка в виде положительных целых чисел, можно представить их в виде полиномов с двоичными коэффициентами или в виде строки бит, каждый из которых является коэффициентом полинома. Например, десятичное число 23 в шестнадцатеричной системе счисления имеет вид 17,в двоичном – 10111, что совпадает с полиномом:

1*x ^4 +0*x ^3 +1*x ^2 +1*x ^1 +1*x ^0

или, упрощенно:

x ^4 +x ^2 +x ^1 +x ^0

И сообщение, и делитель могут быть представлены в виде полиномов, с которыми, как и раньше можно выполнять любые арифметические действия; только теперь надо не забывать о “иксах”.

CRC-арифметика.

Что же такое особенное есть в CRC, что позволяет обнаруживать большую часть ошибок? Это безусловно метод его нахождения. Для нахождения CRC используется специальная CRC-арифметика, которая является полиномиальной арифметикой по модулю 2. Т.е. битовое значение 1001001112=16710 может быть представлено в виде: 1*x^8+0*x^7+0*x^6+1*x^5+0*x^4+0*x^3+1*x^2+1*x^1+1. Для того чтобы получить искомое значение, можно подставить x=2. Эта арифметика обладает одним весьма не маловажным свойством - операции сложения и вычитания идентичны, и поэтому можно оставить лишь одну операцию, т.е.

1001011
1100010

0101001

Сложение и вычитание ведется по следующим правилам: 0-0=0, 0-1=1, 1-0=1, 1-1=0. Таким образом можно заметить, что результат вычитания в арифметике без переносов, аналогичен операции XOR. Для нашего рассмотрения особый интерес представляет операция деления, так как в основе любого алгоритма вычисления CRC лежит представление исходного сообщения в виде огромного двоичного числа, делении его на другое двоичное число и использование остатка от этого деления в качестве значения CRC. Говоря иначе: M - исходное сообщение, n - степень порождающего полинома, G - порождающий полином, тогда CRC - это остаток от M*x^n/G.

Например, сообщение:

M=0x54, G=0x11021 или в бинарном виде

M'=0101,0100,0000,0000,0000,0000 G=1,0001,0000,0010,0001

```
01010100000000000000000|00010001000000100001
10001000000100001    |частное никого не интересует
 ----------------------
00100000000100001000000
 10001000000100001
 ----------------------
 00001000010100101000 0
      10001000000100001
 ----------------------
      00001101001110001
      0001,1010,0111,0001=0x1a71 - остаток от деления
```

После получения этого остатка - остаток добавляется в конец передаваемого сообщения и сообщение передается. Есть 2 способа обнаружения ошибок:

Приемник может получить сообщение отделить последние 16 бит, посчитать CRC оставшегося и сравнить с тем 16 битами.

Или же приемник может поделить полученное сообщение на G, и таким образом обнаружить ошибку

Алгоритм

В самом алгоритме вычисления CRC нет ничего сложного, его можно на любом языке, потому что операции XOR и SHL встроены практически в любой язык. Коротко весь алгоритм можно записать так:

G - полином, M - битовое сообщение.

1. Дополнить исходное сообщение 16-ю нулями. M'=M*x16.
2. Инициализировать начальное значение CRC 0x0000.
3. Выполнять операцию сдвиг влево последовательности бит сообщения M' до тех пор, пока бит в ячейке (ячейка - то место, где изначально находился старший бит M') не станет равным единице или количество бит станет меньше чем в делителе.
4. Если старший бит станет равным единице, то производим операцию XOR между сообщением и полиномом. И повторяем шаг 2.
5. То что в конце остается от последовательности M' и называется CRC.
6. Возможно потребуется обратить результат.

В случае зеркального алгоритма применяемого при вычислении CCITT X.25 и IBM Bisynch CRC есть некоторые отличия, а именно M - не само сообщение, а его зеркальное отражение и еще в X.25 начальное значение - инициализируется в 0xffff - это всего лишь операция XOR первых 16 бит сообщения с 0xffff. Во всем остальном алгоритм работает так же.

Таким образом, алгоритм вычисления CRC имеет несколько параметров:

- начальное значение
- порядок прохода битов сообщения (зеркальный или прямой)

- обращение результата.

```
uint16_t RHcrc16_update(uint16_t crc, uint8_t a)
{
  int i;
  crc ^= a;
  for (i = 0; i < 8; ++i)
  {
      if (crc & 1)
        crc = (crc >> 1) ^ 0xA001;
      else
        crc = (crc >> 1);
  }
  return crc;
}
```

Примеры используемых полиномов

Номер варианта	Название	Образующий полином	Примечание
1	CRC-16-IBM	$x^{16}+x^{15}+x^2+1$	Bisync, Modbus, USB, ANSI X3.28, многие другие; также известен как CRC-16 и CRC-16-ANSI
2	CRC-8-CCITT	x^8+x^2+x+1	(ATM HEC), ISDN Header Error Control and Cell Delineation ITU-T I.432.1 (02/99)
3	CRC-24	$x^{24}+x^{22}+x^{20}+x^{19}+x^{18}+x^{16}+x^{14}+x^{13}+x^{11}+x^{10}+$ $+x^8+x^7+x^6+x^3+x+1$	FlexRay
4	CRC-16-T10-DIF	$x^{16}+x^{15}+x^{11}+x^9+x^8+x^7+x^5+x^4+x^2+x+1$	SCSI DIF
5	CRC-5-USB	x^5+x^2+1	USB token packets
6	CRC-32C (Castagnoli)	$x^{32}+x^{28}+x^{27}+x^{26}+x^{25}+x^{23}+x^{22}+x^{20}+x^{19}+x^{18}+x^{14}+$ $+x^{13}+x^{11}+x^{10}+x^9+x^8+x^6+1$	iSCSI, G.hn payload
7	CRC-10	$x^{10}+x^9+x^5+x^4+x+1$	
8	CRC-24-Radix-64	$x^{24}+x^{23}+x^{18}+x^{17}+x^{14}+x^{11}+x^{10}+x^7+x^6+x^5+x^4+$ $+x^3+x+1$	OpenPGP
9	CRC-8-Dallas/Maxim	$x^8+x^5+x^4+1$	1-Wire bus
10	CRC-32K (Koopman)	$x^{32}+x^{30}+x^{29}+x^{28}+x^{26}+x^{20}+x^{19}+x^{17}+x^{16}+x^{15}+$ $+x^{11}+x^{10}+x^7+x^6+x^4+x^2+x+1$	
11	CRC-15-CAN	$x^{15}+x^{14}+x^{10}+x^8+x^7+x^4+x^3+1$	
12	CRC-4-ITU	x^4+x+1	ITU G.704 [2)]
13	CRC-32Q	$x^{32}+x^{31}+x^{24}+x^{22}+x^{16}+x^{14}+x^8+x^7+x^5+x^3+x+1$	aviation; AIXM
14	CRC-16-CCITT	$x^{16}+x^{12}+x^5+1$	X.25, HDLC, XMODEM,

Номер варианта	Название	Образующий полином	Примечание
			Bluetooth, SD и др.
15	CRC-8	$x^8+x^7+x^6+x^4+x^2+1$	ETSI EN 302 307 [3], 5.1.4
16	CRC-32-IEEE 802.3	$x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+$ $+x^5+x^4+x^2+x+1$	V.42, MPEG-2, PNG, POSIX cksum
17	CRC-6-ITU	x^6+x+1	ITU G.704 [2]
18	CRC-64-ISO	$x^{64}+x^4+x^3+x+1$	HDLC — ISO 3309
19	CRC-11	$x^{11}+x^9+x^8+x^7+x^2+1$	FlexRay [4]
20	CRC-16-CCITT	$x^{16}+x^{12}+x^5+1$	X.25, HDLC, XMODEM, Bluetooth, SD и др.
21	CRC-5-EPC	x^5+x^3+1	Gen 2 RFID [5]
22	CRC-6-ITU	x^6+x+1	ITU G.704 [2]
23	CRC-16-DNP	$x^{16}+x^{13}+x^{12}+x^{11}+x^{10}+x^8+x^6+x^5+x^2+1$	DNP, IEC 870, M-Bus
24	CRC-7	x^7+x^3+1	системы телекоммуникации, ITU-T G.707, ITU-T G.832, MMC, SD. [6]
25	CRC-8-Dallas/Maxim	$x^8+x^5+x^4+1$	1-Wire bus
26	CRC-8-SAE J1850	$x^8+x^4+x^3+x^2+1$	
27	CRC-30	$x^{32}+x^{29}+x^{21}+x^{20}+x^{15}+x^{13}+x^{12}+x^{11}+x^8+x^7+x^6+$ $+x^2+x+1$	CDMA
28	CRC-5-ITU	$x^5+x^4+x^2+1$	ITU G.704 [2]
29	CRC-12	$x^{12}+x^{11}+x^3+x^2+x+1$	системы телекоммуникации [7]
30	CRC-64-ECMA-182	$x^{64}+x^{62}+x^{57}+x^{55}+x^{54}+x^{53}+x^{52}+x^{47}+x^{46}+x^{45}+x^{40}+$ $+x^{39}+x^{38}+x^{37}+x^{35}+x^{33}+x^{32}+x^{31}+x^{29}+x^{27}+x^{24}+x^{23}$ $+x^{22}+x^{21}+x^{19}+x^{17}+x^{13}+x^{12}+x^{10}+x^9+x^7+x^4+x+1$	

Задания к лабораторной работе №6
Задание пишется для лабораторного макета (определение аппаратной части)

Задание для очного обучения
Данная работа предполагает парность, т.е задействована ПАРА макетов, один из которых приемник, второй передатчик.
1. Сделать передачу сообщений от передатчика к приемнику.
2. Передача данных датчика DHT11 на приемник и отображение этих данных на дисплее/
3. Сделать термостат с удаленным датчиком температуры и удаленным управлением. Передатчик должен передавать температуру, влажность, установленную температуру работы термостата (придумать свой метод), включить/выключить термостат. Приемник должен получать и обрабатывать полученные данные, и отображать данные на дисплее.

Задание для дистанционного обучения

Разработать программу удаленного пульта (интеллектуального датчика) простейшего терморегулятора, со следующими характеристиками:

1. В качестве датчика температуры использовать датчик LM35(A2). Текущая температура Тт. Точность расчетов и хранения температуры не ниже 0.1 градуса.

2. Устройство должно передавать данные по беспроводной сети используя передатчик на основе SYN113/SYN115, подключенного к выходу D7 макета. Устройство только передает данные и не принимает решения о включении нагревательного элемента. Должны передаваться:

id — устройства для его идентификации
Тт — текущая температура
Тц — целевая температура
N - номер пакета
F — период передачи

3. Установку целевой температуры (Тц) проводить с помощью переменного резистора присоединенного в выходу A0. Точность установки до 0.1 градуса. Диапазон регулирования +5 … +30 градусов.

4. Частота передачи данных задается кнопками D2 (увеличение 1 сек.) и D3 (уменьшение 1 сек.). Диапазон регулирования от 1 до 12 секунд.

5. Для показа состояния регулятора использовать LCD дисплей подключенный к шине i2c тип дисплея 1602. Должны отображаться Тт, Тц, период посылки данных и число посланных пакетов (5 знаков).

6. В момент передачи для индикации загорается красный светодиод D12.

Для продвинутых пользователей.
7. Добавить выполнение через консоль следующих команд:
state — вывод состояния устройства. Выводится:
текущая температура
целевая температурам
id устройства
число посланных пакетов
период передачи
setTime — ввод значения периода передачи данных. Диапазон регулирования от 1 до 12 секунд. Проверка на валидность введенных данных.
setID — ввод идентификатора устройства. Диапазон 1-100. Проверка на валидность введенных данных.
ON — включить передачу данных
OFF — выключить передачу данных, при этом зажигаем светодиод D13.
help — вывести список команд.

info — информация о разработчике и версии программы, времени и даты сборки прошивки (макросы __TIME__ __DATE__).
При старте программы в консоли выводится приглашение и подсказка об использовании команды help.