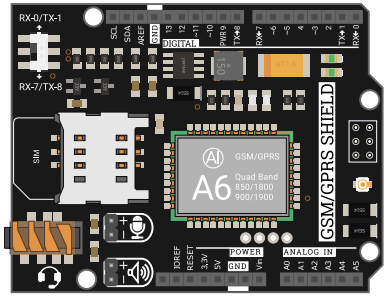
GSM/GPRS Shield: руководство по использованию



Общие сведения

Плата расширения GSM/GPRS Shield подарит вашему устройству на Arduino мобильную связь для осуществления входящих/исходящих голосовых звонков, отправки/получения SMS и приёма/передачи данных через мобильный интернет.

На основе GSM/GPRS Shield можно создать радионяню с радиусом действия всего земного шара, сделать SMS-сигнализацию складского помещения или разработать систему мониторинга климатический условий в вашей загородной тепличке. Главное условие — SIM-карта с наличием мобильной связи.

Плата GSM/GPRS Shield работает с беспроводными стандартами связи 2G GSM/GPRS. К таким оператором в России относится популярная тройка: MTS, Megafon и Beeline. Для установки SIM-карты на плате расположен соответствующий слот под карту. Если вы хотите использовать SIM-карту другого оператора или находитесь в другой стране, уточните у своего провайдера поддержку стандарта GSM.

Версии плат

Функция	GSM/GPRS Shield A6	GSM/GPRS Shield A9
Совершать входящие и исходящие звонки	Да	Да
Отправлять и принимать SMS-сообщения	Да	Да
Выходить в сеть через мобильный интернет	Нет	Да
Размер SIM-карты	mini SIM (2FF)	nano SIM (4FF)

Видеообзор

Обзор GSM/GPRS Shield A6

Обзор GSM/GPRS Shield A9

Подключение и настройка

Платы расширения GSM/GPRS Shield A6 и A9 предусмотрены для установки на платформы форм-фактора Arduino Shield R3. Для коммуникации с контроллером используются контакты интерфейса UART (TX и RX) с дополнительным пином управления PWR.

> Для связи двух устройств в интерфейсе UART существует правило: линия TX подключается к выводу RX, а линия RX к выводу TX. В Arduino UART часто называют Serial. Подробнее про интерфейс UART читайте в нашей статье на вики.

В зависимости от версии GSM/GPRS Shield и управляющей платформы, интерфейс UART может располагаться на разных пинах. Выберите свой вариант подключения GSM/GPRS-модуля.

GSM/GPRS A9 к Arduino Leonardo

На Arduino Leonardo и других платформах с микроконтроллером ATmega32U4, данные по USB и аппаратный интерфейс UART не связаны между собой. Это даёт возможность подключить GSM/GPRS Shield A9 к аппаратному UART платформы на пинах `RX-0` и `TX-1`.

Что понадобится

- 1× Arduino Leonardo
- 1× GSM/GPRS Shield A9
- 1× Кабель micro-USB

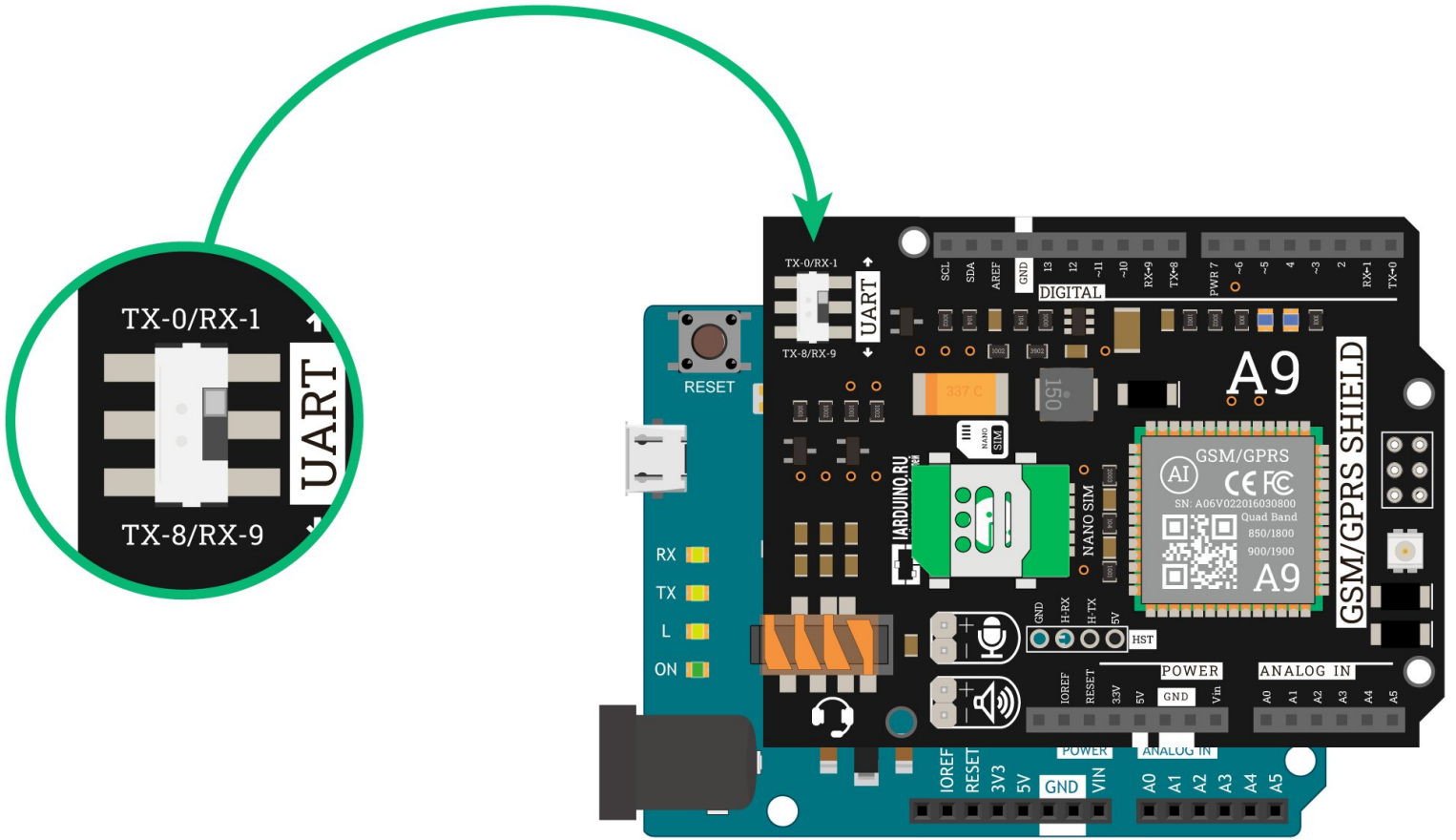
Таблица сигналов

Контакт GSM/GPRS Shield A9	Контакт Arduino Leonardo
TX-0	RX-0
RX-1	TX-1

PWR	7

Схема устройства

1. Вставьте SIM-карту в GSM/GPRS Shield A9.
2. Установите GSM/GPRS Shield A9 сверху на управляющую платформу Arduino Leonardo.
3. Установите переключатель шины UART в положение `TX-0/RX-1` .



Код инициализации

```
/*
*  Код инициализации GSM/GPRS Shield A9 с платами Arduino Leonardo
*/

// Подключаем библиотеку iarduino_GSM
#include <iarduino_GSM.h>

// Назначаем GPIO пины для связи с GSM/GPRS Shield A9
// Пин включения GSM/GPRS-модуля
constexpr int PIN_PWR = 7;

// Создаём объект gsm для работы с функциями библиотеки iarduino_GSM
// В параметрах указываем пин PWR
iarduino_GSM gsm(PIN_PWR);

void setup () {
  // Инициируем работу с GSM/GPRS Shield A9
  // В параметрах передаём объект Serial к которому подключён GSM/GPRS-модуль
  gsm.begin(Serial1);
}

void loop() {
}
```

GSM/GPRS A9 к Arduino Uno

На Arduino Uno и других платформах с микроконтроллером ATmega328, данные по USB и аппаратный интерфейс UART связаны между собой. Это не даёт возможность подключить GSM/GPRS Shield A9 к аппаратному UART платформы на пинах `RX-0` и `TX-1`. Выход есть — программный UART, который можно назначить на другие пины управляющей платы. В примере будем использовать программный UART на пинах `RX-8` и `TX-9`.

Что понадобится

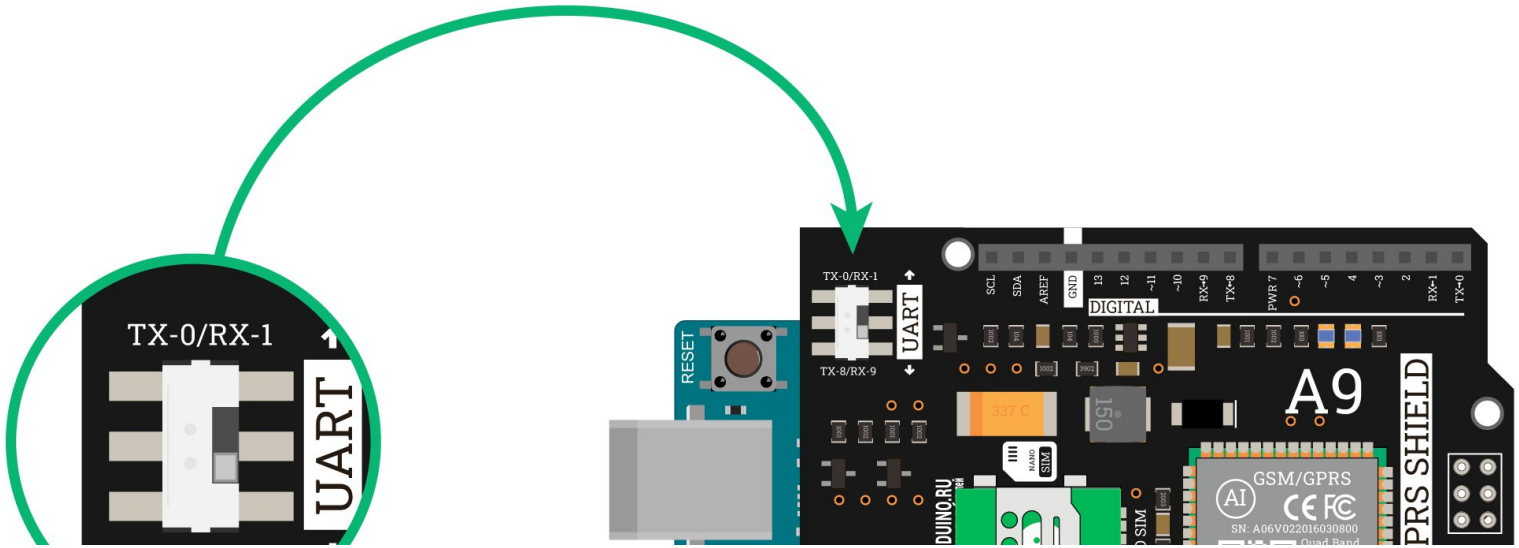
- 1× Arduino Uno
- 1× GSM/GPRS Shield A9
- 1× Кабель USB (A — B)

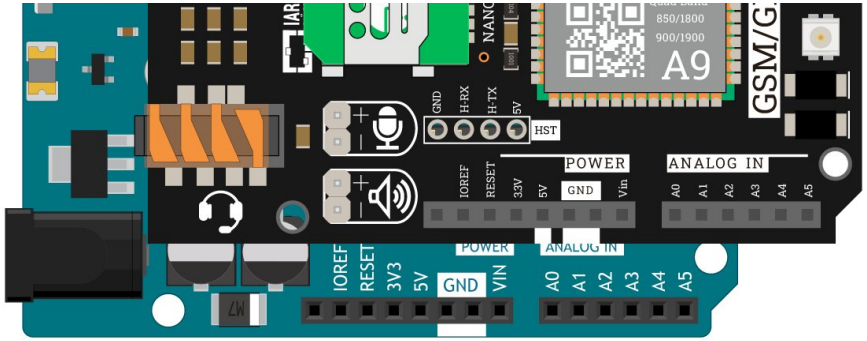
Таблица сигналов

Контакт GSM/GPRS Shield A9	Контакт Arduino Uno
TX-8	RX-8
RX-9	TX-9
PWR	7

Схема устройства

1. Вставьте SIM-карту в GSM/GPRS Shield A9.
2. Установите GSM/GPRS Shield A9 сверху на управляющую платформу Arduino Uno.
3. Установите переключатель шины UART в положение `TX-8/RX-9` .





Код инициализации

```
/*
*  Код инициализации GSM/GPRS Shield A9 с платами Arduino Uno
*/

// Подключаем библиотеку iarduino_GSM
#include <iarduino_GSM.h>
// Подключаем библиотеку SoftwareSerial
#include <SoftwareSerial.h>

// Назначаем GPIO пины для связи с GSM/GPRS Shield A9
// Пин включения GSM/GPRS-модуля
constexpr int PIN_PWR = 7;
// Пин приёма данных из контроллера в GSM/GPRS-модуль
constexpr int PIN_RX = 8;
// Пин передачи данных из контроллера в GSM/GPRS-модуль
constexpr int PIN_TX = 9;

// Создаём объект mySerial для работы с функциями библиотеки SoftwareSerial
// В параметрах указываем пины RX и TX
SoftwareSerial mySerial(PIN_RX, PIN_TX);
```

```
// Создаём объект gsm для работы с функциями библиотеки iarduino_GSM
// В параметрах указываем пин PWR
iarduino_GSM gsm(PIN_PWR);

void setup () {
  // Инициируем работу с GSM/GPRS Shield A9
  // В параметрах передаём объект Serial к которому подключён GSM/GPRS-модуль
  gsm.begin(mySerial);
}

void loop() {
}
```

GSM/GPRS A9 к Arduino Mega

На Arduino Mega и других платформах с микроконтроллером ATmega2560, данные по USB и аппаратный интерфейс UART связаны между собой. Это не даёт возможность подключить GSM/GPRS Shield A9 к аппаратному UART платформы на пинах `RX-0` и `TX-1`. Однако на платах форм-фактора Mega есть ещё дополнительно три аппаратных UART:

- UART1: RX-19 и TX-18
- UART2: RX-17 и TX-16
- UART3: RX-15 и TX-14

В примере будем использовать аппаратный UART3 на пинах `RX-15` и `TX-14`.

Что понадобится

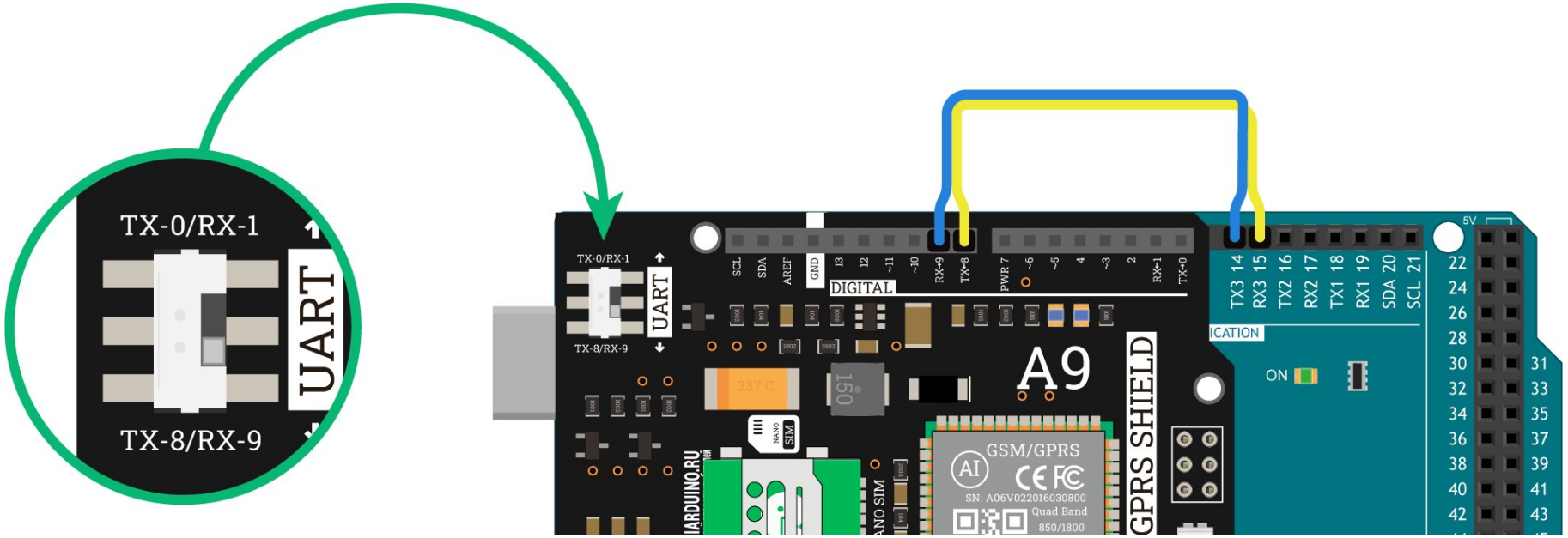
- 1× Arduino Mega 2560
- 1× GSM/GPRS Shield A9
- 1× Кабель USB (A — B)
- 1× Соединительные провода «папа-папа»

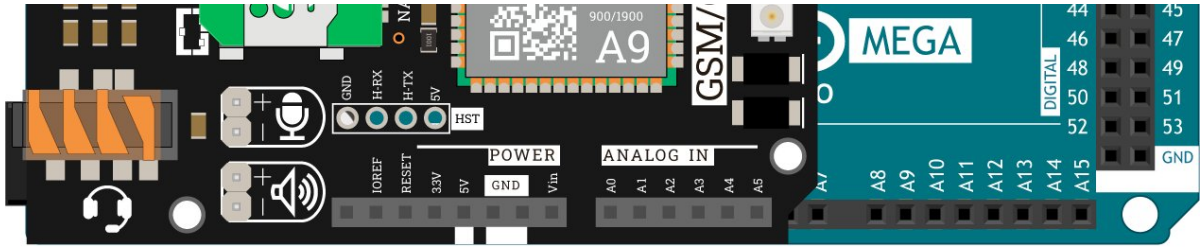
Таблица сигналов

Контакт GSM/GPRS Shield A9	Контакт Arduino Mega 2560
TX-8	RX-15
RX-9	TX-14
PWR	7

Схема устройства

1. Вставьте SIM-карту в GSM/GPRS Shield A9.
2. Установите GSM/GPRS Shield A9 сверху на управляющую платформу Arduino Mega 2560.
3. Установите переключатель шины UART в положение `TX-8/RX-9.`
4. Соедините контакты GSM/GPRS Shield A9 `TX-8` и `RX-9` с платой Arduino Mega с помощью проводов «папа-папа» согласно таблице сигналов.





Код инициализации

```
/*
*  Код инициализации GSM/GPRS Shield A9 с платами Arduino Mega
*/

// Подключаем библиотеку iarduino_GSM
#include <iarduino_GSM.h>

// Назначаем GPIO пины для связи с GSM/GPRS Shield A9
// Пин включения GSM/GPRS-модуля
constexpr int PIN_PWR = 7;

// Создаём объект gsm для работы с функциями библиотеки iarduino_GSM
// В параметрах указываем пин PWR
iarduino_GSM gsm(PIN_PWR);

void setup () {
  // Инициируем работу с GSM/GPRS Shield A9
  // В параметрах передаём объект Serial к которому подключён GSM/GPRS-модуль
  gsm.begin(Serial3);
}

void loop() {
}
```

GSM/GPRS A6 к Arduino Leonardo

На Arduino Leonardo и других платформах с микроконтроллером ATmega32U4, данные по USB и аппаратный интерфейс UART не связаны между собой. Это даёт возможность подключить GSM/GPRS Shield A6 к аппаратному UART платформы на пинах `RX-0` и `TX-1`.

Что понадобится

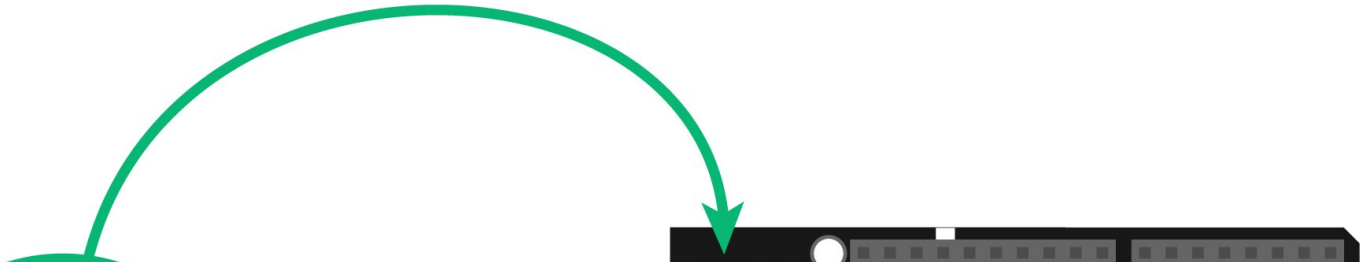
- 1× Arduino Leonardo
- 1× GSM/GPRS Shield A6
- 1× Кабель micro-USB

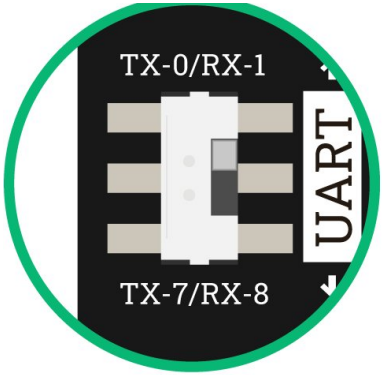
Таблица сигналов

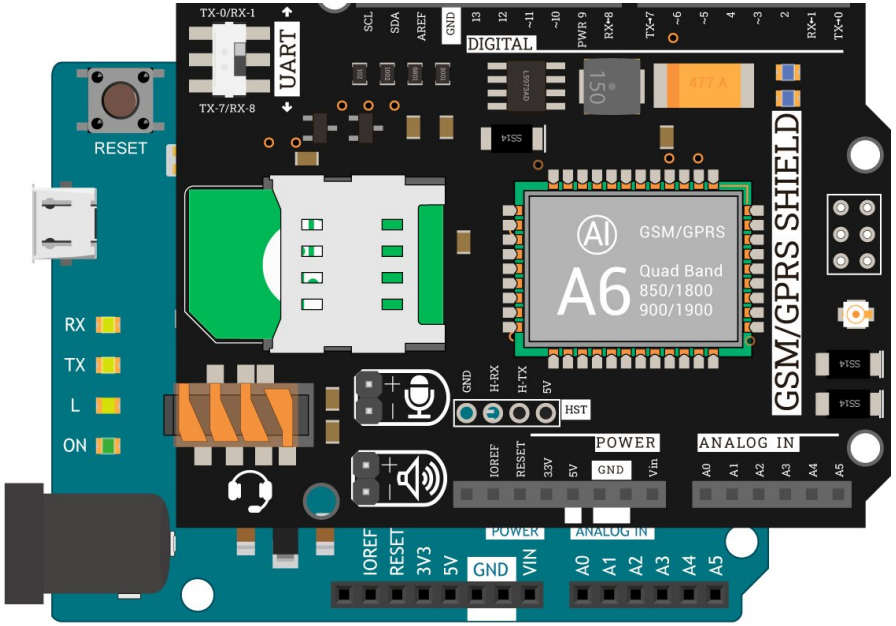
Контакт GSM/GPRS Shield A6	Контакт Arduino Leonardo
TX-0	RX-0
RX-1	TX-1
PWR	9

Схема устройства

1. Вставьте SIM-карту в GSM/GPRS Shield A6.
2. Установите GSM/GPRS Shield A6 сверху на управляющую платформу Arduino Leonardo.
3. Установите переключатель шины UART в положение `TX-0/RX-1`.







Код инициализации

```
/*
*  Код инициализации GSM/GPRS Shield A6 с платами Arduino Leonardo
*/

// Подключаем библиотеку iarduino_GSM
#include <iarduino_GSM.h>

// Назначаем GPIO пины для связи с GSM/GPRS Shield A6
// Пин включения GSM/GPRS-модуля
constexpr int PIN_PWR = 9;

// Создаём объект gsm для работы с функциями библиотеки iarduino_GSM
// В параметрах указываем пин PWR
iarduino_GSM gsm(PIN_PWR);
```

```
void setup () {
  // Инициируем работу с GSM/GPRS Shield A6
  // В параметрах передаём объект Serial к которому подключён GSM/GPRS-модуль
  gsm.begin(Serial1);
}

void loop() {
}
```

GSM/GPRS A6 к Arduino Uno

На Arduino Uno и других платформах с микроконтроллером ATmega328, данные по USB и аппаратный интерфейс UART связаны между собой. Это не даёт возможность подключить GSM/GPRS Shield A6 к аппаратному UART платформы на пинах `RX-0` и `TX-1`. Выход есть — программный UART, который можно назначить на другие пины управляющей платы. В примере будем использовать программный UART на пинах `RX-8` и `TX-9`.

Что понадобится

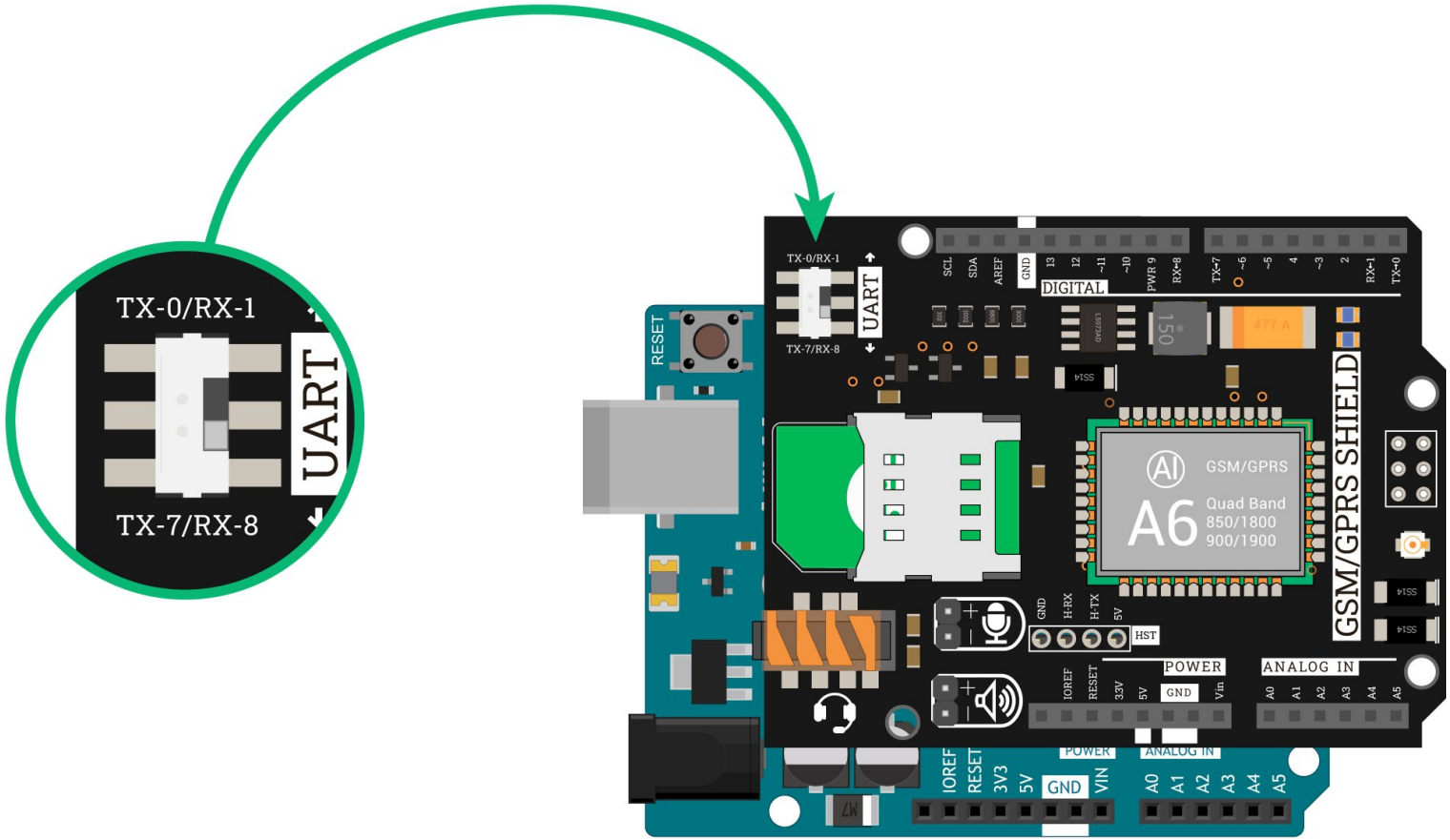
- 1× Arduino Uno
- 1× GSM/GPRS Shield A6
- 1× Кабель USB (A — B)

Таблица сигналов

Контакт GSM/GPRS Shield A6	Контакт Arduino Uno
TX-7	RX-7
RX-8	TX-8
PWR	9

Схема устройства

1. Вставьте SIM-карту в GSM/GPRS Shield A6.
2. Установите GSM/GPRS Shield A6 сверху на управляющую платформу Arduino Uno.
3. Установите переключатель шины UART в положение `TX-7/RX-8` .



Код инициализации

```
/*
*  Код инициализации GSM/GPRS Shield A6 с платами Arduino Uno
```

```
*/

// Подключаем библиотеку iarduino_GSM
#include <iarduino_GSM.h>
// Подключаем библиотеку SoftwareSerial
#include <SoftwareSerial.h>

// Назначаем GPIO пины для связи с GSM/GPRS Shield A6
// Пин включения GSM/GPRS-модуля
constexpr int PIN_PWR = 9;
// Пин приёма данных из контроллера в GSM/GPRS-модуль
constexpr int PIN_RX = 7;
// Пин передачи данных из контроллера в GSM/GPRS-модуль
constexpr int PIN_TX = 8;

// Создаём объект mySerial для работы с функциями библиотеки SoftwareSerial
// В параметрах указываем пины RX и TX
SoftwareSerial mySerial(PIN_RX, PIN_TX);

// Создаём объект gsm для работы с функциями библиотеки iarduino_GSM
// В параметрах указываем пин PWR
iarduino_GSM gsm(PIN_PWR);

void setup () {
  // Инициируем работу с GSM/GPRS Shield A6
  // В параметрах передаём объект Serial к которому подключён GSM/GPRS-модуль
  gsm.begin(mySerial);
}

void loop() {
}
```

GSM/GPRS A6 к Arduino Mega

На Arduino Mega и других платформах с микроконтроллером ATmega2560, данные по USB и аппаратный интерфейс UART связаны между собой. Это не даёт возможность подключить GSM/GPRS Shield A6 к аппаратному UART платформы на пинах `RX-0` и `TX-1`. Однако на платах форм-фактора Mega есть ещё дополнительно три аппаратных UART:

- UART1: RX-19 и TX-18
- UART2: RX-17 и TX-16
- UART3: RX-15 и TX-14

В примере будем использовать аппаратный UART3 на пинах `RX-15` и `TX-14`.

Что понадобится

- 1× Arduino Mega 2560
- 1× GSM/GPRS Shield A6
- 1× Кабель USB (A — B)
- 1× Соединительные провода «папа-папа»

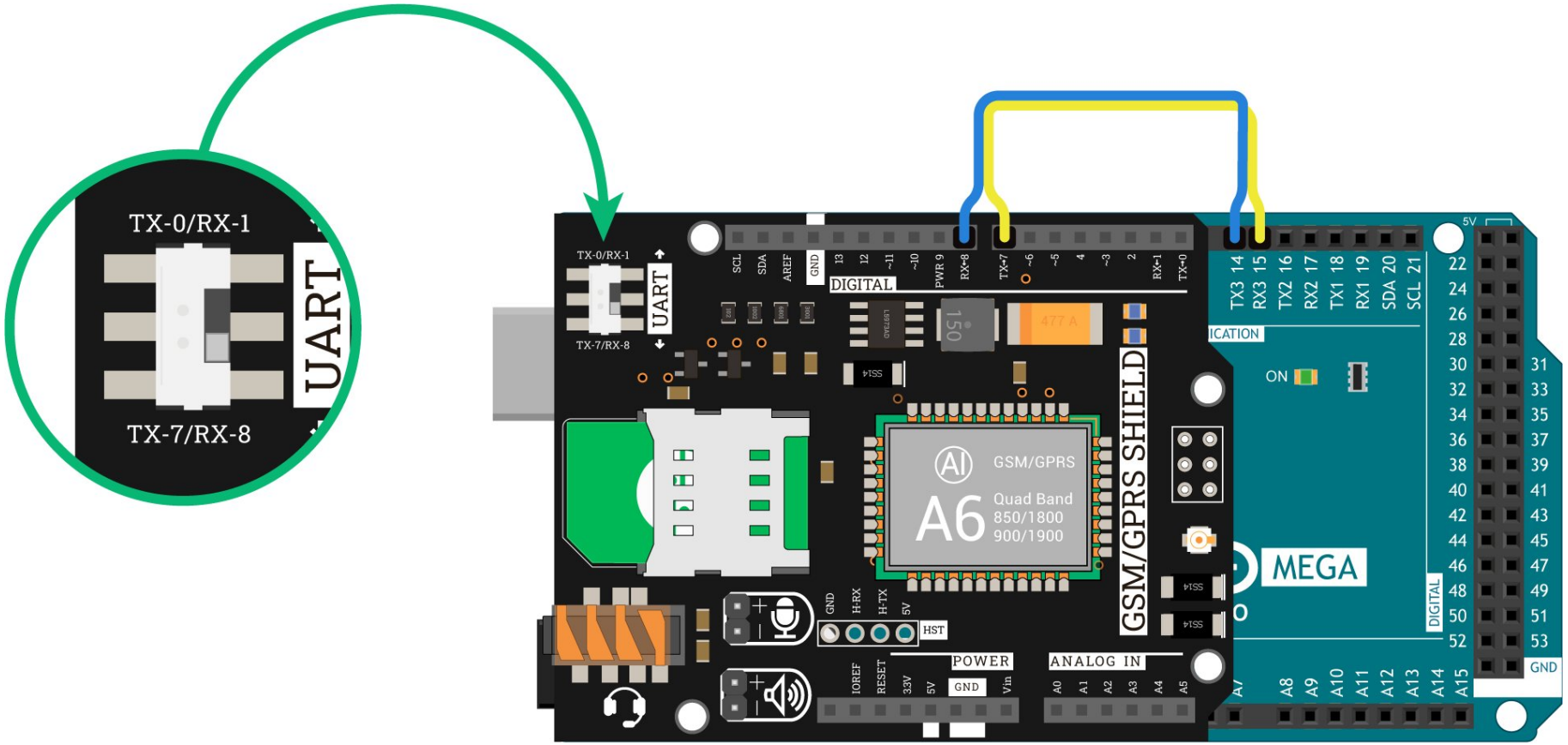
Таблица сигналов

Контакт GSM/GPRS Shield A6	Контакт Arduino Mega 2560
TX-7	RX-15
RX-8	TX-14
PWR	9

Схема устройства

1. Вставьте SIM-карту в GSM/GPRS Shield A6.
2. Установите GSM/GPRS Shield A6 сверху на управляющую платформу Arduino Mega 2560.

3. Установите переключатель шины UART в положение `TX-7/RX-8` .
4. Соедините контакты GSM/GPRS Shield A6 `TX-7` и `RX-8` с платой Arduino Mega с помощью проводов «папа-папа» согласно таблице сигналов.



Код инициализации

```
/*
*  Код инициализации GSM/GPRS Shield A6 с платами Arduino Mega
*/

// Подключаем библиотеку iarduino_GSM
```

```
#include <iarduino_GSM.h>

// Назначаем GPIO пины для связи с GSM/GPRS Shield A6
// Пин включения GSM/GPRS-модуля
constexpr int PIN_PWR = 9;

// Создаём объект gsm для работы с функциями библиотеки iarduino_GSM
// В параметрах указываем пин PWR
iarduino_GSM gsm(PIN_PWR);

void setup () {
  // Инициируем работу с GSM/GPRS Shield A6
  // В параметрах передаём объект Serial к которому подключён GSM/GPRS-модуль
  gsm.begin(Serial3);
}

void loop() {
}
```

Примеры работы

Приведённые ниже примеры написаны для работы GSM/GPRS Shield версии A9 с контроллером Arduino Leonardo. Если у вас другой шилд, например GSM/GPRS Shield версии A6 или другая Arduino, например Arduino Uno или Arduino Mega, необходимо изменить инициализацию в начале кода программы. Все базовые варианты мы рассмотрели в разделе подключение и настройка.

Запрос баланса

Для начала запросим баланс на установленной сим-карте. Баланс запрашивается методом `runUSSD()` с параметром USSD команды в виде последовательности символов. Для оператора МТС команда запроса баланса — `*100#"` . Комбинацию символов для запроса баланса других мобильных операторов можно найти при серфинге в интернете. Метод `runUSSD()` возвращает ответ в текстовом виде.

```
/*
```

```
*  Код запроса баланса
*  для GSM/GPRS Shield A9 с контроллером Arduino Leonardo
*  Если у вас другой шилд или контроллер,
*  необходимо изменить инициализацию в начале кода программы
*  Подробности: https://wiki.iarduino.ru/page/gsm-gprs-shield
*/

// Подключаем библиотеку iarduino_GSM
#include <iarduino_GSM.h>

// Назначаем GPIO пины для связи с GSM/GPRS Shield A9
// Пин включения GSM/GPRS-модуля
constexpr int PIN_PWR = 7;

// Создаём объект gsm для работы с функциями библиотеки iarduino_GSM
// В параметрах указываем пин PWR
iarduino_GSM gsm(PIN_PWR);

void setup () {
  // Инициируем работу с GSM/GPRS Shield A9
  // В параметрах передаём объект Serial к которому подключён GSM/GPRS-модуль
  gsm.begin(Serial1);
  // Открываем консоль
  Serial.begin(9600);
  // Выводим текст
  Serial.print("Initialization, please wait...");

  // Ждём готовность GSM/GPRS Shield к работе
  while (gsm.status() != GSM_OK) {
    Serial.print(".");
    delay(1000);
  }
  Serial.println("OK!");
```

```
  // Отправка USSD запроса *100# с выводом ответа в монитор последовательного порта
  Serial.println("Sending USSD request balance...");
  Serial.print("Answer: ");
  Serial.println(gsm.runUSSD("*100#"));
}

void loop() {
}
```

> Многие Российские операторы связи придерживаются правила, что ответ на команду начинающуюся с символа `*` возвращается на Кириллице, а при замене `*` на `#` ответ возвращается на Латинице. Например, команда `#100#` вернёт баланс на Латинице.

Отправка короткого SMS сообщения

Отправим короткое сообщение на мобильный телефон. Длина SMS зависит от используемой кодировки:

- Латиница: 160 символов
- Кириллица: 70 символов

Отправка SMS сообщения осуществляется методом `SMSsend()` с двумя параметрами: текст сообщения и номер телефона. Метод `SMSsend()` возвращает результат отправки `true` или `false`.

> Измените номер получателя SMS перед загрузкой скетча.

```
/*
*  Код отправки короткого SMS
*  для GSM/GPRS Shield A9 с контроллером Arduino Leonardo
```

```
*  Если у вас другой шилд или контроллер,
*  необходимо изменить инициализацию в начале кода программы
*  Подробности: https://wiki.iarduino.ru/page/gsm-gprs-shield
*/

// Подключаем библиотеку iarduino_GSM
#include <iarduino_GSM.h>

// Назначаем GPIO пины для связи с GSM/GPRS Shield A9
// Пин включения GSM/GPRS-модуля
constexpr int PIN_PWR = 7;

// Назначаем номер получателя SMS без знака +
// Измените предложенный номер на номер вашего получателя
String phone = "+79684541738";

// Создаём объект gsm для работы с функциями библиотеки iarduino_GSM
// В параметрах указываем пин PWR
iarduino_GSM gsm(PIN_PWR);


void setup (){
  // Инициируем работу с GSM/GPRS Shield A9
  // В параметрах передаём объект Serial к которому подключён GSM/GPRS-модуль
  gsm.begin(Serial1);
  // Открываем консоль
  Serial.begin(9600);
  // Выводим текст
  Serial.print("Initialization, please wait...");

  // Ждём готовность GSM/GPRS Shield к работе
  while (gsm.status() != GSM_OK) {
    Serial.print(".");
```

```
    delay(1000);
  }
  Serial.println("OK!");

  // Установка кодировки для символов Кириллицы
  gsm.TXTsendCodingDetect("п");

  // Отправка SMS сообщения
  // Выводим результат отправки SMS сообщения
  if (gsm.SMSsend("Это короткое SMS сообщение.", phone)) {
    Serial.println("SMS Sent!");
  } else {
    Serial.println("SMS Error!");
  }
}

void loop () {
}
```

Наиболее частой причиной возникновения ошибки при отправке SMS является плохая связь с оператором связи. В таких случаях рекомендуем сделать несколько попыток отправки. Для этого измените часть кода для отправки SMS сообщения:

```
// Отправка SMS сообщения в условиях плохой связи
// Выводим результат отправки SMS сообщения
// Цикл попыток отправки SMS сообщения на 10 попыток
for (int i = 0; i < 10; i++){
  if (gsm.SMSsend("Это короткое SMS сообщение.", phone)) {
    Serial.println("SMS Sent!");
    break;
  } else {
    Serial.println("SMS Error!");
  }
```

```
  // Запрашиваем статус связи после каждой попытки отправки SMS сообщения
  gsm.status();
}
```

Отправка длинного SMS сообщения

Не всегда всю информацию можно уместить в одно короткое сообщение. Для этого придумали длинные или так называемые составные сообщения, которые просто склеиваются из коротких в одно целое. Когда мы пишем SMS в телефоне и вылазим за диапазон символов одного SMS, мобильный редактор автоматически переводит текст во второе сообщение, далее в третье и т.д. На стороне получателя письмо приходит в уже готовом склеенном виде. Однако с GSM/GPRS Shield составное сообщение необходимо составлять в явном виде, а на стороне получателя письмо также приходит в уже готовом склеенном виде.

Отправка длинного SMS сообщения осуществляется методом `SMSsend()` с пятью параметрами: текст сообщения, номер телефона, идентификатор всего сообщения, количество SMS в сообщении и порядковый номер данной SMS в сообщении. Метод `SMSsend()` возвращает результат отправки `true` или `false`.

Измените номер получателя SMS перед загрузкой скетча.

```
/*
*  Код отправки составного SMS
*  для GSM/GPRS Shield A9 с контроллером Arduino Leonardo
*  Если у вас другой шилд или контроллер,
*  необходимо изменить инициализацию в начале кода программы
*  Подробности: https://wiki.iarduino.ru/page/gsm-gprs-shield
*/

// Подключаем библиотеку iarduino_GSM
#include <iarduino_GSM.h>

// Назначаем GPIO пины для связи с GSM/GPRS Shield A9
```

```ino
// Пин включения GSM/GPRS-модуля
constexpr int PIN_PWR = 7;

// Назначаем номер получателя SMS
// Измените предложенный номер на номер вашего получателя
String phone = "+79684541738";

// Создаём объект gsm для работы с функциями библиотеки iarduino_GSM
// В параметрах указываем пин PWR
iarduino_GSM gsm(PIN_PWR);

void setup (){
  // Инициируем работу с GSM/GPRS Shield A9
  // В параметрах передаём объект Serial к которому подключён GSM/GPRS-модуль
  gsm.begin(Serial1);
  // Открываем консоль
  Serial.begin(9600);
  // Выводим текст
  Serial.print("Initialization, please wait...");

  // Ждём готовность GSM/GPRS Shield к работе
  while (gsm.status() != GSM_OK) {
    Serial.print(".");
    delay(1000);
  }
  Serial.println("OK!");

  // Установка кодировки для символов Кириллицы
  gsm.TXTsendCodingDetect("n");

  // Отправляем SMS1 составного SMS сообщения
  if (gsm.SMSsend("Это длинное SMS сообщение, состоящее из двух", phone, 0x1234, 2, 1)) {
    Serial.println("SMS 1/2 Sent!");
```

```
  } else {
    Serial.println("SMS 1/2 Error!" );
  }
  // Отправляем SMS2 составного SMS сообщения
  if (gsm.SMSsend(" SMS, которые придут получателю одним сообщением", phone, 0x1234, 2, 2)) {
    Serial.println("SMS 2/2 Sent!");
  } else {
    Serial.println("SMS 2/2 Error!" );
  }
}

void loop () {
}
```

Получение короткого SMS сообщения

А теперь попробуем получить короткое текстовое сообщение. Чтение SMS осуществляется методом `SMSread()` с тремя параметрами: текст SMS сообщения, номер отправителя и дата отправки. Метод `SMSread()` возвращает результат чтения `true` или `false`.

```
/*
*  Код получения короткого SMS
*  для GSM/GPRS Shield A9 с контроллером Arduino Leonardo
*  Если у вас другой шилд или контроллер,
*  необходимо изменить инициализацию в начале кода программы
*  Подробности: https://wiki.iarduino.ru/page/gsm-gprs-shield
*/

// Подключаем библиотеку iarduino_GSM
#include <iarduino_GSM.h>

// Назначаем GPIO пины для связи с GSM/GPRS Shield A9
// Пин включения GSM/GPRS-модуля
```

```
constexpr int PIN_PWR = 7;

// Создаём объект gsm для работы с функциями библиотеки iarduino_GSM
// В параметрах указываем пин PWR
iarduino_GSM gsm(PIN_PWR);

// Переменная для хранения текста принятого SMS сообщения
char SMStxt[161];
// Переменная для хранения номера отправителя SMS сообщения
char SMSnum[13];
// Переменная для хранения даты и времени отправки SMS сообщения
char SMStim[18];

void setup (){
  // Инициируем работу с GSM/GPRS Shield A9
  // В параметрах передаём объект Serial к которому подключён GSM/GPRS-модуль
  gsm.begin(Serial1);
  // Открываем консоль
  Serial.begin(9600);
  // Выводим текст
  Serial.print("Initialization, please wait...");

  // Ждём готовность GSM/GPRS Shield к работе
  while (gsm.status() != GSM_OK) {
    Serial.print(".");
    delay(1000);
  }
  Serial.println("OK!");
}

void loop () {
  // Если есть входящие непрочитанные SMS сообщения
  if(gsm.SMSavailable()) {
```

```
    Serial.println("There are new SMS." );
    // Читаем SMS сообщение, а точнее считываем данные в переменные:
    // текст SMS сообщения, номер отправителя, дату отправки
    gsm.SMSread(SMStxt, SMSnum, SMStim);
    // Выводим дату отправки, номер отправителя и текст SMS
    Serial.print("Data: ");
    Serial.println(SMStim);
    Serial.print("Number: ");
    Serial.println(SMSnum);
    Serial.print("Text SMS: ");
    Serial.println(SMStxt);
  }
}
```

При получении длинного сообщения из нескольких SMS, данный код не сможет понять, что эти SMS относятся к одному сообщению. Для правильного получения составных сообщений используйте пример «Получение длинного SMS сообщения».

Получение длинного SMS сообщения

Чтение составного SMS осуществляется методом `SMSread()` с шестью параметрами: текст SMS сообщения, номер отправителя, дата отправки, идентификатор SMS, количество SMS и порядковый номер SMS. Метод `SMSread()` возвращает результат чтения `true` или `false` .

```
/*
*  Код получения составного SMS
*  для GSM/GPRS Shield A9 с контроллером Arduino Leonardo
*  Если у вас другой шилд или контроллер,
*  необходимо изменить инициализацию в начале кода программы
*  Подробности: https://wiki.iarduino.ru/page/gsm-gprs-shield
```

```
*/

// Подключаем библиотеку iarduino_GSM
#include <iarduino_GSM.h>

// Назначаем GPIO пины для связи с GSM/GPRS Shield A9
// Пин включения GSM/GPRS-модуля
constexpr int PIN_PWR = 7;

// Создаём объект gsm для работы с функциями библиотеки iarduino_GSM
// В параметрах указываем пин PWR
iarduino_GSM gsm(PIN_PWR);

// Переменная для хранения текста принятого SMS сообщения
char SMStxt[161];
// Переменная для хранения номера отправителя SMS сообщения
char SMSnum[13];
// Переменная для хранения даты и времени отправки SMS сообщения
char SMStim[18];
// Переменная для хранения идентификатора составного SMS сообщения
uint16_t SMSlongID;
// Переменная для хранения количества SMS в составном сообщении
uint8_t SMSlongSUM;
// Переменная для хранения номера SMS в составном сообщении
uint8_t SMSlongNUM;

void setup (){
  // Инициируем работу с GSM/GPRS Shield A9
  // В параметрах передаём объект Serial к которому подключён GSM/GPRS-модуль
  gsm.begin(Serial1);
  // Открываем консоль
  Serial.begin(9600);
  // Выводим текст
```

```
  Serial.print("Initialization, please wait...");

  // Ждём готовность GSM/GPRS Shield к работе
  while (gsm.status() != GSM_OK) {
    Serial.print(".");
    delay(1000);
  }
  Serial.println("OK!");
}

void loop () {
  // Если есть входящие непрочитанные SMS сообщения
  if(gsm.SMSavailable()) {
    Serial.println("There are new SMS." );
    // Читаем SMS сообщение, а точнее считываем данные в переменные:
    // текст SMS сообщения, номер отправителя, дату отправки
    // идентификатор SMS, количество SMS и порядковый номер SMS
    gsm.SMSread(SMStxt, SMSnum, SMStim, SMSlongID, SMSlongSUM, SMSlongNUM);
    // Выводим порядковый номер SMS, количество SMS, идентификатор SMS
    // дату отправки, номер отправителя и текст SMS
    Serial.print("SMS:");
    Serial.print(SMSlongNUM);
    Serial.print("/");
    Serial.println(SMSlongSUM);
    Serial.print("ID: ");
    Serial.println(SMSlongID);
    Serial.print("Data: ");
    Serial.println(SMStim);
    Serial.print("Number: ");
    Serial.println(SMSnum);
    Serial.print("Text SMS: ");
    Serial.println(SMStxt);
  }
```

```
}
```

Совершение исходящего вызова

Хватит с нас текста и сообщений, совершим исходящий голосовой вызов. Исходящий вызов совершается методом `CALLdial` с параметром номера вызываемого абонента. Для передачи и приёма голоса подключите к GSM/GPRS Shield гарнитуру или внешний динамик с микрофоном.

> Измените номер вызываемого абонента перед загрузкой скетча.

```
/*
*  Код совершения исходящего вызова
*  для GSM/GPRS Shield A9 с контроллером Arduino Leonardo
*  Если у вас другой шилд или контроллер,
*  необходимо изменить инициализацию в начале кода программы
*  Подробности: https://wiki.iarduino.ru/page/gsm-gprs-shield
*/

// Подключаем библиотеку iarduino_GSM
#include <iarduino_GSM.h>

// Назначаем GPIO пины для связи с GSM/GPRS Shield A9
// Пин включения GSM/GPRS-модуля
constexpr int PIN_PWR = 7;

// Создаём объект gsm для работы с функциями библиотеки iarduino_GSM
// В параметрах указываем пин PWR
iarduino_GSM gsm(PIN_PWR);

// Назначаем номер абонента для совершения исходящего звонка
```

```
// Измените предложенный номер на номер вашего получателя
String phone = "+79684541738";

void setup () {
  // Инициируем работу с GSM/GPRS Shield A9
  // В параметрах передаём объект Serial к которому подключён GSM/GPRS-модуль
  gsm.begin(Serial1);
  // Открываем консоль
  Serial.begin(9600);
  // Выводим текст
  Serial.print("Initialization, please wait...");

  // Ждём готовность GSM/GPRS Shield к работе
  while (gsm.status() != GSM_OK) {
    Serial.print(".");
    delay(1000);
  }
  Serial.println("OK!");

  // Совершаем исходящий вызов на указанный номер
  if (gsm.CALLdial(phone)) {
    Serial.println("Dialing...");
  } else {
    Serial.println("Error!");
  }
}

void loop() {
}
```

Ответ на входящий вызов

После пройденных уроков, вы услышите звонок. Для передачи и приёма голоса подключите к GSM/GPRS Shield гарнитуру или внешний

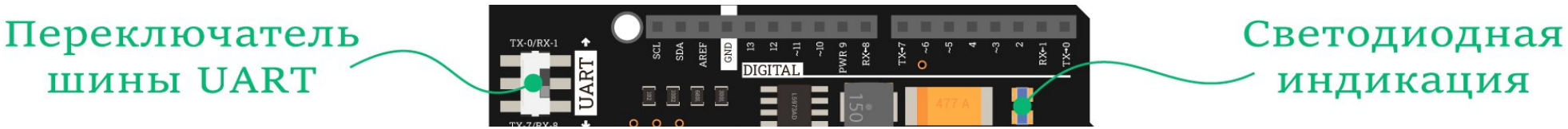
динамик с микрофоном.

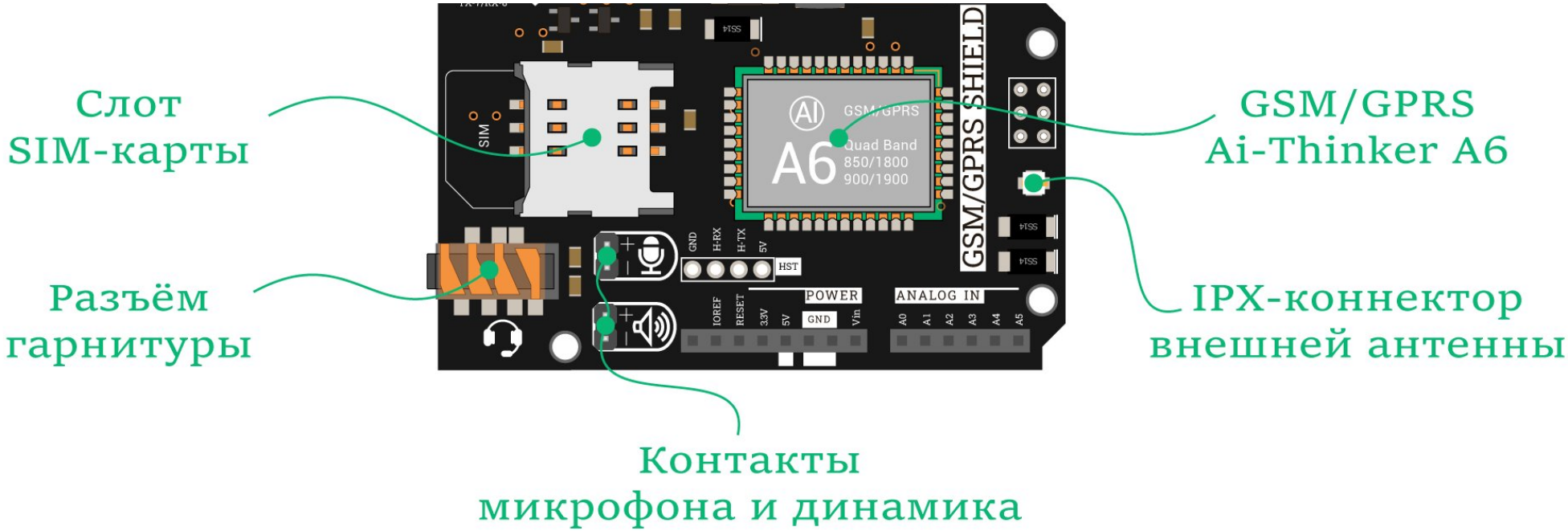
```
/*
*  Код ответа входящего вызова
*  для GSM/GPRS Shield A9 с контроллером Arduino Leonardo
*  Если у вас другой шилд или контроллер,
*  необходимо изменить инициализацию в начале кода программы
*  Подробности: https://wiki.iarduino.ru/page/gsm-gprs-shield
*/

// Подключаем библиотеку iarduino_GSM
#include <iarduino_GSM.h>

// Назначаем GPIO пины для связи с GSM/GPRS Shield A9
// Пин включения GSM/GPRS-модуля
constexpr int PIN_PWR = 7;

// Создаём объект gsm для работы с функциями библиотеки iarduino_GSM
// В параметрах указываем пин PWR
iarduino_GSM gsm(PIN_PWR);

// Переменная для хранения номера входящего абонента
char phone[13];

void setup() {
  // Инициируем работу с GSM/GPRS Shield A9
  // В параметрах передаём объект Serial к которому подключён GSM/GPRS-модуль
  gsm.begin(Serial1);
  // Открываем консоль
  Serial.begin(9600);
  // Выводим текст
  Serial.print("Initialization, please wait...");
```

```
  // Ждём готовность GSM/GPRS Shield к работе
  while (gsm.status() != GSM_OK) {
    Serial.print(".");
    delay(1000);
  }
  Serial.println("OK!");
}

void loop() {
  // Если есть входящий вызов
  if (gsm.CALLavailable()) {
    // Выводим номер звонящего
    Serial.print("Incoming call from ");
    Serial.println(phone);
    // Отвечаем на вызов
    gsm.CALLup();
    // Ждём перехода входящего вызова в разряд активных
    // Это не даст многократно ответить на один и тот же входящий вызов
    // Цикл выполняется пока есть входящий дозванивающийся вызов
    while(gsm.CALLavailable());
    // Проверяем наличие входящих звонков один раз в секунду
    delay(1000);
  }
}
```

Элементы платы

Элементы GSM/GPRS Shield A6





GSM/GPRS Ai-Thinker A6

Плата GSM/GPRS Shield A6 выполнена на чипе Ai-Thinker A6 с беспроводными стандартами связи 2G GSM/GPRS.

Слот SIM-карты

Для установки SIM-карты на плате расположен слот под карту формата nano SIM (4FF). Если у вас сим-карта другого размера, например mini SIM или micro SIM, воспользуйтесь специальными переходниками в магазинах мобильной связи.

Индикаторные светодиоды

Имя	Цвет	Назначение
TX	Синий	Мигает при отправлении данных из GSM/GPRS-модуля в контроллер.
RX	Синий	Мигает при получении данных из контроллера в GSM/GPRS-модуль.

Разъём гарнитуры

Для передачи и приёма звука, на плате предусмотрен разъём под гарнитуру Jack TRRS 3,5 мм. Вставьте наушники с микрофоном в соответствующее гнездо и вы сможете общаться с абонентом аналогично обычному телефону.

По умолчанию, при совершении голосовых звонков, аудиопоток выведен на громкую связь через разъёмы PLS для микрофона и динамика. Перевод звука между громкой связью и гарнитурой осуществляется программно методом `SOUNDdevice` из библиотеки iarduino_GSM. Подробности смотрите в описании библиотеки.

Разъём микрофона и динамика

Для передачи и приёма звука в режиме громкой связи, на плате предусмотрено два разъема PLS-2 для подключения динамика и микрофона соответственно.

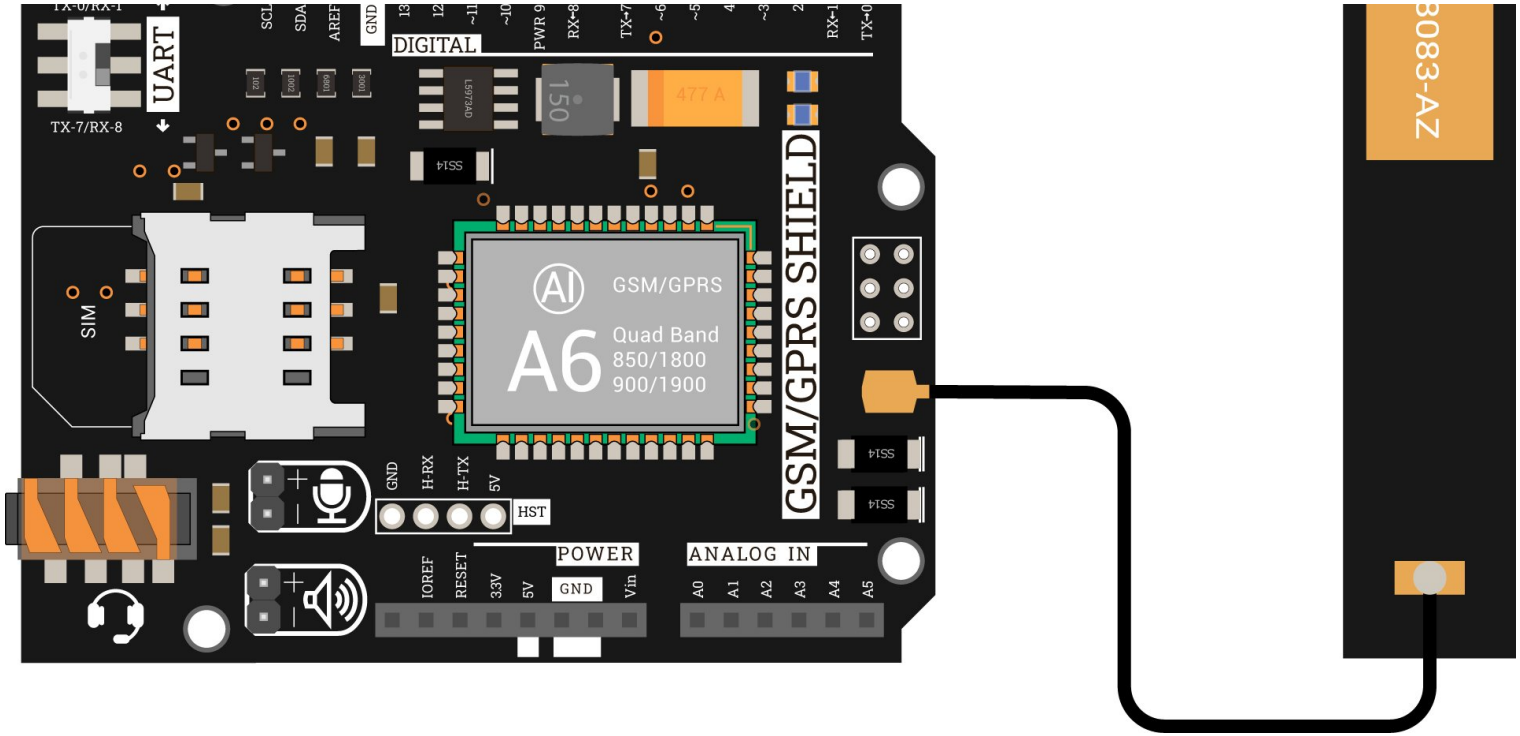
Переключатель шины UART

GSM/GPRS Shield A6 общается с внешними платформами по интерфейсу UART через контакты `TX` и `RX`. Переключатель шины UART позволяет выбрать две разных группы пинов:

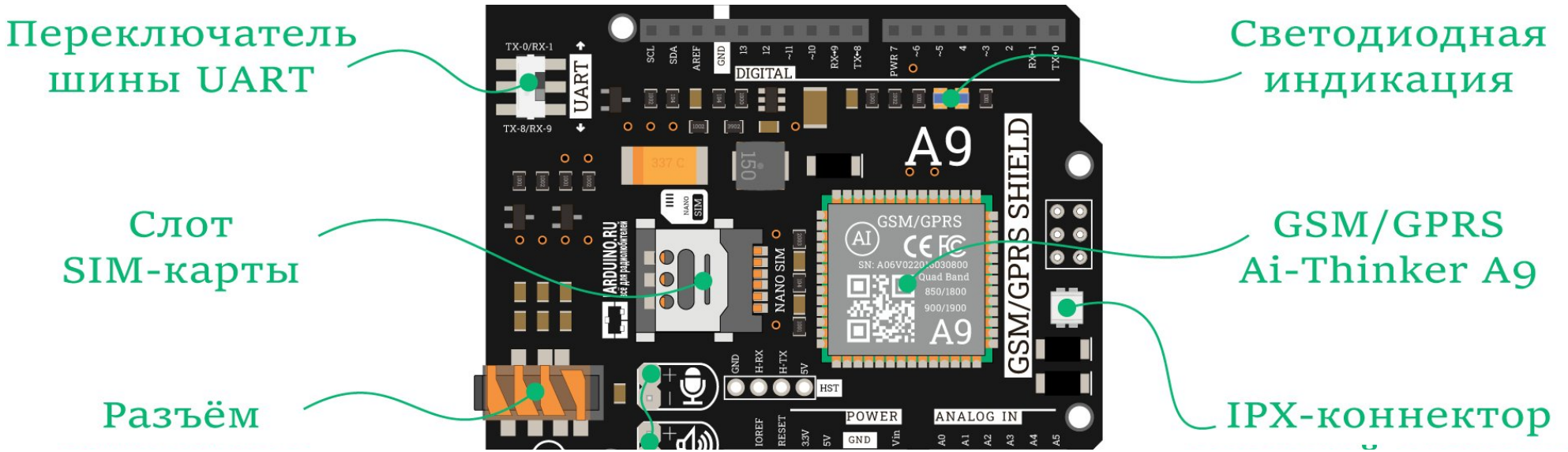
- TX-0/RX-1: линии `TX` и `RX` выведены на контакты `0` и `1` платы GSM/GPRS Shield A6.
- TX-7/RX-8: линии `TX` и `RX` выведены на контакты `7` и `8` платы GSM/GPRS Shield A6.

IPX-коннектор внешней антенны

Для качественного приёма сигнала мы развели внутреннюю антенну PSB на обратной стороне платы. А если вы находитесь далеко от вышки мобильного оператора, на плате расположен разъём IPX UFL для подключения внешней антенны.



Элементы GSM/GPRS Shield A9



GSM/GPRS Ai-Thinker A9

Плата GSM/GPRS Shield A9 выполнена на чипе Ai-Thinker A9 с беспроводными стандартами связи 2G GSM/GPRS.

Слот SIM-карты

Для установки SIM-карты на плате расположен слот под карту формата mini SIM (2FF). Если у вас сим-карта другого размера, например micro SIM или nano SIM, воспользуйтесь специальными инструментами для подрезки SIM-карт в магазинах мобильной связи.

Индикаторные светодиоды

Имя	Цвет	Назначение
TX	Синий	Мигает при отправлении данных из GSM/GPRS-модуля в контроллер.
RX	Синий	Мигает при получении данных из контроллера в GSM/GPRS-модуль.

Разъём гарнитуры

Для передачи и приёма звука, на плате предусмотрен разъём под гарнитуру Jack TRRS 3,5 мм. Вставьте наушники с микрофоном в соответствующее гнездо и вы сможете общаться с абонентом аналогично обычному телефону.

По умолчанию, при совершении голосовых звонков, аудиопоток выведен на громкую связь через разъёмы PLS для микрофона и динамика. Перевод звука между громкой связью и гарнитурой осуществляется программно методом `SOUNDdevice` из библиотеки iarduino_GSM. Подробности смотрите в описании библиотеки.

Разъём микрофона и динамика

Для передачи и приёма звука в режиме громкой связи, на плате предусмотрено два разъема PLS-2 для подключения динамика и микрофона соответственно.

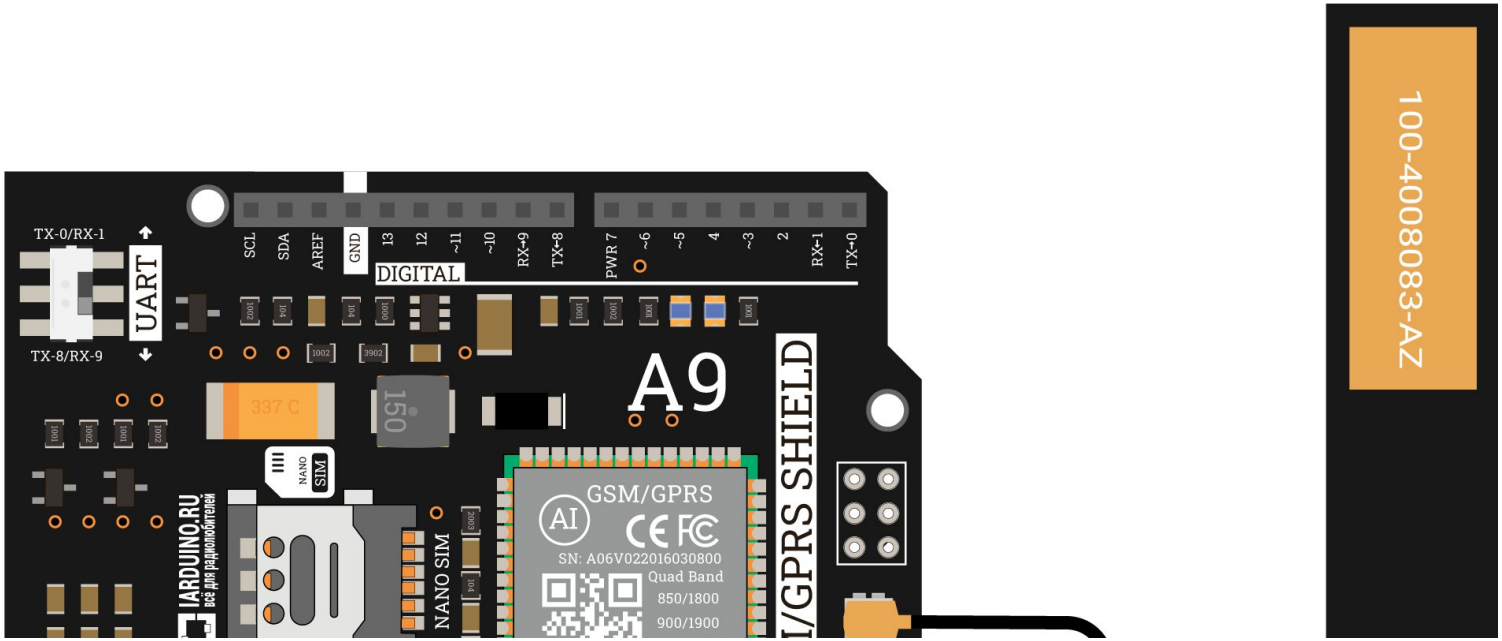
Переключатель шины UART

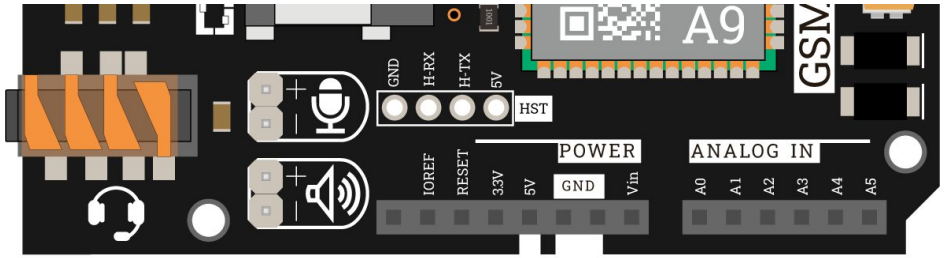
GSM/GPRS Shield A9 общается с внешними платформами по интерфейсу UART через контакты `TX` и `RX`. Переключатель шины UART позволяет выбрать две разных группы пинов:

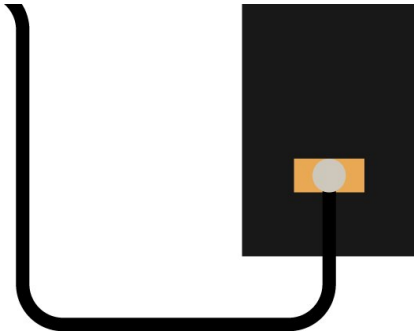
- TX-0/RX-1: линии `TX` и `RX` выведены на контакты `0` и `1` платы GSM/GPRS Shield A9.
- TX-8/RX-9: линии `TX` и `RX` выведены на контакты `8` и `9` платы GSM/GPRS Shield A9.

IPX-коннектор внешней антенны

Для качественного приёма сигнала мы развели внутреннюю антенну PSB на обратной стороне платы. А если вы находитесь далеко от вышки мобильного оператора, на плате расположен разъём IPX UFL для подключения внешней антенны.







Библиотека для Arduino

GSM/GPRS Shield общается с Arduino через интерфейс UART по протоколу AT-команд. Однако вы можете не вникать в детали программного управления, используйте готовые библиотеки для работы с модулем:

- Библиотека iarduino_GSM служит для работы со звонками и СМС. Подробности и описание методов програмного модуля `iarduino_GSM` читайте ниже.
- Библиотека iarduino_GprsClientA9 служит для работы с мобильным интернетом. Подробности и описание методов програмного модуля `iarduino_GprsClientA9` читайте в отдельной инструкции по работе модуля в сети интернет.

Установка

Для старта скачайте и установите библиотеку iarduino_GSM. Для инсталляции рекомендуем использовать нашу инструкцию по установке библиотек для Arduino.

Подключение

- Назначение: подключение библиотеки.
- Синтаксис: `#include <iarduino_GSM.h>`
- Примечания:
  - Библиотека подключается в самом начале программы.
  - Подключение библиотеки обязательное действие, иначе её методы работать не будут.
  - В библиотеках функции называются методами.
- Примеры:

```
// Подключаем библиотеку для работы с GSM/GPRS Shield
#include <iarduino_GSM.h>
```

Конструктор

- Назначение: создание объекта для работы с методами библиотеки `iarduino_GSM`
- Синтаксис: `iarduino_GSM tds(uint8_t pinPWR)`
- Параметры:
  - `pinPWR` : пин включения модуля.
    - Для GSM/GPRS Shield A6 указывайте пин `9`
    - Для GSM/GPRS Shield A9 указывайте пин `7`
- Возвращаемое значение: нет
- Примечания:
  - Конструктор вызывается в самом начале программы.
  - Вызов конструктор обязателен, иначе методы библиотеки работать не будут.
- Пример:

Конструктор для GSM/GPRS Shield A6

```
// Создаём объект gsm для работы с функциями библиотеки iarduino_GSM
// В параметре конструктора передаём пин включения модуля
iarduino_I2C_TDS tds(9);
```

Конструктор для GSM/GPRS Shield A9

```
// Создаём объект gsm для работы с функциями библиотеки iarduino_GSM
// В параметре конструктора передаём пин включения модуля
iarduino_I2C_TDS tds(7);
```

Метод begin()

- Назначение: инициализация работы с модулем.
- Синтаксис: `bool begin(Serial)`
- Параметры:
  - `Serial` : объект или класс используемый для работы с интерфейсом UART.
- Возвращаемое значение:
  - `true` : инициализация модуля прошла успешно.
  - `false` : инициализация модуля прошла не успешно.
- Примечания:
  - Метод достаточно вызывать один раз в функции `setup()` .
  - Serial может быть аппаратный или программный. Для программного Serial необходимо использовать библиотеку `SoftwareSerial` .
- Пример: все базовые варианты мы рассмотрели в разделе подключение и настройка.

Метод status()

- Назначение: получение состояния модуля.
- Синтаксис: `uint8_t status()`
- Параметры: нет
- Возвращаемое значение:
  - `GSM_OK` : модуль готов к работе.
  - `GSM_SPEED_ERR` : не удалось согласовать скорость UART.
  - `GSM_UNAVAILABLE` : модуль недоступен (AT-команды не выполняются).
  - `GSM_UNKNOWN` : статус неизвестен (AT-команды могут не выполнятся).
  - `GSM_SLEEP` : модуль в режиме ограниченной функциональности.
  - `GSM_SIM_PIN` : требуется ввести PIN1.
  - `GSM_SIM_PUK` : требуется ввести PUK1 и новый PIN1.
  - `GSM_SIM_PIN2` : требуется ввести PIN2.
  - `GSM_SIM_PUK2` : требуется ввести PUK2 и новый PIN2.
  - `GSM_SIM_NO` : нет SIM-карты.

    - `GSM_SIM_FAULT` : SIM-карта неисправна.
    - `GSM_SIM_ERR` : неопределённое состояние SIM-карты.
    - `GSM_REG_NO` : модем не зарегистрирован в сети оператора.
    - `GSM_REG_FAULT` : регистрация модема в сети оператора отклонена.
    - `GSM_REG_ERR` : статус регистрации модема в сети оператора не читается.
- Примечания:
    - Метод `status()` удобно использовать сразу после метода `begin()` для ожидания регистрации модема в сети оператора связи, определения необходимости ввода PIN-кода, информирования об отсутствии SIM-карты и т.д.
    - Метод `status()` удобно использовать в функции `loop()` для отслеживания разрыва регистрации с сетью оператора связи.
- Пример:

```
// Вводим PIN-код «1234» если он нужен
if (gsm.status() == GSM_SIM_PIN) {
  gsm.pin("1234");
}
```

```
// Ждём завершения регистрации модема в сети оператора связи
while (gsm.status() == GSM_REG_NO) {
  delay(1000);
}
```

```
// Ждём полной готовности модуля к работе
while (gsm.status() != GSM_OK) {
  delay(1000);
}
```

Метод pin()

- Назначение: ввод PIN-кода.
- Синтаксис:

- `bool pin(String pinCode)`
- `bool pin(const char* pinCode)`

- Параметры:
  - `pinCode` : строка с текущим PIN-кодом.
- Возвращаемое значение:
  - `true` : PIN-код введен верно.
  - `false` : PIN-код введен не верно.
- Примечания:
  - Вводите PIN-код только, если метод `status()` вернул значение `GSM_SIM_PIN` .
  - Некорректный ввод PIN-кода три раза, приведёт к необходимости смены PIN-кода с вводом PUK-кода с помощью метода `puk()` .

Метод puk()

- Назначение: ввод PUK-кода для смены PIN-кода.
- Синтаксис:
  - `bool puk(String pukCode, String pinCodeNew)`
  - `bool puk(String pukCode, const char* pinCodeNew)`
  - `bool puk(const char* pukCode, String pinCodeNew)`
  - `bool puk(const char* pukCode, const char* pinCodeNew)`
- Параметры:
  - `pukCode` : строка с текущим PUK-кодом.
  - `pinCodeNew` : строка с новым PIN-кодом.
- Возвращаемое значение:
  - `true` : PUK-код введен верно.
  - `false` : PUK-код введен не верно.
- Примечания:
  - Вводите PUK-код только, если метод `status()` вернул значение `GSM_SIM_PUK` .
  - Некорректный ввод PUK-кода десять раз, приведёт к блокировке SIM-карты.
  - Если новый PIN-код указать `0000` , то ввод PIN-кода и PUK-кода более не потребуется.

- Пример:

```
// Вводим текущий PUK-код «123456789» и новый PIN-код «0000»
if (gsm.status() == GSM_SIM_PUK) {
gsm.puk("12345678", "0000");
}
```

Метод pwr()

- Назначение: включение и выключение модуля.
- Синтаксис: `void pwr(state)`
- Параметры:
  - `state` :
    - `true` : включить модуль
    - `false` : выключить модуль
- Возвращаемое значение: нет
- Примечания:
  - По умолчанию модуль включён.
  - Метод `pwr` использует номер вывода Arduino указанный при создании объекта через конструктор `iarduino_I2C_TDS` .
  - При включении модуля методом `pwr()` его инициализация происходит автоматически.
  - Для перезагрузки модуля используйте метод `reset()` .
- Пример:

```
// Включить модуль
gsm.pwr(true);
// Выключить модуль
gsm.pwr(false);
```

Метод reset()

- Назначение: перезагрузка модуля.

- Синтаксис: `void reset()`
- Параметры: нет
- Возвращаемое значение: нет
- Примечания:
  - Метод `reset` выключает модуль, ждёт две секунды и включает модуль.
  - Метод `reset` использует номер вывода Arduino указанный при создании объекта через конструктор `iarduino_I2C_TDS`.
- Пример:

```
  // Перезагружаем модуль
  gsm.reset();
}
```

Метод runAT()

- Назначение: выполнение AT-команды.
- Синтаксис:
  - `String runAT(String command, uint32_t timeOut = 200, bool noWait = true)`
  - `String runAT(const char* command, uint32_t timeOut = 200, bool noWait = true)`
- Параметры:
  - `command`: строка с AT-командой которую требуется выполнить.
  - `timeOut`: время отводимое на выполнение AT-команды. Необязательный параметр и по умолчанию равен `200`.
  - `noWait`: досрочный выход из метода `runAT()` при наличии в ответе комбинации символов `OK` или `ERROR`. Необязательный параметр и по умолчанию равен `true`.
- Возвращаемое значение: строка с ответом на AT-команду.
- Примечания:
  - Текст AT-команды должен заканчиваться символом `\r` или символами `\r\n`.
  - Текст ответа на AT-команды так же содержит символы `\r\n`.
- Пример:

```
```

```
// Вывести результат AT-команды «AT+CSQ» (запрос уровня сигнала)
// выделить на выполнение команды 1000 мс
// запретить досрочный выход, т.е ждать указанное время вне зависимости от ответа
Serial.println(gsm.runAT("AT+CSQ\r\n", 1000, false));
```

```
// Вывести результат AT-команды «AT+CSQ» (запрос уровня сигнала)
// выделить на выполнение команды 1000 мс
// разрешить досрочный выход при получении «OK» или «ERROR»
Serial.println(gsm.runAT("AT+CSQ\r\n",1000));
```

```
// Вывести результат AT-команды «AT+CSQ» (запрос уровня сигнала)
// выделить на выполнение команды 200 мс
// разрешить досрочный выход при получении «OK» или «ERROR»
Serial.println(gsm.runAT("AT+CSQ\r\n"));
```

Метод runUSSD()

- Назначение: выполнение USSD запроса.
- Синтаксис:
  - `String runUSSD(String command, uint32_t timeOut = 10000)`
  - `String runUSSD(const char* command, uint32_t timeOut = 10000)`
- Параметры:
  - `command` : строка с командой USSD запроса.
  - `timeOut` : время отводимое на выполнение USSD запроса. Необязательный параметр и по умолчанию равен `10000` .
- Возвращаемое значение: строка с ответом на USSD запрос.
- Примечания:
  - Если ответ отображается некорректно, возможно он приходит на Кириллице. Для смены кодировки используйте метод `TXTreadCoding()` .
  - Многие Российские операторы связи придерживаются правила, что ответ на команду начинающуюся с символа `*` возвращается на Кириллице, а при замене `*` на `#` ответ возвращается на Латинице. Например, команда `#100#` вернёт баланс на Латинице.

- Пример:

```
// Вывести баланс, а точнее ответ на команду *100#
Serial.println(gsm.runUSSD("*100#"));
```

```
// Создаём строку
char str[161];
// Получаем ответ на команду *100# в строку str, указывая её размер
gsm.runUSSD("*100#").toCharArray(str, 161);
// Выводим строку str в монитор
Serial.println( str );
```

Метод signal()

- Назначение: запрос уровня принимаемого сигнала.
- Синтаксис: `uint8_t signal()`
- Параметры: нет
- Возвращаемое значение: число от `0` до `31`.
- Примечания:
  - `0` : уровень сигнала -113 дБм и ниже
  - `1–30` :уровень сигнала от -111 дБм до -53 дБм (шаг 2 дБм)
  - `31` : уровень сигнала -51 дБм и выше.
- Пример:

```
  // Получаем уровень приёма сигнала
uint8_t i = gsm.signal();
if (i == 0) {
  // При уровне приёма ниже или равном -113дБм
  Serial.println("    ");
} else if (i < 8) {
```

```
  // При уровне приёма ниже -97дБм
  Serial.println("▌   ");
} else if (i < 16) {
  // При уровне приёма ниже -81дБм
  Serial.println("▌▌  ");
} else if(i < 24) {
  // При уровне приёма ниже -65дБм
  Serial.println("▌▌▌ ");
} else {
  // При уровне приёма выше или равном -65дБм
  Serial.println("▌▌▌▌");
}
```

Метод SMSavailable()

- Назначение: запрос количества принятых непрочитанных SMS сообщений.
- Синтаксис: `uint8_t SMSavailable()`
- Параметры: нет
- Возвращаемое значение: количество принятых непрочитанных SMS.
- Примечания:
  - Принятые SMS сообщения хранятся на SIM-карте, пока они не будут прочитаны.
  - Если память SIM-карты заполнена, то модуль не будет принимать новые входящие сообщения, пока не будут прочитаны старые.
  - Сообщения удаляются с SIM-карты сразу после их чтения методом `SMSread()`.
  - Общий объем памяти SIM-карты для SMS можно запросить методом `SMSmax()`.
- Пример:

```
// Если есть входящие непрочитанные SMS сообщения
if (gsm.SMSavailable()) {
  // Выводим количество непрочитанных SMS
  Serial.print("I have incoming SMS:");
  Serial.println(gsm.SMSavailable());
```

```
}
```

Метод SMSmax()

- Назначение: запрос объема памяти SMS на SIM карте.
- Синтаксис: `uint8_t SMSmax()`
- Параметры: нет
- Возвращаемое значение: максимальное количество хранимых SMS.
- Примечания:
  - Принятые SMS сообщения хранятся на SIM-карте, пока они не будут прочитаны.
  - Если память SIM-карты заполнена, то модуль не будет принимать новые входящие сообщения, пока не будут прочитаны старые.
  - Сообщения удаляются с SIM-карты сразу после их чтения методом `SMSread()` .
  - Количество принятых непрочитанных SMS можно запросить методом `SMSavailable()` .
- Пример:

```
// Вывод максимального количество смс на SIM-карте
Serial.println("Max number of sms on sim card: ");
Serial.println(gsm.SMSmax());
```

Метод SMSread()

- Назначение: чтение одного входящего не прочитанного SMS сообщения.
- Синтаксис:
  - `bool SMSread(char* text)`
  - `bool SMSread(char* text, char* phone)`
  - `bool SMSread(char* text, char* phone, char* date)`
  - `bool SMSread(char* text, char* phone, char* date, uint16_t& ID, uint8_t& sum, uint8_t& num)`
- Параметры:
  - `text` : переменная для сохранения текста SMS сообщения. Текст SMS сообщения может достигать 160 символов Латиницей или 70 символов Кириллицей.

- `phone` : переменная для сохранения номера отправителя SMS сообщения. Номер отправителя может достигать 12 символов. Необязательный параметр.
- `date` : переменная для сохранения даты отправки SMS сообщения. Дата отправки сообщения содержит 17 символов «ДД.ММ.ГГ ЧЧ:ММ:СС». Необязательный параметр.
- `ID` : переменная для сохранения числового идентификатора. Если сообщение состоит из одного SMS, то идентификатор будет равен 0. Если сообщение состоит из нескольких SMS, то у каждой SMS данного сообщения будет один и тот же идентификатор. Необязательный параметр.
- `sum` : переменная для сохранения количество SMS в составном сообщении. Если сообщение состоит из одного SMS, то количество будет равно 1. Необязательный параметр.
- `num` : переменная для сохранения номера данной SMS в составном сообщении. Если сообщение состоит из одного SMS, то номер будет равен 1. Необязательный параметр.

- Возвращаемое значение:
  - `true` : SMS прочтено успешно.
  - `false` : SMS прочтено не успешно.
- Примечания:
  - Если предполагается принимать только короткие SMS сообщения, то параметры `ID` , `sum` и `num` можно не указывать.
  - Параметры `phone` и `date` так же являются необязательными.
  - Параметр `text` короткого SMS сообщения может достигать 160 символов Латиницей или 70 символов Кириллицей.
  - Параметр `text` одной SMS в составе составного сообщения может достигать 152 символов Латиницей или 66 символов Кириллицей.
  - Метод `SMSread()` удаляет SMS сообщение из памяти SIM-карты, сразу после его чтения.
  - SMS сообщения читаются в порядке их поступления.
  - Если сообщение отображается некорректно, возможно оно приходит в другой кодировке. Для смены кодировки используйте метод `TXTreadCoding()` .
- Пример:

```
// Переменная для хранения текста SMS
char SMStxt[161];
// Переменная для хранения адреса отправителя SMS
char SMSadr[13];
```

```
// Если есть непрочитанные входящие SMS сообщения
if(gsm.SMSavailable()){
  // Читаем SMS сообщение
  gsm.SMSread(SMStxt, SMSadr);
  // Выводим номер отправителя SMS
  Serial.println(SMSadr);
  // Выводим текст SMS сообщения.
  Serial.println(SMStxt);
}
```

Метод SMSsend()

- Назначение: отправка одного SMS сообщения.
- Синтаксис:
  - `bool SMSsend(String text, String phone, uint16_t ID = 0, uint8_t sum = 1, uint8_t num = 1)`
  - `bool SMSsend(const char* text, String phone, uint16_t ID = 0, uint8_t sum = 1, uint8_t num = 1)`
  - `bool SMSsend(String text, const char* phone, uint16_t ID = 0, uint8_t sum = 1, uint8_t num = 1)`
  - `bool SMSsend(const char* text, const char* phone, uint16_t ID = 0, uint8_t sum = 1, uint8_t num = 1)`
- Параметры:
  - `text` : текст отправляемого SMS сообщения. Текст SMS сообщения может достигать 160 символов Латиницей или 70 символов Кириллицей.
  - `phone` : номера отправителя SMS сообщения. Номер отправителя может достигать 12 символов.
  - `ID` : идентификатор составного сообщения, которое состоит из нескольких SMS. Если сообщение состоит из одного SMS, то идентификатор можно не указывать. Необязательный параметр.
  - `sum` : количество SMS в составном сообщении. Если сообщение состоит из одного SMS, то количество можно не указывать. Необязательный параметр.
  - `num` : номера данной SMS в составном сообщении. Если сообщение состоит из одного SMS, то количество можно не указывать. Необязательный параметр.
- Возвращаемое значение:
  - `true` : SMS отправлено успешно.

    - `false` : SMS отправлено не успешно.
- Примечание:
    - Если предполагается отправлять только короткие SMS сообщения, то параметры `ID` , `sum` и `num` можно не указывать.
    - Параметр `text` составного SMS сообщения может достигать 152 символов Латиницей или 66 символов Кириллицей.
    - Отправленные SMS сообщения не сохраняются в памяти.
    - Если на телефоне принимающей стороны сообщение отображается некорректно, возможно оно приходит в другой кодировке. Для смены кодировки используйте методы `TXTsendCoding()` и `TXTsendCodingDetect()` .
    - Если принимающей стороне одновременно поступят несколько составных сообщений с одинаковым `ID` и `sum` , то существует вероятность неправильной склейки текста SMS. По этому значение `ID` должно генерироваться как случайное число от `1` до `65535` .
- Пример:

```
// Отправка SMS с текстом «Привет, мир!» на номер +79684541738
gsm.SMSsend("Привет", "+79684541738");
```

```
// Отправка SMS 1/3 в составе составного сообщения
// с идентификатором 0x1234 на номер получателя +79684541738
gsm.SMSsend("Данное SMS сообщение является составным, так как содержит более 70", "+79684541738", 0x1234, 3, 1);

// Отправка SMS 2/3 в составе составного сообщения
// с идентификатором 0x1234 на номер получателя +79684541738
gsm.SMSsend(" символов. По этому это сообщение будет отправлено как 3 SMS сообщ", "+79684541738", 0x1234, 3, 2);

// Отправка SMS 2/3 в составе составного сообщения
// с идентификатором 0x1234 на номер получателя +79684541738
gsm.SMSsend("ения. Но получатель увидит его как один большой целый текст.", "+79684541738", 0x1234, 3, 3);
```

Метод SMSsendClass()

- Назначение: указание класса для всех отправляемых SMS сообщений.

- Синтаксис: `void SMSsendClass(uint8_t smsClass)`
- Параметры:
  - `smsClass` : значение соответствующее одной из констант.
    - `GSM_SMS_CLASS_NO` : отправлять SMS сообщения без класса (по умолчанию).
    - `GSM_SMS_CLASS_0` : отправлять SMS сообщения с указанием 0 класса.
    - `GSM_SMS_CLASS_1` : отправлять SMS сообщения с указанием 1 класса.
    - `GSM_SMS_CLASS_2` : отправлять SMS сообщения с указанием 2 класса.
    - `GSM_SMS_CLASS_3` : отправлять SMS сообщения с указанием 3 класса.
- Возвращаемое значение: нет
- Примечания:
  - Сообщения без класса — это обычные SMS сообщения отправляемые с телефона на телефон.
  - Сообщения 0 класса выводятся напрямую на дисплей телефона. Такие сообщения либо не сохраняются вообще, либо телефон запрашивает разрешение на сохранение сообщения в памяти входящих SMS.
  - Сообщения 1 класса сохраняются в памяти телефона.
  - Сообщения 2 класса сохраняются в памяти SIM-карты.
  - Сообщения 3 класса передаются терминальному оборудованию или приложению.
  - Выбранный класс будет применяться ко всем SMS сообщениям отправляемым методом `SMSsend()` .
- Пример:

```
// Отправлять все последующие SMS с указанием класса 0
gsm.SMSsendClass(GSM_SMS_CLASS_0);
// Данное сообщение будет отправлено как SMS класса 0
gsm.SMSsend("Привет, мир!", "+79684541738");
// Отправлять все последующие SMS без класса
gsm.SMSsendClass(GSM_SMS_CLASS_NO);
```

Метод TXTreadCoding()

- Назначение: указание кодировки для получаемого текста.
- Синтаксис: `void TXTreadCoding(uint8_t txtCoding)`

- Параметры:
  - `txtCoding` : кодировка текста.
    - `GSM_TXT_UTF8` : использовать кодировку UTF8.
    - `GSM_TXT_WIN1251` : использовать кодировку Windows-1251.
    - `GSM_TXT_CP866` : использовать кодировку CP-866.
- Возвращаемое значение: нет
- Примечание:
  - Если метод чтения SMS сообщений `SMSread()` или метод выполнения USSD запросов `runUSSD()` некорректно возвращают символы в принятом тексте, то до их выполнения нужно вызвать метод `TXTreadCoding()` указав одну из предложенных кодировок.
  - По умолчанию используется кодировка UTF-8.
- Пример:

```
// Указываем выводить весь входящий текст в кодировке UTF8
gsm.TXTreadCoding(GSM_TXT_UTF8);
```

```
// Указываем выводить весь входящий текст в кодировке Windows-1251
gsm.TXTreadCoding(GSM_TXT_WIN1251);
```

```
// Указываем выводить весь входящий текст в кодировке CP-866
gsm.TXTreadCoding(GSM_TXT_CP-866);
```

Метод TXTsendCoding()

- Назначение: указание кодировки для отправляемого текста.
- Синтаксис: `void TXTsendCoding(uint8_t txtCoding)`
- Параметры:
  - `txtCoding` : кодировка текста.
    - `GSM_TXT_UTF8` : использовать кодировку UTF8.
    - `GSM_TXT_WIN1251` : использовать кодировку Windows-1251.

    - `GSM_TXT_CP866` : использовать кодировку CP-866.
- Возвращаемое значение: нет
- Примечания:
    - Метод `TXTsendCoding()` указывает в какой кодировке написан текст SMS сообщения отправляемого методом `SMSsend()` .
    - Если на телефоне получателя SMS сообщений некорректно отображаются символы, то до обращения к методу `SMSsend()` нужно вызвать метод `TXTsendCoding()` указав одну из предложенных кодировок.
    - По умолчанию используется кодировка UTF-8.
    - Так как в большинстве случаев отправляемый текст SMS сообщений хранится в скетче, то вместо ручного метода выбора кодировки `TXTsendCoding()` можно использовать метод автоопределения кодировки отправляемого текста `TXTsendCodingDetect()` .
- Пример:

```
// Указываем отправлять весь текст в кодировке UTF8
gsm.TXTsendCoding(GSM_TXT_UTF8);
```

```
// Указываем отправлять весь текст в кодировке Windows-1251
gsm.TXTsendCoding(GSM_TXT_WIN1251);
```

```
// Указываем отправлять весь текст в кодировке CP-866
gsm.TXTsendCoding(GSM_TXT_CP-866);
```

Метод TXTsendCodingDetect()

- Назначение: автоопределение кодировки отправляемого текста.
- Синтаксис: `void TXTsendCodingDetect(const char* strRusP)`
- Параметры: `strRusP` : символ кодировки. Указывайте строчная букву `п` .
- Возвращаемое значение: нет
- Примечание:
    - Метод `TXTsendCodingDetect()` определяет кодировку скетча и сообщает полученную кодировку методу `SMSsend()` .
    - В качестве параметра допускается указывать только символ `п` , так как кодировка определяется по коду указанного символа.

- Пример:

```
// Автоопределение кодировки отправляемого текста
gsm.TXTsendCodingDetect('п');
```

Метод CALLavailable()

- Назначение: проверка наличия входящего голосового вызова.
- Синтаксис:
  - `void CALLavailable()`
  - `void CALLavailable(char* phone)`
  - Параметры:
    - `phone` : переменная для сохранения номера входящего абонента. Номер отправителя может достигать 12 символов. Необязательный параметр.
  - Возвращаемое значение:
    - `true` : есть входящий вызов.
    - `false` : нет входящего вызова.
- Примечания:
  - Метод `CALLavailable()` удобно использовать перед ответом на входящий голосовой вызов методом поднятия трубки `CALLup()` .
- Пример:

```
if (gsm.CALLavailable()) {
  Serial.println("I have incoming call!");
}
```

```
// Переменная для хранения номера входящего абонента
char phone[13];
// Если поступает входящий звонок
if (gsm.CALLavailable(SMSadr))
  // Выводим номер звонящего
```

```
  Serial.print("I have incoming call from ");
  Serial.println(phone);
  }
```

Метод CALLup()

- Назначение: ответ на входящий голосовой вызов (поднятие трубки).
- Синтаксис: `void CALLup()`
- Параметры: нет
- Возвращаемое значение: нет
- Примечания:
  - Перед методом `CALLup()` удобно использовать метод проверки наличия входящего голосового вызова методом `CALLavailable()`.
  - При установке активного голосового соединения, используется громкая связь. Для смены аудио потока на гарнитуру используйте метод `SOUNDdevice()`.
- Пример:

```
// Если есть входящий вызов
if (gsm.CALLavailable()) {
  // Поднимаем трубку
  gsm.CALLup()
}
```

Метод CALLend()

- Назначение: завершение голосового вызова (опускание трубки).
- Синтаксис: `void CALLend()`
- Параметры: нет
- Возвращаемое значение: нет
- Примечание:
  - Метод `CALLend()` завершает все вызовы: входящие, исходящие, активные, удерживаемые и ожидающие.

- Пример:

```
// Если есть входящий вызов
if (gsm.CALLavailable()) {
  // Бросаем трубку
  gsm.CALLend()
}
```

Метод CALLdial()

- Назначение: инициализация исходящего голосового вызова (набор номера).
- Синтаксис:
  - `bool CALLdial(String phone)`
  - `bool CALLdial(const char* phone)`
- Параметры:
  - `phone` : номера вызывающего абонента. Номер может достигать 12 символов.
- Возвращаемое значение:
  - `true` : исходящий вызов совершен успешно.
  - `false` : исходящий вызов совершен не успешно.
- Примечания:
  - Статус голосового вызова можно отследить методом `CALLstatus()` .
  - После успешной инициализации установится статус вызова «исходящий набираемый».
  - После успешного набора номера установится статус вызова «исходящий в режиме дозвона».
  - После ответа вызываемой стороной установится статус вызова «активный голосовой».
  - После сброса вызова установится статус вызова «разъединение».
  - После разъединения установится статус «свободен для звонков».
  - При установке активного голосового соединения, используется громкая связь. Для смены аудио потока на гарнитуру используйте метод `SOUNDdevice()` .

Метод CALLstatus()

- Назначение: получение состояния голосового вызова.
- Синтаксис: uint8_t CALLstatus()
- Параметры: нет
- Возвращаемое значение:
  - `GSM_OK` : свободен для звонков
  - `GSM_CALL_ACTIVE` : активное голосовое соединение
  - `GSM_CALL_OUT_DIAL` : исходящий вызов в режиме набора номера
  - `GSM_CALL_OUT_BEEP` : исходящий вызов в режиме дозвона (ждём поднятия трубки)
  - `GSM_CALL_IN_BEEP` : входящий вызов в режиме дозвона (ждёт поднятия трубки)
  - `GSM_CALL_IN_WAIT` : входящий вызов в режиме ожидания
  - `GSM_CALL_HELD` : вызов в режиме удержания
  - `GSM_CALL_END` : вызов разъединяется
  - `GSM_CALL_ERR` : состояние вызова не определено
- Примечания:
  - Метод `CALLstatus()` удобно использовать для распределения функционала на разных стадиях голосового вызова.
- Пример:

```
// Если совершен исходящий вызов
if (gsm.CALLdial("+79684541738")) {
  Serial.println("Dialing number...");
  // Ждём завершения набора номера
  while (gsm.CALLstatus() == GSM_CALL_OUT_DIAL) {
    // Код будет выполняться в процессе набора номера
  }
  delay(500);
  // Если начались гудки дозвона
  if (gsm.CALLstatus() == GSM_CALL_OUT_BEEP) {
    // Ждём ответа
    Serial.println("Waiting for answer...");
    // Ждём поднятия трубки на вызываемой стороне
```

```
    while (gsm.CALLstatus() == GSM_CALL_OUT_BEEP) {
      // Код будет выполняться в процессе ожидания ответа
    }
    delay(500);
  }
  // Если соединение установлено (абонент ответил)
  if (gsm.CALLstatus() == GSM_CALL_ACTIVE) {
    // Установлено голосовое соединение!
    Serial.println("Voice connection is OK");
    // Ждём завершения активного голосового соединения
    while (gsm.CALLstatus() == GSM_CALL_ACTIVE) {
      // Код будет выполняться в процессе разговора
    }
    // Вызов завершён
    Serial.println("Call ended");
  } else {
    // Если соединение не было установлено (абонент не ответил)
    Serial.println("No answer");
  }
}
```

Метод SOUNDdevice()

- Назначение: выбор устройства ввода/вывода звука
- Синтаксис: `uint8_t SOUNDdevice(uint8_t soundDevice)`
- Параметры:
  - `device` : устройство для ввода/вывода звука, которое требуется выбрать.
    - `GSM_HEADSET` : использовать гарнитуру.
    - `GSM_SPEAKER` : использовать громкую связь. Используется по умолчанию.
    - `GSM_MICROPHONE` : использовать только микрофон громкой связи.
- Возвращаемое значение:
  - `device` : устройство для ввода/вывода звука, которое сейчас используется.

    - `GSM_HEADSET` : использовать гарнитуру.
    - `GSM_SPEAKER` : использовать громкую связь.
    - `GSM_MICROPHONE` : использовать только микрофон громкой связи.
- Примечания:
    - Если метод `SOUNDdevice()` вызвать без параметра, то он вернёт значение соответствующее используемому устройству, не меняя его.
    - Если метод `SOUNDdevice()` вызвать с параметром, то он переключит звук на указанное устройство и вернёт значение соответствующее используемому устройству.
    - По умолчанию используется громкая связь.
- Пример:

```
// Использовать гарнитуру для ввода/вывода звука
gsm.SOUNDdevice(GSM_HEADSET);
```

Метод SOUNDvolume()

- Назначение: установка громкости звука
- Синтаксис: `uint8_t SOUNDvolume(volume)`
- Параметры:
    - `volume` : уровень громкости от `0` до `7` .
        - `7` : максимальная громкость
        - `1` : минимальная громкость
        - `0` : без звука
- Возвращаемое значение:
    - `volume` : текущий уровень громкости от `0` до `7` .
        - `7` : максимальная громкость
        - `1` : минимальная громкость
        - `0` : без звука
- Примечания:

- Если метод `SOUNDvolume()` вызвать без параметра, то он вернёт значение соответствующее установленной громкости, не меняя её.
- Если метод `SOUNDvolume()` вызвать с параметром, то он установит указанный уровень звука и вернёт значение соответствующее установленному уровню звука.

- Пример:

```
// Установить максимальный уровень громкости
gsm.SOUNDvolume(3);
// Увеличить громкость на 1 пункт
gsm.SOUNDvolume(gsm.SOUNDvolume() + 1);
// Уменьшить громкость на 1 пункт
gsm.SOUNDvolume(gsm.SOUNDvolume() - 1);
```

Метод SOUNDmute()

- Назначение: включение режима без микрофона
- Синтаксис:
  - `bool SOUNDmute()`
  - `bool SOUNDmute(state)`
- Параметры:
  - `true` : включить режим без микрофона.
  - `false` : выключить режим без микрофона.
- Возвращаемое значение:
  - `true` : режим без микрофона включен успешно.
  - `false` : режим без микрофона включен не успешно.
- Примечания:
  - Если метод `SOUNDmute()` вызвать без параметра, то он вернёт флаг состояния режима без микрофона.
  - Если метод `SOUNDmute()` вызвать с параметром, он включит или выключит немой режим, а затем вернёт флаг состояния режима без микрофона.
- Пример:

```
```

```
// Включить немой режим
gsm.SOUNDmute(true);
// Выключить немой режим
gsm.SOUNDmute(false);
```

Габаритный чертёж

Чертёж GSM/GPRS Shield A6

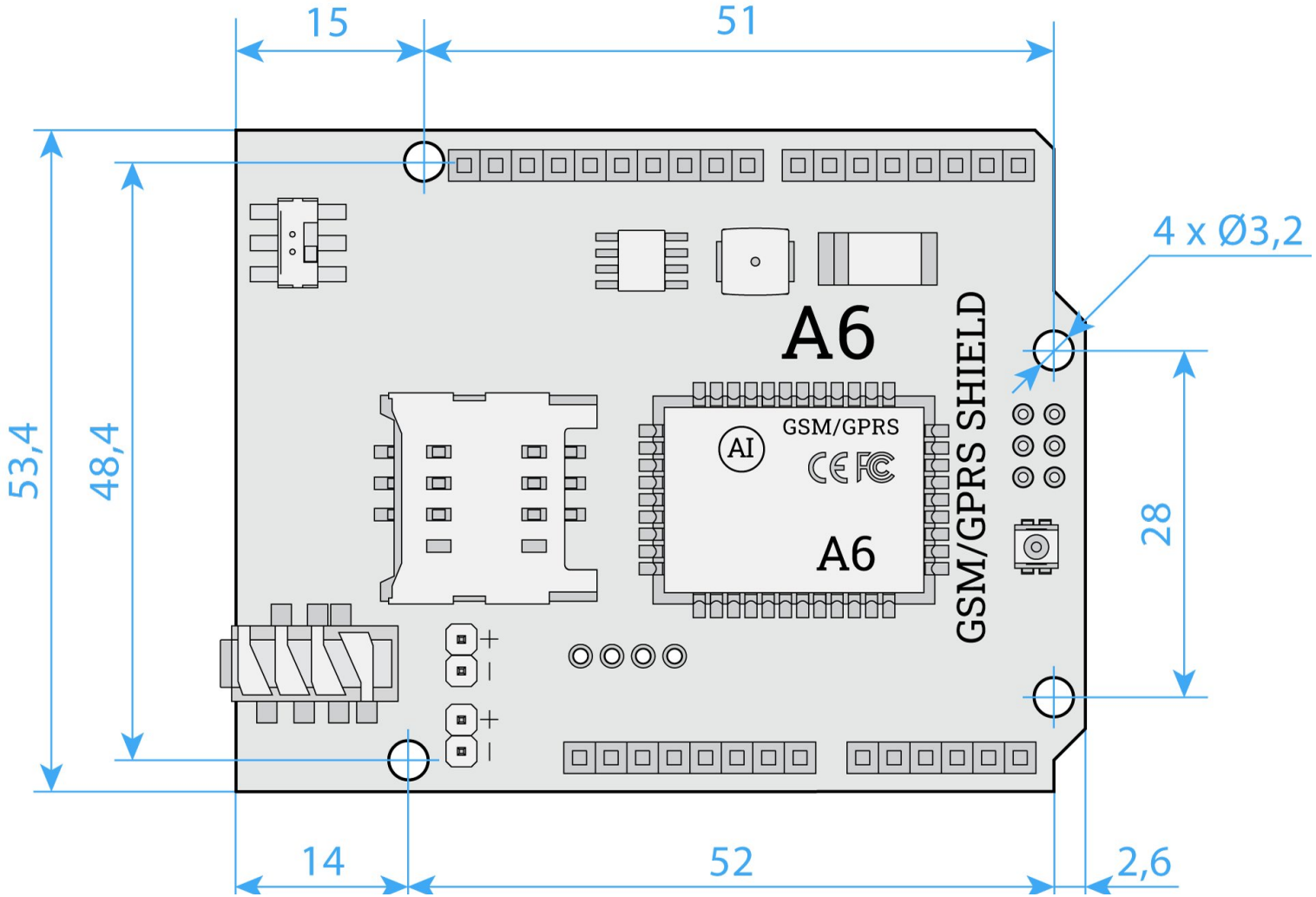
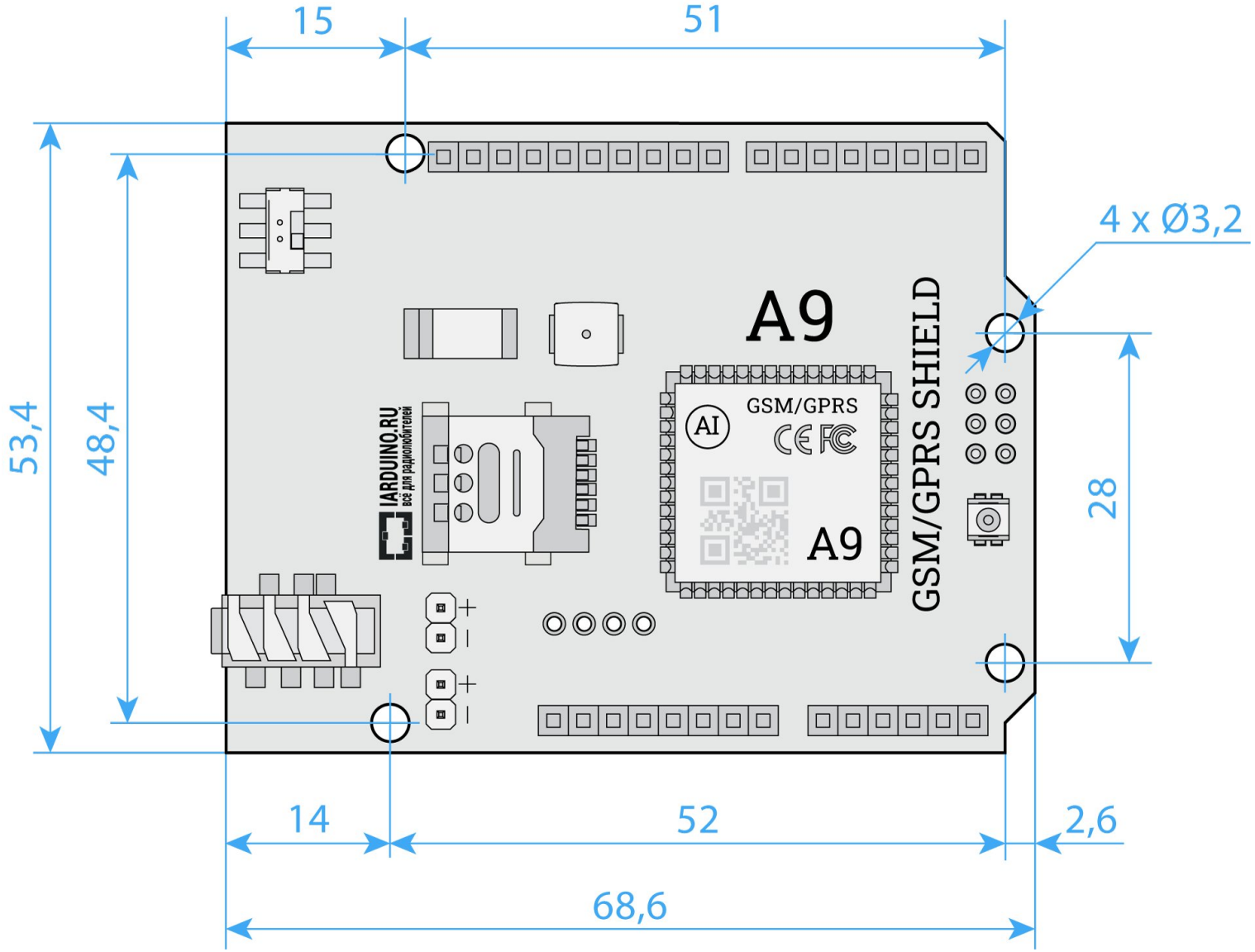


Чертёж GSM/GPRS Shield A6



Характеристики

- Модель:
  - GSM/GPRS Shield A6
  - GSM/GPRS Shield A9
- Стандарт связи: GSM/GPRS
- Возможности:
  - Версия A6: голосовая связь, SMS-сообщения
  - Версия A9: голосовая связь, SMS-сообщения, приём и передача данных
- Поддержка частот: 850/900/1800/1900 МГц
- Слот для SIM-карты:
  - Версия A6: mini SIM (2FF)
  - Версия A9: nano SIM (4FF)
- Антенна:
  - Встроенная: разведена на плате
  - Внешняя: подключается через разъём IPX UFL
- Совместимость: контроллеры форм-фактора Arduino R3
- Программный интерфейс: UART с дополнительными пином управления
- Программный протокол: AT-команды
- Входное напряжение питания:
  - Версия A6: 5–12 В
  - Версия A9: 7–12 В
- Потребляемый ток:
  - В спящем режиме: до 3 мА
  - В режиме ожидания: до 100 мА
  - В активном режиме (соединение, разговор, SMS): до 500 мА
  - Поиск сети: до 2 А
- Логическое напряжение уровней: 3,3–5 В

- Размеры: Arduino Shield R3