

Debugging with Atmel ICE and PlatformIO

NOTE: You must have your board's fuse bits set before you can debug.

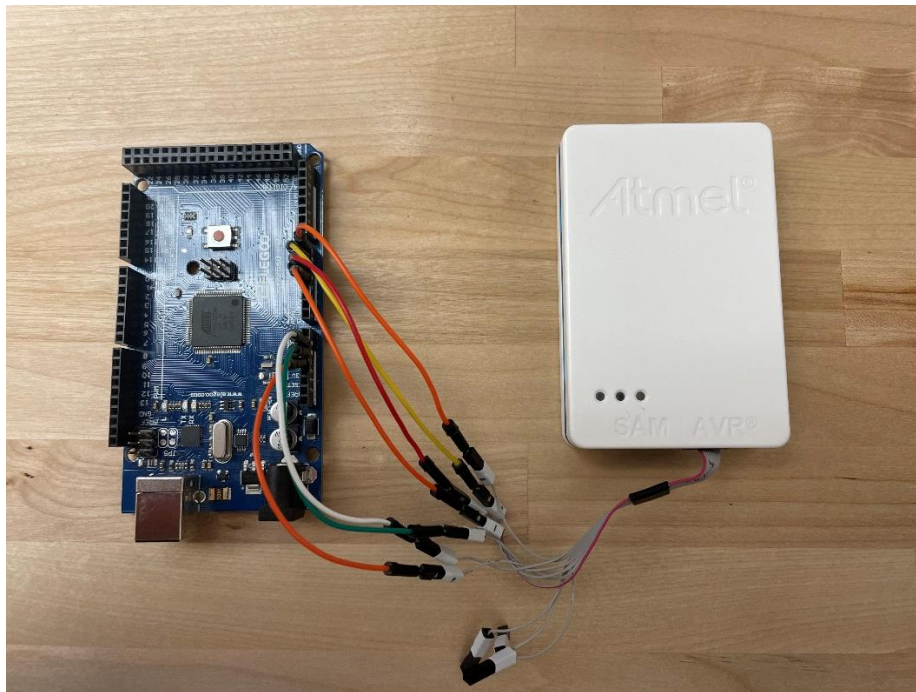
Step 1: Connecting Atmel ICE to JTAG on ATmega2560

To begin, take the 10-pin squid cable and connect the 100 mm connector into the AVR port of the Atmel ICE. With some wires from your kit, connect the cable to the JTAG of the Atmega2560 located in the high nybble of PORTF. The pins are as follows:

	Board →	Squid Cable
[TCK]	PF4 (A4)	→ Pin 1
[TMS]	PF5 (A5)	→ Pin 5
[TDO]	PF6 (A6)	→ Pin 3
[TDI]	PF7 (A7)	→ Pin 9
[VTG]	Vcc (5v)	→ Pin 4
[GND]	Ground	→ Pin 2 & 10

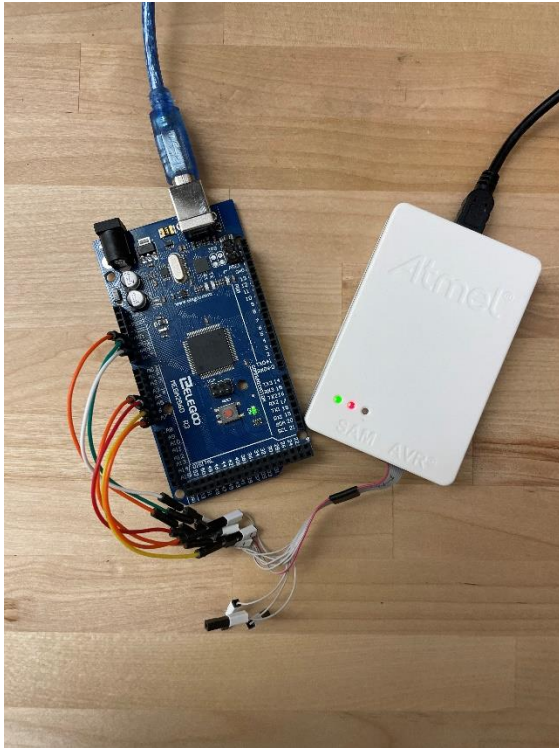
Tips: Pins 6, 7, 8 are not connected. Labels for pins 6 and 9 are not disambiguated; be sure to count the wires to ensure you have the correct one!

You should have something that looks similar to the picture below:



Step 2: Connecting Atmel ICE to AVaRICE

Plug the Atmel ICE into your lab computer. A red light should appear indicating it has power. Then connect your ATmega2560 board to the computer while it is still connected to JTAG, and a green light should appear on the Atmel ICE. If both lights are lit then the debugger is both powered and can see connections to the board.



To validate your connections in the previous step, log into the computer and open a terminal. Type in “avarice -4 -r” and you should see an output similar to what is shown below:

```
s7396626@cenglab03:~$ avarice -4 -r
AVaRICE version 2.14svn20200906, Oct 16 2021 10:40:06

Defaulting JTAG bitrate to 250 kHz.

JTAG config starting.
Found a device, serial number: J42700061816
Reported device ID: 0x9801
Configured for device ID: 0x9801 atmega2560
JTAG config complete.

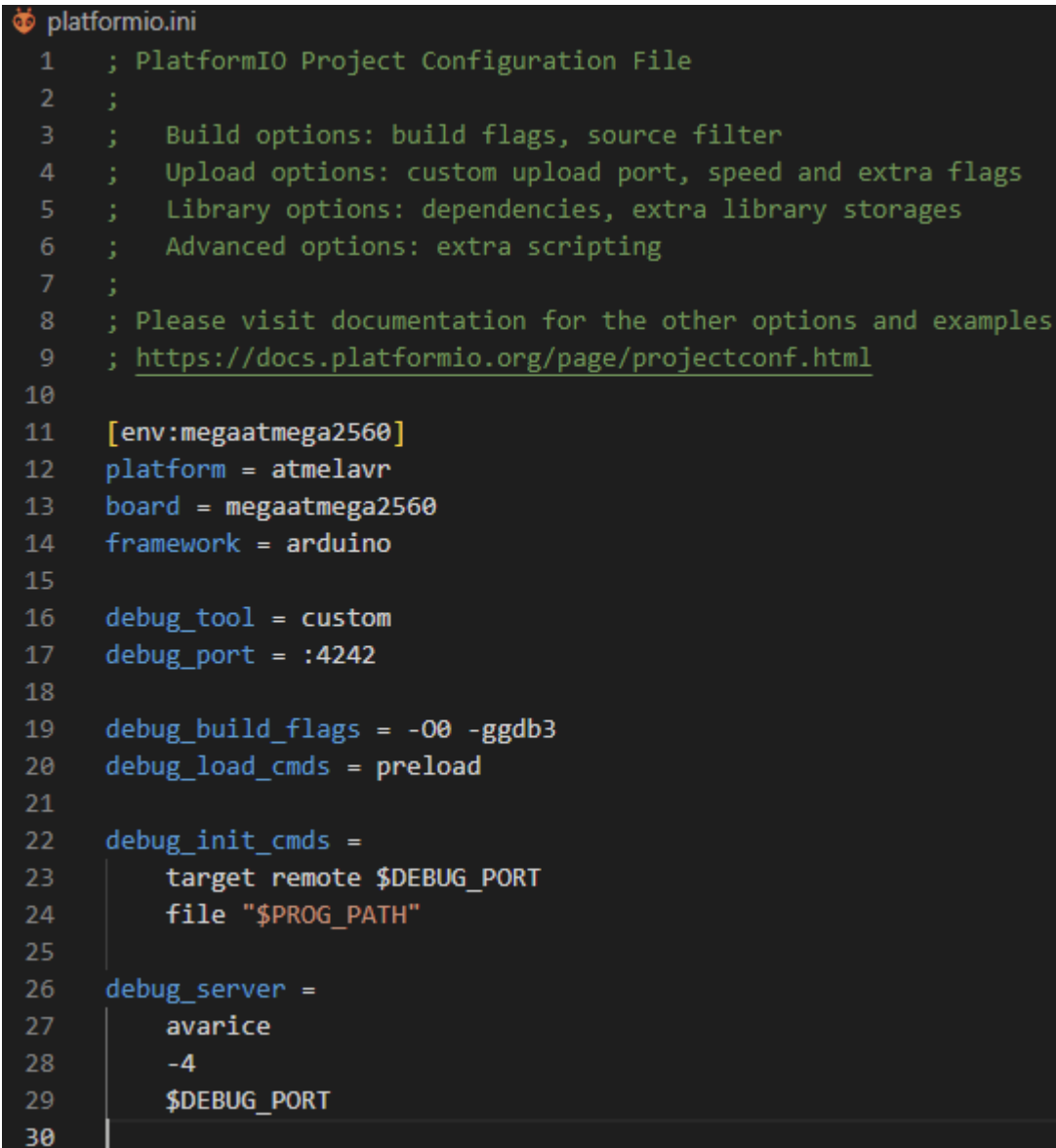
Reading Fuse Bytes:
  Extended Fuse byte -> 0xfd
    High Fuse byte -> 0x18
    Low Fuse byte -> 0xff
```

If you see a message about not receiving an answer from the target, ensure your wiring is correct. Ensure the fuse bytes match EXACTLY. This does not need to be repeated and is only for validation.

Step 3: Connecting AVaRICE to PlatformIO and GDB

The next step is to load up VS Code. You will need to install the extensions required for this lab for the first time (see tutorial on Visual Studio Code and PlatformIO).

Open or create a project and open your “platformio.ini” file. Replace its contents with the following:

A screenshot of a code editor showing the contents of a file named 'platformio.ini'. The file is a PlatformIO project configuration file. It contains comments for build options, upload options, library options, and advanced options. It also sets the environment to 'megaatmega2560', the platform to 'atmelavr', the board to 'megaatmega2560', and the framework to 'arduino'. Debugging options are configured to use 'custom' debug tool on port ':4242', with build flags '-O0 -ggdb3' and preload commands. The debug init commands are 'target remote \$DEBUG_PORT' and 'file "\$PROG_PATH"'. The debug server is set to 'avarice' on port '-4' and '\$DEBUG_PORT'.

```
platformio.ini
1  ; PlatformIO Project Configuration File
2  ;
3  ; Build options: build flags, source filter
4  ; Upload options: custom upload port, speed and extra flags
5  ; Library options: dependencies, extra library storages
6  ; Advanced options: extra scripting
7  ;
8  ; Please visit documentation for the other options and examples
9  ; https://docs.platformio.org/page/projectconf.html
10
11 [env:megaatmega2560]
12 platform = atmelavr
13 board = megaatmega2560
14 framework = arduino
15
16 debug_tool = custom
17 debug_port = :4242
18
19 debug_build_flags = -O0 -ggdb3
20 debug_load_cmds = preload
21
22 debug_init_cmds =
23     target remote $DEBUG_PORT
24     file "$PROG_PATH"
25
26 debug_server =
27     avarice
28     -4
29     $DEBUG_PORT
30
```

This will start AVaRICE every time debugging is started and will have it interface the Atmel ICE with GDB over the specified port. GDB will load the firmware from the built program and debugging can begin.

Select “Start Debugging” from the quick access menu of PlatformIO and VS Code should hook into GDB automatically.

If successful you should be able to set breakpoints, inspect variables, read registers and step through your code!