Программирование микроконтроллера ATmega на базе платы Arduino

11 лекция

«Программирование микроконтроллеров на языке C++»

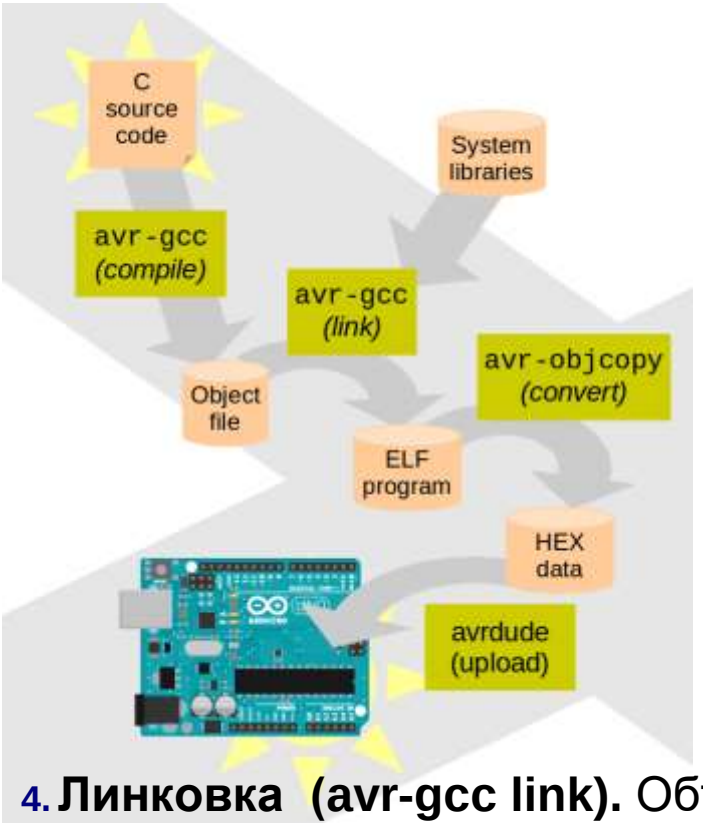
ОБЩИЕ СВЕДЕНИЯ

- Плата **Arduino** построена на базе микроконтроллера семейства **Atmega**. Программирование осуществляется в среде **Arduino IDE**, использующей язык, основанный на **C/C++**, но с упрощённым синтаксисом и встроенными библиотеками для работы с периферией.
- Каждая программа называется **скетчем** и сохраняется с расширением *.ino. Перед загрузкой кода необходимо выбрать модель платы и порт подключения. Программа загружается в энергонезависимую память (Flash) микроконтроллера через загрузчик (bootloader).
- Язык программирования Arduino называется Arduino C он имеет некоторые отличия по части написания кода, который компилируется и собирается с помощью avr-qcc, с особенностями, облегчающими написание работающей программы - имеется набор библиотек, включающий в себя функции и объекты.

ОБЩИЕ СВЕДЕНИЯ

- Программы, написанные программистом Arduino, сохраняются в файлах с расширением *.ino. Эти файлы перед компиляцией обрабатываются препроцессором Arduino. Также существует возможность создавать и подключать к проекту стандартные файлы C++.
- В текст скетча программист может не вставлять заголовочные файлы используемых стандартных библиотек. Эти заголовочные файлы добавит препроцессор Arduino в соответствии с конфигурацией проекта. Однако пользовательские библиотеки нужно указывать.
- Arduino IDE не предлагает никаких настроек компилятора и минимизирует другие настройки, что упрощает начало работы для новичков и уменьшает риск возникновения проблем; но присутствуют директивы препроцессора, такие как #define, #include и много других.

ПРОЦЕСС СБОРКИ и ЗАГРУЗКИ ПРОГРАММЫ



1. **Создание исходного кода (C source code)**
   Программа сохраняется в файле ***.ino** - это скетч Arduino, содержащий основной код с функциями setup() и loop().
2. **Преобразование в C++**
   IDE автоматически добавляет служебные заголовки и превращает файл ***.ino** во временный файл C++ с расширением ***.cpp**.
3. **Компиляция (avr-gcc compile)**
   Компилятор avr-gcc преобразует ***.cpp** в объектные файлы с расширением *.o, содержащие машинный код, но ещё не собранные в единую программу.
4. **Линковка (avr-gcc link).** Объектные файлы и системные библиотеки объединяются в ***.elf** файл с кодом, таблицами символов и данными для отладки.
5. **Конвертация (avr-objcopy convert).** ***.elf** файл преобразуется в ***.hex** файл, содержащий только машинный код для микроконтроллера.
6. **Загрузка (avrdude upload).** Программа avrdude записывает ***.hex** файл в память микроконтроллера.

Пример преобразования Arduino-программы в файл .cpp

Исходный скетч Blink.ino:

```cpp
void setup() {
  pinMode(13, OUTPUT);
}

void loop() {
  digitalWrite(13, HIGH);
  delay(1000);
  digitalWrite(13, LOW);
  delay(1000);
}
```

После обработки Arduino IDE (созданный временный файл `Blink.cpp`):

```cpp
#include <Arduino.h>    // Добавлено IDE

void setup();
void loop();

void setup() {
  pinMode(13, OUTPUT);
}

void loop() {
  digitalWrite(13, HIGH);
  delay(1000);
  digitalWrite(13, LOW);
  delay(1000);
}

// Главная функция, вызываемая при старте МК
int main(void) {
  init();               // Инициализация аппаратуры
  setup();              // Вызов пользовательской функции setup()
  while (true) {
    loop();             // Бесконечный цикл
    if (serialEventRun) serialEventRun();
  }
  return 0;
}
```

СИНТАКСИС

- Тела функций заключаются в фигурные скобки **{ }**
- Каждая команда заканчивается точкой с запятой **;**
- Метод применяется к объекту через **точку**.

  myservo.write (val); // применяем МЕТОД write к ОБЪЕКТУ myservo
- Вызов функции или метода всегда заканчивается скобками, даже если функция не принимает параметров.

  loop(); delay(15);
- Разделитель десятичных дробей – **точка. 0.25**
- Запятыми перечисляются аргументы функций и методов, а также членов массива. Также запятая является самостоятельным оператором.

  digitalWrite(3, HIGH);

  массив – **int myArray[] = {3, 4, 5 ,6};**

СИНТАКСИС

- Одиночный символ заключается в одиночные кавычки **‘a’**
- Строка заключается в двойные кавычки **“строка”**
- Имена переменных могут содержать **латинские буквы** в верхнем и нижнем регистре (большие и маленькие), **цифры** и **нижнее подчеркивание**.

  Пример: **myVal_35; byte _color;**
- Имена переменных **не могут начинаться с цифры**. Только с буквы или нижнего подчёркивания
- Регистр имеет значение, т.е. большая буква отличается от маленькой.

  Пример: переменные **val** и **Val** – не одно и то же.

СИНТАКСИС

К синтаксису также можно отнести комментарии, т.к. в разных языках они выделяются по-разному. Комментарий это обычный текст, который игнорируется на этапе компиляции. Комментарии нужны для пояснения кода, как себе самому, так и другим возможным его читателям. В C++ у нас два типа комментариев:

- Однострочный комментарий

  // однострочный комментарий

  // компилятор меня игнорирует =(

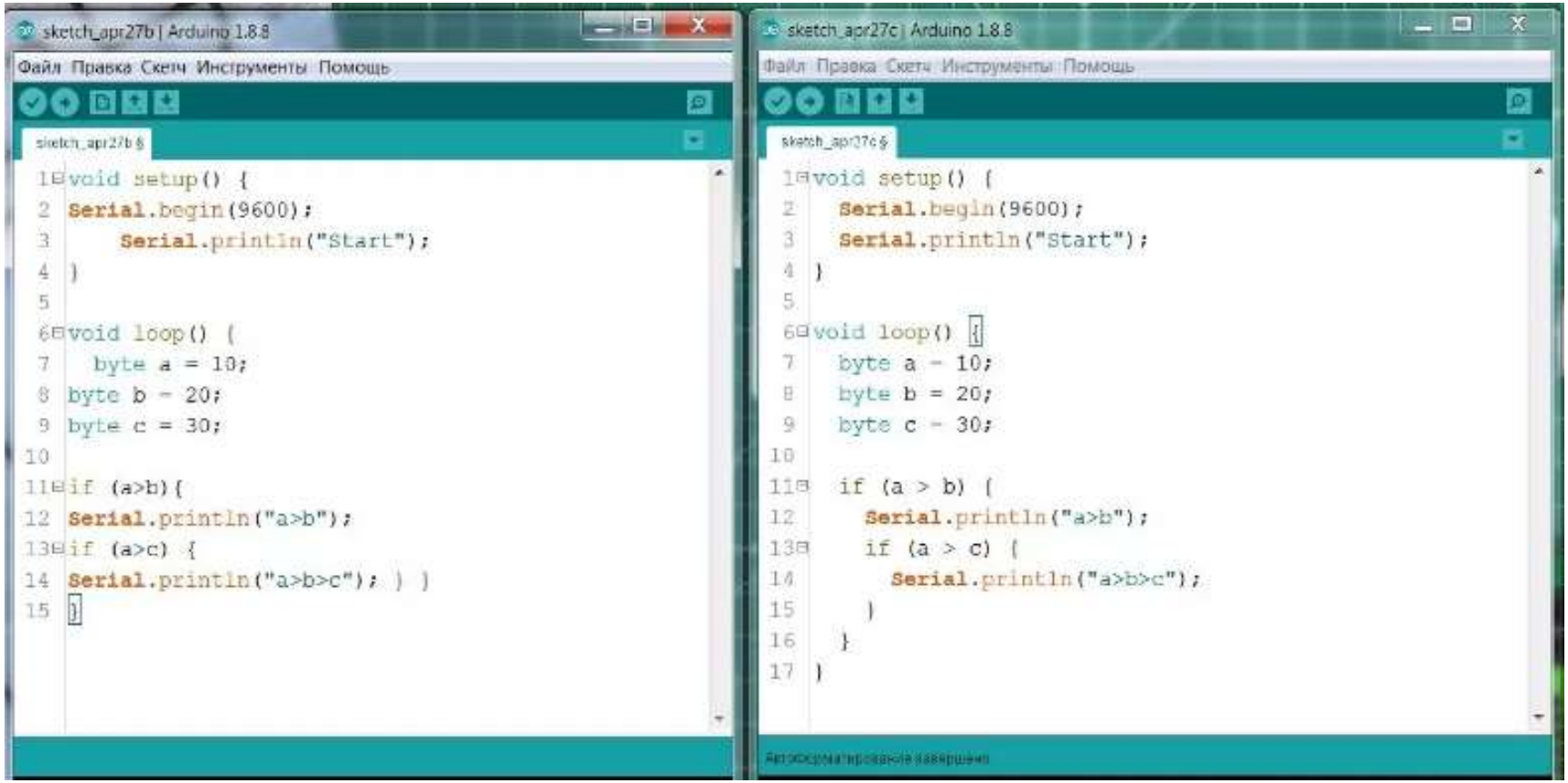
- Многострочный комментарий

  /* Многострочный

  комментарий */

ОФОРМЛЕНИЕ

- Форматирование. Есть такое понятие, как форматирование (выравнивание) кода, то есть соблюдение пробелов и интервалов. Сравните эти два примера. Какой код смотрится более понятно и наглядно?



ОФОРМЛЕНИЕ

Во всех средах разработки есть **автоформатирование кода**, оно работает как в процессе написания, так и по вызову. В Arduino IDE код форматируется комбинацией **Ctrl+T** на клавиатуре или в окне IDE **Инструменты/АвтоФорматирование**

- Между математическими действиями, знаками сравнения, присваивания и всем подобным ставится пробел
- Как и в обычном тексте, пробел ставится после и не ставится перед запятой, двоеточием, точкой с запятой.
- Отступ от левого края экрана – знак табуляции, код сдвигается вправо и на одном расстоянии формируются команды из одного блока кода. В Arduino IDE одна табуляция, равна двум пробелам
- Каждое действие выполняется с новой строки (автоформатирование это не исправляет)
- Фигурные скобки начала и окончания блока кода принято писать на отдельной строке. Также очень многие пишут открывающую скобку на строке с оператором, это экономит место

ИМЕНА ПЕРЕМЕННЫХ

- Имена переменных принято писать начиная с маленькой буквы, называть их так, чтобы было понятно.

  Пример: **value**

- Если название переменной состоит из нескольких слов, они разделяются верхним регистром первой буквы каждого нового слова.

  Пример: **myButtonState**

- Имена типов данных и классов принято писать с большой буквы.

  Пример: **Signal, Servo**

- Имена констант принято писать в верхнем регистре, разделение – нижнее подчеркивание.

  Пример: **MOTOR_SPEED**

- При написании библиотек и классов, имена внутренних переменных принято писать, начиная со знака нижнего подчёркивания.

  Пример: **_position**

ИМЕНА ФУНКЦИЙ и МЕТОДОВ

- Имена функций и методов принято начинать с глагола, кратко описывающего действие функции. Вот те из них, которые вы будете встречать постоянно:
- **get** – получить значение (getValue)
- **set** – установить значение (setTime)
- **print**, **show** – показать что-то
- **read** – прочитать
- **write** – записать
- **change** – изменить
- **clear** – очистить
- **begin**, **start** – начать
- **end**, **stop** – закончить, остановить

ГОРЯЧИЕ КЛАВИШИ

- **Скрытие частей кода** – сворачивайте длинные функции и куски кода для экономии места и времени на скроллинг.

  Файл/Настройки/Включить сворачивание кода

- Полезные комбинации клавиш на клавиатуре:
  - **Ctrl+←** , **Ctrl+→** – переместить курсор влево/вправо НА ОДНО СЛОВО
  - **Home** , **End** – переместить курсор в начало/конец строки
  - **Shift+←** , **Shift+→** – выделить символ слева/справа от курсора
  - **Shift+Ctrl+←** , **Shift+Ctrl+→** – выделить слово слева/справа от курсора
  - **Shift+Home** , **Shift+End** – выделить все символы от текущего положения курсора до начала/конца строки
  - **Ctrl+Z** – отменить последнее действие
  - **Ctrl+Y** – повторить отменённое действие
  - **Ctrl+C** – копировать выделенный текст
  - **Ctrl+X** – вырезать выделенный текст
  - **Ctrl+V** – вставить текст из буфера обмена
  - **Ctrl+U** – загрузить прошивку в Arduino
  - **Ctrl+R** – скомпилировать (проверить)
  - **Ctrl+Shift+M** – открыть монитор порта

СТРУКТУРА КОДА

Прежде чем переходить к структуре и порядку частей кода, нужно кое-что запомнить:

- Переменная любого типа должна вызываться **после** своего объявления. Иначе будет ошибка
- Объявление и использование классов или типов данных из библиотеки/файла должно быть **после** подключения библиотеки/файла
- Функция может вызываться **как до, так и после** объявления, потому что C++ компилируемый язык, компиляция проходит в несколько этапов, и функции “выделяются” отдельно, поэтому могут вызываться в любом месте программы

СТРУКТУРА КОДА

Базовая структура программы для Arduino довольно проста и состоит, по меньшей мере, из двух частей. В этих двух обязательных частях, или функциях, заключён выполняемый код.

При запуске Arduino IDE даёт нам заготовку в виде двух обязательных функций: **setup ()** и **loop ()**. Важно заметить, что обязательную в C++ функцию main() процессор Arduino создаёт сам. Результат того, что видит программист:

```
void setup()
{
  statements;
}

void loop()
{
  statements;
}
```

Структура стандартного скетча

```
// -------- ОПИСАНИЕ ---------
/*Данный скетч плавно крутит
Сервопривод */
// -------- НАСТРОЙКИ ---------
#define SERVO_PIN 13 // пин подключения
#define SERVO_SPEED 3 // скорость серво
// -------- СЛУЖЕБНЫЕ КОНСТАНТЫ ---------
const int PWM_RESOLUTION = 255; // разрешение ШИМ
// ------- БИБЛИОТЕКИ -------
#include <Servo.h>
Servo myservo;
// ------- ПЕРЕМЕННЫЕ -------
uint32_t servoTimer;
boolean servoDirection;
int servoAngle;
// --------- SETUP ----------
void setup()
{
myservo.attach(SERVO_PIN);
}
// ---------- LOOP ----------
void loop()
{
turnServo();
}
// --------- ФУНКЦИИ --------
void turnServo()
{……
```

1 — Описание прошивки, полезные ссылки, заметки, авторство

2 — Константы настройки (define или const)

3 — Служебные константы

4 — Подключаемые библиотеки, объявления типов и классов

5 — Глобальные переменные

6 — Функция setup ()

7 — Функция loop ()

8 — Пользовательские функции

Подключение библиотек и файлов

- В реальной работе вы очень часто будете использовать библиотеки или просто внешние файлы, они подключаются к главному файлу (файлу прошивки) при помощи директивы **#include**, данная директива сообщает препроцессору, что нужно найти и включить в компиляцию указанный файл. Рассмотрим пример:

  #include <Servo.h> // подключает библиотеку Servo.h

  #include “Servo.h” // тоже подключает библиотеку Servo.h

- В чём отличие **<>** и **“”**? Когда указываем название **“в кавычках”**, компилятор сначала ищет файл в папке со скетчем, а затем в папке с библиотеками. При использовании **<галочек>** компилятор ищет файл только в папке с библиотеками.

.

Заключение

Программирование микроконтроллера **ATmega на базе Arduino** основано на языке **C/C++** и реализуется в среде **Arduino IDE**.

Ключевыми элементами программы являются функции **setup()** и **loop()**, задающие структуру выполнения кода.

Следование правилам программирования, правильное использование **констант, библиотек и пользовательских функций** обеспечивает надёжность и читаемость программы.

Знание особенностей синтаксиса и структуры скетча является основой для создания **эффективных и устойчивых систем управления**.