

Pinboard II

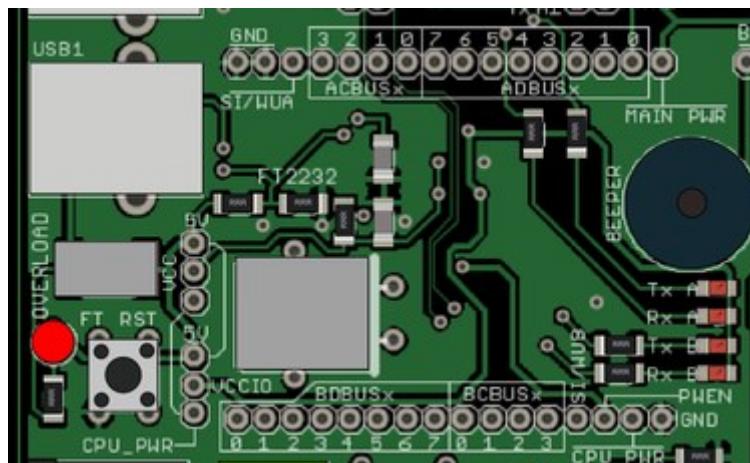
Инструкция по использованию программатора 2FTBV

Интерфейсная часть построена на базе микросхемы FT2232D (или C) которая способна помимо организации двух виртуальных COM портов предоставлять еще и Bitbang сервис. Т.е. у микросхемы есть два восьмиразрядных порта ADBUS и BDBUS выводами которых мы можем управлять произвольно, через драйвер FTDI. Как если бы это был микроконтроллер.

А раз так, то ничего не мешает нам использовать эти выводы для программирования микроконтроллеров AVR. Благо AVRDUDE поддерживает эту особенность.

Железная часть

Для наших целей будет использоваться один из двух портов. ADBUS или BDBUS



При подключении микросхемы FTDI она образует два устройства FTn и FTn+1, где n зависит от наличия других FTDI микросхем подключенных к компьютеру в текущий момент. Если таковых нет, то это будут устройства FT0 для ADBUS и FT1 для BDBUS соответственно. Если же в системе есть другие FTDI устройства , скажем шнурок для сотового телефона на FT232RL, то это будут FT1 и FT2, т.к. FT0 будет занят шнурком.

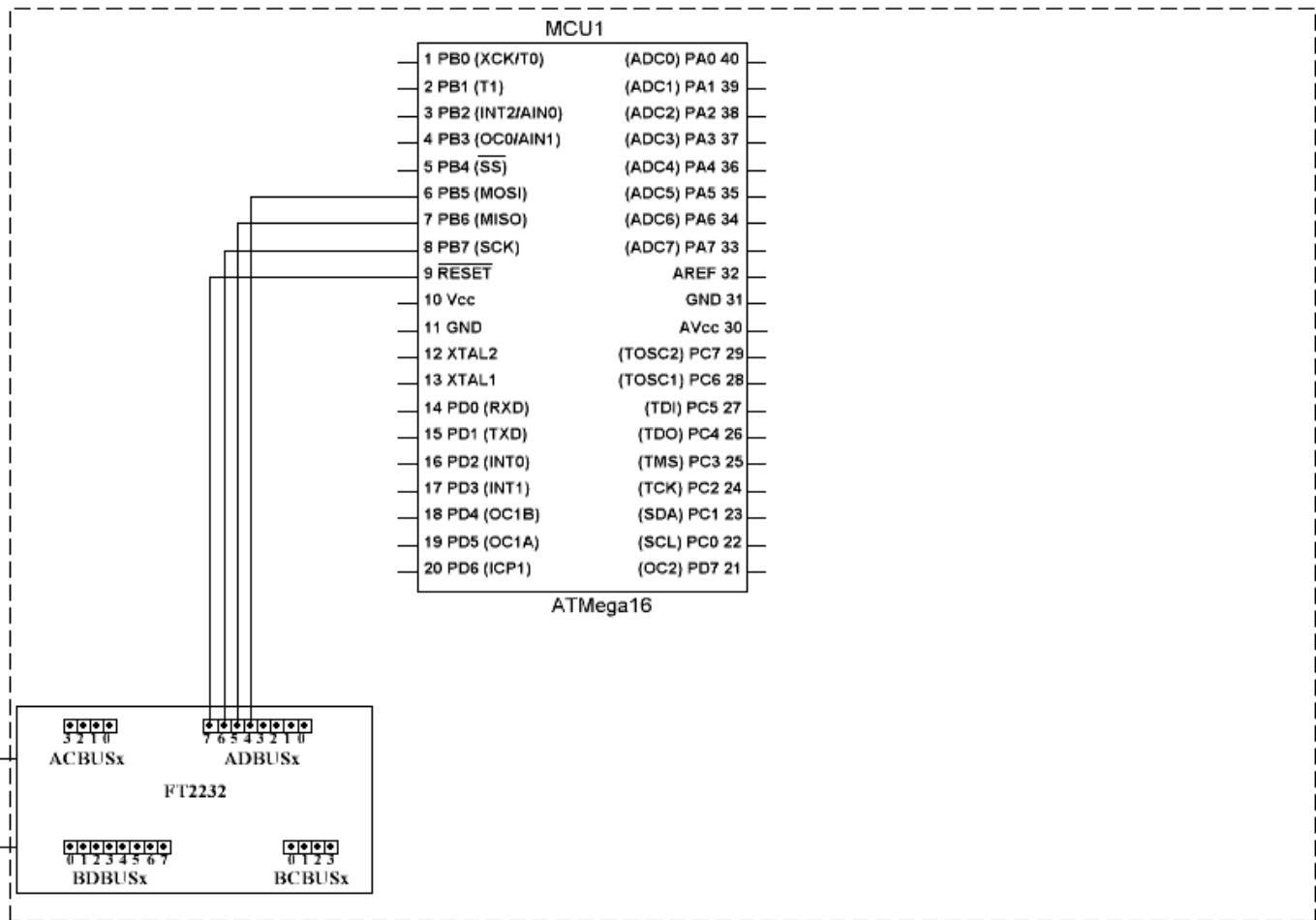
Т.к. портом мы можем пользоваться произвольно, то и выводы для программирования AVR тоже назначаем произвольным образом. Я предпочел выставить их в том порядке, в каком они идут по корпусу ATMEGA16

Напомню, что порядок там такой: RESET, SCK, MISO, MOSI

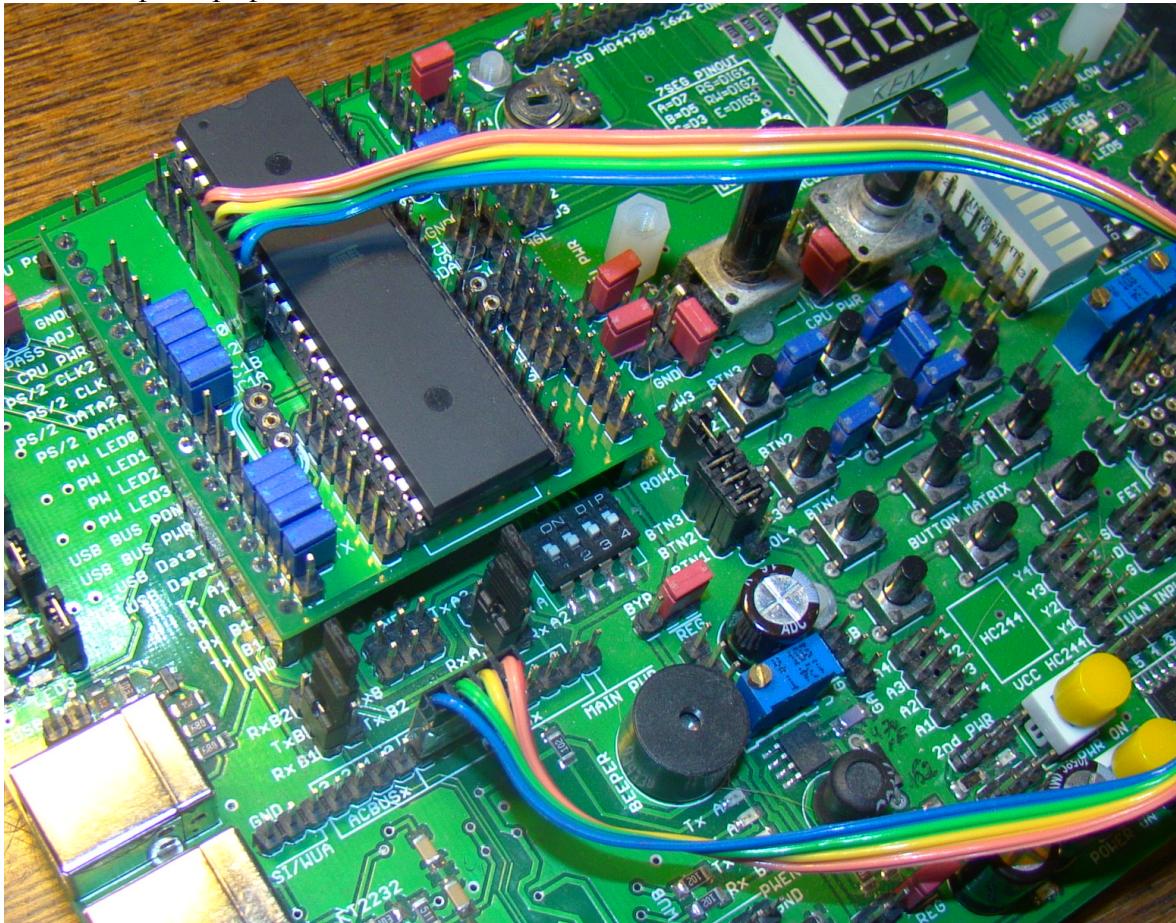
Потому, для удобства, задаем соответствие выводов таким образом:

- RESET (9 вывод ATmega16) – ADBUS 7
- SCK (8 вывод ATmega16) – ADBUS 6
- MISO (7 вывод ATmega16) – ADBUS 5
- MOSI (6 вывод ATmega16) – ADBUS 4

Для прошивки контроллера подключаем проводки. Вот по такой схеме.



Или, как показано на фотографии:



С таким же успехом можно применить и канала B, соединив соответственно.

Точно таким же способом можно будет прошить и любой другой AVR микроконтроллер (кроме тех, что с интерфейсом PDI, это самые новые серии вроде ATxMega), подключить четыре провода интерфейса, а плюс к этому еще землю (GND) и, дополнительно, если прошиваемая схема не имеет своего источника, питание.

Программная часть

Дальше дело за программатором, точнее за софтом. Тут я применяю проверенную годами **avrdude**. Надежная, удобная. Правда консольная, но это не недостаток, это фича. Впрочем, обо всем по порядку.

Итак avrdude надо настроить под наш программатор. В архиве **уже настроенная и готовая к употреблению версия**, но мало ли, может вы свою сборку делаете.

Итак, рассмотрим архив вблизи. Состоит она из

1. **avrdude.exe** – программа прошивальщик. Консольная утилита.
2. **avrdude.conf** – файл конфигурации, тут описаны все программаторы. Не стоит его менять без нужды.
3. **ftd2xx.dll** – библиотека для работы с микросхемой FTDI
4. **PBSelf16 - ft2232.cmd** – пакетный файл для прошивки бутлоадера на Мегу16 (и только на нее!)
5. **PBSelf32 - ft2232_32.cmd** – пакетный файл для прошивки бутлоадера на Мегу32 (и только на нее!)
6. **m16boot.hex** – файл прошивки с бутлоадером для ATMega16.
7. **m32boot.hex** – файл прошивки с бутлоадером для ATMega32
8. **boot_com3.cmd** – командный файл заливки тестовой программы через бутлоадер.
9. **MainDemo16-RTOS.hex** – файл с прошивкой тестовой демопрограммы, которая шла изначально.

Как накатить бутлоадер на свежекупленный контроллер ATMega16 или ATMega32?

0. Вставляем свежий контроллер в панельку модуля.
1. Подсоединяем проводки программатора, как показано на схемах и фотографиях выше.
2. Убеждаемся, что USB кабель включен, а в системе появились COM порты.
3. Убеждаемся, что питание на плату подано, на CPU Power напряжение около 4...5 вольт, не меньше.
4. Запускаем PBSelf16 - ft2232.cmd и наблюдаем за процессом. (Для меги32 это будет PBSelf32 - ft2232_32.cmd)

Готово!

Содержимое командного файла PBSelf16 - ft2232.cmd следующее:

```
avrdude.exe -p m16 -c 2ftbb -P ft0 -B 4800 -U hfuse:w:154:m -U lfuse:w:228:m -U lock:w:63:m  
avrdude.exe -p m16 -c 2ftbb -P ft0 -U flash:w:m16boot.hex:a
```

```
@echo off  
color 0A  
echo ATMega16 Burned ok! Go to Boot!  
pause  
  
boot_com3
```

Первая строчка накатывает нужные фуз биты. Вторая заливает прошивку, а дальше пауза с ожиданием нажатия клавиши и смена цвета, чтобы показать, что действие произошло. А в конце вызывается командный файл boot_com3, но это уже не обязательно. Я это сделал для удобства, чтобы одним тыком накатывать при проверке бутлоадер и сразу же следом за ним, через бутлоадер же заливать тестовую прошивку. За заливку прошивки через бутлоадер и отвечает этот boot_com3. Этую строчку вы можете смело удалить.

Запускаете командный файл и по экрану пробежит длинный лог с множеством прогресс баров. На основании него можно многое сказать о процессе прошивки. Если какие то проблемы возникли, то быстро понять на каком этапе.

Ниже пример такого лога с моими комментариями по каждому событию.

```
d:\Work\Soft\dudeitbang>avrduude.exe -p m16 -c 2ftbb -P ft0 -B 4800 -U hfuse:w:154:m -U lfuse:w:228:m -U lock:w:63:m
//это команда переданная avrdude
// Цель этой команды и данного набора FUSE битов выставить тактовую частоту на 8МГц, чтобы дальше можно было шить на максимальной скорости.
// обратите внимание на параметры передаваемые:
// -p m16 - тип контроллера Mega16
// -c 2ftbb – тип программатора 2ftbb
// -P используемое устройство ft0
// -B 4800 скорость битклоха 4800. Это важно! Т.к. многие контроллеры по дефолту работают на очень низкой частоте и не могут прожевать
// полную скорость FTI
// -U hfuse:w:154:m Запись старшего байта фузов. -U ключ работы с памятью, hfuse – какую память мы адресуем, w – запись. 154 – непосредственное
// значение байта hfuse в десятичном виде. Т.е. все эти 1 и 0 фуз битов которые мы глядим в даташите образуют байт, способный принимать значение
// от 0 до 255, зависимости от комбинации битов. Вот это число тут и внесено, преобразованное в десятичную форму.
// m – manual, признак того, что значения вот они, прям тут. А не из файла.
// -U lfuse:w:228:m и -U lock:w:63:m аналогично, но для младшего байта фузов и для байта защиты
```

```
avrduude.exe: BitBang OK
// программатор на FTI сказал, что он тут есть и с ним все OK
```

```
avrduude.exe: pin assign miso 5 sck 6 mosi 4 reset 7
//avrduude еще раз напомнила нам какой вывод на какой ноге висит.
```

```
avrduude.exe: drain OK
ft245r: bitclk 4800 -> ft baud 2400
// Avrdude сообщила нам, что применена пониженная скорость. Шить будем на 2400.
```

```
avrduude.exe: AVR device initialized and ready to accept instructions
// Готово. Поехали!
```

```
Reading | ##### | 100% 0.00s
// Читаем сигнатуру.
```

```
avrduude.exe: Device signature = 0x1e9403
// Считали. Соответствует меге16, если не будет соответствовать, то дудка ругнется
```

```
avrduude.exe: reading input file "154"
// берем наше значение первого байта фуз битов
```

```
avrduude.exe: writing hfuse (1 bytes):
```

```
Writing | ##### | 100% 0.03s
// записываем
```

```
avrduude.exe: 1 bytes of hfuse written
avrduude.exe: verifying hfuse memory against 154:
avrduude.exe: load data hfuse data from input file 154:
avrduude.exe: input file 154 contains 1 bytes
avrduude.exe: reading on-chip hfuse data:
// avrdude говорит что один байт она записала, теперь дескать возьмем его прочитаем и сравним, что все ок.
```

```
Reading | ##### | 100% 0.03s
```

```
avrduude.exe: verifying ...
// Читаем, сравниваем. Если появилось verifyng значит все ок.
```

```
avrduude.exe: 1 bytes of hfuse verified
// О чём нам дудка и говорит.
```

```
avrduude.exe: reading input file "228"
avrduude.exe: writing lfuse (1 bytes):
```

```
Writing | ##### | 100% 0.03s
```

```
avrduude.exe: 1 bytes of lfuse written
avrduude.exe: verifying lfuse memory against 228:
avrduude.exe: load data lfuse data from input file 228:
avrduude.exe: input file 228 contains 1 bytes
avrduude.exe: reading on-chip lfuse data:
```

```
Reading | ##### | 100% 0.03s
```

```
avrduude.exe: verifying ...
avrduude.exe: 1 bytes of lfuse verified
avrduude.exe: reading input file "63"
avrduude.exe: writing lock (1 bytes):
```

```
Writing | ##### | 100% 0.03s
```

```
avrduude.exe: 1 bytes of lock written
avrduude.exe: verifying lock memory against 63:
avrduude.exe: load data lock data from input file 63:
avrduude.exe: input file 63 contains 1 bytes
avrduude.exe: reading on-chip lock data:
```

```
Reading | ##### | 100% 0.03s
```

```
avrduude.exe: verifying ...
avrduude.exe: 1 bytes of lock verified
```

```
avrduude.exe: safemode: Fuses OK
```

```
avrduude.exe done. Thank you.
```

```
// Таким же макаром зашивается, считывается и верифицируются и второй байт с байтом защиты
```

```
// Теперь следующая команда из нашего командного файла. Именно она вкатывает прошивку.
```

```
d:\Work\Soft\dudebitbang>avrduude.exe -p m16 -c 2ftbb -P ft0 -U flash:w:m16boot.hex:a
// параметры схожи. Разница только в том, что тут у нас параметр -U другой.
// пишем во flash, указано имя файла m16boot.hex (можно и полный путь сюда вписать). А также параметр «а» говорящий дудке, чтобы она
// автоматом определила тип считываемого файла. Он может быть как intel hex, так и bin. Вот это дудка и просчет.
// обрати внимание, что теперь нет ключа -В поникающего скорость. Он и не нужен. Т.к. частоту тактовую мы выставили прошлой командой.
```

```
avrduude.exe: BitBang OK
avrduude.exe: pin assign miso 5 sck 6 mosi 4 reset 7
avrduude.exe: drain OK
```

```
ft245r: bitclk 230400 -> ft baud 115200
// скорость на 115200 и полетели!
```

```
avrduude.exe: AVR device initialized and ready to accept instructions
```

```
Reading | ##### | 100% 0.00s
```

```
avrduude.exe: Device signature = 0x1e9403
avrduude.exe: NOTE: FLASH memory has been specified, an erase cycle will be performed
To disable this feature, specify the -D option.
// дудка предупреждает, что флеш память перед записью будет стерта. Чтобы это выключить надо применить ключ -D. Пусть стирает. Это полезно.
```

```
avrduude.exe: erasing chip
// стираем чип.
```

```
ft245r: bitclk 230400 -> ft baud 115200
avrduude.exe: reading input file "m16boot.hex"
avrduude.exe: input file m16boot.hex auto detected as Intel Hex
avrduude.exe: writing flash (16310 bytes):
```

```
Writing | ##### | 100% 9.05s
```

```
avrduude.exe: 16310 bytes of flash written
// шьем.
```

```
avrduude.exe: verifying flash memory against m16boot.hex:
avrduude.exe: load data flash data from input file m16boot.hex:
avrduude.exe: input file m16boot.hex auto detected as Intel Hex
avrduude.exe: input file m16boot.hex contains 16310 bytes
avrduude.exe: reading on-chip flash data:
```

```
Reading | ##### | 100% 4.44s
```

```
// считываем зашитое
```

```
avrduude.exe: verifying ...
avrduude.exe: 16310 bytes of flash verified
// Верифицируем. И все ОК.
```

```
avrduude.exe: safemode: Fuses OK
```

```
avrduude.exe done. Thank you.
```

```
ATmega16 Burned ok! Go to Boot!
```

Для продолжения нажмите любую клавишу . . . Тут то и запустится boot_com3 его задача накинуть через бутлоадер тестовую прошивку, проверив тем самым работу бутлоадера. Вам это не обязательно, но пример я разжулю до конца, мало ли, вдруг пригодится. Содержимое BootCom3 следующее:

```
avrduude -p m16 -c avr109 -P COM3 -U flash:w:MainDemo16-RTOS.hex
```

```
pause
```

```
"PBSelf16 - ft2232.cmd"
```

Та же самая avrdude, о обратите внимание на ключ «-с». Дудка коннектится уже к другому программатору! В качестве программатора выступает уже не 2FTBB, а avr109! Наш бутлоадер он, с точки зрения avrdude, программатор AVR109 висящий на порту COM3.

Да, тут есть одно замечание. У меня FTDI образует два ком порта. Это COM2 и COM3. У вас может быть по другому. Скажем сидет на COM3 и COM4, в зависимости от наличия свободных COM портов. Это надо бы учесть в командном файле. Так что будте внимательны.

И еще один немаловажный момент. Дело в том, что когда FTDI шьет битбангом, то этот канал блокируется и переводится в режим в котором COM порт недоступен. Выход из него только через RESET FTDI кнопкой FT RST. Поэтому если вы хотите прошить бутлоадер битбангом, а потом сразу же закатать тестовую прошивку через бутлоадер. То выбирайте COM который не пересекается с каналом Bitbang.

Т.е. например у нас FTDI образовала два канала:

- A) FT0 - COM2
- B) FT1 – COM3

Мы шьем битбангом через FT0 и по оконании прошивки у нас порт COM2 будет недоступен, т.к. канал A переключится в битбанг режим. Так что бутлоадеру надо будет ломиться через COM3. Ну и, соответственно, перемычки выставить так, чтобы COM3 канала B вел до Rx-Tx контроллера.

Поглядим же на лог AVRDUDE при работе с бутлоадером.

```
Connecting to programmer: .
```

```
// ждем ответа от программатора. В этот момент контроллер должен быть сброшен, чтобы запустился бут и он был готов ответить.  
// Дудка непрерывно шлет комманд S на скорость 19200, ожидая ответа AVRBOOT
```

```
Found programmer: Id = "AVRBOOT"; type = S  
Software Version = 0.8; No Hardware Version given.  
Programmer supports auto addr increment.  
Programmer supports buffered memory access with buffersize=128 bytes.  
// Связались с бутлоадером. Бут подтвердил, что он тут есть, а также выдал режимы в которых он может работать
```

```
Programmer supports the following devices:
```

```
Device code: 0x75
```

```
// Также узнали код устройства
```

```
avrduude: AVR device initialized and ready to accept instructions
```

```
Reading | ##### | 100% 0.02s
```

```
avrduude: Device signature = 0x1e9403
```

```
avrduude: NOTE: FLASH memory has been specified, an erase cycle will be performed
```

```
// Прочитали сигнатуру устройства. На самом деле она тупо зашифрована в бутлоадере
```

```
To disable this feature, specify the -D option.
```

```
avrduude: erasing chip
```

```
avrduude: reading input file "MainDemo16-RTOS.hex"
```

```
avrduude: input file MainDemo16-RTOS.hex auto detected as Intel Hex
```

```
avrduude: writing flash (852 bytes):
```

```
Writing | ##### | 100% 0.56s
```

```
// стерли чип и записали туда новую прошивку. Ту самую MainDemo16-RTOS.hex
```

```
avrduude: 852 bytes of flash written
```

```
avrduude: verifying flash memory against MainDemo16-RTOS.hex:
```

```
avrduude: load data flash data from input file MainDemo16-RTOS.hex:
```

```
avrduude: input file MainDemo16-RTOS.hex auto detected as Intel Hex
```

```
avrduude: input file MainDemo16-RTOS.hex contains 852 bytes
```

```
avrduude: reading on-chip flash data:
```

```
Reading | ##### | 100% 0.56s
```

```
avrduude: verifying ...
```

avrdude: 852 bytes of flash verified

// Считали и верифицировали.

avrdude done. Thank you.

Для продолжения нажмите любую клавишу . . .

Если вы тут нажмете любую клавишу, то снова запустится PBSelf16 - ft2232.cmd ;) Т.к. я обычно шью сразу десятками плат, чтобы удобней было. Можете смело удалять строчку запускающую PBSelf16 - ft2232.cmd в файле boot_com3

Ну и, напоследок, небольшая справочка по AVRDUDE

avrdude.conf содержит в себе описания программаторов и того, что у нас к чему подключено. Там должен быть описан и наш программатор:

Запись вот такого вида:

```
#FTDI_Bitbang
programmer
id      = "2ftbb";
desc    = "FT232R Synchronous BitBang";
type    = ft245r;
miso   = 5; # ADBUS 5
sck    = 6; # ADBUS 6
mosi   = 4; # ADBUS 4
reset  = 7; # ADBUS 7
;
```

id - это то как мы его будем вызывать из командной строки.

desc – описание для выбора типа работы.

type – тип программатора. Впервые такое сделали на FT245R

Ну и дальше идет описание выводов какие куда. Тут, думаю, пояснений не нужно. Хотите сделать другую распиновку? Перепишите как нужно. Я же предпочитаю так.

-р – ключ определяющий тип прошиваемого контроллера. m16 это, в данном случае, ATMega16, обозначения других контроллеров можно посмотреть в документации на программу avrdude (или запустить ее без параметров в командной строке и она сама подскажет что к чему).

-с – ключ после которого идет имя программатора. В данном случае это “2ftbb” Если покопаться в файле avrdude.conf то можно найти там секцию Pinboard II в которой указано описание выводов микросхемы FT232B применительно к плате Pinboard II

-P – имя порта на котором висит программатор. В данном случае это FTDI и порт ft0, но если в системе несколько микросхем ftdi то номер может быть другим. В этом случае номер порта определяется экспериментально.

-B скорость прошивки. Дело в том, что на медленных тактовых скоростях (1МГц, стоящий по умолчанию на ATMega16) микроконтроллер не успевает отвечать на сигналы программатора и процесс прошивки завершается ошибкой. Чтобы этого избежать мы принудительно занижаем скорость. Не путать с ключом -b РЕГИСТР ВАЖЕН!

-U тип памяти куда мы будем записывать. Формат параметра этого ключа следующий

[тип памяти]:[тип операции]:[данные]:[тип данных]

Тип памяти бывает

- hfuse,lfuse – это старший и младший байт FUSE
- flash – это основная память программ,
- eeprom – память EEPROM.
- lock – байт защитных битов.
- calibration – байты калибровки RC генератора.

Тип операции бывает

- w - означает, что мы пишем в выбранную память.
- r - читаем выбранную память. Чтение идет в файл (например в lfuse,h)

Данные:

Либо путь к файлу (если он в том же каталоге, то просто имя файла), либо непосредственные данные (в случае с одним байтом FUSE)

Тип данных:

- m – ручной ввод данных (тот самый байт, записанный числом). Если число с 0-восьмеричная система, 0x... - шестнадцатеричная система. В ином случае – десятичная.
- a – автоматическое определение (только для записи, при чтении надо указывать в каком виде мы хотим получить данные)
- i - Intel Hex формат.

Одновременно может быть несколько ключей -U с разными типами памяти.

Подробное описание ключей программы avrdude можно найти тут:

http://www.nongnu.org/avrdude/user-manual/avrdude_4.html

Если внимательно посмотреть пакетные файлы, то будет видно, что вначале я на медленной скорости меняю FUSE биты как мне нужно (переставляю на повышенную скорость), после чего на полной скорости заливаю FLASH.

Возможные ошибки:

В консоли нет длинного лога, а сразу же пишется что то вроде:

```
d:\Work\Soft\dudebitbang>avrdude.exe -p m16 -c pinb -P ft0 -B 4800 -U hfuse:w:1  
54:m -U lfuse:w:228:m -U lock:w:63:m  
avrdude.exe: ft0 open failed
```

```
d:\Work\Soft\dudebitbang>avrdude.exe -p m16 -c pinb -P ft0 -U flash:w:m16boot.hex:a  
avrdude.exe: ft0 open failed
```

```
d:\Work\Soft\dudebitbang>pause  
Для продолжения нажмите любую клавишу . . .
```

Такая ошибка возникает если программатор не обнаружил ни одной микросхемы FTDI в системе.

Проверьте включено ли питание платы.

Проверьте правильно ли установился драйвер FTDI (в системе должен быть виртуальный COM порт)

Возможно вы поторопились запустить командный файл и операционная система в тот момент еще не обнаружила новое устройство.

При попытке прошить контроллер система выдает что то вроде:

```
d:\Work\Soft\dudebitbang>avrdude.exe -p m16 -c pinb -P ft0 -B 4800 -U hfuse:w:1
54:m -U lfuse:w:228:m -U lock:w:63:m
avrdude.exe: BitBang OK
avrdude.exe: pin assign miso 6 sck 5 mosi 3 reset 7
avrdude.exe: drain OK

ft245r: bitclk 4800 -> ft baud 2400
avrdude.exe: ft245r_program_enable: failed
avrdude.exe: initialization failed, rc=-1
    Double check connections and try again, or use -F to override
    this check.
```

Сейчас программатор нормально определился системой, но микроконтроллер не отвечает на его команды. Возможные причины неисправности:

- Неправильно подключен микроконтроллер. Проверьте соответствие выводов FTBB и прошиваемого микроконтроллера.
- Плохой контакт или обрыв провода.
- Не угадали с FT девайсом. Т.е. avrdude постучался в ft0, а на самом деле наш программатор сел на ft1. Тут можно перебрать все FT от 0 до 9.
- Микроконтроллер заблокирован FUSE битами или выставлен на неактивный способ тактирования.
- В этом случае надо вспомнить каким образом последний раз зашивались FUSE биты или методом подбора попытаться достучаться до микроконтроллера (подключить кварц, внешний генератор, RC цепочку и т.д.)

Графические оболочки для AVRDUDE

Их существует великое множество. Самые удачные из всех которые я видел это

SinaProg. Подробное описание которой можно найти тут:

<http://easyelectronics.ru/sinaprof-graficheskaya-obolochka-dlya-avrdude.html>

И ADS, написанная для себя одним из наших читателей, с учетом моих рекомендаций. Правда в процессе консультирования я показал как клево и удобно пользоваться командными файлами и оболочка для него стала ненужна. Так что не уверен, что проект не сдох ;) Найти ее последний раз можно было тут:

<http://forum.easyelectronics.ru/viewtopic.php?f=16&t=8498>