Enabling Ethernet Connectivity on Octoclock and Octoclock-G

Contents

AN-800 by Sam Reiter and Michael Dickens

This application note covers, in detail, the steps required to program an Octoclock or Octoclock-G to allow Ethernet connectivity and communication with UHD. This guide serves as a supplement to the UHD Manual's coverage of the topic, and has been tested with Ubuntu 19.10 and 20.04. This guide applies to Octoclock, Octoclock-G, and CDA-2990 devices. Any of these names is interchangeable with "Octoclock" in this document.

To get started, make sure that this guide is appropriate for your device(s). Currently shipping Octoclocks will come with the new firmware pre-loaded and do not require the update in this guide. You can *usually* determine which firmware / bootloader on an Octoclock with a simple `ping` test:

1. Connect the 6V DC power supply to the Octoclock
2. Connect your Octoclock's RJ-45 port to a host PC via an Ethernet cable
3. Configure your host's Ethernet port to a static connection with the IPV4 address: `192.168.10.1`
4. Configure your host's Ethernet port with a subnet mast of: `255.255.255.0`
5. Turn your host's Ethernet port off and back on for changes to take effect
6. In a terminal, issue the command:

```
ping 192.168.10.3
```

A device with the old bootloader will *not* respond to a `ping`; nor will a device that is bricked or one where the EEPROM settings are garbled. In this case, proceed to the next section and try updating the firmware. If after a few tries the device will still not `ping`, then see the NOTES below.

A successful `ping` between devices means that your device is already configured with updated firmware, and this guide is not necessary. You can, of course, do the steps in this guide and they should work; you might want to do this if EEPROM settings are not sticking, where the firmware might be corrupted, or where you cannot otherwise access the device via Ethernet.

NOTES:

1. This guide assumes the IP address of the Octoclock is in subnet 192.168.10.X; this is the default EEPROM setting. If the Octoclock's IP address has been changed to some other subnet *and you know what it is*, then use it instead of "192.168.10" throughout this guide.
2. If you do *not* know the assigned non-default IP address of the Octoclock, then this guide will not help and the ethernet portion of the Octoclock may or not be functional; there is no good way to know.
3. This guide will *not* work for recovering bricked devices.

If this guide does not allow Ethernet connectivity with an Octoclock, please contact Ettus Support for assistance, as there are ways to use this guide with custom bootloader and firmware that *might* restore the device to functionality.

- Philips head screwdriver
- ATMEL-ICE Programmer (or comparable AVR programmer) with SPI/ISP cable; the BASIC model will work
- Ethernet cable

To use the ATMEL-ICE programmer, the `avrdude` utility must be installed. The version of `avrdude` should be >= 6.1. At the time of this guide, version 7.0 is the latest version and seems to work the same as versions 6.1 through 6.4. This guide will cover a build of avrdude 6.1 from source as it is known to work.

Install the following dependencies:

```
sudo apt-get install bison flex libftdi1-dev libftdi-dev
```

Download the `avrdude-6.1.tar.gz` release here, for example via the following command:

```
wget https://download.savannah.gnu.org/releases/avrdude/avrdude-6.1.tar.gz
```

Uncompress the tarball:

```
tar xf avrdude-6.1.tar.gz
```

Enter the source directory:

```
cd avrdude-6.1
```

Run the configure script:

```
./configure
```

Expected output:

```
<truncated output>
```

```
Configuration summary:
----------------------
DO HAVE    libelf
DO HAVE    libusb
DO HAVE    libusb_1_0
DO HAVE    libftdi1
DO HAVE    libftdi (but prefer to use libftdi1)
DON'T HAVE libhid
DO HAVE    pthread
DISABLED   doc
ENABLED    parport
DISABLED   linuxgpio
```

Build the code:

```
make
```

Install the executable:

```
sudo make install
```

Update Linux dynamic library cache:

```
sudo ldconfig
```

Test your `avrdude` installation

```
avrdude -?
```

Expected output:

```
<truncated output>

avrdude version 6.1, URL: <http://savannah.nongnu.org/projects/avrdude/>
```

If UHD is not already installed, install your preferred version with this guide: Building and Installing the USRP Open-Source Toolchain (UHD and GNU Radio) on Linux

Download images for UHD:

```
sudo uhd_images_downloader
```

Verify that you have **octoclock_bootloader.hex** and **octoclock_r4_fw.hex**

```
ls -l /usr/local/share/uhd/images | grep octoclock
```

Expected output

```
-rw-r--r-- 1 root root       17332 Jun  6  2019 octoclock_bootloader.hex
-rw-r--r-- 1 root root       22845 Jun  6  2019 octoclock_r4_fw.hex
```

Change to your images directory:

```
cd /usr/local/share/uhd/images
```

Leave this terminal open for future steps.

If you are building UHD from source then you can pretty easily build the bootloader and firmware from source. Doing this is particularly useful for creating hex files that ignore EEPROM networking settings, instead using the defaults.

The Octoclock firmware sources are located relative to the top-level UHD GIT clone directory in `firmware/octoclock`. We require the AVR cross-compiler suite for this build, installed via:

```
sudo apt install avr-libc
```

From the directory noted above, do the "usual" build commands:

```
mkdir build
cd build
cmake ..
```

Expected output:

```
-- The C compiler identification is GNU 5.4.0
-- Check for working C compiler: /usr/bin/avr-gcc
-- Check for working C compiler: /usr/bin/avr-gcc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: XXXX/firmware/octoclock/build
```

Note the C compiler must be `/usr/bin/avr-gcc`. Assuming it is, then:

```
make
```

Assuming this works, then doing a listing on the `build` directory will show the built HEX files (`ls *.hex`):

```
octoclock_bootloader.hex  octoclock_r4_fw.hex
```

The bootloader file can be used instead of the one in UHD images for the `avrdude` command by issuing the `avrdude` command from this build directory instead of the `images` directory.

To use the firmware hex file, use the following command from this build directory:

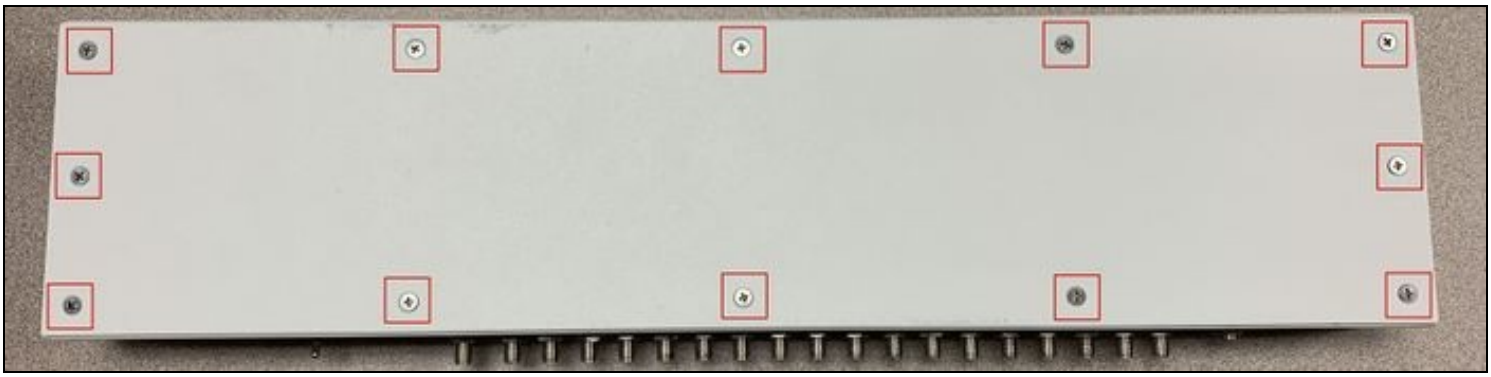
```
uhd_image_loader --args="type=octoclock,addr=192.168.10.3" --fpga=octoclock_r4_fw.hex
```

NOTES:

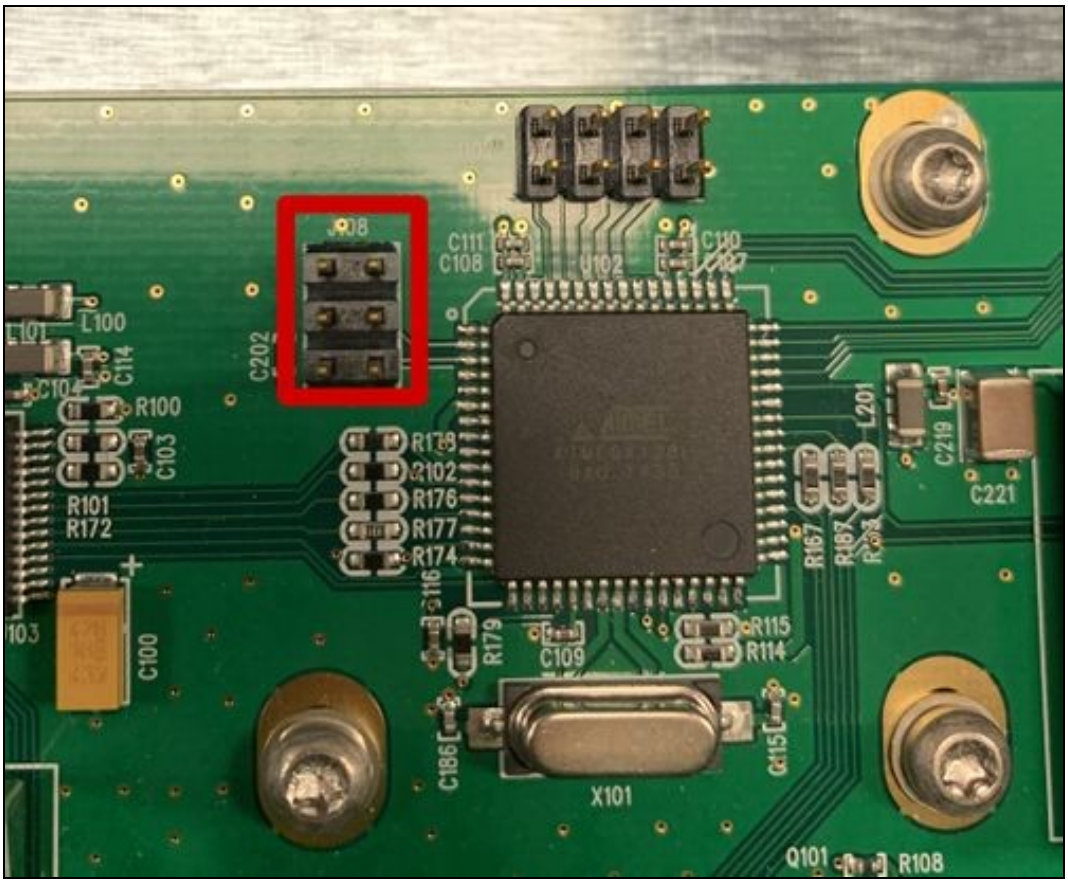
1. We strongly recommend using the bootloader and firmware HEX files from the same place, *either* UHD images *or* this build directory.
2. To install updated firmware, you must generally install updated bootloader first, as described later in this AppNote.

Begin this section with the Octoclock completely unplugged.

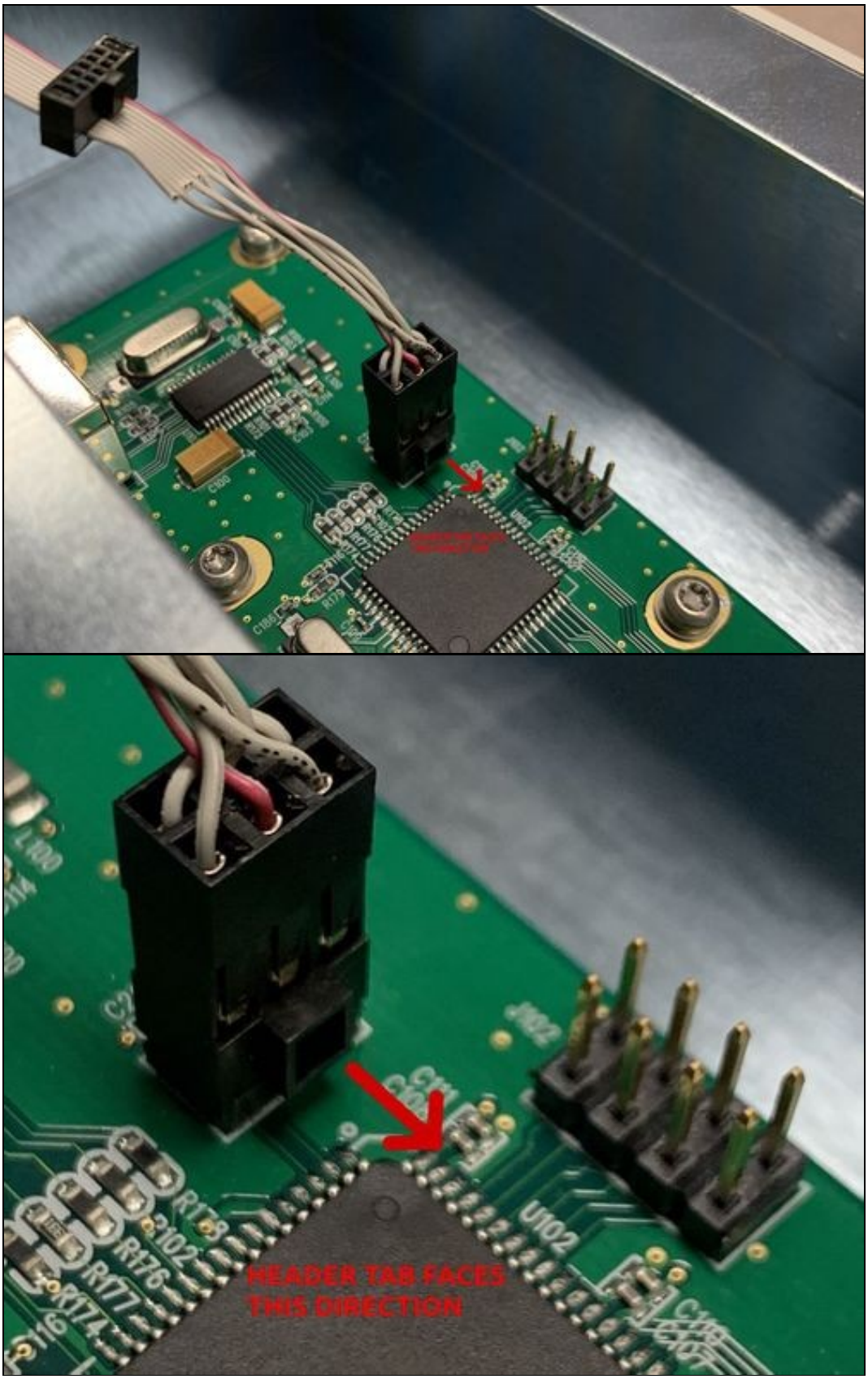
Remove the top plate from the Octoclock, exposing the PCB. There are 12 screws securing the top plate.



With the top plate removed, locate the 6-pin header (J108, male) for SPI communication with the ATmega128.

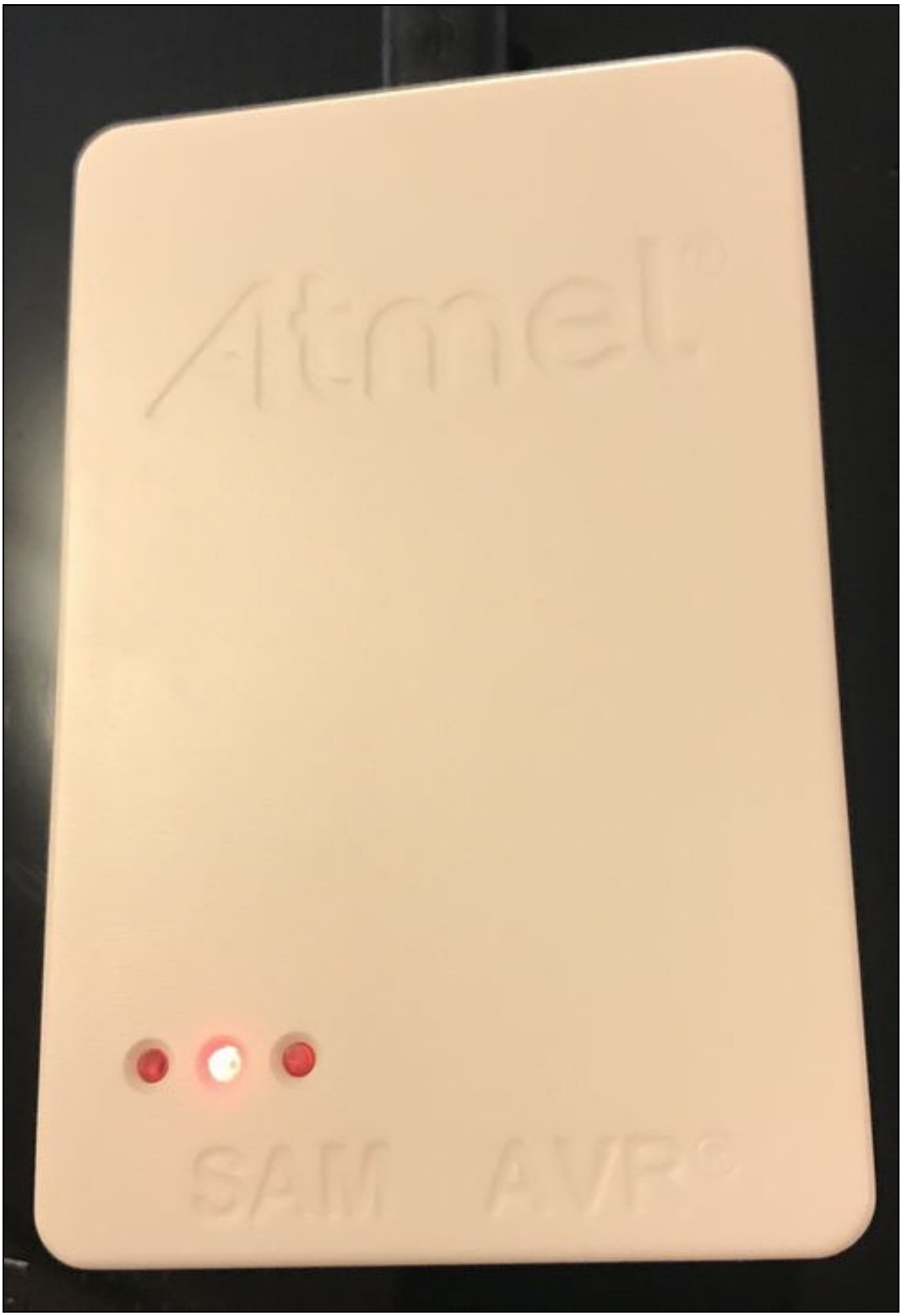


Connect the ATMEL-ICE's 6-pin header (female) to the SPI header of the Octoclock as shown below. The tab of the header must be facing the ATmega128 chip.



Note: If you plug the header in backwards, `avrdude` is expected to return a `please check your connections` message after a programming failure. See the section "Troubleshooting" at the bottom of this document for more detail.

Disconnect the Octoclock from power, then hook up the AMTEL-ICE connections: Connect the other end of the SPI cable to the ATMEL-ICE's AVR squid connector, and connect the ATMEL-ICE to the host computer using the micro-usb cable. The connections will be `Host --> Micro USB Cable --> ATMEL-ICE --> Squid to SPI Cable --> Octoclock SPI header`. There should be a single LED lit -- the middle one (red) -- on the programmer:



Supply power to the Octoclock with the 6V power brick. You should see the Octoclock's Power LED come on and the left LED (green) on the ATMEL-ICE programmer illuminate, such that the left 2 LEDs are lit. The overall setup should look something like this:

With the terminal that is open in the same directory as the .hex images, run the following command:

```
sudo avrdude -p atmega128 -c atmelice_isp -P usb -U efuse:w:0xFF:m -U hfuse:w:0x80:m -U lfuse:w:0xEF:m -U flash:w:octoclock_bootloader.hex:i
```

NOTES:

1. This command sometimes fails the first time; if so, try running it again.
2. This command *must* be executed with `sudo`, otherwise `avrdude` will error out trying to open the USB interface. See the "Troubleshooting" section for details.
3. If you are using a programmer other than the ATMEL-ICE, you will need to change the -c parameter to match your programmer. Valid programmers for your version of avrdude can be found by running:

```
avrdude -c help
```

The expected output from a successful run of `avrdude` is as follows:

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

Reading | ################################################## | 100% 0.01s

avrdude: Device signature = 0x1e9702 (probably m128)
avrdude: NOTE: "flash" memory has been specified, an erase cycle will be performed
         To disable this feature, specify the -D option.
avrdude: erasing chip
avrdude: reading input file "0xFF"
avrdude: writing efuse (1 bytes):

Writing | ################################################## | 100% 0.01s

avrdude: 1 bytes of efuse written
avrdude: verifying efuse memory against 0xFF:
avrdude: load data efuse data from input file 0xFF:
avrdude: input file 0xFF contains 1 bytes
avrdude: reading on-chip efuse data:

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

avrdude: verifying ...
avrdude: 1 bytes of efuse verified
avrdude: reading input file "0x80"
avrdude: writing hfuse (1 bytes):

Writing | ################################################## | 100% 0.01s

avrdude: 1 bytes of hfuse written
avrdude: verifying hfuse memory against 0x80:
avrdude: load data hfuse data from input file 0x80:
avrdude: input file 0x80 contains 1 bytes
avrdude: reading on-chip hfuse data:

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

avrdude: verifying ...
avrdude: 1 bytes of hfuse verified
avrdude: reading input file "0xEF"
avrdude: writing lfuse (1 bytes):

Writing | ################################################## | 100% 0.01s

avrdude: 1 bytes of lfuse written
avrdude: verifying lfuse memory against 0xEF:
```

```
avrdude: load data lfuse data from input file 0xEF:
avrdude: input file 0xEF contains 1 bytes
avrdude: reading on-chip lfuse data:

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

avrdude: verifying ...
avrdude: 1 bytes of lfuse verified
avrdude: reading input file "octoclock_bootloader.hex"
avrdude: writing flash (129012 bytes):

Writing | ################################################## | 100% 0.00s

avrdude: 129012 bytes of flash written
avrdude: verifying flash memory against octoclock_bootloader.hex:
avrdude: load data flash data from input file octoclock_bootloader.hex:
avrdude: input file octoclock_bootloader.hex contains 129012 bytes
avrdude: reading on-chip flash data:

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

avrdude: verifying ...
avrdude: 129012 bytes of flash verified

avrdude: safemode: Fuses OK (E:FF, H:80, L:EF)

avrdude done.  Thank you.
```

Upon successful burning of a bootloader, updated firmware must be loaded onto the Octoclock. Connect the USRP to your host via Ethernet and configure your host's IP as noted in the "Verify Current Octoclock Configuration" section. Next, run

```
uhd_find_devices
```

Expected output:

```
--------------------------------------------------
-- UHD Device 0
--------------------------------------------------
Device Address:
    addr: 192.168.10.3
    type: octoclock-bootloader
```

This means that UHD successfully recognizes your device's bootloader and can download the firmware image. If you get a "No Devices Found" return from UHD, something went wrong during the bootloader burn. In this case you should rerun the `avrdude` command with verbose output enabled; see the "Troubleshooting" section for more detail.

If UHD successfully recognized the Octoclock's bootloader, run the following command:

```
uhd_image_loader --args="type=octoclock,addr=192.168.10.3"
```

Once this completes, your OctoClock will load its firmware. Power cycle the device, then run the `uhd_find_devices` utility again, and the output should be similar to the following:

```
--------------------------------------------------
-- UHD Device 0
--------------------------------------------------
Device Address:
    addr: 192.168.10.3
    type: octoclock
    name:
    serial: XXXXXX
```

Note that the Octoclock will enter its bootloader once it first receives power. It will take ~10s to boot and be recognized as an Octoclock as seen in the above output.

As a final step, the device's EEPROM will need to be updated. On the back of your device, you will see a label sticker with a serial number (labeled S/N) and a MAC address (labeled MAC). For later use, the MAC address will have to be used in a different format than is on the label. As an example, if the label lists the MAC address as `00802F112233`, you will need to format it as `00:80:2F:11:22:33`.

Update the Octoclock's EEPROM with the following command:

```
/usr/local/lib/uhd/utils/octoclock_burn_eeprom --args="addr=192.168.10.3" --values="mac-addr=<FORMATTED MAC HERE>,ip-addr=192.168.10.3,netmas
```

Verify everything with

```
/usr/local/lib/uhd/utils/octoclock_burn_eeprom --args="addr=192.168.10.3" --read-all
```

Expected output:

```
Creating OctoClock device from args: addr=192.168.10.3
[INFO] [UHD] linux; GNU C++ version 9.2.1 20191008; Boost_106700; UHD_3.15.0.HEAD-0-gaea0e2de
[INFO] [OCTOCLOCK] Opening an OctoClock device...
[INFO] [OCTOCLOCK] Detecting internal GPSDO...
[INFO] [GPS] Found an internal GPSDO: LC_XO, Firmware Rev 0.929a
[INFO] [OCTOCLOCK] Detecting external reference...false
[INFO] [OCTOCLOCK] Detecting switch position...Prefer internal

Fetching current settings from EEPROM...
EEPROM ["mac-addr"] is "<MAC ADDR>"
EEPROM ["ip-addr"] is "192.168.10.3"
EEPROM ["gateway"] is "192.168.10.1"
EEPROM ["netmask"] is "255.255.255.0"
Device is using internal reference

    EEPROM ["serial"] is "<SERIAL NUMBER>"
    EEPROM ["name"] is ""
    EEPROM ["revision"] is "4"

Power-cycle your device to allow any changes to take effect.
```

Power cycle your device and your Octoclock firmware and EEPROM have been updated!

- UHD Manual - Octoclock Update
- Octoclock Schematic
- ATMEL-ICE User Guide
- ATMEL SPI Pinout
- AVRDUDE Releases

This process has been run and confirmed in Ubuntu 19.10 and 20.04. Other versions of Linux may require different versions of dependencies to be installed. `avrdude` also runs natively on Windows.

1. If you run the `avrdude` command *without* `sudo`, then it should fail showing something like the following. Make sure to use `sudo` with running the `avrdude` command!

```
avrdude: usb_open(): cannot read serial number "error sending control message: Operation not permitted"
avrdude: usb_open(): cannot read product name "error sending control message: Operation not permitted"
avrdude: usbdev_open(): WARNING: failed to set configuration 1: could not set config 1: Operation not permitted
avrdude: usbdev_open(): error claiming interface 0: could not claim interface 0: Operation not permitted
avrdude: usbdev_open(): error claiming interface 1: could not claim interface 1: Operation not permitted
avrdude: usbdev_open(): no usable interface found
avrdude: jtag3_open_common(): Did not find any device matching VID 0x03eb and PID list: 0x2141
```

2. If `avrdude` fails more than once when using `sudo`, then try running it again with `-v` flags to make the output verbose. Here is the output of `avrdude` with verbose flags set:

```
sudo avrdude -v -v -p atmega128 -c atmelice_isp -P usb -U efuse:w:0xFF:m -U hfuse:w:0x80:m -U lfuse:w:0xEF:m -U flash:w:octoclock_bootloader.
```

Output when SPI plug is plugged into the header backwards:

```
avrdude: Version 6.1, compiled on Jan 10 2020 at 15:41:02
         Copyright (c) 2000-2005 Brian Dean, http://www.bdmicro.com/
         Copyright (c) 2007-2014 Joerg Wunsch

         System wide configuration file is "/usr/local/etc/avrdude.conf"
         User configuration file is "/root/.avrduderc"
         User configuration file does not exist or is not a regular file, skipping

         Using Port                    : usb
         Using Programmer              : atmelice_isp
avrdude: stk500v2_jtag3_open()
avrdude: usbdev_open(): Found Atmel-ICE CMSIS-DAP, serno: J42700007132
avrdude: Found CMSIS-DAP compliant device, using EDBG protocol
avrdude: jtag3_edbg_prepare(): connection status 0x01
avrdude: Sending sign-on command: 0x80 (509 bytes msg)
         AVR Part                      : ATmega128
         Chip Erase delay              : 9000 us
         PAGEL                         : PD7
         BS2                           : PA0
         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         4    12    64    0 no       4096    8      0  9000  9000 0xff 0xff
           flash         33     6   128    0 yes    131072  256    512  4500  4500 0xff 0xff
           lfuse          0     0     0    0 no          1    0      0  9000  9000 0x00 0x00
           hfuse          0     0     0    0 no          1    0      0  9000  9000 0x00 0x00
           efuse          0     0     0    0 no          1    0      0  9000  9000 0x00 0x00
           lock           0     0     0    0 no          1    0      0  9000  9000 0x00 0x00
           calibration    0     0     0    0 no          4    0      0     0     0 0x00 0x00
           signature      0     0     0    0 no          3    0      0     0     0 0x00 0x00

         Programmer Type : JTAG3_ISP
         Description     : Atmel-ICE (ARM/AVR) in ISP mode
avrdude: jtag3_getparm()
avrdude: Sending get parameter (scope 0x01, section 1, parm 0) command: 0x84 (509 bytes msg)
         Vtarget         : 2.7 V
         SCK period      : 125.00 us

avrdude: jtag3_setparm()
avrdude: Sending set parameter (scope 0x12, section 0, parm 0) command: 0x80 (509 bytes msg)
avrdude: jtag3_setparm()
avrdude: Sending set parameter (scope 0x12, section 0, parm 1) command: 0x80 (509 bytes msg)
avrdude: jtag3_setparm()
avrdude: Sending set parameter (scope 0x12, section 1, parm 0) command: 0x80 (509 bytes msg)
avrdude: stk500v2_command(): command failed
avrdude: initialization failed, rc=-1
         Double check connections and try again, or use -F to override
         this check.

avrdude: stk500v2_jtag3_close()
avrdude: jtag3_close()
avrdude: Sending AVR sign-off command: 0x80 (509 bytes msg)
avrdude: Sending sign-off command: 0x80 (509 bytes msg)

avrdude done.  Thank you.
```