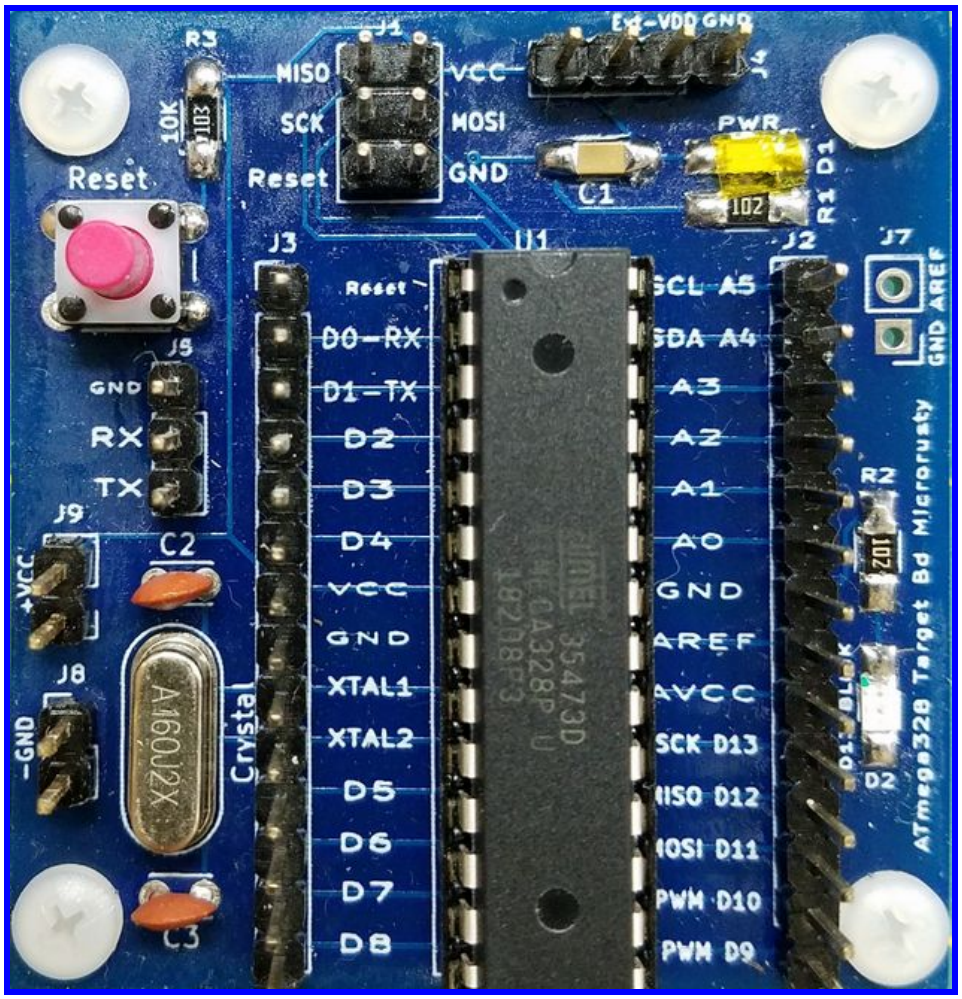
# ATmega328p Target Bd - USB Programmer
# User Manual Version 1




**Rusty Cain**
**Microrusty**
**August 28, 2018**

# ATmega328p Target Bd - AVRdude

## Programming using AVRdude

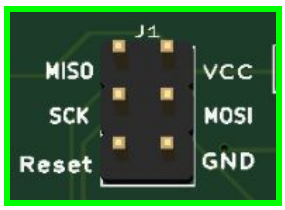
**Download AVRdude**: https://www.nongnu.org/avrdude/

- Create a new folder for Avrdude and install Avrdude in that folder.
- Open Terminal (Linux). Open CMD.exe (Windows Command Line)
- Find and Open the AVRDude program.
- Windows Example: *C: Documents\AVRDude-V6-3. cd Documents\AVRDude-V6-3*
- Type "*avrdude*" The avrdude help menu should display a list of commands.

```
Usage: avrdude [options]
Options:
  -p <partno>                Required. Specify AVR device.
  -b <baudrate>              Override RS-232 baud rate.
  -B <bitclock>              Specify JTAG/STK500v2 bit clock period (us).
  -C <config-file>           Specify location of configuration file.
  -c <programmer>            Specify programmer type.
  -D                         Disable auto erase for flash memory
  -i <delay>                 ISP Clock Delay [in microseconds]
  -P <port>                  Specify connection port.
  -F                         Override invalid signature check.
  -e                         Perform a chip erase.
  -O                         Perform RC oscillator calibration (see AVR053).
  -U <memtype>:r|w|v:<filename>[:format]
                             Memory operation specification.
                             Multiple -U options are allowed, each request
                             is performed in the order specified.
  -n                         Do not write anything to the device.
  -V                         Do not verify.
  -u                         Disable safemode, default when running from a script.
  -s                         Silent safemode operation, will not ask you if
                             fuses should be changed back.
  -t                         Enter terminal mode.
  -E <exitspec>[,<exitspec>] List programmer exit specifications.
  -x <extended_param>        Pass <extended_param> to programmer.
  -y                         Count # erase cycles in EEPROM.
  -Y <number>                Initialize erase cycle # in EEPROM.
  -v                         Verbose output. -v -v for more.
  -q                         Quell progress output. -q -q for less.
  -l logfile                 Use logfile rather than stderr for diagnostics.
  -?                         Display this usage.
```

### Connect USB programmer

Connect USB programmer to ISP connector port J1 on Target board.




The ISP header consists of 2×3 pins.

### Test connection to the Target Board (Reference AVRdude on page 4)

- **Test Device:** avrdude -c usbtiny -p m328p -v
- **Test Device:** avrdude -c usbtiny -p atmega328p –v
- **Test Device:** avrdude -c usbasp -p m328p -v
- **Test Device:** avrdude -p m328p -P /dev/ttyACM0 -c usbasp -v

### Flash Write Examples: -U <memtype>:r|w|v:<filename>[:format]:

- Flash a hex file:

*avrdude -c usbtiny -p m328p -U flash:w:**blink.hex***

- Flash Fuses 1 MHz:

*avrdude -p m328p -c usbtiny -v -U lfuse:w:0x62:m -U hfuse:w:0xD9:m -U efuse:w:0xFF:m*

Refer to Fuse setting page.

### Flash Reading Hex Files Examples:

Test this *avrdude -c usbtiny -p m328p -U flash:r:mystery.hex:r*
Test this *avrdude -c usbtiny -p atmega328p -v -U flash:w:"i2cscan.hex":r*

### Using Terminal Mode: *avrdude -c usbtiny -p m328p -t*

Commands for AVRdude **terminal mode**:

- dump : dump memory : dump <memtype> <addr> <N-Bytes>
- read : alias for dump
- write : write memory : write <memtype> <addr> <b1> <b2> ... <bN>
- erase : perform a chip erase
- sig : display device signature bytes
- part : display the current part information
- pgm : return to programming mode
- help : help
- ? : help
- quit : quit

Refer to Terminal Mode page.

**Fuses Settings:**

Rules for Changing Fuse settings:

- Never change DWEN, SPIEN and RSTDSBL fuses. These settings cannot be changed while in ISP programming mode.
- Double check CKSEL fuses twice before writing. Wrong CKSEL settings can cause problems when the wrong clock source is selected.
- Don't set lock bits unless you are producing a commercial product and want to protect the software.

**If unsure – read datasheet or ask questions.**

**Warning:**

- One standard to understand before setting fuse and lock bits. Atmel uses the following standard:
  - **fuse bit = 1 is UN-programmed (inactive);**
  - **fuse bit = 0 is programmed (active);**
- Settings fuse bits with the wrong settings may lead to permanent damage of the microcontroller chip.
- This documentation may not be 100 percent accurate and could include incorrect data and mistypes. Please read these additional references.

**Additonal Referance:**

http://www.instructables.com/id/How-to-change-fuse-bits-of-AVR-Atmega328p-8bit-mic/
http://www.martyncurrey.com/arduino-atmega-328p-fuse-settings/

---

**Example settings for ATmega328P**

**Refer to ATmega328P DataSheet:** https://www.microchip.com/wwwproducts/en/ATMEGA328P

**Atmega328P:** Low Fuse Setting Example set to 8 MHZ E2

| Bit | Name | Description |
|---|---|---|
| 7 | CKDIV8 | When set, divides the clock speed by 8. |
| 6 | CKOUT | When set, the clock pulse is output on PB0 (pin 14) |
| 5 | SUT1 | Sets start up delay time |
| 4 | SUT0 | |
| 3 | CKSEL3 | Sets the Clock Source |
| 2 | CKSEL2 | |
| 1 | CKSEL1 | |
| 0 | CKSEL0 | |

**Low Byte Fuse**

**These 8 bits are explained here:**

| Example set to 8 MHZ | E2 (1 = unprogrammed) |
|---|---|
| Bit-6 : CKOUT When set clock pulses are output on PB0 | 1 (unprogrammed) |
| Bit-5 : SUT1 Startup time delay | 1 (Slow rising power option see data sheet section 9.6 table 9-12) |
| Bit-4 : SUT0 Startup time delay | 0 (Slow rising power option see data sheet section 9.6 table 9-12) |
| Bit-3 : CKSEL3 Set the clock source | 0 (4 bits to set Internal Clock see data sheet Section 9.2 Table 9.1) |
| Bit-2 : CKSEL2 Set the clock source | 0 (4 bits to set Internal Clock see data sheet Section 9.2 Table 9.1) |
| Bit-1 : CKSEL1 Set the clock source | 1 (4 bits to set Internal Clock see data sheet Section 9.2 Table 9.1) |
| Bit-0 : CKSEL0 Set the clock source | 0 (4 bits to set Internal Clock see data sheet Section 9.2 Table 9.1) |

7 6 5 4 | 3 2 1 0
8 4 2 1 | 8 4 2 1
0 1 1 0 | 0 0 1 0 = 62 1MHz
1 1 1 0 | 0 0 1 0 = E2 8MHz
1 1 1 0 | 0 0 0 0 = E0 ?? 16Mhz

| Device Clocking Option | CKSEL3 CKSEL2 CKSEL1 CKSEL0 |
|---|---|
| Low Power Crystal Oscillator | 1111 - 1000 |
| Full Swing Crystal Oscillator | 0111 - 0110 |
| Low Frequency Crystal Oscillator | 0101 - 0100 |
| Internal 128kHz RC Oscillator | 0011 |
| Calibrated Internal RC Oscillator | 0010 |
| External Clock | 0000 |
| Reserved / Not used | 0001 |

CKSEL0 = 0. CKSEL1 = 1 CKSEL2 = 0, CLSEL3 = 0, SUT0 = 0. SUT1 = 1 CKOUT=1. CKDIV8 = 1.
**Low Byte Fuses:** Now combining all the 8 bits, the required low fuse byte is **11100010**. **lfuse : 0xE2**
**High Byte Fuses:** Don't need to change the high fuse bits in order to change the clock source and operating frequency. So here is the default hex value: **11011001** **hfuse : 0xD9**
**Extended Byte Fuse** By default all the bits in this byte is set as **11111111**. **efuse : 0xFF**

---

## Fuse settings for different clock speeds.

Atmega328P Default Fuse Settings:
Low fuse = 0x62 (B01100010)
High fuse = 0xD9 (B11011001)
Extended fuse = 0xFF (B11111111)

**1 MHz Default:**
avrdude -p m328p -c usbtiny -v -U lfuse:w:0x62:m -U hfuse:w:0xD9:m -U efuse:w:0xFF:m
**8 MHz:**
avrdude -p m328p -c usbtiny -v -U lfuse:w:0xE2:m -U hfuse:w:0xD6:m -U efuse:w:0xFF:m
8 MHz crystal oscillator medium rising power with EEPROM preserve:
avrdude -p m328p -c usbtiny -v -U lfuse:w:0xFE:m -U hfuse:w:0xD1:m -U efuse:w:0xFF:m
Internal RC oscillator with prescaller off. 8 MHz system clock.
avrdude -p m328p -c usbtiny -v -U lfuse:w:0xE2:m -U hfuse:w:0xD9:m -U efuse:w:0x07:m
**16 MHz:**
avrdude -p m328p -c usbtiny -v -U lfuse:w:0xFF:m -U hfuse:w:0xDE:m -U efuse:w:0xFF:m
16 MHz ceramic resonator fast start:
avrdude -p m328p -c usbtiny -v -U lfuse:w:0xCE:m -U hfuse:w:0xD9:m -U efuse:w:0xFF:m
16 MHz crystal, slow start + CLKO clock:
avrdude -p m328p -c usbtiny -v -U lfuse:w:0xF7:m -U hfuse:w:0xD9:m -U efuse:w:0xFF:m

**Standard Arduino Uno settings.** **ATmega328P-PU**

16 MHz slow power starting, Bootloader enabled with boot size 128 words (optiboot). Typical brown out voltage selected at 2.7V.

Arduino or Nano w/ ATmega328 Default Fuse Settings

Low fuse = 0xFF (B11111111)
High fuse = 0xDE (B11011110)
Extended fuse = 0x05 (B00000101)

avrdude -p m328p -c usbtiny -v -U lfuse:w:0xFF:m -U hfuse:w:0xDE:m -U efuse:w:0x05:m

**Memory Lock Bits**

avrdude -c usbasp -P usb -p m328p -U lock:w:0x00:m -v

| Memory Lock Bits | | | |
|---|---|---|---|
| **Mode** | **LB1** | **LB2** | **Protection Type** |
| 1 | 1 | 1 | Unprogrammed, no protection enabled |
| 2 | 0 | 1 | Further Programming disabled, Read back possible |
| 3 | 0 | 0 | Further programming and read back is disabled |

Lock bits (LB1 and LB2) when low should prevent hackers from stealing your firmware. It is still possible for hackers to reverse engineer your code.
To clear the Lock bits, a complete Chip Erase is required, which erase the Flash memory

**Fuse Calculators:**

http://www.engbedded.com/fusecalc/
https://eleccelerator.com/fusecalc/

## Entering Terminal Mode: avrdude -c usbasp -P usb -p m328p -t

### Avrdude Terminal Mode Valid commands:

```
dump   : dump memory  : dump <memtype> <addr> <N-Bytes> dump flash 00 1024
read   : alias for dump
write  : write memory : write <memtype> <addr> <b1> <b2> ... <bN>
erase  : perform a chip erase erase
sig    : display device signature bytes
part   : display the current part information
send   : send a raw command : send <b1> <b2> <b3> <b4>
Parms  : display adjustable parameters (STK500 only)
vtarg  : set <V[target]> (STK500 only)
varef  : set <V[aref]> (STK500 only)
fosc   : set <oscillator frequency> (STK500 only)
sck    : set <SCK period> (STK500 only)
spi    : enter direct SPI mode
pgm    : return to programming mode
verbose: change verbosity
help   : help
?      : help
quit   : quit
```

Use the 'part' command to display valid memory types for use with the 'dump' and 'write' commands.

### Examples of commands in Terminal Mode

**Dump Flash or eeprom memory:**
**Read 1024 bytes of flash memory to display**

```
avrdude> dump flash 00 128
        >>> dump flash 00 128
        0000  0c 94 5c 00 0c 94 6e 00  0c 94 6e 00 0c 94 6e 00  | .\. .n. .n. .n.|
        0010  0c 94 6e 00 0c 94 6e 00  0c 94 6e 00 0c 94 6e 00  | .n. .n. .n. .n.|
        0020  0c 94 6e 00 0c 94 6e 00  0c 94 6e 00 0c 94 6e 00  | .n. .n. .n. .n.|
        0030  0c 94 6e 00 0c 94 6e 00  0c 94 6e 00 0c 94 6e 00  | .n. .n. .n. .n.|
        0040  0c 94 14 01 0c 94 6e 00  0c 94 6e 00 0c 94 6e 00  | ... .n. .n. .n.|
        0050  0c 94 6e 00 0c 94 6e 00  0c 94 6e 00 0c 94 6e 00  | .n. .n. .n. .n.|
        0060  0c 94 6e 00 0c 94 6e 00  00 00 00 00 24 00 27 00  | .n. .n.....$.'.|
        0070  2a 00 00 00 00 00 25 00  28 00 2b 00 04 04 04 04  |*.....%.(.+.....|

avrdude> dump eeprom 00 32
        >>> dump eeprom 00 32
        0000  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  |................|
        0010  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  |................|

avrdude> write eeprom 04 128
        0000  ff ff ff ff 80 ff ff ff  ff ff ff ff ff ff ff ff  |................|

avrdude> write eeprom 00 0x57 0x6f 0x72 0x6b 0x73 0x68 0x6f 0x70 0x20 0x32 0x30 0x31 0x38
        >>> write eeprom 00 0x57 0x6f 0x72 0x6b 0x73 0x68 0x6f 0x70 0x20 0x32 0x30 0x31 0x38

avrdude> dump eeprom 00 16
        >>> dump eeprom 00 16
        0000  57 6f 72 6b 73 68 6f 70  20 32 30 31 38 ff ff ff  |Workshop 2018...|

avrdude> sig
        Reading | ################################################## | 100% 0.06s
```

```
        Device signature = 0x1e950f

avrdude> verbose 0 - 4 ( setting anything past 3 = full verbosity. Default is Zero)
        >>> verbose 3
        New verbosity level: 3

avrdude> dump eeprom 00 4
        >>> dump eeprom 00 4
        avrdude: usbasp_spi_cmd(0xa0, 0x00, 0x00, 0x00) => 0x00, 0xa0, 0x00, 0x57
        avrdude: usbasp_spi_cmd(0xa0, 0x00, 0x01, 0x00) => 0x00, 0xa0, 0x00, 0x6f
        avrdude: usbasp_spi_cmd(0xa0, 0x00, 0x02, 0x00) => 0x00, 0xa0, 0x00, 0x72
        avrdude: usbasp_spi_cmd(0xa0, 0x00, 0x03, 0x00) => 0x00, 0xa0, 0x00, 0x6b
        0000  57 6f 72 6b                              |Work          |

avrdude> part
        >>> part
        AVR Part                      : ATmega328P
        Chip Erase delay              : 9000 us
        PAGEL                         : PD7
        BS2                           : PC2
        RESET disposition             : dedicated
        RETRY pulse                   : SCK
        serial program mode           : yes
        parallel program mode         : yes
        Timeout                       : 200
        StabDelay                     : 100
        CmdexeDelay                   : 25
        SyncLoops                     : 32
        ByteDelay                     : 0
        PollIndex                     : 3
        PollValue                     : 0x53
        Memory Detail                 :

                                 Block Poll               Page                       Polled
  Memory Type Mode Delay  Size  Indx Paged  Size   Size #Pages MinW  MaxW   ReadBack
  ----------- ---- ----- ----- ---- ------ ------ ---- ------ ----- ----- ---------
  eeprom          65    20     4    0 no      1024    4      0  3600  3600 0xff 0xff
  flash           65     6   128    0 yes    32768  128    256  4500  4500 0xff 0xff
  lfuse            0     0     0    0 no         1    0      0  4500  4500 0x00 0x00
  hfuse            0     0     0    0 no         1    0      0  4500  4500 0x00 0x00
  efuse            0     0     0    0 no         1    0      0  4500  4500 0x00 0x00
  lock             0     0     0    0 no         1    0      0  4500  4500 0x00 0x00
  calibration      0     0     0    0 no         1    0      0     0     0 0x00 0x00
  signature        0     0     0    0 no         3    0      0     0     0 0x00 0x00

avrdude> erase
        >>> erase
        avrdude: erasing chip
        avrdude: warning: cannot set sck period. please check for usbasp firmware update.
avrdude> quit
        >>> quit
        avrdude: safemode: Fuses OK (E:FF, H:D9, L:62)
```

## Appendix

---

**Avrdude Command line Examples:**

| | |
|---|---|
| List of Avrdude commands: | **avrdude** |
| USBTiny Display device info: | **avrdude -c usbtiny -p m328p** |
| USBasp Display device info: | **avrdude -c usbasp -P usb -p m328p** |
| USBTiny Display Verbose info: | **avrdude -c usbtiny -p atmega328p -v** |
| Write hex file to Microcontroller: | **avrdude -c usbtiny -p m328p -U flash:w:blink.hex:** |
| Read Microcontroller and write file: | **avrdude -c usbtiny -p m328p -U flash:r:blink.hex:r** |
| Erase flash: | **avrdude -c usbtiny -p atmega328p -e** |
| Terminal Mode**:** | **avrdude -c usbtiny -p atmega328p -t** |
| Capture flash memory data: | **avrdude -c usbtiny -p m328p -t > dumpflash.txt** |
| Using the terminal Mode: | then type the following: **dump flash 00 1024**<br>then wait a min for it to create and write the file then type **quit.** |

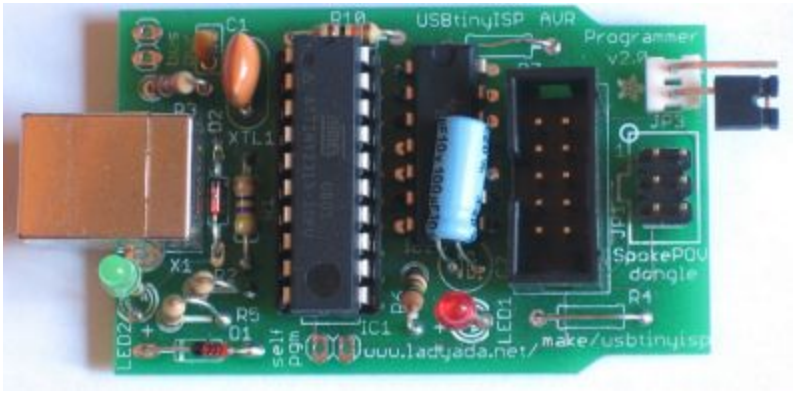
**End of Part 1.**
**AVRdude - USB Programmer - ATmega328**
**Author Rusty Cain Microrusty 2018**

# USB Programmers

**USBTiny**

USBtiny is a software implementation of the USB low-speed protocol for the Atmel ATtiny microcontrollers. Of course, it will also work on the ATmega series. The software is written for an AVR clocked at 12 MHz. At this frequency, each bit on the USB bus takes 8 clock cycles, and with a lot of trickery, it is possible to decode and encode the USB waveforms by software. The USB driver needs approximately 1250 to 1350 bytes of flash space (excluding the optional identification strings), depending on the configuration and compiler version, and 46 bytes RAM (excluding stack space). The C interface consists of 3 to 5 functions, depending on the configuration.

The USBTinyISP is a step up from the USBasp. Keep in mind that it cannot program certain high end Atmega devices due to memory limitations.




**Reference: https://learn.adafruit.com/usbtinyisp/use-it**

**Indicator LEDs**

There are two LEDs, a green one near the USB port and a red one near the cables.

The green LED indicates that the USB connection was successful. If you're using a Windows or Linux machine and the green LED does not light up when you plug it in, there's a problem.

The red LED indicates that the USBtinyISP is 'busy' programming. Do not unplug the device being programmed while it's lit.

**Programming Cables**

There are two cables for programming: a 10-pin ISP cable and a 6-pin ISP cable. They are the two prevailing standards for in-circuit AVR programming. This programmer doesn't do JTAG programming

**Jumper JP3 (USB power to target)**

JP3. When the jumper is in place (connecting the two wires) then that means that the USBtinyISP is providing 5V power to the device being programmed. If you don't want to power the device then just take the jumper out or make sure it's only on one of the wires.

The USBtinyISP can only provide 5V, up to about 100mA to the device. If you need more power then you should remove the jumper and power the device separately. Version 1.0 of USBtinyISP sends data to the device at 5V level no matter whether it's powering the device or not so make sure it's 5V compliant! (Note that there are 2 1.5K resistors in series with the data lines for protection)

Version 2.0 which is almost certainly what you've got, uses a level shifter so that if the jumper is not in place, it will use whatever the target voltage is, a lot better for your low-voltage devices!

**So, if you have a device that needs to run at 3.3V, don't have the jumper in place!**

**USBasp**
http://www.fischl.de/usbasp/
USBasp - USB programmer for Atmel AVR controllers
USBasp is a USB in-circuit programmer for Atmel AVR controllers. It simply consists of an ATMega88 or an ATMega8 and a couple of passive components. The programmer uses a firmware-only USB driver, no special USB controller is needed.

## Features

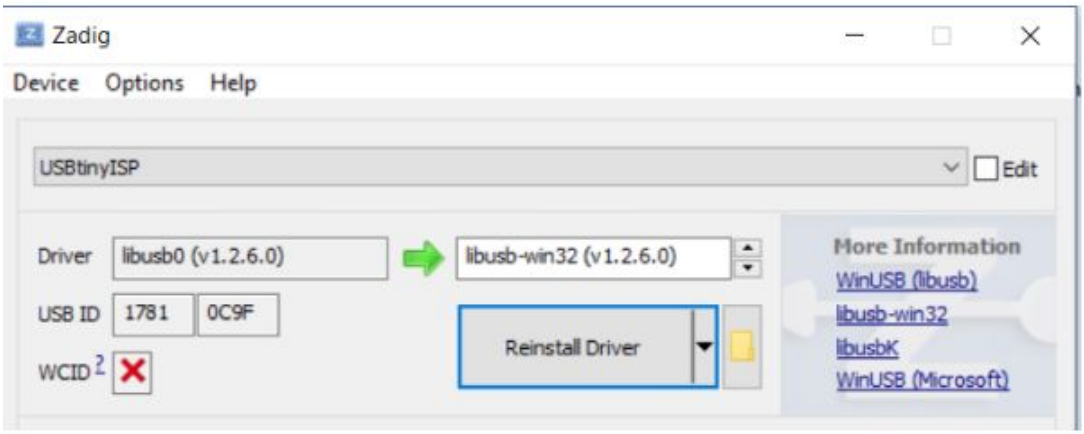
- Works under multiple platforms. Linux, Mac OS X and Windows are tested.
- No special controllers or smd components are needed.
- Programming speed is up to 5kBytes/sec.
- SCK option to support targets with low clock speed (< 1,5MHz).
- Planned: serial interface to target (e.g. for debugging).

The USBasp is quite possibly the cheapest programmer out there, but you should be careful when buying them; some versions use outdated firmware or are missing jumpers. Make sure that your model has three jumpers (or three pairs of holes with J1, J2, and J3 printed next to them). You can find them on eBay starting at only $3. Be wary of the shipping times that the Chinese ones come with.

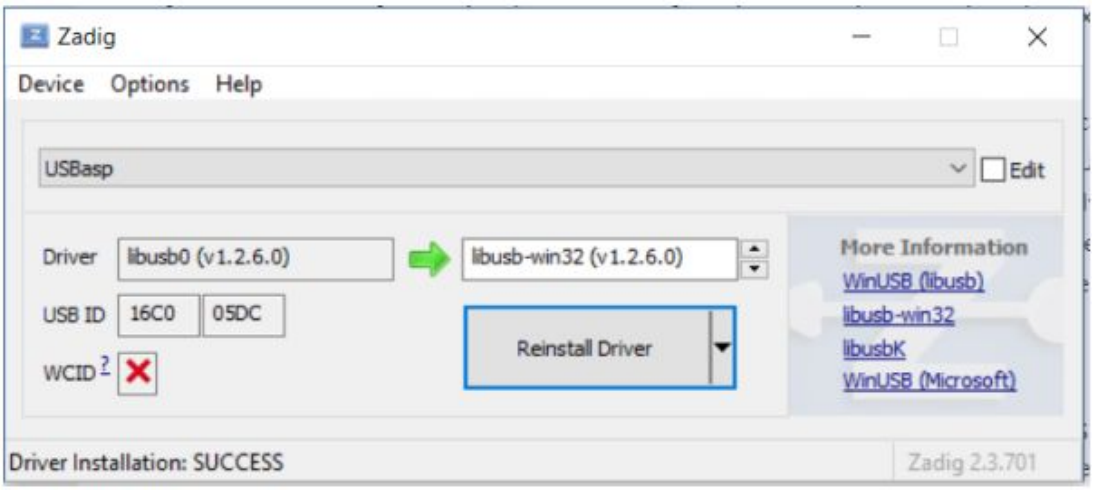
## Installing Windows 10 Drivers for USB devices

**Due to windows 10 USB changes**

**Zadig USBTiny**




**Zadig USBasp**



## List of Avrdude commands

Options:

-p <partno> Required. Specify AVR device.
-b <baudrate> Override RS-232 baud rate.
-B <bitclock> Specify JTAG/STK500v2 bit clock period (us).
-C <config-file>Specify location of configuration file.
-c <programmer> Specify programmer type.
-D Disable auto erase for flash memory
-i <delay> ISP Clock Delay [in microseconds]
-P <port>Specify connection port.
-F Override invalid signature check.
-e Perform a chip erase.
-O Perform RC oscillator calibration (see AVR053).
-U <memtype>:r|w|v:<filename>[:format] Memory operation specification.
Multiple -U options are allowed, each request is performed in the order specified.
-n Do not write anything to the device.
-V Do not verify.
-u Disable safemode, default when running from a script.
-s Silent safemode operation, will not ask you if fuses should be changed back.
-t Enter terminal mode.
-E <exitspec>[,<exitspec>] List programmer exit specifications.
-x <extended_param> Pass <extended_param> to programmer.
-y Count # erase cycles in EEPROM.
-Y <number> Initialize erase cycle # in EEPROM.
-v Verbose output. -v -v for more.
-q Quell progress output. -q -q for less.
-l logfile Use logfile rather than stderr for diagnostics.
-? Display this usage.

**Microrusty 2018**