**Программируем Pololu 3pi на Ubuntu/Debian, для начинающих**

6 мин

1.4K

[Чулан](https://habr.com/ru/hubs/closet/)

Вы только что приобрели робота от Pololu и в качестве операционной системы используете Linux Ubuntu или Debian, тогда эта статья для вас. В первую очередь статья рассчитана на новичков, которые решили попробовать себя в робототехнике.   
  
Начну с того, что доставили мне подобного робота, а точнее Pololu 3pi, и принялся я изучать что это за зверь такой. Из магазина в него была прошита демо-программа, с помощью которой можно было проверить все функции устройства. А умел он сверкать светодиодами, вращать двигателями, играть классическую мелодию через свою пищалку (Buzzer), выводить текст на двухстрочный дисплей по 8 символов в строчку и наконец распознавать под датчиками на нижней части устройства чёрные области (как пример, чёрные дорожки на белом фоне по которым можно двигаться).  
  
После изучения функционала устройства возникла следующая мысль «А давайте-ка мы запишем в него свою программу». И тут встал следующий вопрос как это сделать на Linux Ubuntu. В этом вопросе очень помог официальный сайт [www.pololu.com](http://www.pololu.com), зарубежные форумы и изучение исходников скачиваемых библиотек.  
Теперь вкратце расскажу как и что нужно сделать что бы записать свою программу в это устройство. Сразу уточню, что для программирования Pololu кроме самого устройства необходим специальный программатор с интерфейсом USB для подключения к компьютеру. Я использовал PGM03A — замечательный и недорогой программатор для микроконтроллеров построенных на базе AVR, в том числе и наш ATMega328P.  
Писать программы мы будем на C, но можно писать и на С++, для этого необходимо использовать соответствующий компилятор и библиотеки. Для начала ставим компилятор, дополнительные библиотеки и утилиты с помощью «sudo apt-get install»:  
  
gcc-avr — компилятор исходных кодов в машинный код контроллеров AVR, если вы пишете на C++ необходимо ставить соответствующие компилятор и библиотеки;  
avr-libs — библиотеки AVR для языка С;  
avra — ассемблер, необходим в составе AVR-GCC для получения ассемблерного кода;  
binutils-avr — дополнительные утилиты для сборки программ;  
avrdude — утилита для записи собранного бинарного hex-файла через программатор в контроллеры AVR.  
  
Это все библиотеки которые необходимы для сборки и записи бинарных hex-файлов в контроллер AVR. Теперь необходимо собрать и записать в контроллер простейшую программу.  
Для примера рассмотрим следующий код написанный на C, он будет циклично включать и отключать красный светодиод Pololu 3 pi. Пример был взят с официального сайта [www.pololu.com](http://www.pololu.com) и будет правильно работать только с микроконтроллером ATMega328P.  
  
#define F\_CPU 20000000UL // Baby Orangutan frequency (20MHz)  
#include  
#include  
void delayms( uint16\_t millis ) {  
while ( millis ) {  
\_delay\_ms( 1 );  
millis--;  
}  
}  
  
int main( void ) {  
DDRD |= 1 << DDD1; // set LED pin PD1 to output  
while ( 1 ) {  
PORTD &= ~( 1 << PORTD1 ); // LED off  
delayms( 900 ); // delay 900 ms  
PORTD |= 1 << PORTD1; // LED on  
delayms( 100 ); // delay 100 ms  
}  
return 0;  
}  
  
Этот фрагмент кода необходимо поместить в файл «BlinkLed.c». Расширение файла «\*.c» говорит о том, что перед нами исходный код программы написанный на языке С. Теперь компилируем файл в машинный код, для этого в консоли заходим в рабочую директории с файлом BlinkLed.c и выполняем следующую команду:  
  
$ avr-gcc -g -Wall -mcall-prologues -mmcu=atmega328p -Os -c -o BlinkLed.o BlinkLed.c  
  
И на выходе получаем файл BlinkLed.o с объектным кодом. Объектный код это исполняемый код с некоторыми вкраплениями дополнительной информации для линковщика, например, таблица символов. На этом шаге ваша программа уже собрана, осталось объединить её с другими объектными модулями линковщиком (если вы их используете) и собрать бинарный файл с расширением \*.hex, данные которого мы и будем писать в микропроцессор. В консоли нужно выполнить команды:  
  
$ avr-gcc -g -Wall -mcall-prologues -mmcu=atmega328p -Os BlinkLed.o -Wl,-gc-sections -lpololu\_atmega328p -Wl,-relax -o BlinkLed.obj  
$ avr-objcopy -R .eeprom -O ihex BlinkLed.obj BlinkLed.hex  
  
В директории появился файл BlinkLed.hex с командами для микроконтроллера ATMega328P, которые теперь необходимо в него прошить. Но прежде нужно узнать в какой девайс посылать команды. Для этого нам необходимо подключить к компьютеру программатор и найти новые устройства в директории /dev. В моём случае появилось 2 новых устройства: /dev/ttyACM0 и /dev/ttyACM1. Теперь подключаем к программатору микроконтроллер и выполняем команду для соответствующего устройства (устройство должно быть включено и заряда батарей должно хватить на время процесса прошивки):  
$ avrdude -c avrispv2 -p m328p -P /dev/ttyACM0 -e  
  
Команда читает данные из контроллера.  
avrdude: AVR device initialized and ready to accept instructions  
Reading | |||||||||||||||||||||||||||||||||||||||||||||| | 100% 0.01s  
avrdude: Device signature = 0x1e654f  
avrdude: erasing chip  
avrdude: safemode: Fuses OK  
avrdude done. Thank you.  
  
Если Вы увидели ответ от устройства, то тест соединения фактически прошёл и теперь можно смело прошивать наши подготовленные инструкции. Выполняем в консоли:  
  
$ avrdude -p m328p -c avrisp2 -P /dev/ttyACM0 -U flash:w:BlinkLed.hex  
  
Через несколько секунд инструкции в устройство будут успешно прошиты, контроллер автоматически перезагружен и Вы увидите как Pololu 3pi активно подмигивает вам своим светодиодом. Для автоматизации процесса сборки приложения можно написать Makefile.  
Мы научились прошивать микроконтроллер из исходного кода написанного на языке С, теперь для более комфортной работы с устройством необходимо подключить библиотеку Pololu, в которой уже реализованы низкоуровневые операции управления компонентами робота. Скачиваем последнюю версию библиотеки с официального сайта [www.pololu.com](http://www.pololu.com), на момент написания статьи библиотека доступна на странице [www.pololu.com/docs/0J20/3](http://www.pololu.com/docs/0J20/3).  
  
$ wget www.pololu.com/file/download/libpololu-avr-100607.zip?file\_id=0J381  
  
Распаковываем:  
  
$ unzip libpololu-avr-yymmdd.zip, где «yymmdd» необходимо заменить на версию вашей библиотеки.  
  
Заходим в корневую директорию с распакованной библиотекой и выполняем команду «make» для сборки пакетов. После сборки осталось её поставить, но тут меня ожидала пара сюрпризов. Во-первых необходимо открыть файл «Makefile», найти 2 строчки:  
  
# LIB := /usr/lib/avr/lib  
# INCLUDE\_POLOLU := /usr/lib/avr/include  
  
И раскомментировать их, проверить существование директорий в вашей системе и при необходимости изменить. После этого можно выполнять команду «sudo make install» для установки библиотеки. При этом в «/usr/lib/avr/lib » должны скопироваться пакеты с расширением «\*.a» для различных версий микроконтроллеров и в «/usr/lib/avr/include» заголовочные файлы «popolu/\*.h» для подключения необходимого функционала библиотеки.  
Тут меня ждал второй сюрприз, заголовочные файлы скопировались в директорию указанную в «NCLUDE\_POLOLU», то есть в «/usr/lib/avr/include», хотя поиск их ведётся в директории «/usr/lib/avr/include/pololu». Проблема решилась созданием директории «/usr/lib/avr/include/pololu» и повторным копированием в неё всех файлов «pololu/\*.h».  
Напоследок напишем ещё одну маленькую программку для закрепления материала, с использованием библиотеки Pololu. Для этого в начале файла программы подключим библиотеку написав такую строчку "#include <pololu/3pi.h>". Ниже размещаю код с комментариями, который необходимо поместить в файл «MyRobot.c». Как видите кода получилось меньше, но функционала мы задействовали гораздо больше:  
  
#include <pololu/3pi.h> // подключаем Pololu  
#include <stdlib.h> // необходима для генератора случайных чисел  
int main()  
{  
print("Hello :)"); // печатаем на дисплей  
lcd\_goto\_xy(0,1); // переводим коретку на вторую строчку  
int val;  
while(1)  
{  
green\_led(0); // отключаем зелёный светодиод  
red\_led(0); // отключаем красный светодиод  
val = random()%3; // получаем случайное число из {0,1,2}  
if(val == 0) {  
green\_led(1); // включаем зелёный светодиод  
red\_led(1); // включаем красный светодиод  
}  
delay\_ms(1000); //пауза 1 секунда  
}  
}  
  
Для автоматической сборки и прошивки программы мы создадим специальный файл Makefile в который необходимо поместить следующий текст, все отступы необходимо сохранить в виде 1 табуляции от начала строки:

DEVICE = atmega328p

AVRDUDE\_DEVICE = m328p

CFLAGS=-g -Wall -mcall-prologues -mmcu=$(DEVICE) -Os

CC=avr-gcc

OBJ2HEX=avr-objcopy

LDFLAGS=-Wl,-gc-sections -lpololu\_$(DEVICE) -Wl,-relax

PORT=/dev/ttyACM0

AVRDUDE=avrdude

TARGET=MyRobot

all: $(TARGET).hex

clean:

rm -f \*.o \*.hex \*.obj \*.hex

%.hex: %.obj

$(OBJ2HEX) -R .eeprom -O ihex $< $@

%.obj: %.o

$(CC) $(CFLAGS) $< $(LDFLAGS) -o $@

install: $(TARGET).hex

$(AVRDUDE) -p $(AVRDUDE\_DEVICE) -c avrisp2 -P $(PORT) -U flash:w:$(TARGET).hex

Теперь для сборки программы достаточно набрать команду «make», а для прошивки её в устройство соответственно «make install».  
Настало время пройтись по многочисленным примерам, которые вы найдёте в распакованных файлах библиотеки, и там же лежат уже компилированные hex файлы, если вдруг у вас возникнут проблемы с компилятором. В качестве IDE для написание исходного кода на С я использую NetBeans с установленым плагином C/C++.   
  
Спасибо за внимание. Буду благодарен, если вы сообщите о неточностях и ошибках в статье.

Теги:

* [popolu](https://habr.com/ru/search/?target_type=posts&order=relevance&q=%5bpopolu%5d)
* [3pi](https://habr.com/ru/search/?target_type=posts&order=relevance&q=%5b3pi%5d)
* [робот](https://habr.com/ru/search/?target_type=posts&order=relevance&q=%5b%D1%80%D0%BE%D0%B1%D0%BE%D1%82%5d)
* [ATMega328P](https://habr.com/ru/search/?target_type=posts&order=relevance&q=%5bATMega328P%5d)
* [avr-gcc](https://habr.com/ru/search/?target_type=posts&order=relevance&q=%5bavr-gcc%5d)

Хабы:

* [Чулан](https://habr.com/ru/hubs/closet/)