Guide to open-source debugging ATMega328

This guide explains how to use avarice, avr-gdb and avrdude on Fedora 31 to do some basic debugging using debugWIRE protocol.

Used hardware

- JTAG debugger MKII-CN (EUR 74)
- 10pin to 6pin AVR ISP adapter
- Arduino Uno with ATMega328

MKII is the only debugger that seems to be supported by open-source software. Atmel released details about the MKII protocol in application note AVR069: AVRISP mkII Communication Protocol. The original MKII is not produced anymore and is out of stock. However a Chinese company called Mcuzone has developed a product called MKII-CN that claims to be 100% compatible to the original Atmel MKII. It costs about EUR 74 in 2020 via Aliexpress.

Please be aware when buying the USB AVR JTAGICE MKII-CN that there are various programmers and JTAG dongles available that will not work for debugging most AVR micro controllers.

- Programmers exists that use the MKII protocol, these programmers will usually sell for around EUR 20 and have *AVRISP mkII AVR ISP mk2* in the description. These programmers can only program using ISP protocol and cannot do debugging.
- JTAG ICE programmers exist that cost around EUR 10. Most of these programmers will list a small number of AVR devices in their product description that are supported. These are all older AVR's that have JTAG support. ATmega 16/32/128 and such. It probably won't work on Linux.

The USB AVR JTAGICE MKII-CN or MKII-CN in short supports both JTAG and debugWIRE protocol on older and newer AVR's. So it is the best option to get debugging on a wide range of devices.

Used software

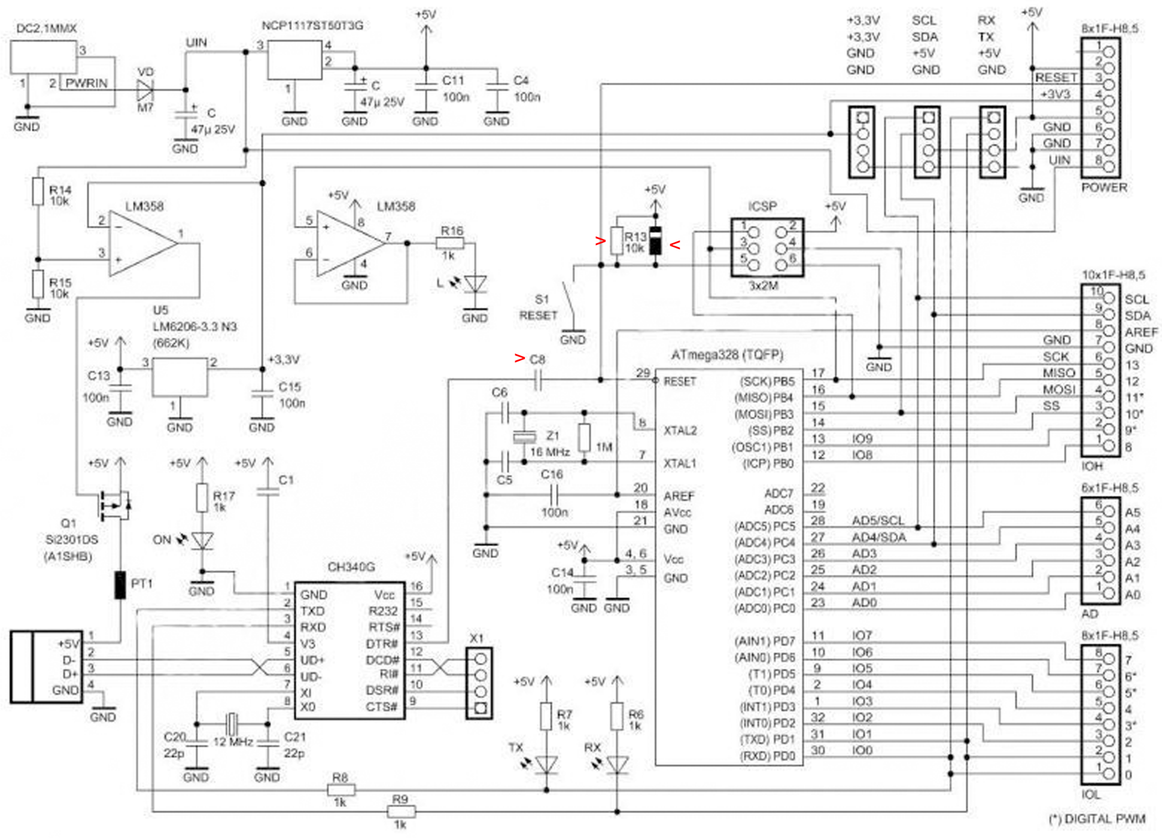
- Fedora 31
- AVaRICE version 2.13, Jul 24 2019
- avr-gdb 8.1-4
- avrdude version 6.3
- Arduino IDE 1.8.13

Preparing Arduino Uno board

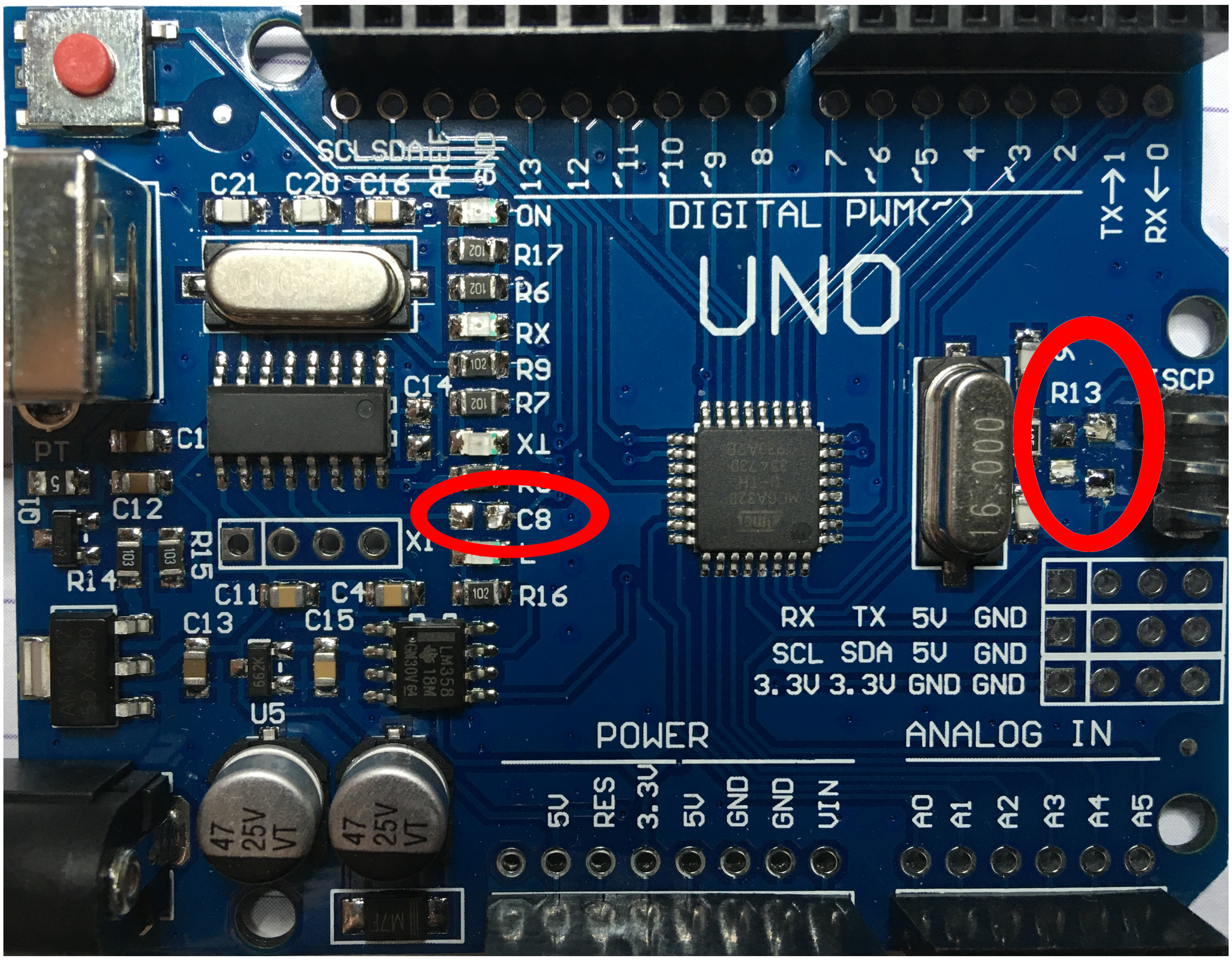
For the Atmel ATmega328 datasheet: When designing a system where debugWIRE will be used, the following observations must be made for correct operation:

- Pull-up resistors on the dW/(RESET) line must not be smaller than 10kΩ. The pull-up resistor is not required for debugWIRE functionality
- Connecting the RESET pin directly to VCC will not work.
- Capacitors connected to the RESET pin must be disconnected when using debugWire.
- All external reset sources must be disconnected.

This means for stable operation a number of components need to be removed from the Arduino Uno board. After removing these components you will no longer be able to use the DTR line on the serial port to do a reset.



Remove C8, R13 and unnumbered diode.



Remove C8, R13 and unnumbered diode.

With all components still on the board, you will be able to program the debugWIRE enable fuse (DWEN), but then you probably end up with a board that cannot connect via debugWIRE. And since enable debugWIRE will disable ISP programming, you may no longer have a way to program your board. Initially I used an oscilloscope in component tester mode to find out what components are connected to the reset line. But in case you have no such option you will have to find the correct schematics for your board.

Installing software

```
dnf install avarice avr-gdb avr-gcc arduino avrdude
```

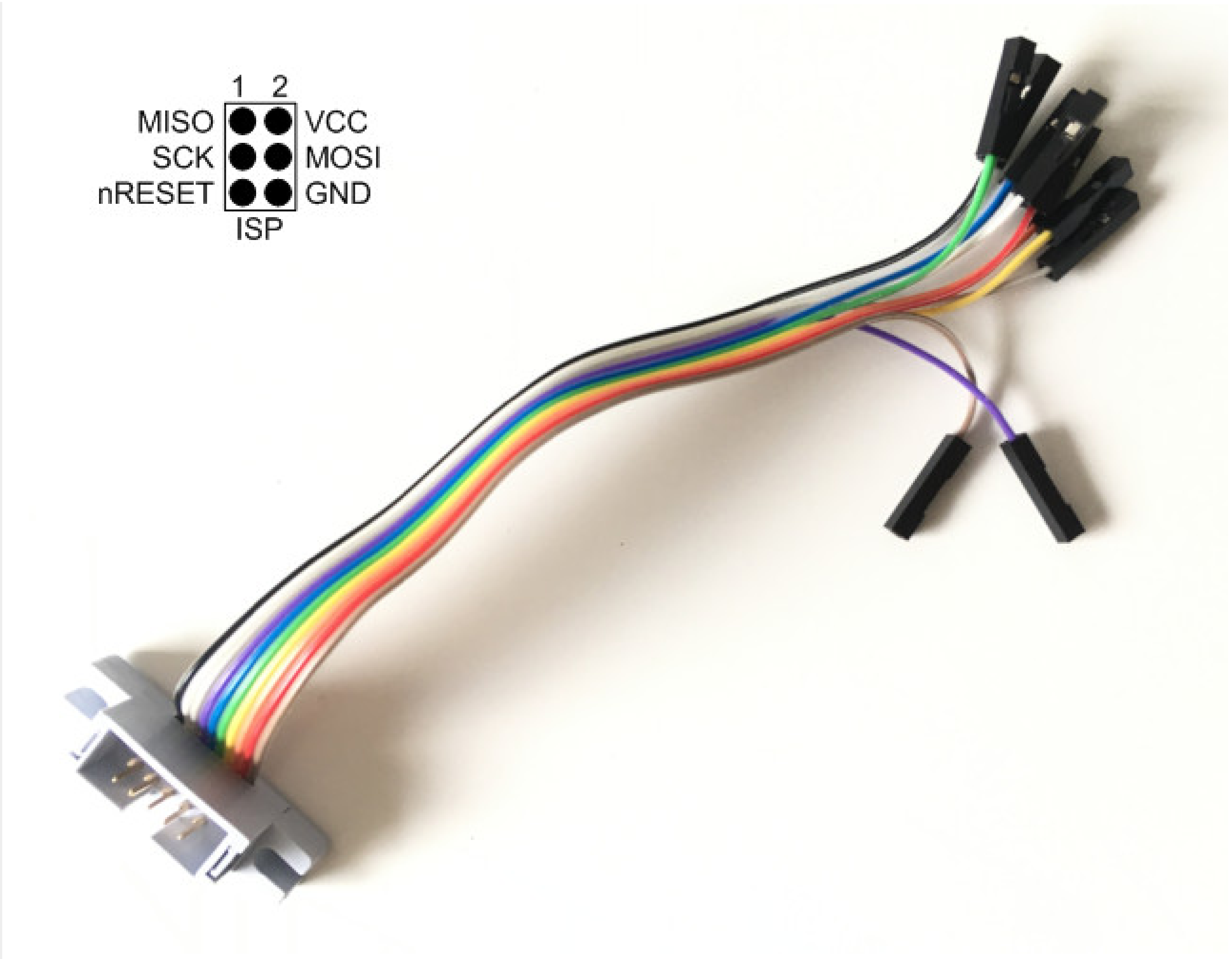
I have set up my system so I can run all these commands from my normal user account, but in case you have issues, sudo it.

Connecting Arduino Uno to MKII-CN using debugWIRE

You have 2 options to connect the Arduino Uno to the MKII-CN. Option one is using the so called squid cable that comes with the MKII-CN (or the original MKII if you are lucky enough to have one).

The debugWIRE interface only requires RESET, VCC, and GND. These pins can be found on the 6pin ISP header.

Arduino	Squid Cable color
GND	white
VCC	purple
RESET	green

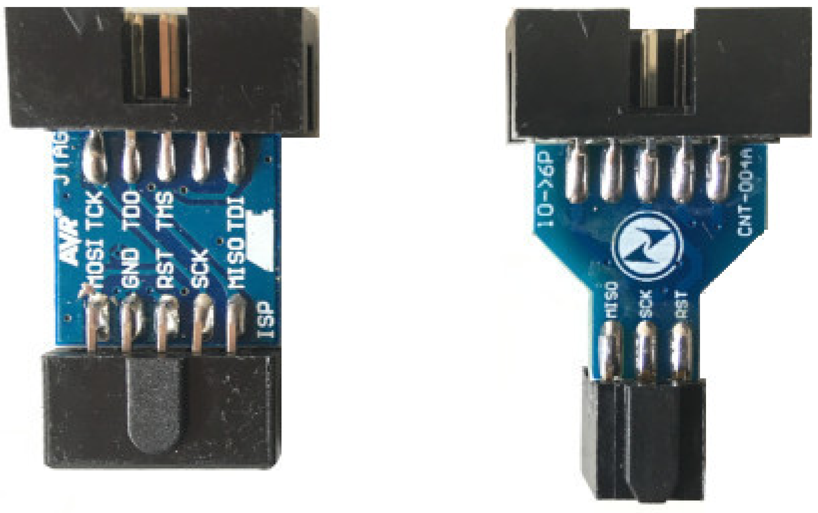


ISP header pinouts and squid cable

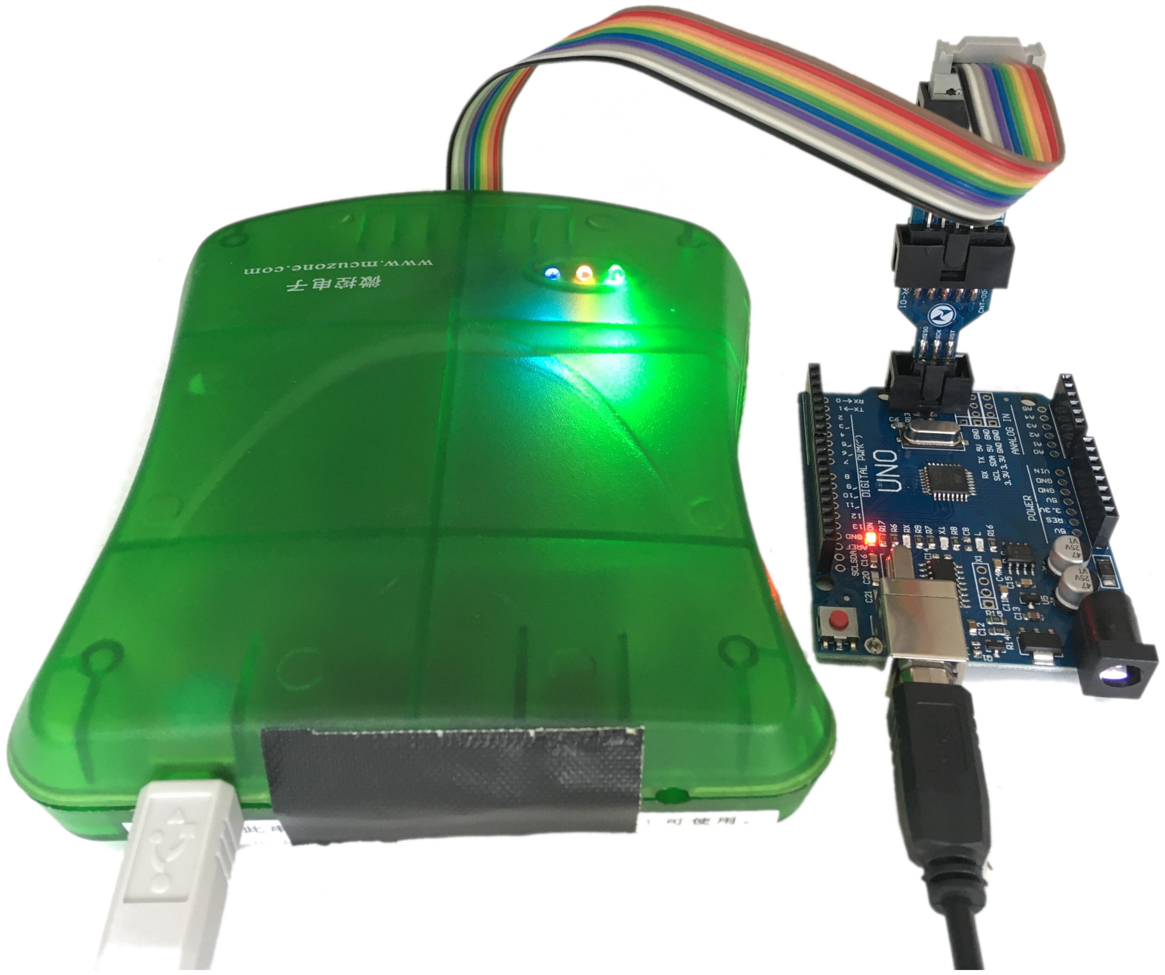
The VCC line is NOT used to supply power. It is used by the JTAG debugger to measure the operating voltage of the Arduino. This is why it is referred to as VTref in Atmel documentation. This means you also need to connect power to the Arduino Uno. Easiest way is to just plug-in the usb cable to the Arduino and use that for power.

On my board the RESET pin on the ISP header is on the side closest to the center of the board.

The other option to make the connection is using a 10-pin to 6-pin ISP converter and the ISP to JTAG converter. The latter one uses 10 pin connectors and comes with the MKII-CN.



ISP-to-JTAG converter and 10pin-to-6pin ISP converter



Connection using adapter boards

You may notice a piece of duct tape on the back of my MKII-CN covering a db9 and power jack. This is because a sticker on the MKII-CN says in Chinese one should not use these connectors. I just covered these up in case I forget.

Do not program any lock bits if using debugWIRE. This will brick your Arduino.

Create a basic example for debugging

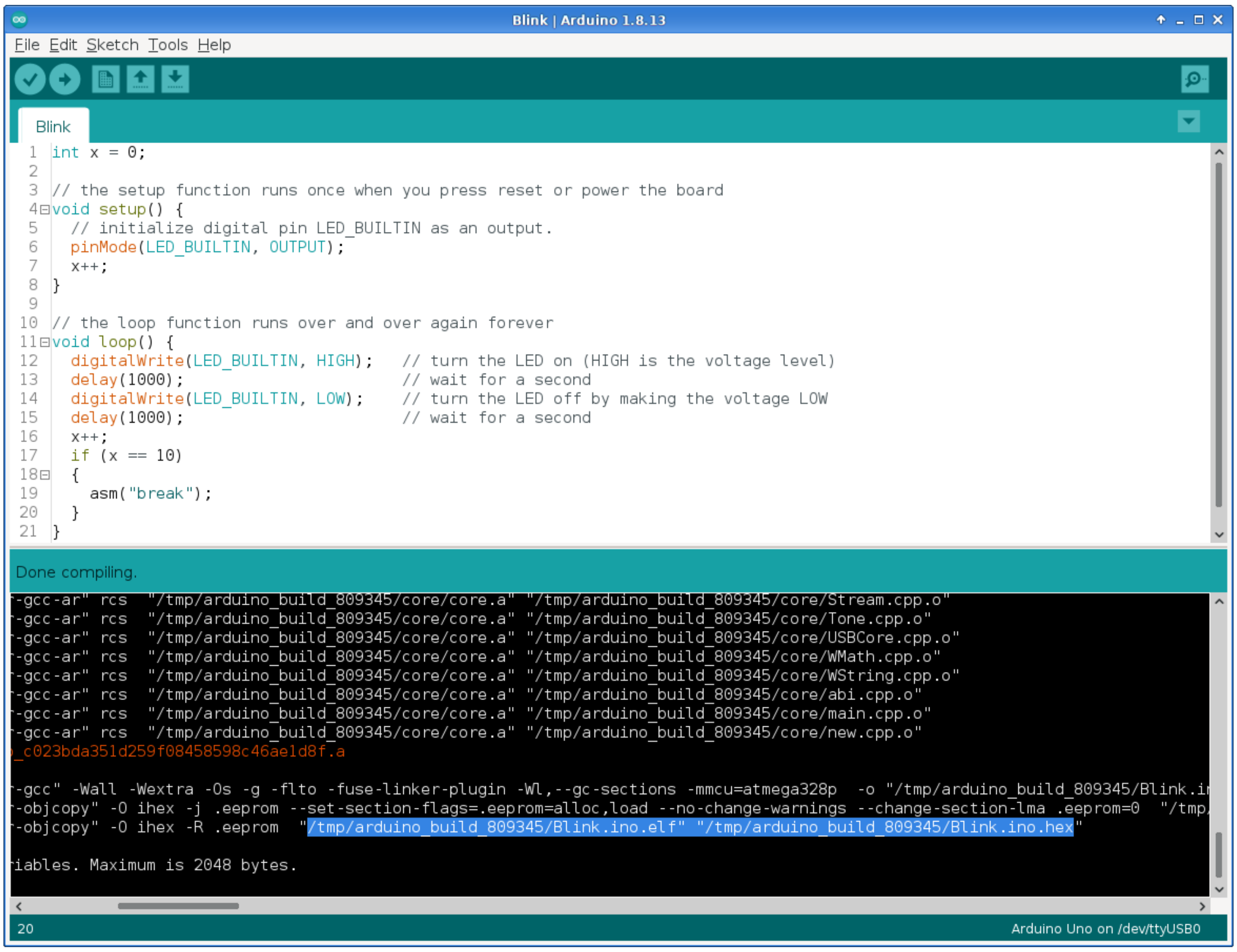
The Arduino IDE does not support debugging and the ATMega328 does not support hardware breakpoints. This means the easiest way to have breakpoints is adding them to your C code using an inline assembly instruction like this:

```
asm("break");
```

To test this breakpoint we use the Blink example in Arduino IDE and add a variable `x` that we increment in a loop:

```
int x = 0;

// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
  x++;
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);                       // wait for a second
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);                       // wait for a second
  x++;
  if (x == 10)
  {
    asm("break");
  }
}
```

To start debugging you will need the compiled .hex file and the .elf file. Which you can get by doing *verify/compile* in the Arduino IDE and then look for the paths in the compiler output. In this case the files can be found at */tmp/arduino_build_809345/Blink.ino.elf* and */tmp/arduino_build_809345/Blink.ino.hex*.



```
int x = 0;

// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
  x++;
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);                       // wait for a second
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);                       // wait for a second
  x++;
  if (x == 10)
  {
    asm("break");
  }
}
```

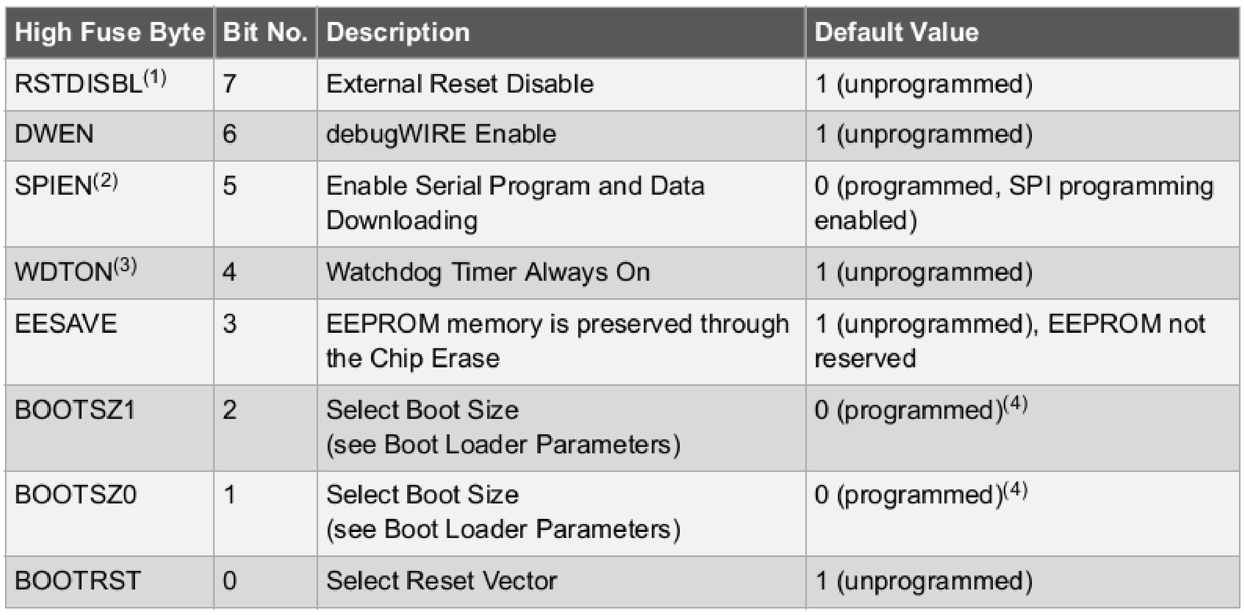
```
-gcc-ar" rcs  "/tmp/arduino_build_809345/core/core.a" "/tmp/arduino_build_809345/core/Stream.cpp.o"
-gcc-ar" rcs  "/tmp/arduino_build_809345/core/core.a" "/tmp/arduino_build_809345/core/Tone.cpp.o"
-gcc-ar" rcs  "/tmp/arduino_build_809345/core/core.a" "/tmp/arduino_build_809345/core/USBCore.cpp.o"
-gcc-ar" rcs  "/tmp/arduino_build_809345/core/core.a" "/tmp/arduino_build_809345/core/WMath.cpp.o"
-gcc-ar" rcs  "/tmp/arduino_build_809345/core/core.a" "/tmp/arduino_build_809345/core/WString.cpp.o"
-gcc-ar" rcs  "/tmp/arduino_build_809345/core/core.a" "/tmp/arduino_build_809345/core/abi.cpp.o"
-gcc-ar" rcs  "/tmp/arduino_build_809345/core/core.a" "/tmp/arduino_build_809345/core/main.cpp.o"
-gcc-ar" rcs  "/tmp/arduino_build_809345/core/core.a" "/tmp/arduino_build_809345/core/new.cpp.o"
_c023bda351d259f08458598c46ae1d8f.a

-gcc" -Wall -Wextra -Os -g -flto -fuse-linker-plugin -Wl,--gc-sections -mmcu=atmega328p  -o "/tmp/arduino_build_809345/Blink.i
-objcopy" -O ihex -j .eeprom --set-section-flags=.eeprom=alloc,load --no-change-warnings --change-section-lma .eeprom=0  "/tmp
-objcopy" -O ihex -R .eeprom  "/tmp/arduino_build_809345/Blink.ino.elf" "/tmp/arduino_build_809345/Blink.ino.hex"

iables. Maximum is 2048 bytes.
```

Example of compile output in Arduino IDE.

Start debugging

To keep things simple I copied the .hex and .elf file to my home folder and renamed them main.hex and main.elf. Flash main.hex and enable the debugWIRE fusebit (DWEN) using avrdude on the command prompt like this:

```
  /usr/bin/avrdude -C/etc/avrdude.conf -cjtag2isp -patmega328p -Pusb -V -Uflash:w:main.hex -U
hfuse:w:0x99:m
```



High Fuse Byte	Bit No.	Description	Default Value
RSTDISBL(1)	7	External Reset Disable	1 (unprogrammed)
DWEN	6	debugWIRE Enable	1 (unprogrammed)
SPIEN(2)	5	Enable Serial Program and Data Downloading	0 (programmed, SPI programming enabled)
WDTON(3)	4	Watchdog Timer Always On	1 (unprogrammed)
EESAVE	3	EEPROM memory is preserved through the Chip Erase	1 (unprogrammed), EEPROM not reserved
BOOTSZ1	2	Select Boot Size (see Boot Loader Parameters)	0 (programmed)(4)
BOOTSZ0	1	Select Boot Size (see Boot Loader Parameters)	0 (programmed)(4)
BOOTRST	0	Select Reset Vector	1 (unprogrammed)

Description of Fuse High Byte from Atmel datasheet.

In this example I used 0x99 for the Fuse High Byte. In binary form this is 10011001. Which means use all Atmel defaults and enable DWEN. Please note that this will remove and disable the bootloader. You can flash it back using Arduino IDE in the last step if you want.

NOW YOU HAVE TO REMOVE THE POWER FROM THE ARDUINO AND RE-APPLY POWER. This will put your device in debugWIRE mode.

To start the debugger use the following command:

```
avarice -j usb -w -P atmega328p :4242
```

Open a new terminal and connect to the debugger:

```
avr-gdb main.elf
```

You should now be in the gdb prompt. Here you can use the following command to connect:

```
target remote localhost:4242
```

The blinking led is a nice indicator of when the AVR is halted by the debugger. It halts when you connect the debugger. To make the program run, use the `continue` command in gdb.

Useful GDB commands

command	description
CTRL+C	will halt the AVR
continue	will resume the AVR

command	description
print x	shows value of x
set variable x = 1	sets value of x
whatis x	show type of x

Special case, if an assembly breakpoint is reached you have to use the following command to resume AVR program execution:

```
set $pc += 2
continue
```

Untested, you can also call functions using GDB like so:

```
  call send_string("Test of UART")
```

Going back to normal mode

This is kind of a trick that avrdude has, you can attempt to write a hex file like this:

```
  /usr/bin/avrdude -C/etc/avrdude.conf -cjtag2isp -patmega328p -Pusb -V -Uflash:w:main.hex
```

The response from avrdude will be something like this:

```
avrdude: jtagmkII_setparm(): bad response to set parameter command: RSP_FAILED
avrdude: jtagmkII_getsync(): ISP activation failed, trying debugWire
avrdude: Target prepared for ISP, signed off.
avrdude: Now retrying without power-cycling the target.
avrdude: jtagmkII_setparm(): bad response to set parameter command: RSP_FAILED
avrdude: jtagmkII_getsync(): ISP activation failed, trying debugWire
avrdude: jtagmkII_setparm(): bad response to set parameter command: RSP_DEBUGWIRE_SYNC_FAILED
avrdude: failed to sync with the JTAG ICE mkII in ISP mode

avrdude done.  Thank you.
```

As you can see the programming failed, but avrdude did instruct the AVR to temporary disable debugWIRE as shown by *Target prepared for ISP*. You can now open the Arduino IDE and program the bootloader or another program and power cycle the board. This will resume normal operation.

If you power cycle the board without programming anything, debugWIRE will re-enable.

References:

- http://winavr.sourceforge.net/AVR-GDB_and_AVaRICE_Guide.pdf
- JTAGICE mkII user guide: https://onlinedocs.microchip.com/pr/GUID-73C92233-8EC5-497C-92C3-

D52ED257761E-en-US-1/index.html?GUID-5580C285-5789-4BC7-ACD1-229832FC4487

- DebugWIRE how-to https://gist.github.com/mattvenn/930590aabbb46beba6a9306312d3e620
- JTAGICE how-to https://christophe.vg/technology/JTAGICE_mkII
- Using dragon for debugWIRE https://blag.pseudoberries.com/post/16374999524/avr-debugwire-on-linux
- ATmega328p datasheet https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf

Public Domain notice and disclaimer