Ответы на викторину "Arduino: программная часть-6"

("Радио", 2021, № 9, с. 63, 64)

С. РЮМИК, г. Чернигов, Украина

1 **Ответ — 1**. Платы A1 и A2 на электрической схеме соединяются одним информационным проводом. Однако для протекания тока требуется замкнуть контур, т. е. нужен ещё один проводник. На схеме он обозначается в виде общего провода ("земли") — специального гостированного знака "соединение с корпусом", который подключается к контактам **GND** плат A1 и A2. Это упрощает рисование электрических схем.

Наличие двух одинаковых знаков означает, что монтажник должен проложить между этими цепями отдельный провод. Итого, хотя интерфейс и называется однопроводным, но физически платы Arduino соединяются с помощью двух проводников. Именно этот момент иногда не учитывают начинающие радиолюбители, когда ищут ответ на вопрос: "Почему не работает?"

2 **Ответ — 1**. Назначение скетча — передача по однопроводному интерфейсу GBus [1] последовательности цифр от 1 до 5 с паузой между посылками 2 с. Используется внешняя библиотека GyverBus, которую надо установить в среде Arduino IDE, выбрав пункты **Инструменты→Управлять библиотеками→Поиск Gyver Bus→Установка**.

Для ответа на вопрос о местонахождении счётчика передаваемых байтов не обязательно вникать в тонкости незнакомой библиотеки функций. Можно воспользоваться методом аналогий и проанализировать ключевые термины в строках 11 и 12. В частности, слово **sizeof** является стандартным в языке Си (C++) и обозначает унарный оператор, с помощью которого вычисляется число байт, занимаемое объектом в памяти. В строке 12 скетча этот оператор пересчитывает цифры 1—5 в буфере **buf[ ]**, т. е. является своеобразным "счётчиком".

3 **Ответ — 0**. Назначение скетча — приём данных, поступающих по однопроводному интерфейсу GBus (строки 7, 8), загрузка их в буфер **buf[ ]** (строки 9, 10) и посимвольная индикация в терминале компьютера (строка 11). Протокол обмена данными интерфейса GBus похож на UART с одним стартовым, одним стоповым и восемью битами данных. Поскольку скорость передачи протокола UART принято указывать в бодах, то логично предположить, что число 1000 — это тоже боды, чему есть подтверждение в официальной документации GBus [1].

Бод — это единица измерения символьной скорости в отличие от скорости передачи полезной информации, которая представляется в битах в секунду. Перевод осуществляется по формуле [2]: $V[\text{бит/с}]=V[\text{бод}]\cdot d/(d+s+1)=1000\cdot 8/(8+1+1)=800$ бит/с, где **d**, **s** — количество битов соответственно в поле данных и в "стопах".

4 **Ответ — 1**. Как известно, платформа Arduino не накладывает ограничений на тип микроконтроллера, его тактовую частоту и напряжение питания. В частности, семейство плат Arduino Pro Mini поставляется в двух версиях — с питанием микроконтроллера ATmega168 (ATmega328) от напряжения 3,3 В при тактовой частоте 8 МГц и с питанием от напряжения 5 В при тактовой частоте 16 МГц. Отличаются версии добавлением надписи 3,3V или 5V в конце названия.

Связь между платами A1 и A2 осуществляется с помощью двухпроводного интерфейса UART. Если заменить плату A2 платой Arduino Pro Mini 3,3V, появится риск повредить её микроконтроллер высоким напряжением 4,5...5 В, поступающим в виде лог. 1 с линии 11 цифрового выхода платы A1.

Согласно справочным данным на микроконтроллеры ATmega168, ATmega328, напряжение на их информационных линиях не должно превышать VCC+0,5 В, где VCC — напряжение питания в вольтах [3]. Значит, при VCC = 3,3 В нельзя подавать на входы больше, чем 3,8 В. Требуется согласование уровней, но это уже доработка существующей конструкции, а не замена "один к одному".

5 **Ответ — 0**. Назначение скетча — передача по интерфейсу UART фразы Hello, world! и цифр 1—4 с паузой 2 с. Используется встроенная в Arduino IDE библиотека функций **SoftwareSerial**, которая позволяет работать с любыми линиями портов, а не только с **RX** (D0), **TX** (D1).

По умолчанию считается, что кадр информации UART передаётся в формате 8-N-1, т. е. один стартовый, один стоповый и 8 бит данных без контроля чётности. Если надо изменить формат, то в стандартном операторе **Serial.begin()** через запятую после скорости 9600 бод явно указывают тип, например 8-N-2, при этом происходит смена режима и в конце кадра добавляется второй стоповый бит.

Однако это справедливо только для аппаратного UART. В рассматриваемом скетче используется программный UART, а в нём не предусмотрена опция смены формата, поэтому добавление надписи **SERIAL_8N2** приведёт к ошибке компиляции.

6 **Ответ — 0**. Назначение скетча — принять по интерфейсу UART информацию с канала **RX1 (Serial1)** и выдать её в терминал компьютера **(Serial)**, а также принять информацию, вводимую с клавиатуры компьютера (Serial), и передать её в канал **TX1 (Serial1)**. Скетч **MultiSerial.ino** взят из каталога примеров среды Arduino IDE, поэтому содержит как приёмную, так и передающую части. За приём внешней информации отвечают операторы в строках 6—8.

7 **Ответ — 1**. Двухпроцессорная система, собранная на платах A1 и A2, использует для связи трёхпроводной интерфейс SPI. Плата A1 является ведущей, а плата A2 — ведомой, поскольку её контакт 53 соединяется с общим проводом **GND**.

Питание 5 В на платы A1 и A2 поступает от разъёмов USB. Подключаться они могут к одному или к разным компьютерам, ноутбукам. Одна незадача — на электрической схеме отсутствует связь по сигнальной земле между контактами **GND** обеих плат. Следовательно, при подключении разъёмов USB к одному компьютеру связь цепей **GND** будет производиться через внутреннее соединение общих проводов, и система передачи данных будет функционировать нормально.

С другой стороны, при использовании портов USB в двух разных удалённых компьютерах цепи **GND** плат A1 и A2 будут разобщены со всеми вытекающими негативными последствиями, влияющими на появление ошибок и сбоев.

8 **Ответ — 1**. Назначение скетча — передача по трёхпроводному интерфейсу SPI последовательности цифр 0—3 с паузой 2 с. Параметры протокола SPI (**S**erial **P**eripheral **I**nterface) и его технические характеристики настраиваются в строках 5, 6 методом **SPI.beginTransaction()**. Именно так рекомендуется делать, начиная с версии Arduino IDE 1.6.0, вместо использования устаревших программных конструкций.

Параметр **MSBFIRST** в строке 6 указывает, что байты данных передаются старшим битом вперёд. Это подтверждает расшифровка аббревиатуры MSB (**M**ost **S**ignificant **B**it) — старший значащий бит. Если указать в строке 6 параметр **LSBFIRST**, первые три буквы которого расшифровываются как **L**east **S**ignificant **B**it — младший значащий бит, то байты данных будут передаваться младшим битом вперёд.

Анализ строки 9 скетча показывает, что первой цифрой в посылке будет 0, поскольку **i=48**, в шестнадцатеричном коде 0x30 (параметр MSB). Если перевернуть биты в обратную сторону (параметр LSB), то получится число 0x0C (**рис. 1**).

9 **Ответ — 0**. Назначение скетча — при старте программы в терминал компьютера выводится надпись Receive: (строка 8), затем в цикле принимаются байты по интерфейсу SPI и печатаются в терминале в шестнадцатеричном виде (строки 11—13).

В строке 6 скетча контакт **MISO** платы A2 настраивается как цифровой выход. Это обязательная процедура при работе устройства в режиме ведомого и двухстороннем обмене данными. Ключевое слово здесь — двухстороннее. Интерфейс SPI имеет полное дуплексное соединение. Ведущий (master) может передавать данные ведомому (slave), а ведомый одновременно может передавать данные ведущему по одним и тем же тактовым импульсам **SCK**.

Поскольку информация в плату A2 поступает только от ведущего и назад не возвращается, то удаление или закомментирование строки 6 ни на что не повлияет. На **рис. 2** показаны направления передачи сигналов в интерфейсе SPI, из чего видно, что сигнал **MISO** платы A2 принимает участие в передаче, но не в приёме данных. Если оба контакта **MISO** на платах A1 и A2 будут настроены как цифровые входы, то с электрической точки зрения аварийная ситуация не возникнет.

10 **Ответ — 1**. Для передачи данных используется четырёхпроводный интерфейс SPI. В нём в дополнение к стандартным сигналам **MOSI**, **MISO**, **SCK** добавляется сигнал

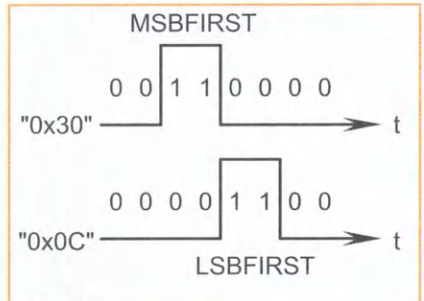

Рис. 1

разрешения **SS** (**S**lave **S**elect), который позволяет организовать сетевое соединение нескольких устройств на одной шине. В документации Arduino сигнал **SS** пишется без общепринятого знака инверсии. Это не запрещается ввиду многочисленных вариаций у разных разработчиков: **NSS**, **nSS**, **nCS**, **/CS**, **CSN**, **SE**.

Питание платы A1 производится от двух последовательно соединённых Li-Ion аккумуляторов суммарным напряжением 7,4 В. Если случайно перепутать полярность подключения их клемм, то может выйти из строя электролитический конденсатор PC1 или микросхема стабилизатора U1 (**рис. 3**). Защита от переполюсовки в виде диода D1 сработает лишь при подключении питания к разъёму X1, но в рассматриваемом варианте аккумуляторы подключаются именно к цепи VIN, которая выводится на гребёнку контактов Arduino UNO.

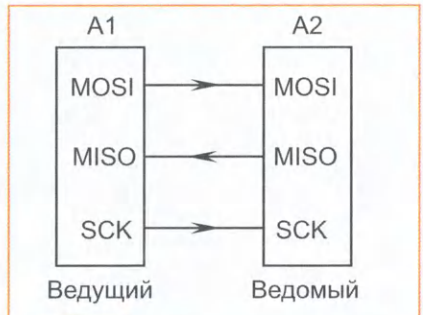

Рис. 2

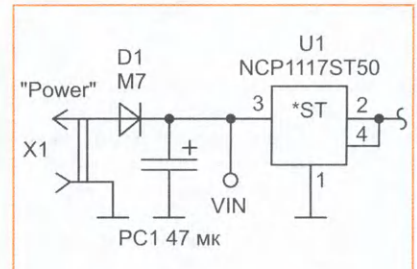

Рис. 3

11 **Ответ — 0**. Назначение скетча — передача по четырёхпроводному интерфейсу SPI последовательности цифр 0—9 (строка 11) с паузой в 1 с (строка 13). В строках 10 и 12 на линии 10 (**SS**) выставляются уровни лог. 0 и лог. 1. Однако самого оператора, задающего режим цифрового выхода для линии 10, в скетче не видно. Логично предположить, что он спрятан внутри библиотеки функций SPI, которая изначально встроена в Arduino IDE. Но в какой именно функции — **SPI.begin()** или **SPI.beginTransaction()**?

Для разгадки необходимо открыть в компьютере папку **C:\Program Files\Arduino\Hardware\arduino\avr\libraries\SPI\srs** и просмотреть текстовым редактором файл **SPI.cpp**. В нём описан прототип функции **void SPIClass::begin()**, где операторами языка Си вывод **SS** микроконтроллера переводится в режим цифрового выхода и на нём устанавливается высокий уровень лог. 1. Следовательно, инициализация линии **SS** проводится в строке 5 в операторе **SPI.begin()**.

12 **Ответ — 0**. Назначение скетча — принять по интерфейсу SPI текстовую информацию в виде последовательности цифр 0—9 и передать её в терминал компьютера. В скетче организуется приём данных через прерывание **ISR** (строка 11), что позволяет высвободить вычислительные мощности для выполнения других задач.

В AVR-контроллерах, применяемых в платах Arduino, прерывания делятся на внешние и внутренние. Внешние прерывания формируются в результате анализа логических уровней на определённых линиях портов микроконтроллера. Внутренние прерывания формируются аппаратными модулями, например таймерами. Поскольку модуль SPI в AVR-контроллере является внутренним, то и прерывания, возникающие при анализе состояния его регистров, будут внутренними.

ЛИТЕРАТУРА

1. Связь нескольких Arduino по проводу. — URL: **https://alexgyver.ru/gyverbus/** (14.07.21).

2. Скорость передачи данных через UART. — URL: **https://bravikov.wordpress.com/2012/08/24/скорость-передачи-данных-через-uart/** (14.07.21).

3. ATmega48P/V/88P/V/168P/V. — URL: **https://ww1.microchip.com/downloads/en/DeviceDoc/ATmega48P_88P_168P-DS40002065A.pdf** (10.07.21).