
Guide to open-source debugging ATMega2560

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

Used hardware

- JTAG debugger MKII-CN (EUR 74)
- 10pin to 6pin AVR ISP adapter
- JTAG Squid cable
- Arduino Mega with ATMega2560

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 Mega board

The board can be used without modification.

Installing software

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

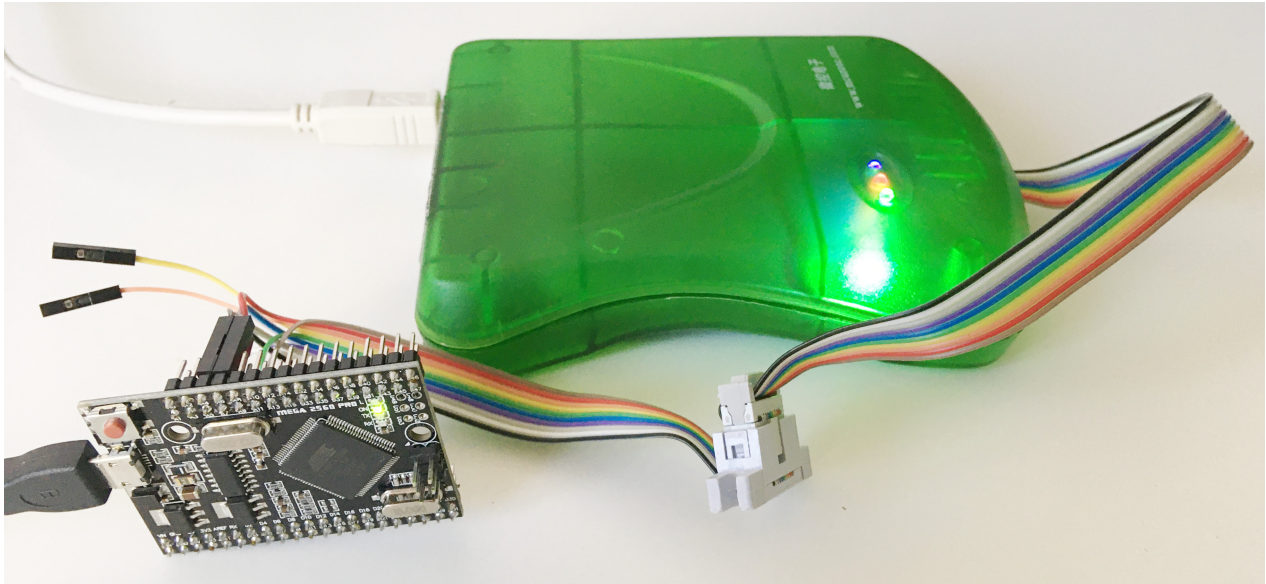
I have set up my system so I can run all these commands from my normal user account, but in case you have issues, sudo it.

Connecting Arduino Mega to MKII-CN using JTAG

To connect the Arduino Mega to the MKII-CN, you have to use the so called squid cable that comes with the MKII-CN (or the original MKII if you are lucky enough to have one).

The following connections must be made:

JTAG pin	Arduino	Squid Cable Color
1	A4	Black
2	GND	White
3	A6	Grey
4	VCC	Purple
5	A5	Blue
6	RST	Green
7		Yellow
8		Orange
9	A7	Red
10	GND	Brown



Connecting 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 Mega. Easiest way is to just plug-in the usb cable to the Arduino and use that for power.

Create a basic example for debugging

The Arduino IDE does not support debugging. 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:

```

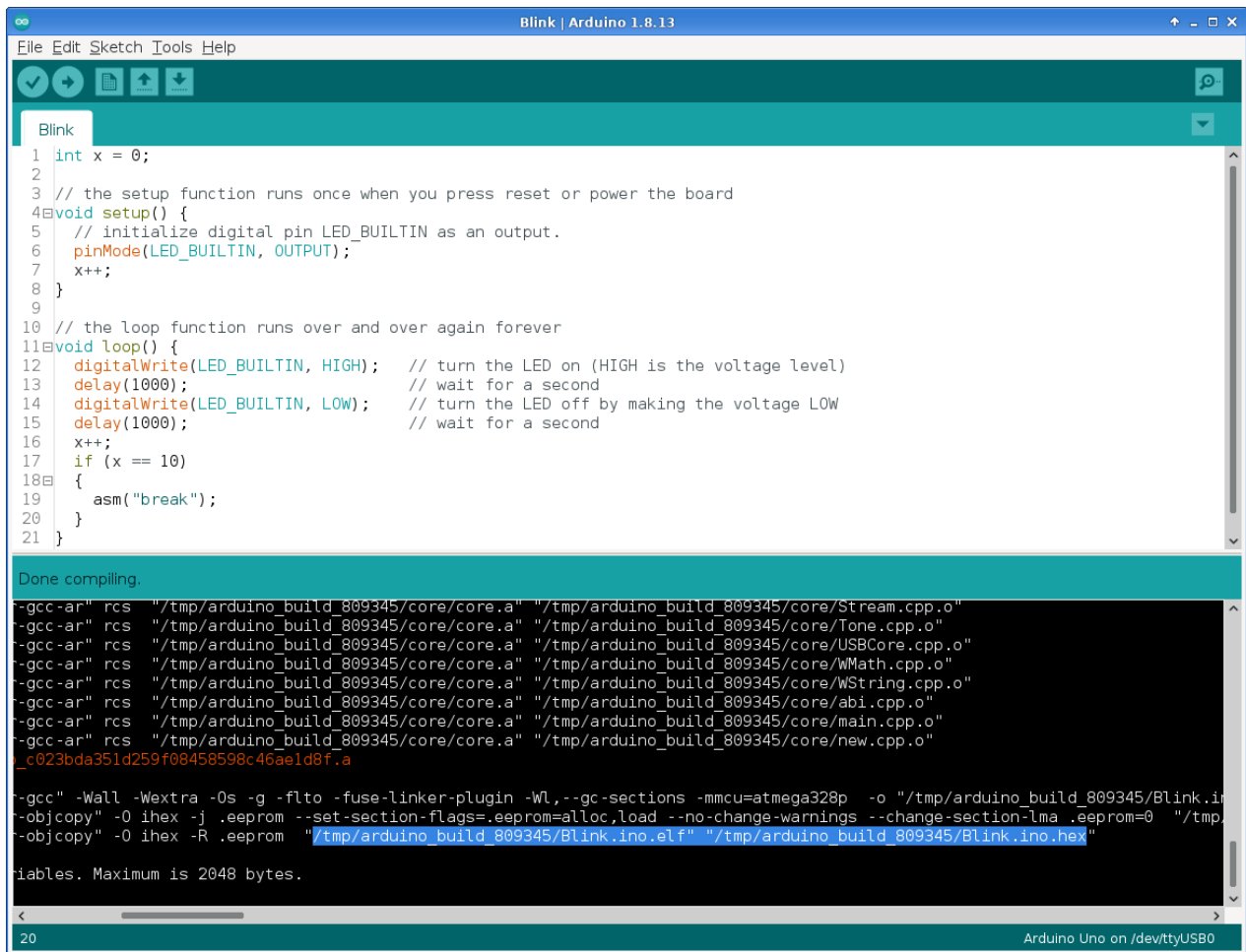
int x = 0;

// 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*.



The screenshot shows the Arduino IDE interface. The top window displays the 'Blink' sketch with the following code:

```
1 int x = 0;
2
3 // the setup function runs once when you press reset or power the board
4 void setup() {
5   // initialize digital pin LED_BUILTIN as an output.
6   pinMode(LED_BUILTIN, OUTPUT);
7   x++;
8 }
9
10 // the loop function runs over and over again forever
11 void loop() {
12   digitalWrite(LED_BUILTIN, HIGH); // turn the LED on (HIGH is the voltage level)
13   delay(1000); // wait for a second
14   digitalWrite(LED_BUILTIN, LOW); // turn the LED off by making the voltage LOW
15   delay(1000); // wait for a second
16   x++;
17   if (x == 10)
18   {
19     asm("break");
20   }
21 }
```

The bottom window shows the compilation output, including the following lines:

```
Done compiling.
-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.ino.elf" -objcopy" -O ihex -j .eeprom --set-section-flags=.eeprom=alloc,load --no-change-warnings --change-section-lma .eeprom=0 "/tmp/arduino_build_809345/Blink.ino.elf" "/tmp/arduino_build_809345/Blink.ino.hex"
iables. Maximum is 2048 bytes.
```

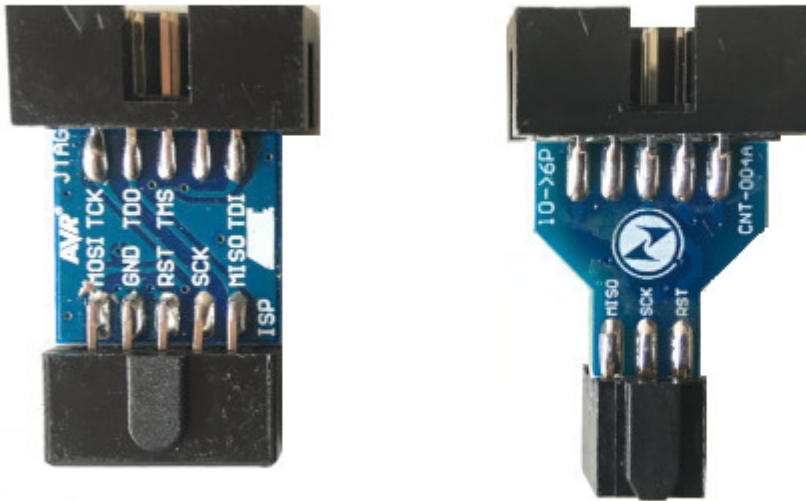
Example of compile output in Arduino IDE.

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 JTAG and OCD fusebits (JTAGEN, OCDEN) using avrdude on the command line like so:

```
/usr/bin/avrdude -C/etc/avrdude.conf -cjtag2isp -patmega2560 -Pusb -V -Uflash:w:main.hex -Uhfuse:w:0x19:m
```

You have to use ISP programming for flashing the program and fuses. You can use the 6 pin ICSP header on the Arduino Mega board and these adapters:



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

Fuse High Byte	Bit No	Description	Default Value
OCDEN ⁽⁴⁾	7	Enable OCD	1 (unprogrammed, OCD disabled)
JTAGEN	6	Enable JTAG	0 (programmed, JTAG enabled)
SPIEN ⁽¹⁾	5	Enable Serial Program and Data Downloading	0 (programmed, SPI prog. enabled)
WDTON ⁽³⁾	4	Watchdog Timer always on	1 (unprogrammed)
EESAVE	3	EEPROM memory is preserved through the Chip Erase	1 (unprogrammed, EEPROM not preserved)
BOOTSZ1	2	Select Boot Size (see Table 30-9 on page 329 for details)	0 (programmed) ⁽²⁾
BOOTSZ0	1	Select Boot Size (see Table 30-9 on page 329 for details)	0 (programmed) ⁽²⁾
BOOTRST	0	Select Reset Vector	1 (unprogrammed)

Description of Fuse High Byte from Atmel datasheet.

In this example I used 0x19 for the Fuse High Byte. In binary form this is 11001. Which means use all Atmel defaults and enable JTAGEN, OCDEN. 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.

To start the debugger use the following command:

```
avarice -j usb -2 -P atmega2560 :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
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:

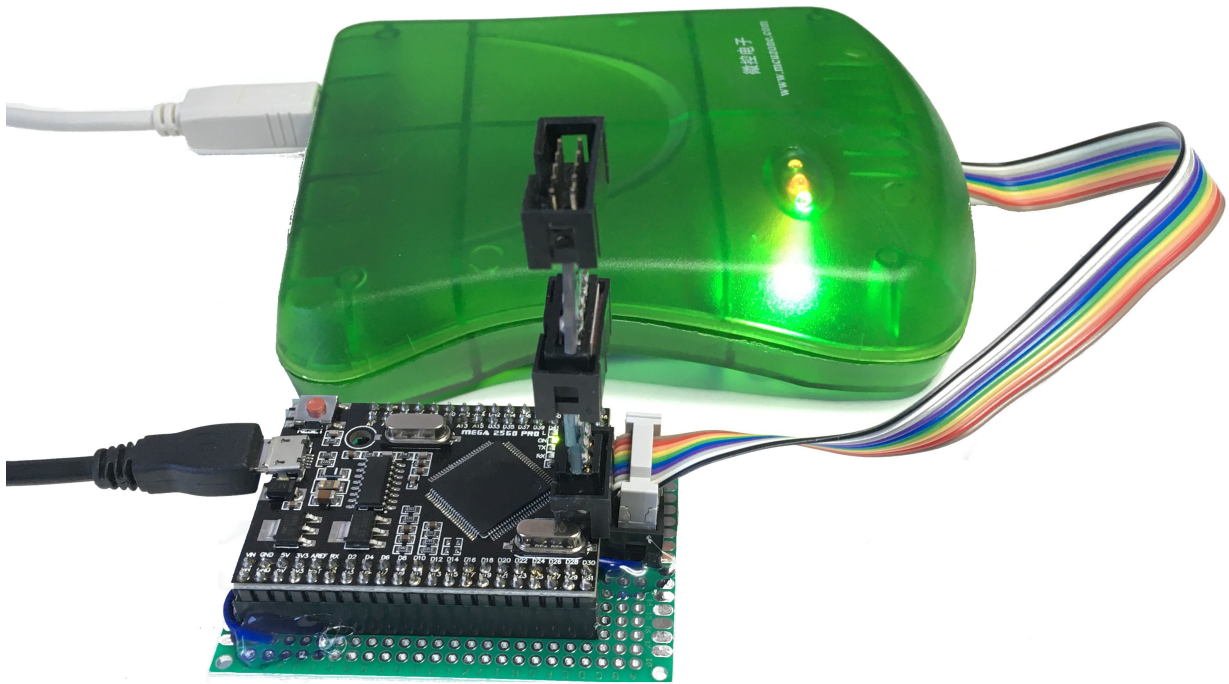
```
set $pc += 2
continue
```

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

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

Homebrew adapter board

Since connecting the squid cable each time is tedious, I created this adapter board that adds a boxheader to connect the JTAG. The board still allows the use of Dupont wires on the Arduino Mega Pro keeping it as development friendly as possible while retaining the smaller form factor as compared to the regular Arduino Mega.



Homebrew adapter board.

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>
- ATmega2560 datasheet https://ww1.microchip.com/downloads/en/devicedoc/atmel-2549-8-bit-avr-microcontroller-atmega640-1280-1281-2560-2561_datasheet.pdf

Public Domain notice and disclaimer

The author has placed this work in the Public Domain, thereby relinquishing all copyrights. Everyone is free to use, modify, republish, sell or give away this work without prior consent from anybody.

This documentation is provided on an ``as is'' basis, without warranty of any kind. Use at your own risk! Under no circumstances shall the author(s) or contributor(s) be liable for damages resulting directly or indirectly from the use or non-use of this documentation.