Ada Development on AVR Microcontroller

Sowebio SARL
15, rue du Temple
17310 – St Pierre d'Oléron – France

Capital 15 000 EUR – SIRET 844 060 046 00019 – RCS La Rochelle – APE 6201Z – TVA FR00844060046

Ed.	*Release*	*Comments*	
1	20050505	Initial release in Texinfo.	sr
2	20120104	First windows setup with JTAGICE MkII.	sr
3	20171114	Port from GNU Texinfo format to ISO OpenDocument format.	sr
23	20171122	Preliminary edition for Paris Open Source Summit 2017.	sr
24	20171204	Add appendix, GNU/Linux build section. Some typos corrected.	sr
35	20221205	GNU/Linux edition, on going work starting	sr
48	20230108	Initial release with full features setup	sr
50	20230109	First public release	sr
54	20230112	New Ada logo “Horizon”	sr
55			

- Authors

  Stéphane Rivière [Number Six] - stef@genesix.org [CTO Sowebio]

  Some documentation parts [mainly related to the AVR-Ada chapter] are borrowed from Rolf Ebert rolf.ebert.gcc@gmx.de work https://github.com/RREE. Rolf is the AVR Ada tool chain long haul developer. AVR-Ada environment is a amazing work.

- Acknowledgments

  I would like to thank, at one hand, the AVR-Ada project leader Rolf Ebert and his dream team : Tero Koskinen, Warren W. Gay [VE3WWG[1]], Bernd Trog and, at the other hand, John Leimon, author of a valuable AVR-Ada fork[2].

- Manual

  Stéphane Rivière [Number Six] - stef@genesix.org [CTO Sowebio - FM1US/F1USA[3]]

  The “Excuse me I’m French” speech - The main author of this manual is a Frenchman with basic English skills. Frenchmen are essentially famous as frog eaters[4]. They have recently discovered that others ~~forms of communication~~ languages are widely used on earth. So, as a frog eater, I've tried to write some stuff in this foreign dialect loosely known here under the name of English. However, it’s a well known fact that frogs don’t really speak English. So your help is welcome to correct this bloody manual, for the sake of the wildebeests, and penguins too.

- Syntax notation

  Inside a command line:
  - A parameter between brackets **[ ]** is optional;
  - Two parameters separated by **|** are mutually exclusives.

  An important notice:

  ⇨ This is an important notice !

- Edition

  1 55 - 2023-01-12

[1] International amateur radio call sign - https://en.wikipedia.org/wiki/Amateur_radio.

[2] https://github.com/evilspacepirate/avr-ada

[3] International amateur radio call sign - https://en.wikipedia.org/wiki/Amateur_radio.

[4] We could be famous as designers of the Concorde, Ariane rockets, Airbus planes or even Ada computer language but, definitely, Frenchmen have to wear beret with bread baguette under their arm to go eating frogs in a smokey tavern. That's *le cliché* :]

https://this-page-intentionally-left-blank.org

Contents

https://this-page-intentionally-left-blank.org

Introduction

Keep It Simple, Stupid.
Clarence Leonard "Kelly" Johnson - Director of the Blackbirds program, the most complex aircrafts of their time, with performances still unmatched 60 years later.

1 About this manual

1.1 Introduction

Ada Development on AVR Microcontroller (ADAM) is based on the **latest GNAT, Alire and GNAT Studio releases and allows real-time AVR debugging in GNAT Studio**[5].

We also wanted to get acquainted with Alire while keeping our usual environment, based on GNAT Studio. This IDE has become really stable, keeping its distinctive qualities: perfect GNAT integration, high customizability, smooth operation, intuitive interface.

Thanks to the author of Alire, Alejandro R. Mosteo, we also discover that GNAT Studio integrates perfectly with Alire, giving the whole thing a nice coherence.

This work is part of a more general desire to empower the community with respect to the defunct CE editions of Adacore. We adhere to this new Adacore policy.

Between this new direction, the arrival of Alire, the availability of many Crates and the first successes of the community building GNAT Studio independently, the Ada community is probably entering a new era.

Today in 2023, thanks to the work (and their friendly help) of Rolf Ebert (AVR-Ada environment and AVR_Ada to Alire conversion), Fabien Chouteau and Adacore (GNAT-AVR to Alire conversion and Alire promotion), here is a **comprehensive tutorial to get the easiest environment to develop in Ada on 8-bits AVR targets under Linux.**

1.2 History

In 2005, this document was originally a Texinfo based manual, created under Windows, with the help of the AIDE (Ada Instant Development Environment) distribution[6].

[5] We used to debug an AVR-Ada target in real-time on Windows with GNAT and GDB, with the help of the DDD or Insight front-ends, but unfortunately not with GNAT Studio, whereas it was natural with GVD (GNU Visual Debugger), the ancestor of GPS and GNAT Studio. In addition, Sowebio has been a full-penguin company since 2018. We can't imagine virtualize a Windows for this job. The honor of the penguin was at stake.

[6] https://stef.genesix.org/aide/aide.html

In 2012, this manual has been updated to the OpenDocument ISO format[7], with the help of LibreOffice, describing the first windows setup with JTAGICE MkII.

While the previous editions were Windows centric. The 2017 edition could be seen as a research work about :
- All the possibilities offered to develop projects in Ada with AVR microcontrollers ;
- Implementing a full functional, up-to-date, Linux Ada 2012[8] toolchain with true real-time debugging capabilities. This last point failed at this time.

2 Ada

For general information about the Ada language, see the AIDE manual on the Sowebio Github repository https://github.com/sowebio/aide-doc.

3 AVR Ada environment

Mailing list (devel) : https://lists.sourceforge.net/lists/listinfo/avr-ada-devel
Mailing list (cvs/ro) : https://lists.sourceforge.net/lists/listinfo/avr-ada-cvs-commit
Mailing list web archive : https://sourceforge.net/p/avr-ada/mailman/avr-ada-devel

4 Microchip AVR Resources

Atmel has been acquired by Microchip in 2016.

Home page : https://www.microchip.com/en-us/products/microcontrollers-and-microprocessors/8-bit-mcus/avr-mcus

Parametric search : https://www.microchip.com/en-us/parametric-search.html/716

Documentation : https://microchipsupport.force.com/s/

https://microchipsupport.force.com/s/communityknowledge then search avr.

https://www.microchip.com/doclisting/TechDoc.aspx?type=ReferenceManuals

[7] ISO/IEC 26300

[8] Last Ada compiler level (ISO/IEC 8652-2012)

GNAT toolchain

Doubling the number of programmers on a late project does not make anything else than double the delay.
Second Brook's Law

1 GNAT Studio

1.1 Installation

Pick the latest in https://github.com/AdaCore/gnatstudio/releases.

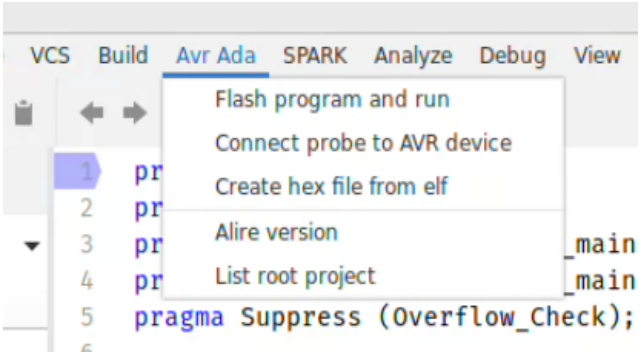
Taking 23.0w release as example:

```
user@system : wget https://github.com/AdaCore/gnatstudio/releases/download/gnatstudio-cr-20220512/gnat-studio-23.0w-20220512-x86_64-linux-bin.tar.gz

user@system : tar -xvf gnatstudio-23.0w-20220512-x86_64-linux-bin.tar.gz

user@system : cd gnatstudio-23.0w-20220512-x86_64-linux-bin

user@system : sudo ./doinstall
```

1.2 Additional setup

An extra menu is created to ease the AVR Ada workflow.

1.2.1 Extra menu



Create $HOME/.config/.gnatstudio/plug-ins/avrada.xml:

```
<?xml version="1.0" ?>

<!--
GNAT Studio menu extender for AVR-Ada utilities
```

```
-->

<avrada>

  <action name="probeflash" output="default output">
    <external output="Avr Ada" > xterm -e ./avrada-flash.sh</external>
  </action>

  <action name="probedebug" output="default output">
    <external output="Avr Ada" > xterm -hold -e ./avrada-debug.sh</external>
  </action>

  <action name="createhex" output="default output">
    <external output="Avr Ada" > xterm -e ./avrada-hex.sh</external>
  </action>

  <action name="alireversion">
    <external>alr version</external>
  </action>

  <action name="projectdir">
    <shell>pwd</shell>
    <external>ls -lpX --time-style=long-iso</external>
  </action>

  <submenu>

    <title>Avr Ada</title>
      <menu after ="Build" action="probeflash">
        <title>Flash program and run</title>
      </menu>
      <menu action="probedebug">
        <title>Connect probe to AVR device</title>
      </menu>
      <menu action="createhex">
        <title>Create hex file from elf</title>
      </menu>

      <menu><title/></menu>

      <menu action="alireversion">
        <title>Alire version</title>
      </menu>
      <menu action="projectdir">
        <title>List root project</title>
      </menu>

  </submenu>

</avrada>
```

1.2.2 Extra menu files for each project

At the root of each project, create theses files. You must adjust the bolded parameters for your environment and/or your project name.

avrada-debug.sh

```
echo "Connect probe to AVR device, waiting for GNAT Studio debug session..."
echo ""

avarice --part atmega1280 --jtag usb --mkII --reset-srst :4242
```

avrada-flash.sh

```
echo "Create hex file, flash device and run program..."
echo ""

UTILITY="$(find $HOME/.config/alire -name avr-objcopy) -v -O ihex flash_led.elf flash_led.hex"
echo $UTILITY
$UTILITY

avrdude -p m1280 -c jtagmkII -P usb -F -U flash_led.hex

echo "Wait 20 seconds before closing..."
```

```
ping -c 20 localhost > /dev/null
```

avrada-hex.sh

```
echo "Create hex file..."
echo ""

UTILITY="$(find $HOME/.config/alire -name avr-objcopy) -v -O ihex flash_led.elf flash_led.hex"
echo $UTILITY
$UTILITY

echo ""
echo "Wait 20 seconds before closing..."
ping -c 20 localhost > /dev/null
```

Give them execution rights:

```
user@system : chmod +x avrada-*.sh
```

1.2.3 Extra menu terminal

Xterm is the simplest and agnostic choice for extra menu needs:

```
user@system : sudo apt install xterm
```

2 AVR tools

2.1 AVaRICE

2.1.1 Introduction

AVaRICE is a program which interfaces the GNU Debugger with the AVR JTAGICE mkI and mkII and other debuggers available from Atmel.

It connects to GDB via a TCP socket and communicates via GDB's « serial debug protocol ». This protocol allows GDB to send commands like « set/remove breakpoint » and « read/write memory ». AVaRICE translates this commands into the Atmel protocol used to control the JTAG ICE (or other) debugger.

Because the GDB-AVaRICE connection is via a TCP socket, the two programs do not need to run on the same machine.

2.1.2 Links

https://sourceforge.net/projects/avarice

2.1.3 Install

The last available release in Ubuntu 22.04 LTS is 2.14svn20200906, so no build is mandatory. The last stable release is also 2.14svn20200906.

```
user@system : sudo apt install avarice
```

This command also installs gdb-avr7.7-4.1, which will not be used.

2.1.4 List all accepted devices

Full list is around 100 devices:

```
user@system : avarice -k

AVaRICE version 2.14svn20200906, Oct 16 2021 10:40:06

List of known AVR devices:

Device Name       Device ID     Flash    EEPROM
---------------   ---------   -------   -------
at90can128           0x9781   128 KiB     4 KiB
.../...
atxmega8e5           0x9341    10 KiB   0.5 KiB
```

2.1.5 List all accepted probes

The currently supported debuggers are:
- JTAGICE mkI ;
- JTAGICE mkII ;
- JTAGICE3
- AVR Dragon;
- EDBG Integrated Debugger.

2.2 Avrdude

2.2.1 Introduction

Avrdude is a tool to program ATMEL AVR devices, through many available hardware interfaces.

2.2.2 Links

https://www.nongnu.org/avrdude

2.2.3 Install

The last available release in Ubuntu 22.04 LTS is 6.3-20171130, so no build is mandatory. The last stable release is 6.4 and a 7.0 is on its go.

```
user@system : sudo apt install avrdude avrdude-doc
```

2.2.4 List all accepted devices

```
user@system : avrdude -p?

uc3a0512 = AT32UC3A0512
.../...
m128     = ATmega128
m1280    = ATmega1280
m1281    = ATmega1281
m1284    = ATmega1284
m1284p   = ATmega1284P
m1284rfr2= ATmega1284RFR2
m128rfa1 = ATmega128RFA1
m128rfr2 = ATmega128RFR2
m16      = ATmega16
m161     = ATmega161
m162     = ATmega162
m163     = ATmega163
```

```
.../...
x64d4    = ATxmega64D4
x8e5     = ATxmega8E5
ucr2     = deprecated, use 'uc3a0512'
```

2.2.5 List all accepted probes

```
user@system : avrdude -c?

2232HIO          = FT2232H based generic programmer
4232h            = FT4232H based generic programmer
89isp            = Atmel at89isp cable
.../...
jtag2            = Atmel JTAG ICE mkII
jtag2avr32       = Atmel JTAG ICE mkII im AVR32 mode
jtag2dw          = Atmel JTAG ICE mkII in debugWire mode
jtag2fast        = Atmel JTAG ICE mkII
jtag2isp         = Atmel JTAG ICE mkII in ISP mode
jtag2pdi         = Atmel JTAG ICE mkII PDI mode
jtag2slow        = Atmel JTAG ICE mkII
jtagmkII         = Atmel JTAG ICE mkII
jtagmkII_avr32   = Atmel JTAG ICE mkII im AVR32 mode
.../...
xplainedpro      = Atmel AVR XplainedPro in JTAG mode
xplainedpro_updi = Atmel AVR XplainedPro in UPDI mode
```

2.3 Simavr

2.3.1 Introduction

Simavr introduce itself as a “lean and mean” ATMEL AVR simulator for GNU/Linux.

It handles common cores like ATMega2560, AT90USB162 (with USB!), ATMega1281, ATMega1280, ATMega128, ATMega128rf1, ATMega16M1, ATMega169, ATMega162, ATMega164 / 324 / 644, ATMega48 / 88 / 168 / 328, ATMega8 / 16 / 32, ATTiny25 / 45 / 85, ATTIny44 / 84, ATTiny2313 / 2313v, ATTiny13 / 13a.

According to its home site, it also emulates common real world things like, eeprom, watchdog, IO ports (including pin interrupts), timers, 8 &16 (Normal, CTC and Fast PWM, the overflow interrupt too), UART, including tx & rx interrupts (there is a loop-back/local echo test mode too), SPI, master/slave (including the interrupt i2c Master & Slave), external Interrupts, INT0 and so on, ADC and even self-programming (ie bootloaders).

2.3.2 Links

https://github.com/buserror/simavr
https://www.instructables.com/id/Debugging-AVR-code-in-Linux-with-simavr

2.3.3 Install

The last available release is 1.6 dated 20180110, so build could be mandatory. The previous release is available in Ubuntu 18.04 LTS (simavr version 1.5).

```
user@system : sudo apt install simavr libsimavr-examples
```

3 Alire repository manager

Alire is the Ada Library REpository manager for the Libre and Open Source Ada ecosystem.

A comprehensive presentation paper, from the Alire author, can be found in AUJ Vol 39, Number 3, Sept 2018, P 189. http://www.ada-europe.org/archive/auj/auj-39-3.pdf

3.1 Links

https://github.com/alire-project
https://alire.ada.dev
https://alire.ada.dev/docs
https://gitter.im/ada-lang/Alire
https://www.reddit.com/r/ada
https://twitter.com/mosteobotic

3.2 Install

```
user@system : wget https://github.com/alire-project/alire/releases/download/v1.2.1/alr-1.2.1-bin-x86_64-linux.zip

user@system : sudo unzip -j -o  alr-1.2.1-bin-x86_64-linux.zip bin/alr -d /usr/local/bin

user@system : mkdir $HOME/opt/alire ; cd $HOME/opt/alire
```

3.3 Use

3.3.1 List all crates

```
user@system : alr search --list
```

3.3.2 Search for some specific packages

```
user@system : alr search --full --external-detect avr (all versions)

user@system : alr search --external-detect avr (last versions)
```

4 Native compiler

This installation is somewhat beyond the scope of this document. It is not mandatory and only proposed as a simple example of how to get started with Alire.

4.1 Installing Gprbuild

```
user@system : alr get gprbuild
```

4.2 List all GNAT packages

```
user@system : alr search --external-detect gnat

gnat_arm_elf                 12.2.1    The GNAT Ada compiler - ARM cross-compiler
gnat_avr_elf                 12.2.1    The GNAT Ada compiler - AVR cross-compiler
gnat_math_extensions         1.1.0     Eigenvalues eigenvectors for non-symmetric, Hermitian matrices
gnat_native                  12.2.1    The GNAT Ada compiler - Native
gnat_riscv64_elf             12.2.1    The GNAT Ada compiler - RISC-V cross-compiler
gnatcoll                     23.0.0    GNAT Components Collection - Core packages
gnatcoll_gmp                 23.0.0    GNAT Components Collection - GNU Mult. Precision Arithmetic
gnatcoll_iconv               23.0.0    GNAT Components Collection - iconv binding
gnatcoll_lzma                23.0.0    GNAT Components Collection - lzma binding
gnatcoll_omp                 23.0.0    GNAT Components Collection - OpenMP binding
gnatcoll_postgres            23.0.0    GNAT Components Collection - postgres
gnatcoll_python              21.0.0    GNAT Components Collection - python2 binding
```

```
gnatcoll_python3              23.0.0    GNAT Components Collection - python3 binding
gnatcoll_readline             23.0.0    GNAT Components Collection - readline binding
gnatcoll_sql                  23.0.0    GNAT Components Collection - sql
gnatcoll_sqlite               23.0.0    GNAT Components Collection - sqlite
gnatcoll_syslog               23.0.0    GNAT Components Collection - syslog binding
gnatcoll_xref                 23.0.0    GNAT Components Collection - xref
gnatcoll_zlib                 23.0.0    GNAT Components Collection - zlib binding
gnatcov                       22.0.1    Coverage Analysis Tool
gnatdist_garlic               6.0.1     The configuration tool gnatdist for GARLIC
gnatdoc                       23.0.0    GNAT Documentation Generation Tool (as `gnatdoc4` binary)
gnatprove                     12.1.1    Automatic formal verification of SPARK code
libgnatdoc                    23.0.0    GNAT Documentation Generation Tool (as a library)
```

4.3 Installing AVR packages

```
user@system : alr get gnat_native
```

4.3.1 Build Hello

- Get Hello :

```
user@system : alr get hello

Clonage dans '/home/sr/.config/alire/indexes/community/repo'...
remote: Enumerating objects: 7784, done.
remote: Counting objects: 100% (94/94), done.
remote: Compressing objects: 100% (67/67), done.
remote: Total 7784 (delta 29), reused 72 (delta 16), pack-reused 7690
Réception d'objets: 100% (7784/7784), 1.33 Mio | 9.30 Mio/s, fait.
Résolution des deltas: 100% (4228/4228), fait.
ⓘ Deploying hello=1.0.2...
Clonage dans '/home/sr/opt/alire-test/alr-olkw.tmp'...
remote: Enumerating objects: 34, done.
remote: Counting objects: 100% (10/10), done.
remote: Compressing objects: 100% (10/10), done.
remote: Total 34 (delta 0), reused 10 (delta 0), pack-reused 24
Réception d'objets: 100% (34/34), 5.44 Kio | 5.44 Mio/s, fait.
Résolution des deltas: 100% (5/5), fait.
ⓘ Deploying libhello=1.0.1...
Clonage dans '/home/sr/opt/alire-test/hello_1.0.2_5715870b/alire/cache/dependencies/alr-lwuj.tmp'...
remote: Enumerating objects: 30, done.
remote: Counting objects: 100% (13/13), done.
remote: Compressing objects: 100% (13/13), done.
remote: Total 30 (delta 0), reused 12 (delta 0), pack-reused 17
Réception d'objets: 100% (30/30), 5.20 Kio | 5.20 Mio/s, fait.
Résolution des deltas: 100% (3/3), fait.

hello=1.0.2 successfully retrieved.
Dependencies were solved as follows:

   + libhello 1.0.1 (new)
```

Get and build hello :

```
user@system : alr get --build hello

ⓘ Deploying hello=1.0.2...
Clonage dans '/home/sr/opt/alire/alr-qwis.tmp'...
remote: Enumerating objects: 34, done.
remote: Counting objects: 100% (10/10), done.
remote: Compressing objects: 100% (10/10), done.
remote: Total 34 (delta 0), reused 10 (delta 0), pack-reused 24
Réception d'objets: 100% (34/34), 5.44 Kio | 5.44 Mio/s, fait.
Résolution des deltas: 100% (5/5), fait.
ⓘ Deploying libhello=1.0.1...
Clonage dans '/home/sr/opt/alire/hello_1.0.2_5715870b/alire/cache/dependencies/alr-pqja.tmp'...
remote: Enumerating objects: 30, done.
remote: Counting objects: 100% (13/13), done.
remote: Compressing objects: 100% (13/13), done.
remote: Total 30 (delta 0), reused 12 (delta 0), pack-reused 17
Réception d'objets: 100% (30/30), 5.20 Kio | 5.20 Mio/s, fait.
Résolution des deltas: 100% (3/3), fait.
ⓘ Building hello/hello.gpr...
Setup
```

```
   [mkdir]        object directory for project Libhello
   [mkdir]        library directory for project Libhello
   [mkdir]        object directory for project Hello
   [mkdir]        exec directory for project Hello
Compile
   [Ada]          hello.adb
   [Ada]          libhello_config.ads
   [Ada]          libhello.adb
Build Libraries
   [gprlib]       Libhello.lexch
   [archive]      libLibhello.a
   [index]        libLibhello.a
Bind
   [gprbind]      hello.bexch
   [Ada]          hello.ali
Link
   [link]         hello.adb
Build finished successfully in 0.23 seconds.

hello=1.0.2 successfully retrieved and built.
Dependencies were solved as follows:

   + libhello 1.0.1 (new)
```

5 Cross Compiler

Installation is done with ALIRE manager.

5.1 Installing Gprbuild

```
user@system : alr get gprbuild
```

5.2 List all AVR packages

```
user@system : alr search --full --external-detect avr (all versions)

user@system : alr search --external-detect avr

avrada_examples          1.0.1    Sample applications in Ada for AVR microcontrollers
avrada_lib               2.0.2    Library of drivers for AVR microcontrollers
avrada_mcu               2.0.2    Device (MCU) specific definitions for AVR microcontrollers
avrada_rts               2.0.1    Minimal run time system (RTS) for AVR 8bit controllers
gnat_avr_elf             12.2.1   The GNAT Ada compiler - AVR cross-compiler
```

5.3 Installing AVR packages

```
user@system : alr get gnat_avr_elf

user@system : alr get avrada_rts

user@system : alr get avrada_mcu

user@system : alr get avrada_lib

user@system : alr get avrada_examples
```

5.4 Select toolchain

```
user@system : alr toolchain --select

  5. gnat_avr_elf=12.2.1
  1. gprbuild=22.0.1

user@system : alr toolchain

CRATE         VERSION  STATUS    NOTES
gprbuild      22.0.1   Default
gprbuild      2021.0.0 Available Detected at /home/sr/opt/gnat-2021a/bin/gprbuild
gnat_avr_elf  12.2.1   Default
gnat_external 2021.0.0 Available Detected at /home/sr/opt/gnat-2021a/bin/gnat
```

6 Documentation

Build the GCC documentation from scratch appears to be more difficult than build the compiler. Don't have time to investigate and suggest you pick the documentation here : https://gcc.gnu.org/onlinedocs

Choose the corresponding GCC version. In our context, it is the last one [beginning of 2023]: version 12.2

These manuals below have a total of nearly 1800 pages. GNAT is well documented:

- GCC 12.2 Manual.pdf
- GCC 12.2 GNAT User's Guide.pdf
- GCC 12.2 GNAT Reference Manual.pdf
- GNAT Coding Style Manual.pdf

https://this-page-intentionally-left-blank.org

AVR Ada environment

There are 10 types of people in the world: those who understand binary and those who don't.
Anonymous

1 Introduction

This chapter is mainly borrowed from Rolf Ebert's text available in https://github.com/AdaCore/Ada-SPARK-Crate-Of-The-Year/issues/19

1.1 History

From Rolf Ebert's own words, creator of AVR-Ada :

The development for AVR-Ada started in 2004 (or even before, I don't remember). I read somewhere around that time that the new AVR processors have many registers (32) to support high level languages. And then the article continued to use C. Having used Ada in the 1990ies I considered Ada a much more appropriate choice for a high level language.

I plunged into building gcc as a cross compiler and activating the Ada frontend. I learned a lot about GCC's internals and GNAT's run-time-system. I started to write scripts for building the compiler and the binutils (linker, assembler, and other tools), developed a minimalist run-time-system, and a support library for the on-chip or on-board peripherals. Providing all the different parts to other users was always the most tedious part.

AVR-Ada was once part of the popular WinAVR distribution in 2009, which was the most user friendly installation so far. It was limited to Windows only, but everything was included and it was nicely integrated in the development environment of that time AVR-Studio. Before and after that the installation was quite awkward.

AdaCore also distributed an AVR cross-compiler in 2011 and 2012. There were only pin and register definitions for a single target and no driver library at all. The compiler itself was probably more stable than the one from AVR-Ada.

Now, for the GNU/Linux sake in 2023, we have a full GCC AVR-Ada crate, including AVR-GDB debugger, which is maintained by Fabien Chouteau from AdaCore.

1.2 Purpose

AVR-Ada strives to provide a more or less complete compilation environment targeting the 8-bit AVR micro-controllers. It is not an integrated development environment like Emacs, GNAT-Studio, AVR-Studio or the newer VS Code but integrates well into each of them.

More specifically the project provides:

- A GNAT compiler based on the existing AVR and Ada support in GCC;
- A minimalist Ada runtime system;
- Register and pin definitions for over an hundred AVR micro-controllers;
- An extensive and useful AVR specific support library;
- Support packages for accessing LCDs, Dallas 1-wire sensors, or several humidity and temperature sensors;
- Some sample programs that show how to use the compiler and the library.

Some applications that had been built using AVR-Ada:

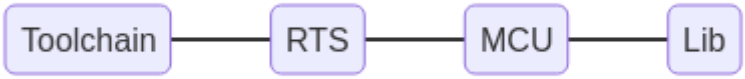
- A data logger for a weather station;
- A closed loop heating control system;
- An astronomical "GoTo" mount for a telescope on an AVR90USB128;
- A small robot based on the Asuro platform;
- A limited IP stack with ARP, ICMP and UDP [no TCP];
- Sample programs for the very popular Arduino platform.

2 Architecture

2.1 AVR-Ada split into modular crates

Originally AVR-Ada provided a one stop shop solution including everything in a single download. It was tedious to maintain and to explain how everything works together.

With the repackaging of AVR-Ada in Alire, Rolf Ebert took the chance to apply a modular approach. AVR-Ada is now split into five separate crates following the examples of other bigger packages [e.g. gnatcoll, Ada_Drivers_Library].



2.2 Introduction

The AVR-Ada environment is divided in six Alire crates:
- gprbuild, GNAT project builder;
- gnat_avr_elf, AVR-Ada cross-compiler;
- AVRAda_RTS, run time system;
- AVRAda_MCU, device definitions;
- AVRAda_Lib, library drivers;
- AVRAda_Examples, examples using the above crates.

Alire allows to describe their dependencies and versions. That alone turns maintenance a lot easier.

2.3 AVRAda_RTS

2.3.1 Description

Minimal Run Time System for AVR 8bit controllers. Supports local exceptions and tagged types.

https://github.com/RREE/AVRAda_RTS

The files were borrowed from subsets of GNAT's RTS and then carefully tailored to the AVR architecture;

It is also used to specify MCU type, clock frequency and stack size.

2.4 AVRAda_MCU

2.4.1 Description

Device (MCU[9]) specific definitions for AVR micro controllers.

https://github.com/RREE/AVRAda_MCU

This crate contains the bit and port definitions for over an hundred MCUs that Atmel (now Microchip Technology) released in the years around 2003 - 2015. If you use the more modern atxmega or attiny MCUs you currently (fall 2022) have to write you own.

This crate is mostly useful with the run-time-system AVRAda_RTS.
A library for most on-chip peripherals like timers, GPIOs, AD-converters, etc. is in

2.4.2 (Re-)creating the port and bit definitions

The port and bit definitions in this crate were for the most part automatically generated from Atmel's XML files that were provided at that time together with the development environment AVR-Studio

2.5 AVRAda_Lib

2.5.1 Description

Support library for on- and off-board peripherals (e.g. timers, interrupts, string handling).

The library strives to abstract away many of AVR's irregularities among the different MCUs. There is on one side drives for the on-chip capabilities, like GPIO, timers, A/D converters, UART, etc. On the other side the library provides some functionality that is typically in the Ada run time system or standard library, like real-time functions or string handling functions.

The main incentive to write special packages in the AVR hierarchy instead of the Ada hierarchy are the size constraints of the 8bit micro controllers. Some of them only have a few bytes of RAM. You then think twice if you want your string variables indexed by a 8bit wide or 16bit wide index.

[9] Micro Controller Unit

The library make heavy use of the GNAT preprocessor. All compilations have the symbol of the respective MCU name and the UART/Serial interface distinguishes between UART, USART, USART0, and USART1 for example. See the file mcu_capabilities. gpr.

3 Usage

Add avrada_rts in the dependency list and set some configuration values, i.e. the actual MCU in your project and the clock frequency.

The typical Arduino boards have an atmeg328p running at 16MHz. The secondary stack permits some fancy Ada constructs like returning unconstrained strings or class values. The generated AVR object code requires quite some space, however. The reserved stack space are 63 bytes per default. If you generate bigger objects you can increase the stack with the configuration value of Sec_Stack_Size :

```
[configuration.values]
avrada_rts.AVR_MCU = "atmega328p"
avrada_rts.Clock_Frequency = 16000000
avrada_rts.Sec_Stack_Size = 100

[[depends-on]]
avrada_rts = "*"
```

Then edit your project file to add the following elements:

“with" the run-time project file. With this, gprbuild will compile the run-time before your application

```
with "avrada_rts.gpr";
```

Specify the Target and Runtime attributes:

```
for Target use "avr";
for Runtime ("Ada") use AVRAda_RTS'Runtime ("Ada");
```

4 Hacking

4.1 Code Re-Use Across Different Micro-Processors

4.1.1 Generic Micro-controller Definitions

As there are so many different AVR micro-controllers you generally want to reuse your code in new projects with different controllers. It is therefor highly recommended *not* to refer to your specific controller in your Ada code. Use a generic renaming instead. That is, instead of:

```
with AVR.atmega328p;     use AVR.atmega328p;
```

You better use:

```
with AVR.MCU;            use AVR.MCU;
```

and define the actual processor at a single location. See Alire configuration variables below.

4.1.2 Preprocessor

That aforementioned technique works at an algorithmic level. But sometimes you have to dive into the low level details of registers and pin names. In order to keep the resulting code as re-usable as possible the AVR-Ada drivers make heavy use of the GNAT preprocessor. The AVR controllers tend to have different names on different controllers for actually the same thing. You will see a lot of code like this

```
   procedure Enable_External_Interrupt_0 is
   begin
#if MCU = "attiny85" then
      MCU.GIMSK_Bits (MCU.INT0_Bit) := True;
#elsif MCU = "atmega162" then
      MCU.GICR_Bits (MCU.INT0_Bit) := True;
#else
      MCU.EIMSK_Bits (MCU.INT0_Bit) := True;
#end if;
   end Enable_External_Interrupt_0;
```

The interrupt mask register is called `GIMSK` on an attiny85, `GICR` on the atmega162, and `EIMSK` on most other controllers.

If you want to add a new microprocessor or want to code different solutions for different controllers you can use the preprocessor variable `MCU` and the name of the controller.

> For the most part, the use of the preprocessor is limited to the internals of the support library.

The library makes heavy use of the GNAT preprocessor. All compilations have the symbol of the respective MCU name and the UART/Serial interface distinguishes between UART, USART, USART0, and USART1 for example. See the file `mcu_capabilities.gpr`. Typical user code rarely has to resort to the preprocessor.

4.1.3 Capabilities and Controller Names

In an attempt to better organize the driver sources and to shorten the preprocessor lines in the source code I started to describe the capabilities of the different microcontrollers. Right now there is only a few capabilities extracted to `mcu_capabilities.gpr` [EEPROM_Width, name of the EEPROM Write Enable bit, and the name of the UART subsystem].

For GNAT package manager file `avrada_lib.gpr` puts several source files into groups for handling the different drivers. there is for example the `UART_Sources` or `Timer0_Sources` for handling the UART or the internal timer0, respectively. Depending on the name of the micro-controller the source file groups are made available for the support lib. Take a look at the lower part of `avrada_lib.gpr`. If there is any interest and as time permits most of these options could be moved to the Alire configuration variables.

4.2 Alire configuration variables

Currently we mainly use two configuration variables. In the file `alire.toml` of your application you have to set the name of the micro-controller as a string and the speed of the processor as an integer. The typical Arduino boards have an atmeg328p running at 16MHz.

```
[configuration.values]
avrada_rts.AVR_MCU = "atmega328p"
avrada_rts.Clock_Frequency = 16000000
```

In your project you can refer to the name of the micro-controller in the preprocessor variable MCU. The example above for the busy waiting blinking LED must know about the processor speed to generate an appropriate loop of `NOP` statements. It refers to the clock frequency as `Avrada_Rts_Config.Clock_Frequency`:

```
procedure Wait_1_Sec is new
  AVR.Wait.Generic_Wait_USecs (Crystal_Hertz => Avrada_Rts_Config.Clock_Frequency,
                              Micro_Seconds => 1_000_000);
```

These are now compile time constants and permit to generate optimal code.

The secondary stack permits some fancy Ada constructs like returning unconstrained strings or class values. The generated AVR object code requires quite some space, however. The reserved stack space are 63 bytes per default. If you generate bigger objects you can increase the stack with the configuration value of `Sec_Stack_Size`:

```
[configuration.values]
avrada_rts.Sec_Stack_Size  =  100
```

There is a fourth configuration variable that is probably rarely used. If you want to use Timing Events you probably want them to be started at the timer tick. That can be achieved if you set `avrada_lib.Process_Timing_Events_In_Ticks` to True.

4.3 Use of a `Makefile`

Despite the rich possibilities of the GNAT project manager (i.e. `gpr`-files) and now Alire in `alire.toml` there are still quite some recurring functionalities missing. Since the first days we therefore provide a `Makefile` with appropriate targets.

If your program is called `sample` you can call the following commands:

- `make sample.prog`: that is probably the most important one, uploading the resulting binary to the target after generating appropriate ihex and eep files for flash and EEPROM.
- `make sample.size`: show the absolute and relative use of flash and eeprom memory.
- `make sample.s`: generate a commented AVR assembler file from `sample.adb`. That proves to be very helpful for debugging or when you suspect problems in the pin definitions or in the driver code.
- `make`: the standard make target simply calls `alr build`to rebuild your application.

Most tasks of the Makefile can be delegated to Alire. Some commands, however, require the name of the target processor. A simple AWK one-liner extracts that information from the configuration in `alire.toml`.

4.4 Adding a new microcontroller

At first, you must inspect and modify:
- AVRAda_Lib/mcu_capabilities.gpr
- AVRAda_Lib/avrada_lib.gpr

Then, in AVRAda_examples/basic_examples:
- Limit project file basic_examples.gpr scope to the simplest test program flash_led: for Main use ("flash_led");
- Adjust alire.toml with the new microcontroller id (avrada_rts_AVR.MCU line).

Then build basic_examples with the make command.

Fix each error creating new preprocessing condition with the new microcontroller id.

https://this-page-intentionally-left-blank.org

AVR Micro-controllers

Investment in C programs reliability will increase up to exceed the probable cost of errors or until someone insists on re-coding everything in Ada.
Gilb's laws synthesis

1 Introduction

1.1 Examples of common prototyping devices

Model	*ROM [Ko]*	*RAM*	*E²PROM*	*I/O*	*Mips*	*ARCH -mmcu=*	*Comments*
ATmega162	16	1024	512	35	16	avr5	old but okay for prototyping
ATmega168	16	1024	512	23	20	avr5	old but okay for prototyping
ATmega1280	128	8192	4096	86	16	avr51	seeduino card evaluation card
ATmega2560	256	8192	4096	86	16	avr6	

1.1.1 ATmega162

1 Watchdog
4 timers/counters with compare modes
2 USART
1 SPI

1.1.2 ATmega168

1 Watchdog
1 RTC
3 timers/counters with compare modes and capture and PWM
1 USART
1 8 channels 10 bit ADC
1 SPI

1.1.3 ATmega1280

1 Watchdog
1 RTC
6 timers/counters with compare modes and PWM
4 USART
1 bit oriented 2 wires serial interfaces
1 16 channels 10 bit ADC
1 SPI

1.1.4 ATmega2560

1 Watchdog
1 RTC
6 timers/counters with compare modes and PWM
4 USART
1 bit oriented 2 wires serial interfaces
1 16 channels 10 bit ADC
1 SPI

1.2 JTAG pins

Jtag function	***ATmega162***	***ATmega1280***			
TDI – Test Data Input	PC7	ADC7/PF7			
TDO – Test Data Out-put	PC6	ADC6/PF6			
TMS – Test Mode Se-lect	PC5	ADC5/PF5			
TCK – Test clocK	PC4	ADC4/PF4			

1.3 Quartz frequencies

Any between 1 MHz and 16 MHz for common models.

1.3.1 UART compatible

18.4320, 14.7456, 11.0592, 7.3728, 3.6864 and 1.8432 MHz

Basics on the bench

Weinberg's Second Law : If builders built buildings the way programmers wrote programs, then the first woodpecker that came along would destroy civilization.
Gerald Weinberg

1 Introduction

A sample test bed with a jtagice mkII and a seeduino test board :

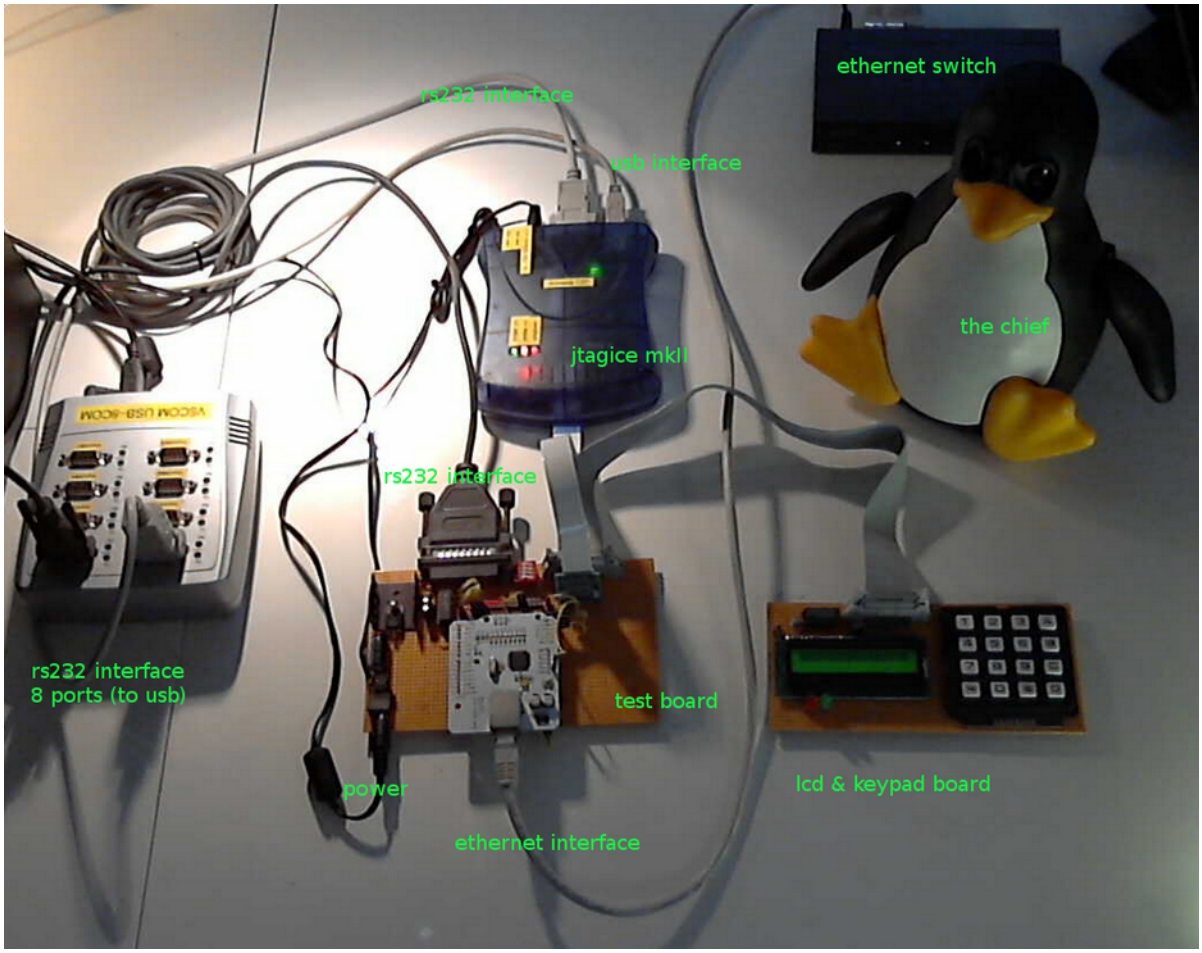


Fig. 1: A real world test-bed, supervised by the chief himself.

2 Basic example

Basic examples are located in :

```
$HOME/opt/alire/avrada_examples_1.0.1_72780c76/basic_examples
```

2.1 Build architecture

Basic examples are build through basic_examples.gpr:

```
with "avrada_rts.gpr";
with "avr_tool_options.gpr";
with "avrada_mcu.gpr";
with "avrada_lib.gpr";

project Basic_Examples is

   for Target use "avr";
   for Runtime ("Ada") use AVRAda_RTS'Project_Dir;

   for Source_Dirs use (".");
   for Object_Dir use "obj";
   for Create_Missing_Dirs use "True";

   for Exec_Dir use ".";
   for Main use ("extern_int_main", "flash_led", "voltmeter", "test_eeprom",
                 "hello_led", "local_exception", "pwm_demo", "walking_led_main",
                 "family_main");

   package Builder renames AVR_Tool_Options.Builder;

   package Compiler is
      for Default_Switches ("Ada") use AVR_Tool_Options.ALL_ADAFLAGS;
   end Compiler;

   package Binder is
      for Switches ("Ada") use AVR_Tool_Options.Binder_Switches;
   end Binder;

   package Linker is
      for Switches ("Ada") use AVR_Tool_Options.Linker_Switches;
   end Linker;

end Basic_Examples;
```

Default options settings are grouped together in avr_tools_options.gpr located in:

```
$HOME/opt/alire/avrada_rts_2.0.1_92a5f15a/avr_tools_options.gpr
```

avr_tool_options.gpr defaulted to build binaries in debug mode.

```
--  This is the default version of the target_options.gpr file,
--  imported by runtime_build.gpr. Variables defined here are appended
--  to the corresponding variables there. Target configurations may
--  their own version of this file.

with "config/avrada_rts_config.gpr";

abstract project AVR_Tool_Options is

   type Build_Type is ("Production", "Debug");
   Build : Build_Type := external ("AVR_RUNTIME_BUILD", "Debug");

   MCU := AVRAda_RTS_Config.AVR_MCU;

   --  COMFLAGS here is common flags (used for Assembler, C and Ada).
   COMFLAGS := ("-fcallgraph-info=su,da",
                "-ffunction-sections",  --  create separate function sections
```

```
               "-fdata-sections",      --  create separate data sections
               "-mmcu=" & MCU);        --  set the target device

   --  Per language flags (COMFLAGS will be added later)
   ADAFLAGS := ("-gnatef",           --  full path in error messages
                "-fverbose-asm",     --  include names of variables in asm code
                "-gnateDMCU=" & Mcu  --  name of the micro controler for preprocessor
                --  "-fno-delete-null-pointer-checks" --  This is completely disabled on AVR anyway
               );
   ASMFLAGS := ("");
   CFLAGS := ("-DIN_RTS",
              "-Dinhibit_libc",
              "-Wall");

   case Build is
      when "Production" =>
         --  Optimize for size
         COMFLAGS := COMFLAGS & ("-Os");
         ADAFLAGS := ADAFLAGS & ("-gnatp",     --  suppress run-time checks
                                 "-frename-registers",
                                 --  avoid false dependencies in scheduled code
                                 --  Attempt to avoid false dependencies in scheduled code by
                                 --  making use of registers left over after register
                                 --  allocation.  This optimization will most benefit
                                 --  processors with lots of registers. It can, however, make
                                 --  debugging impossible, since variables will no longer stay
                                 --  in a "home register".
                                 "-gnatVn",     --  no validity checks (smaller code)
                                 "-gnatn2");   --  full inlining across units
      when "Debug" =>
         --  Set optimization for debug and add specific debug symbols
         COMFLAGS := COMFLAGS & ("-Og",        --  optimize for debugging
                                 "-gdwarf-2"   --  generate debug symbols in dwarf-2 format
                                );
         ASMFLAGS := ASMFLAGS & ("-g");
   end case;

   --  Concatenate with common flags
   ALL_ADAFLAGS := ADAFLAGS & COMFLAGS;
   ALL_CFLAGS := CFLAGS & COMFLAGS;
   ALL_ASMFLAGS := ASMFLAGS & COMFLAGS;

   package Builder is
      for Global_Configuration_Pragmas use "gnat.adc";
      for Executable_Suffix use ".elf";
   end Builder;


   --  Provide options for binder and linker.  They are not needed for
   --  the RTS itself, but must be referenced in the application
   --  gpr-files

   Binder_Switches := (--  "--RTS=" & Arch, -- binder must know about RTS
                       "-D" & AVRAda_RTS_Config.Sec_Stack_Size,  -- set secondary stack size
                       "-minimal"       -- smaller binder file
                      );

   Linker_Switches := (-- linking the MCU specific libgcc and lib<<mcu>>
                       "-mmcu=" & MCU,
                       "-Wl,--gc-sections", -- remove sections of unused code
                       "-g",                -- keep binder file (needed???)
                       "-Wl,--relax"        -- linker relaxation is a must on AVR
                      );

end AVR_Tool_Options;
```

This setup build a binary tagged : ELF 32-bit LSB executable, Atmel 8-bit, version 1 (SYSV), statically linked, with debug_info, not stripped

2.2 Build

2.2.1 Setting target

You need to update your target in:
$HOME/opt/alire/avrada_examples_1.0.1_72780c76/basic_examples/alire.toml

```
name = "avrada_examples_basic_examples"
description = "Ada for AVR microcontrollers, several examples to get started"
version = "1.0.1"
```

```
authors = ["Rolf Ebert"]
maintainers = ["Rolf Ebert <rolf.ebert.gcc@gmx.de>"]
maintainers-logins = ["RREE"]
licenses = "GPL-2.0-or-later WITH GCC-exception-3.1"
website = "https://sourceforge.net/projects/avr-ada/"
tags = ["avr", "embedded", "demo"]

project-files = ["basic_examples.gpr"]

[configuration.values]
avrada_rts.AVR_MCU = "atmega1280"
avrada_rts.Clock_Frequency = 16000000

[[depends-on]]
gnat_avr_elf = "^11 | ^12.2"
avrada_rts = "^2.0.1"
avrada_mcu = "^2.0.2"
avrada_lib = "^2.0.2"
```

2.2.2 Build for target

```
user@system : cd $HOME/opt/alire/avrada_examples_1.0.1_72780c76/basic_examples

user@system : alr run basic_examples

user@system : ls -l *.elf

-rwxrwxr-x 1 sr sr  52780 déc.  10 10:21 extern_int_main.elf
-rwxrwxr-x 1 sr sr  83928 déc.  10 10:21 family_main.elf
-rwxrwxr-x 1 sr sr  15080 déc.  10 17:53 flash_led.elf
-rwxrwxr-x 1 sr sr  14180 déc.  10 10:21 hello_led.elf
-rwxrwxr-x 1 sr sr  41228 déc.  10 10:21 local_exception.elf
-rwxrwxr-x 1 sr sr  24188 déc.  10 10:21 pwm_demo.elf
-rwxrwxr-x 1 sr sr  39576 déc.  10 10:21 test_eeprom.elf
-rwxrwxr-x 1 sr sr 145228 déc.  10 10:21 voltmeter.elf
-rwxrwxr-x 1 sr sr  23200 déc.  10 10:21 walking_led_main.elf
```

2.3 Program target and execute

Elf to hex conversion (15080 to 615 bytes shrinking):

```
user@system :   $HOME/opt/alire/gnat_avr_elf_12.2.1_8e59f77f/bin/avr-objcopy   -O   ihex   flash_led.elf
flash_led.hex
```

Program and execute:

```
user@system : avrdude -p m1280 -c jtagmkII -P usb -F -U flash_led.hex
```

The LED must blink.

3 Basic debug

3.1 AVaRICE probe interface

Open a terminal and execute:

```
user@system : avarice --part atmega1280 --jtag usb --mkII --reset-srst :4242

AVaRICE version 2.14svn20200906, Oct 16 2021 10:40:06

Defaulting JTAG bitrate to 250 kHz.

JTAG config starting.
Found a device: JTAGICEmkII
Serial number:  07:00:00:00:61:f8
Reported JTAG device ID: 0x9703
```

```
Configured for device ID: 0x9703 atmega1280 -- Matched with atmega1280
JTAG config complete.
Preparing the target device for On Chip Debugging.
Waiting for connection on port 4242.
```

Monitor this terminal to ensure that AVaRICE does not hang up.

Open an other terminal to start a front-end.

3.2 avr-gdb front-end

3.2.1 Initialize session

```
user@system : cd $HOME/opt/alire/avrada_examples_1.0.1_72780c76/basic_examples

user@system : avr-gdb --tui ./flash_led.elf
```

On the [gdb] terminal, connect to the AVaRICE server:

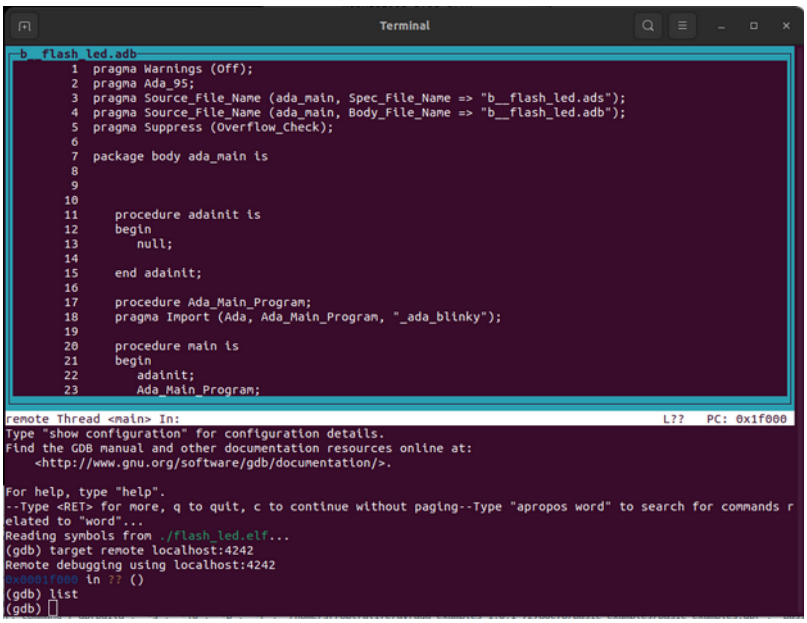
```
[return]
(gdb) target remote localhost:4242 [return]
```

On the AVaRICE terminal, a connection notification occurs:

```
Connection opened by host 127.0.0.1, port 49988.
```

3.2.2 Debug session

```
(gdb) b main (break main)
(gdb) c      (continue to the main break point)
(gdb) n      (next)
(gdb) p Some_Variable (print some variable)
(gdb) b 25   (break point on line 25)
```



3.3 DDD front-end

3.3.1 Introduction

DDD is a graphical debugger frontend. DDD stands for Data Display Debugger.

3.3.2 Installation

```
user@system : sudo apt install ddd
```

Insert at the very top of the file ./ddd/init

./ddd/init

```
Ddd*renderTable: rt
Ddd*rt*fontType: FONT_IS_XFT
Ddd*rt*fontName: Inconsolata
Ddd*rt*fontSize: 8
```

3.3.3 Initialize session

```
user@system : cd $HOME/opt/alire/avrada_examples_1.0.1_72780c76/basic_examples

user@system : ddd ./flash_led.elf --debugger $HOME/opt/alire/gnat_avr_elf_12.2.1_8e59f77f/bin/avr-gdb
```

3.3.4 Run the program

On the [gdb] terminal, connect to the AVaRICE server:

```
(gdb) target remote localhost:4242 [return]
```

On the AVaRICE terminal, a connection notification occurs:

```
Connection opened by host 127.0.0.1, port 49988.
```

3.3.5 Debug session

```
(gdb) b main (break main)
(gdb) c      (continue to the main break point)
(gdb) n      (next)
(gdb) p Some_Variable (print some variable)
(gdb) b 25   (break point on line 25)
```

Or use DDD features:

3.3.6 Useful commands

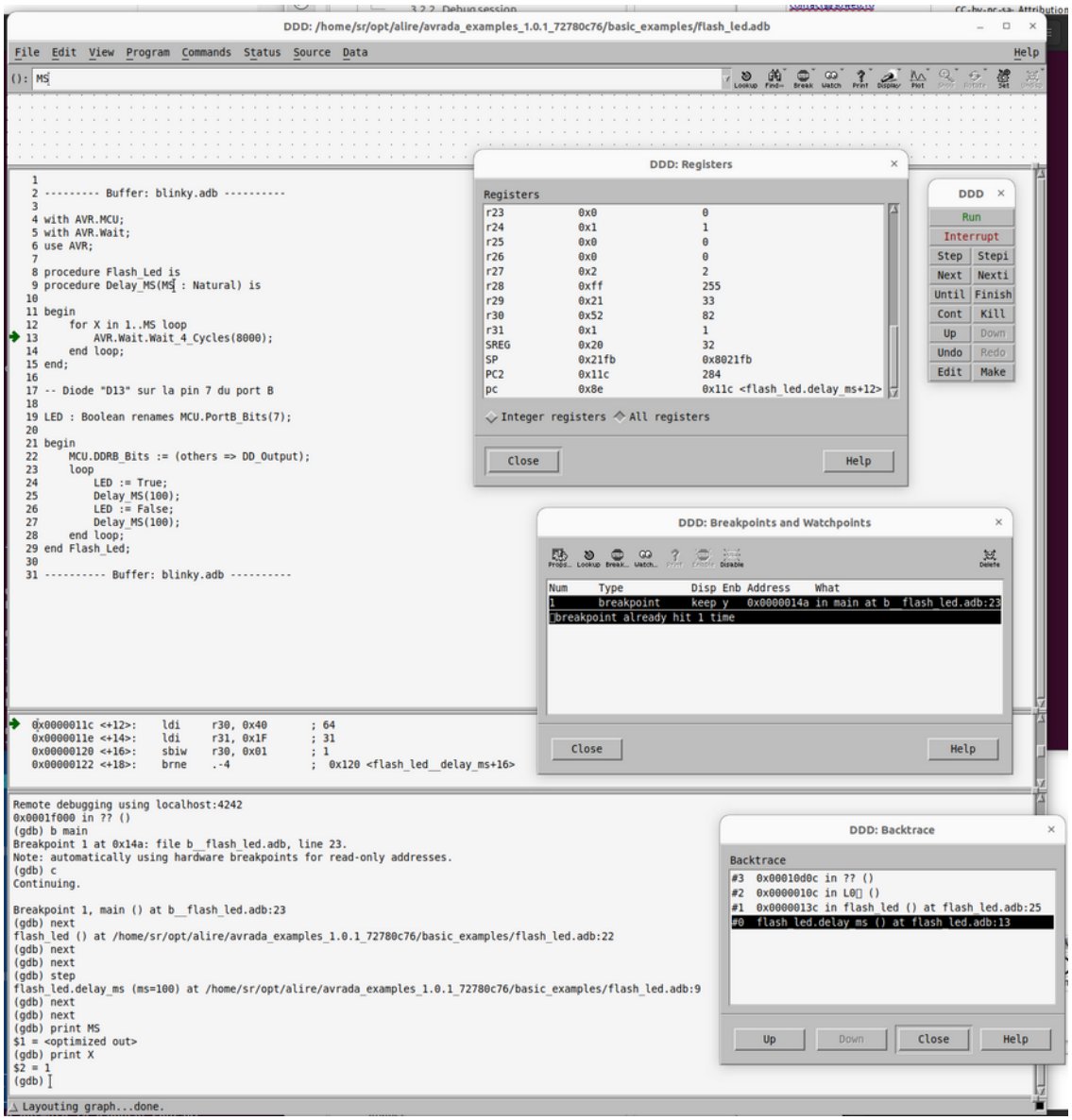
Beyond the basics of run, continue and next, don't forget some other handy commands.

Finish - execute until the current function returns, and break in caller. Useful if you accidentally go too deep, or if the rest of a function is of no interest.

Until - execute until reaching a later line. You can use this on the last line of a loop to run through the rest of the iterations, break out, and stop.

Start - create a temporary breakpoint on the first line of main[] and then run. Starts the program and breaks right away.

Step vs N**ext** - how to remember the difference? Think a flight of "steps" goes downward, "stepping down" into subroutines. Whereas "next" is the next contiguous source line.



3.3.7 Useful links

https://www.gnu.org/software/ddd

https://www.gnu.org/software/ddd/manual

https://begriffs.com/posts/2022-07-17-debugging-gdb-ddd.html#fixing-ddd-freeze-on-startup

https://stackoverflow.com/questions/61953224/jtag-debugging-on-avr-with-avr-gdb-and-ddd

https://jyh890312.gitbooks.io/ddd_manual_cn/content/the_ddd_windows/customizing_ddd.html

4 GNAT Studio full workflow

4.1 Project example flash_led

4.1.1 Directory content before build

```
config <dir>
alire.toml
flash_led.adb
flash_led.gpr
```

4.1.2 file alire.toml

alire.toml

```
name = "flash_led"
description = "Basic example for AVR microcontrollers"
version = "0.0.1"

authors = ["Your_Name"]
maintainers = ["Your_Name <name@domain.tld>"]
maintainers-logins = ["YourLogin"]
licenses = "GPL-2.0-or-later WITH GCC-exception-3.1"
website = "https://www.domain.tld"
tags = ["avr", "embedded", "demo"]

project-files = ["flash_led.gpr"]

[configuration.values]
avrada_rts.AVR_MCU = "atmega1280"
avrada_rts.Clock_Frequency = 16000000

[[depends-on]]
gnat_avr_elf = "^11 | ^12.2"
avrada_rts = "^2.0.1"
avrada_mcu = "^2.0.2"
avrada_lib = "^2.0.2"

[[pins]]
avrada_lib = { path = "../../avrada_lib_2.0.2_be6627e4" }
avrada_mcu = { path = "../../avrada_mcu_2.0.2_f6ef583f" }
```

4.1.3 file flash_led.adb

flash_led.adb

```
with AVR.MCU;
with AVR.Wait;
use AVR;

procedure Flash_Led is
procedure Delay_MS(MS : Natural) is
```

```
begin
    for X in 1..MS loop
        AVR.Wait.Wait_4_Cycles(8000);
    end loop;
end;

-- On seeduino/avr1280, a green LED "D13" is connected to pin 7 of port B
LED : Boolean renames MCU.PortB_Bits(7);

begin
    MCU.DDRB_Bits := (others => DD_Output);
    loop
        LED := True;
        Delay_MS(100);
        LED := False;
        Delay_MS(100);
    end loop;
end Flash_Led;
```

4.1.4 file flash_led.gpr

```
flash_led.gpr

with "avrada_rts.gpr";
with "avr_tool_options.gpr";
with "avrada_mcu.gpr";
with "avrada_lib.gpr";

project Flash_Led is

   for Target use "avr";
   for Runtime ("Ada") use AVRAda_RTS'Project_Dir;

   for Source_Dirs use (".");

   for Object_Dir use "obj";
   for Create_Missing_Dirs use "True";

   for Exec_Dir use ".";

   for Main use ("flash_led");

   package Ide is
     for Program_Host use "localhost:4242";
     for Communication_Protocol use "remote";
   end Ide;

   package Builder renames AVR_Tool_Options.Builder;

   package Compiler is
      for Default_Switches ("Ada") use AVR_Tool_Options.ALL_ADAFLAGS;
   end Compiler;

   package Binder is
      for Switches ("Ada") use AVR_Tool_Options.Binder_Switches;
   end Binder;

   package Linker is
      for Switches ("Ada") use AVR_Tool_Options.Linker_Switches;
   end Linker;

end Flash_Led;
```

The specific definitions of an embedded development with Ada and AVR are in bold.

4.2 GNAT Studio IDE

At the root of your alire project, open a terminal and execute:

```
user@system : alr edit
```

This start GNATstudio with Alire build environment setup.

4.2.1 Clean all

GNAT Studio: Build > Clean > Clean all

4.2.2 Build all

GNAT Studio: Build > Project > Build all [F9]

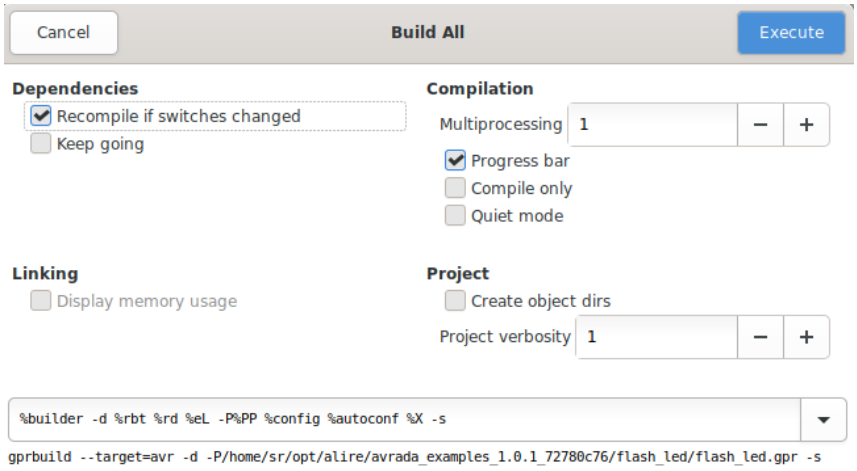
In the build window, check the build string:

```
%builder -d %rbt %rd %eL -P%PP %config %autoconf %X
```

Which is automatically translated by GNAT Studio as:

```
gprbuild --target=avr -d -P/home/sr/opt/alire/avrada_examples_1.0.1_72780c76/flash_led/flash_led.gpr
```

Press [Execute] to build:



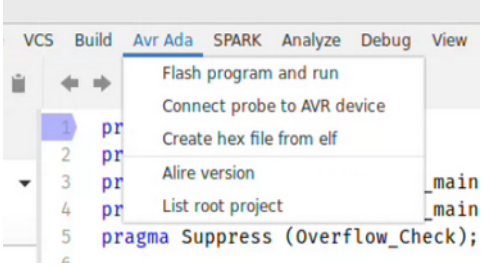
Build log:

```
gprbuild  --target=avr  -d  -P/home/sr/opt/alire/avrada_examples_1.0.1_72780c76/flash_led/flash_led.gpr  -
XAVR_RUNTIME_BUILD=Debug -XADAFLAGS=
Compile
   [Ada]          flash_led.adb
   [Ada]          avrada_lib_config.ads
   [Ada]          avr-strings-progmem.adb
   [Ada]          avr-serial.ads
   [Ada]          avr-adc.adb
   [Ada]          avr-wait.adb
   [Ada]          avr-serial_polled.ads
   [Ada]          avr-spi.adb
   [Ada]          avr-eeprom.adb
   [Ada]          avr-containers8.ads
   [Ada]          avr-programspace.adb
   [Ada]          avr-sleep.adb
   [Ada]          avr-i2c.adb
   [Ada]          avr-i2c-master.adb
   [Ada]          avr-strings-edit-generic_integers.adb
   [Ada]          avr-generic_text_io.adb
   [Ada]          avr-timer2.adb
   [Ada]          avr-strings-search.adb
   [Ada]          avr-uart_config.ads
   [Ada]          avr-int_img.adb
   [Ada]          avr-spi-master.adb
   [Ada]          avr-strings-edit-integers.ads
   [Ada]          avr-containers8-generic_bounded_priority_queues.adb
   [Ada]          avr-timer1.adb
   [Ada]          avr-ext_int.adb
```

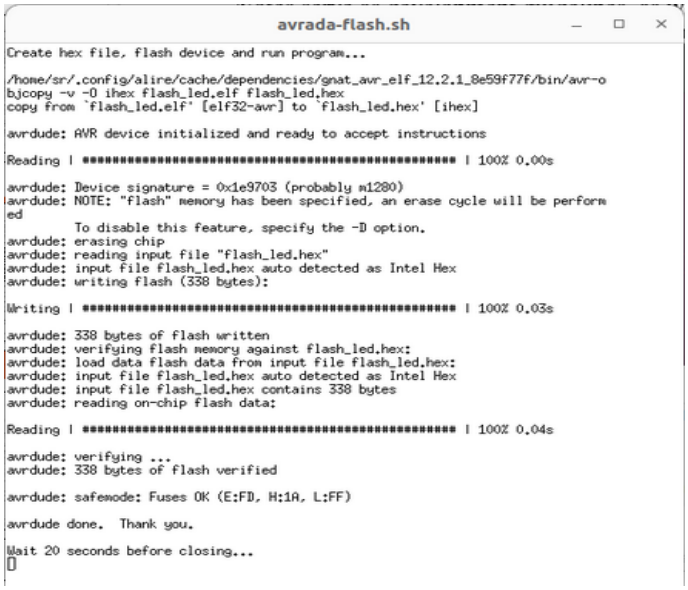
```
   [Ada]          avr-strings-edit.adb
   [Ada]          avr-uart_base_polled.adb
   [Ada]          avr-uart.adb
   [Ada]          avr-int_val.adb
   [Ada]          avr-watchdog.adb
   [Ada]          avr-spi-slave.adb
   [Ada]          avr-interrupts.adb
   [Ada]          avr-timer0.adb
   [Ada]          avr-strings.ads
   [Ada]          avr-atmega1280-bootloader.ads
   [Ada]          avr-atmega1280.ads
   [Ada]          avr-mcu.ads
   [Ada]          avr.adb
   [Asm_CPP]      ada_u32_img.S
   [Ada]          avrada_rts_config.ads
   [Ada]          a-tags.adb
   [Ada]          gnat.ads
   [Ada]          s-stoele.adb
   [Ada]          ada.ads
   [Ada]          a-caldel.ads
   [Ada]          unchdeal.ads
   [Ada]          a-unccon.ads
   [Ada]          s-intimg.ads
   [Ada]          s-secsta.adb
   [Ada]          machcode.ads
   [Ada]          a-calend.ads
   [Ada]          s-stalib.ads
   [Ada]          system.ads
   [Ada]          interfac.ads
   [Ada]          s-bitops.adb
   [Ada]          s-memory.adb
   [Ada]          g-souinf.ads
   [Ada]          a-cgcaso.adb
   [Ada]          unchconv.ads
   [Ada]          s-maccod.ads
   [Ada]          a-except.adb
   [Ada]          a-chlat9.ads
   [Ada]          s-sssita.adb
   [Ada]          a-contai.ads
   [Ada]          i-c.ads
   [Ada]          a-cgarso.adb
   [Ada]          s-parame.ads
   [Ada]          a-chlat1.ads
   [Ada]          a-uncdea.ads
   [Ada]          a-charac.ads
   [Ada]          s-assert.adb
   [Ada]          s-gccbui.ads
   [Ada]          s-unstyp.ads
Build Libraries
   [gprlib]       gnat.lexch
   [gprlib]       avrada_mcu.lexch
   [archive]      libgnat.a
   [index]        libgnat.a
   [gprlib]       avrada.lexch
   [archive]      libavrada_mcu.a
   [index]        libavrada_mcu.a
   [archive]      libavrada.a
   [index]        libavrada.a
Bind
   [gprbind]      flash_led.bexch
   [Ada]          flash_led.ali
Link
   [link]         flash_led.adb
[2023-01-06 15:56:13] process terminated successfully, elapsed time: 01.72s
Output saved in /home/sr/opt/alire/avrada_examples_1.0.1_72780c76/flash_led/obj/messages.txt
```

4.2.3 Flash device and run program

Use extra menu Avr Ada:



Click on Flash program and run:

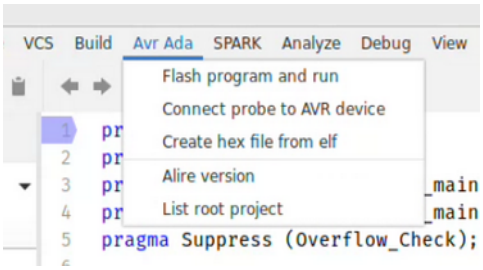


You must see the LED flash...

4.2.4 Debug session

- Setup the probe

Use extra menu Avr Ada:



Click on Connect probe to AVR device. A probee terminal is automatically opened:

Monitor this terminal to ensure that AVaRICE does not hang up.

- Setup Gnat Studio

  Then click on Debug > Initialize... > flash_led

  The probe interface terminal displays the connection:

```
Connection opened by host 127.0.0.1, port 38914.
```

- Open files

  Click on File > Open file > flash_led.adb

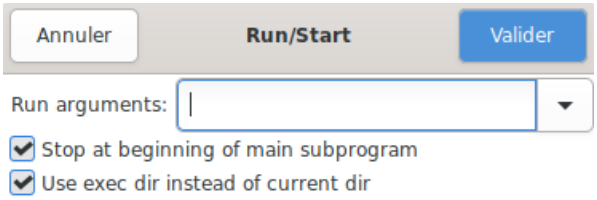
  Click on File > Open file > /obj/b__flash_led.adb

  The b__flash_led.adb file is the startup code automatically generated by the compiler.

b__flash_led.adb

```
pragma Warnings (Off);
pragma Ada_95;
pragma Source_File_Name (ada_main, Spec_File_Name => "b__flash_led.ads");
pragma Source_File_Name (ada_main, Body_File_Name => "b__flash_led.adb");
pragma Suppress (Overflow_Check);

package body ada_main is

   procedure adainit is
   begin
      null;

   end adainit;

   procedure Ada_Main_Program;
   pragma Import (Ada, Ada_Main_Program, "_ada_flash_led");

   procedure main is
   begin
      adainit;
      Ada_Main_Program;
   end;

end ada_main;
```

  When starting a debug session with GNAT Studio: Debug > Run... [Maj+F9]



  Continue/Run pressing [F8]

  As the box “Stop at the beginning...” is checked by default, this first breakpoint appears in b__flash_led.adb instead of flash_led.adb.

  In this context GNAT Studio does not work the same way as in native development:

```
19
20      procedure main is
21      begin
22         adainit;
23         Ada_Main_Program;
24      end;
25
26   end ada_main;
```

Then, you may use regular functions keys:

- Debug - Step

  [F5] Go to the immediate next executable line

- Debug - Step over

  [F6] Execute the program until the next source line stepping over subprograms calls

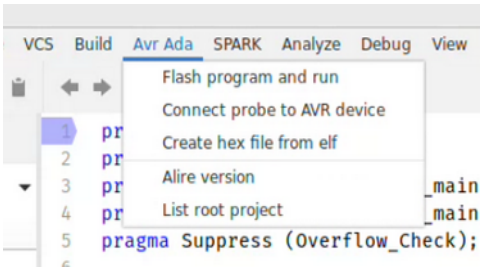
- Debug - Finish

  [F7] Continue execution until selected stack frame returns

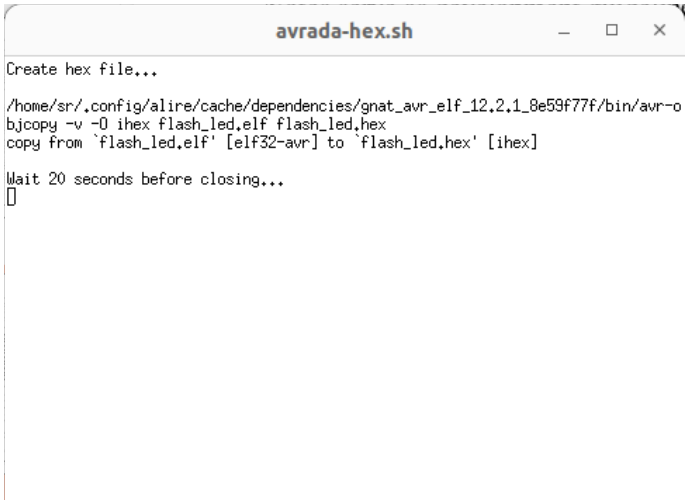
- Debug - Run

  [F8] Continue execution until next breakpoint

4.2.5 Other extra menu commands



Click on Create hex file from elf is self explanatory. A terminal is open for this task and automatically close after 20 s:



- Alire version

  Click on Alire version gives, in GNAT Studio message tab, some Alire information:

```
alr version
APPLICATION
alr version:               1.2.1
libalire version:          1.2.1
compilation date:          2022-08-28 10:41:38
compiler version:          Community 2021 (20210519-103)

CONFIGURATION
config folder:             /home/sr/opt/gnat-2021/etc/alire
force flag:                FALSE
non-interactive flag:      FALSE
community index branch:    stable-1.2.1
compatible index versions: ^1.1 & <=1.2.1
indexes folder:            /home/sr/opt/gnat-2021/etc/alire/indexes
indexes metadata:          OK
index #1:                  (community) git+https://github.com/alire-project/alire-index#stable-1.2.1
toolchain assistant:       enabled
tool #1 gnat:              not configured
tool #2 gprbuild:          not configured

WORKSPACE
root status:               VALID
root release:              flash_led=0.0.1
root load error:           none
root folder:               /home/sr/opt/alire/avrada_examples_1.0.1_72780c76/flash_led
current folder:            /home/sr/opt/alire/avrada_examples_1.0.1_72780c76/flash_led

SYSTEM
distribution:              UBUNTU
host-arch:                 X86_64
os:                        LINUX
target:                    NATIVE
toolchain:                 USER
word-size:                 BITS_64
alr version
APPLICATION
alr version:               1.2.1
libalire version:          1.2.1
compilation date:          2022-08-28 10:41:38
compiler version:          Community 2021 (20210519-103)

CONFIGURATION
config folder:             /home/sr/opt/gnat-2021/etc/alire
force flag:                FALSE
non-interactive flag:      FALSE
community index branch:    stable-1.2.1
compatible index versions: ^1.1 & <=1.2.1
indexes folder:            /home/sr/opt/gnat-2021/etc/alire/indexes
indexes metadata:          OK
index #1:                  (community) git+https://github.com/alire-project/alire-index#stable-1.2.1
toolchain assistant:       enabled
tool #1 gnat:              not configured
tool #2 gprbuild:          not configured

WORKSPACE
root status:               VALID
root release:              flash_led=0.0.1
root load error:           none
root folder:               /home/sr/opt/alire/avrada_examples_1.0.1_72780c76/flash_led
current folder:            /home/sr/opt/alire/avrada_examples_1.0.1_72780c76/flash_led

SYSTEM
distribution:              UBUNTU
host-arch:                 X86_64
os:                        LINUX
target:                    NATIVE
toolchain:                 USER
word-size:                 BITS_64
```

  Note: Target is detected as native as our setup use a GNAT Studio embedded in a native environment.

- List root project

  Click on List root project lists, in GNAT Studio message tab, all files in the root project directory:

```
pwd
/home/sr/opt/alire/avrada_examples_1.0.1_72780c76/flash_led/
ls -lpX --time-style=long-iso
total 76
drwxrwxr-x 3 sr sr  4096 2023-01-08 18:31 alire/
drwxrwxr-x 2 sr sr  4096 2023-01-05 13:52 config/
-rw-rw-r-- 1 sr sr  4793 2022-12-09 16:47 Makefile
drwxrwxr-x 2 sr sr  4096 2023-01-08 18:05 obj/
drwxrwxr-x 2 sr sr  4096 2023-01-08 12:25 show/
-rw-rw-r-- 1 sr sr   447 2023-01-04 17:10 flash_led.adb
-rwxrwxr-x 1 sr sr 21484 2023-01-06 16:23 flash_led.elf
-rw-rw-r-- 1 sr sr   867 2023-01-06 17:30 flash_led.gpr
-rw-rw-r-- 1 sr sr   975 2023-01-08 18:34 flash_led.hex
-rwxrwxr-x 1 sr sr   149 2023-01-08 18:28 avrada-debug.sh
-rwxrwxr-x 1 sr sr   315 2023-01-08 18:28 avrada-flash.sh
-rwxrwxr-x 1 sr sr   236 2023-01-08 18:33 avrada-hex.sh
-rw-rw-r-- 1 sr sr   788 2023-01-07 14:48 alire.toml
```


AVR probes

One can write neatly in any language, including C. One can write badly in any language, including Ada. But Ada is the only language where it is more tedious to write badly than neatly.
Jean-Pierre Rosen

1 Discussion around real-time hardware debugging

For common users, real-time hardware debugging can be avoided, most of the time, with a serial link and/or a LCD display. However, once you work with real-time programs (in which step by step can not be performed) or your work on complex ones (like cyphers or protocols stacks), the practice of breakpoint debugging and step by step debugging become the rule. In theses latter cases simulator or traditional « poor man » debugging practices can't always help.

As the AVR micro-controllers come with all debugging circuitry inside the chip, there are no real emulators for the AVR family. The link between the chip debugging circuitry and the software debugger is achieved by a JTAG interface. JTAG is a standardized interface control in the electronics industry (synchronous serial interface, for example). It is impossible to debug step by step without a JTAG probe.

Basically, three hardware breakpoints are available.

Some low-cost JTAG probes are available for AVR. I've found one at the bottom of one of my drawers. But I can't make it at work with theses more recent ATmega AVR series (my Seedunino card comes with an ATmega1280 on-board). Some others JTAG probes should work with ATmega1280, but the price is only 30 % less than genuine Atmel probes.

In 2012, at the time I have to choose a real-time capable debugging probe, The AVR Studio IDE of the time was very heavy and mostly buggy, and the AVR up-to-date and affordable probe JTAGICE 3 had some problems too.

I've finally chose to buy a JTAGICE mkII[10], which is a field proof genuine AVR probe, ugly looking and expensive, with USB and (important to my point of view) serial links.

10 Still available in late 2017 as compatible stuff for the half of its price - worth 250 € in late 2011 (search « jtagice mkII amazon »).

Luckily, I avoid the hardware revision zero and got a hardware revision one, which supports the PDI and aWire interfaces (serial number starting with A09-0041 or B0... indicate a hardware rev one).

We'll see later in this chapter there are now tons of JTAGICE mkII Probe clones, at incredibly low prices (less than 35 EUR in 2012, more in 2023, *thanks* to covid).

2 JTAGICE mkII Probe (Genuine)

2.1 Introduction

Key features are[11] :
- Supports up to three hardware program breakpoints or one maskable data breakpoint ;
- Supports symbolic debug of complex data types including scope information ;
- Supports up to 128 software breakpoints ;
- Includes on-board 512kB SRAM for fast statement-level stepping ;
- Level converters support 1.8V to 5.5V target operation ;
- Uploads 256Kb code in ~30 seconds (XMEGA using JTAG interface) ;
- Full-speed USB 2.0 compliant (12 MB/s) and RS-232 host interfaces ;
- Externally or USB powered.

Probe page : www.atmel.com/tools/AVRJTAGICEMKII.aspx
Documentation : http://www.atmel.com/webdoc/jtagicemkii/index.html
Protocol : http://www.atmel.com/webdoc/protocoldocs

2.2 Powering

As state in the probe manual, the JTAGICE mkII probe is able to operate using an external power supply providing 9...15V DC or it can be powered directly from the USB bus (if the plug can deliver a minimum of 500mA, as a self powered hub). An internal switch will default select the power from the external power supply.

➪ However, if external power is not connected, or the external power supply drops below a usable level, the power will be taken from the USB (if connected).

This means that, counter-intuitively, when USB cable inserted, the probe lights up only when its power switch is set on external power.

Although any polarity will work, the preferred polarity of the DC jack is negative-centre due to the power switch grounding.

When powering up the JTAGICE mkII, the power LED should illuminate immediately. If the LED does not light up, check that an adequate power supply is being used.

➪ Always **switch off** the target application power before switching off the Atmel AVR JTAGICE mkII.

➪ Never leave a **powered-down** JTAGICE mkII connected to a powered application as current may leak from the application and result in damage to the emulator.

➪ If the target application uses the JTAG pins for general purpose I/O, the JTAGICE mkII can still be used to program the target via the JTAG pins (provided that the JTAG en-

[11] According to http://www.atmel.com/tools/AVRJTAGICEMKII.aspx

able fuse is set.) However, be sure to connect the RESET pin of the target device to the nSRST pin of the emulator. Without this connection, the target application cannot be prevented from running after programming. If the application drives the JTAG pins as outputs, there will be signal contention with the emulator, which may result in damage to the emulator and/or target.

2.3 Tips

2.3.1 Identification

To identify the JTAGICE mkII probe, just plug its USB cable :

```
user@system : dmesg

[1032846.594575] usb 3-2.2: new full-speed USB device number 44 using xhci_hcd
[1032846.716227] usb 3-2.2: New USB device found, idVendor=03eb, idProduct=2103, bcdDevice= 2.00
[1032846.716231] usb 3-2.2: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[1032846.716232] usb 3-2.2: Product: JTAGICE mkII
[1032846.716233] usb 3-2.2: Manufacturer: ATMEL
[1032846.716234] usb 3-2.2: SerialNumber: 0700000061F8
```

2.3.2 Speed

Using USB is much faster than using rs232. Especially in debug mode.

2.3.3 Wiring

⇨ Again, do not forget to wire the RST line of the JTAGICE mkII otherwise the programming is not done and the debugger does not work.

2.3.4 Software

One can program before the debugging session, via the --erase --program flags (which must now go together). However, this causes the loss of the bootloader.

2.4 Programming firmware

2.4.1 GNU/Linux

There is no mean to update firmware with GNU/Linux. You have to create a Windows 7 VM (2Go RAM and 30 Go HD min) and then follow the Windows programming firmware installation.

In a Windows 7 VM hosted on GNU/Linux, Studio 7 install well and then, when using it, behave well.

- Programming firmware from a VM

However, using a VM to update firmware through an USB cable come with some tricks you should aware of, thanks to Clawson, an user of the AvrFreaks well known forum, some mysteries are solved[12].

As he noticed, I also experiment the « Windows bong-bing noise » alerting to the fact that the USB device had disconnected. If he went to the « Devices-USB Devices

[12] http://www.avrfreaks.net/forum/tuthardusing-jtacicemkii-atmel-tools-virtualbox?name=PNphpBB2&file=viewtopic&t=126054

menus » of the Virtual Machine and re-ticked the JTAGICE to reconnect it he then just ended up with the firmware upgrader saying the upgrade had failed.

After a while he realized what's going on : the JTAGICEmkII can actually appear as TWO different USB devices. One called « Atmel JTAGICE mkII » and one called « Atmel AVRBLDR » which is clearly the bootloader in the JTAGICE mkII that is used to apply the firmware update.

So what he ended up doing was applied two USB filter rules to the VM. Using a filter means that if this device ever enumerates to the host (GNU/Linux in our case) it is immediately captured and connected to the virtual machine.

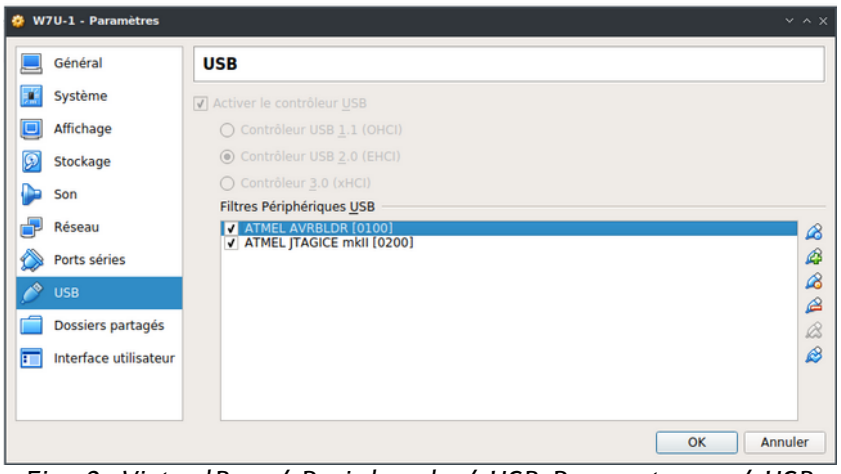


Fig. 2: VirtualBox / Peripherals / USB Parameters... / USB Peripherals Filters

Filters details :

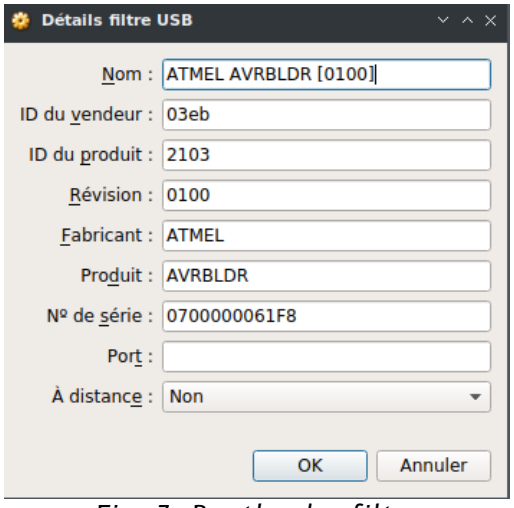


Fig. 3: Bootloader filter

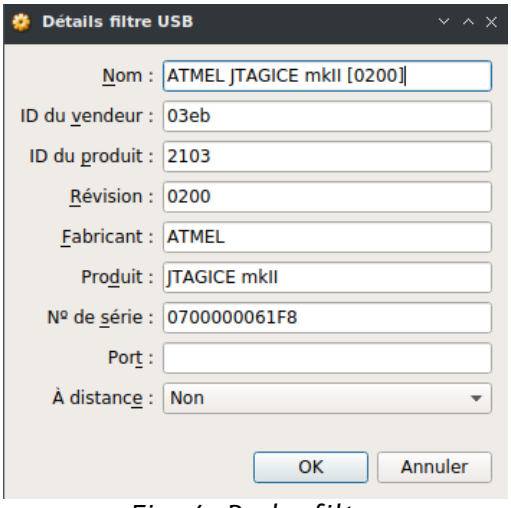


Fig. 4: Probe filter

2.4.2 Windows

An old Studio 4 under XP is ok to update the JTAGICE MkII firmware. In 2013, JTAGICE MkII firmware 6.06 (release included in Studio 4) was ok for Avrdude. But if you need the latest firmware, you need Studio 7 and Windows 7 at least.

Install AVR Studio 7 (as-installer-7.0.1645-full.exe, 887 MB !). It is an incredibly lengthy process. You need .NET 4 dotNetFx40_Full_x86_x64.exe and the KB Update Windows6.1-KB2999226-x64.msu. You have to reboot. And even after, the installation process is not finished ! As a result, your hard disk has « gained » 8231 files, 1238 folders and loss 1113 MB ! This piece of software is incredibly fat, pretty slow and complex.

In return, it offers a very complete environment, has many helpers, and could be used to get acquainted with the Atmel AVR family. The interface is neat and well organized.

Connect the JTAGICE mkII via USB and check the USB driver installation.

- Using Studio 4 (release 6.06)

  Launch AVR studio and go to Tools / Jtagice mkII upgrade...
  Tools / Extension and updates
  To update, click OK and the firmware is updated in less than 20 seconds. The update procedure does not check if the firmware is already up-to-date and always updates.

- Using Studio 7 (release 7.27)

  According to the manual, Studio 7 will automatically upgrade the tool's firmware when needed. A potential firmware upgrade is triggered once you start using a tool. Examples: the first time you launch a debug session or the first time you select the tool in the Device Programming dialog.

  The tool can not be used by Atmel Studio if the user chooses not to upgrade.

  You can also check for firmware used and upgrades availables by using View / Available Atmel Tools, the click right Upgrade... You get On Tool and On Disk releases, as well as the firmware status (up-to-date or need to update). If you decide to upgrade, you need to click on the Upgrade button, or quit the window with the Close button.

- Using Console

  To our taste, this way is a better one.

```
-- Tools

C:> C:\Program Files (x86)\Atmel\Studio\7.0\tools\JTAGICEmkII

-- Check the current version

C:> <Atmel>\7.0\atbackend\atfw.exe -t jtagicemkii -r

Found jtagicemkii:0700000061F8
Master MCU Version: 6.11
Slave MCU Version: 6.11

-- Updating (if the processs freeze, without quitting, a straight probe on/off could help)

C:> <Atmel>\7.0\atbackend\atfw.exe -a c:\Users\sr\jtagicemkii_fw.zip -t jtagicemkii -s 0700000061F8

Upgrading jtagicemkii:0700000061F8
Upgrading Main MCU: [==========]
Upgrading Slave MCU: [==========]
Successful upgrade

-- Check new version

C:> <Atmel>\7.0\atbackend\atfw.exe -t jtagicemkii -r

Found jtagicemkii:0700000061F8
```

```
Master MCU Version: 7.27
Slave MCU Version: 7.27
```

2.5 Using GDB

The GDB debugger is able to use the Atmel JTAGICE mkII probe.

Only connection through JTAG is supported.
Only three breakpoints are supported.

For some operations (such as the next command), GDB uses temporary breakpoints.

Do not forget to correctly program the fuses (be sure OCD and JTAG are enabled).

To use the probe, you simply have to select the « atmel-mkii » target :

```
$ gdb myprog
(gdb) target atmel-mkii
(gdb) load
(gdb) run
```

As shown above, GDB is able to erase and program the AVR with the load command.[13]

When configured for debugging the Atmel AVR, GDB supports the following AVR-specific commands : info io_registers. This command displays information about the AVR I/O registers. For each register, GDB prints its number and value.

2.6 Debugging the debugger

2.6.1 JTAG clock

According to the JTAGICE mkII manual, the target clock frequency must be accurately specified in the software front-end before starting a debug session. For synchronization reasons, the JTAG TCK signal must be less than one fourth of the target clock frequency for reliable debugging.

Setting the target clock frequency too high will cause failure of a debug session shortly after programming completes. This may be accompanied by several spurious SLEEP, WAKEUP, or IDR messages being displayed. When programming via the JTAG interface, the TCK frequency is limited by the maximum frequency rating of the target device, and not the actual clock frequency being used.

When using the internal RC oscillator, be aware that the frequency may vary from device to device and is affected by temperature and VCC changes. Be conservative when specifying the target clock frequency.

2.6.2 GCC related information

Single stepping GCC-generated code in source-level may not always be possible. Set optimization level to lowest for best results, and use the disassemble view when necessary.

[13] Excerps from GNAT User's Guide - Supplement for Cross Platforms - Copyright © 1995-2011, Free Software Foundation.

The JTAGICE mkII is optimized to work the way AVR Studio works, and that way is fairly different than the way GDB works. Thus, the JTAGICE mkII communication in AVaRICE causes much more traffic than it does in AVR Studio. This is most notable when single-stepping. As an alternative, you might consider using temporary breakpoints ["tbreak" or just "tb" in GDB].

Also, AVaRICE is sometimes not as efficient as it could be when communicating with the JTAGICE mkII. More manpower would be needed to clean things like that up [volunteers welcome !].

3 JTAGICE mkII Probe [clone]

IIRC Atmel AVRISP mkII was EOL'd due to EOL of its USB bridge IC. Some companies, like Waveshare, build some fine clones.

https://www.waveshare.com/wiki/USB_AVRISP_XPII
https://www.kanda.com/avr-programmer.html
https://www.avrfreaks.net/s/topic/a5C3l000000UXOtEAO/t142185

⇨ Check if internal quartz is 12 or 16 MHz. 16 MHz is mandatory when using RC calibration.

https://this-page-intentionally-left-blank.org

Lab tools

With the Wildebeest and the Penguin, there's no Bull.
Number Six

1 Introduction

We need some companions tools in our lab. Here is a poor man's list for that.

Buying first class refurbished equipment could be the way to own very valuable equipment for almost nothing. Dig the net for brokers.

2 Soldering station

Top class refurbished equipment.

There are Weller soldering irons and others.

Weller WS 50

3 Desoldering station

Top class refurbished equipment.

Rare equipment

Weller VP 801 EC

4 Power supply

Top class refurbished equipment. An old PC power supply can do it. Anything will do.

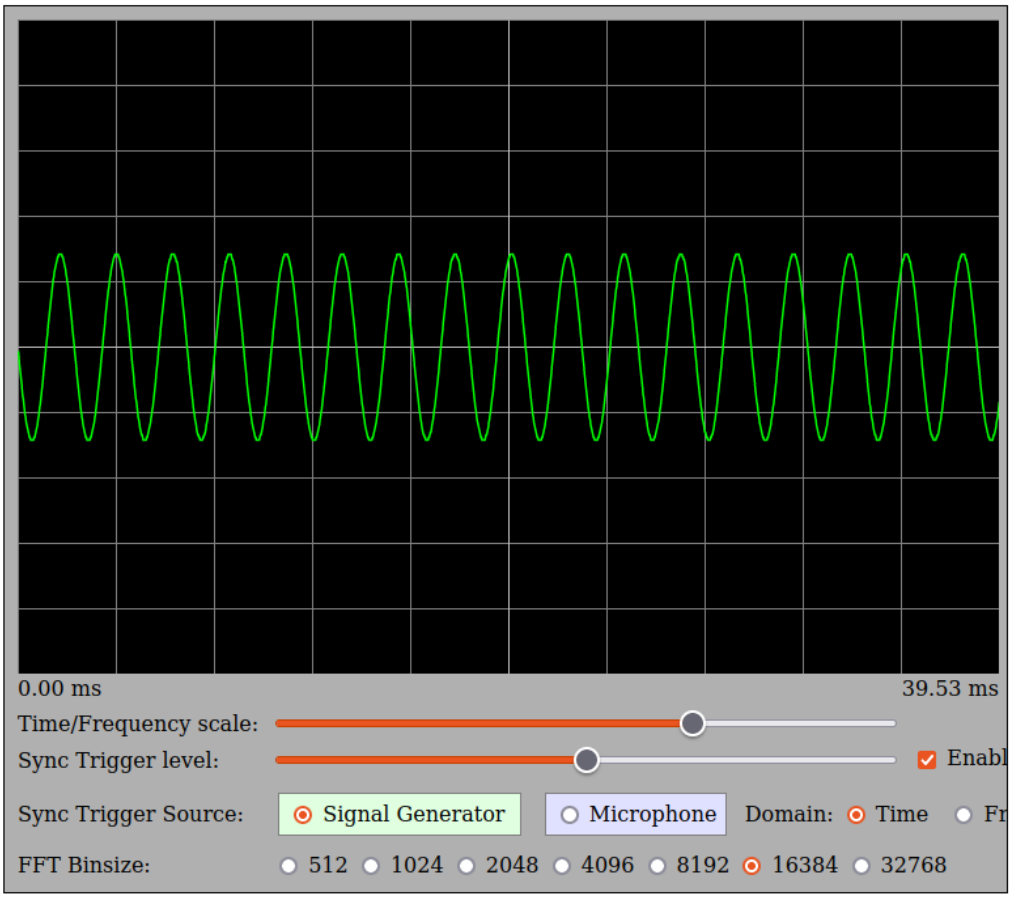
ALINCO DM-130MVZ [12V-30A]

5 Multi-meter

Basic multi-meter. Anything will do.

6 LF Generator

Introducing LF software generator by P. Lutus v1.7. Through software using sound card, there is no need to buy an equipment. You can save the page standalone, using it without an internet connection. The whole web site is amazing. [https://arachnoid.-com/SigGen/index.html](https://arachnoid.com/SigGen/index.html)



Activate **Mute output audio** | **Signal Source:** Signal Generator Microphone

Signal Generator:

Properties:

Carrier: Gain: Frequency: 440
Modulation: Gain: Frequency: 40
Noise: Gain: Enable

Carrier waveform:

Sine Square Triangle Sawtooth

Modulation:

AM FM

Modulation Waveform:

Sine Square Triangle Sawtooth

7 HF Generator

Top class refurbished equipment.

Marconi Instruments signal generator 2018A - 80 KHz to 520MHz

Very good rig. Well designed. Absolutely silent.

8 Frequency meter

Top class refurbished equipment.

Hewlett-Packard 5360A computing counter - 0 to 320 MHz

Bought for nothing, almost indestructible. First sell in 1969, at twice the price of a brand new ford pickup. Mine is 1974.

Always the state of art of frequency meter half a century later. It was HP's golden era, when HP's engineers were building the most advanced equipment, with the best materials, to the highest standards.

The noise level is reminiscent of an small aircraft.

The 12 *really flat* nixie tubes are gorgeous.

https://dopecc.net/calculators/hp/5360a

9 Oscilloscope & Logic analyzer Hantek HT6022BL

9.1 Introduction

Very basic but smart equipment. USB powered. Insanely low priced (~60€).

9.2 Links

9.2.1 Product

http://hantek.com/products/detail/153

9.2.2 OpenHantek oscilloscope software

http://openhantek.org
https://github.com/OpenHantek

9.2.3 Sigrok logic analyzer software

https://sigrok.org

9.3 Specifications

9.3.1 Oscilloscope

20MHz (-3 dB) Bandwidth; 48MSa/s Sample Rate; Gain 20mV~5V/div. DC couplig, +-5V max. Input protection 35V. Display mode Y-T & X-Y.. Timebase from 1ns to 5000s/div.

Measurements: Vp-p, Vmax, Vmin, Vmean, Vrms, Vamp, Vtop, Vbase, Vmid, positive overshoot, negative overshoot, cycle mean, cycle RMS, period, frequency, positive pulse width, negative pulse width, rise time (10%~90%), fall time (10%~90%), positive duty cycle, negative duty cycle.
Math : Addition, Subtraction, Multiplication, Division.
FFT : Rectangular, Hanning, Hamming, Blackman Window.

9.3.2 Logic analyzer

10MHz bandwidth. 16 (8 under GNU/Linux) Channels Logic Analyzer. 5,5V max, TTL, LVTTL & CMOS levels. Memory depth 1M per channel.

9.4 OpenHantek oscilloscope software

9.4.1 Installation

https://github.com/OpenHantek/OpenHantek6022/releases

```
user@system : gdebi openhantek_3.3.1-23-g42b9645_amd64.deb
```

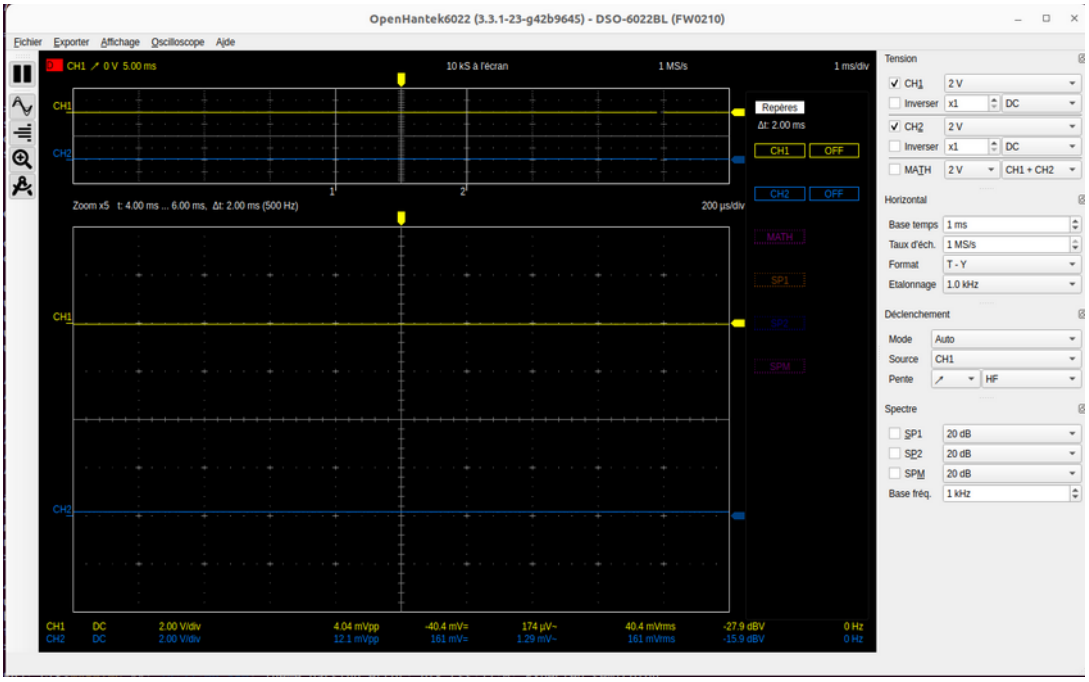
9.4.2 Using oscilloscope

Rear button selects oscilloscope or logic analyzer modes:
- Pressed: oscilloscope

After selecting a mode, you must off/on the equipment to register the new mode.

lsusb: ID 04b4:602a Cypress Semiconductor Corp.

```
user@system : OpenHantek
```



9.5 Sigrok logic analyzer software

9.5.1 Installation

```
user@system : sudo apt install sigrok
```

9.5.2 Using logic analyzer

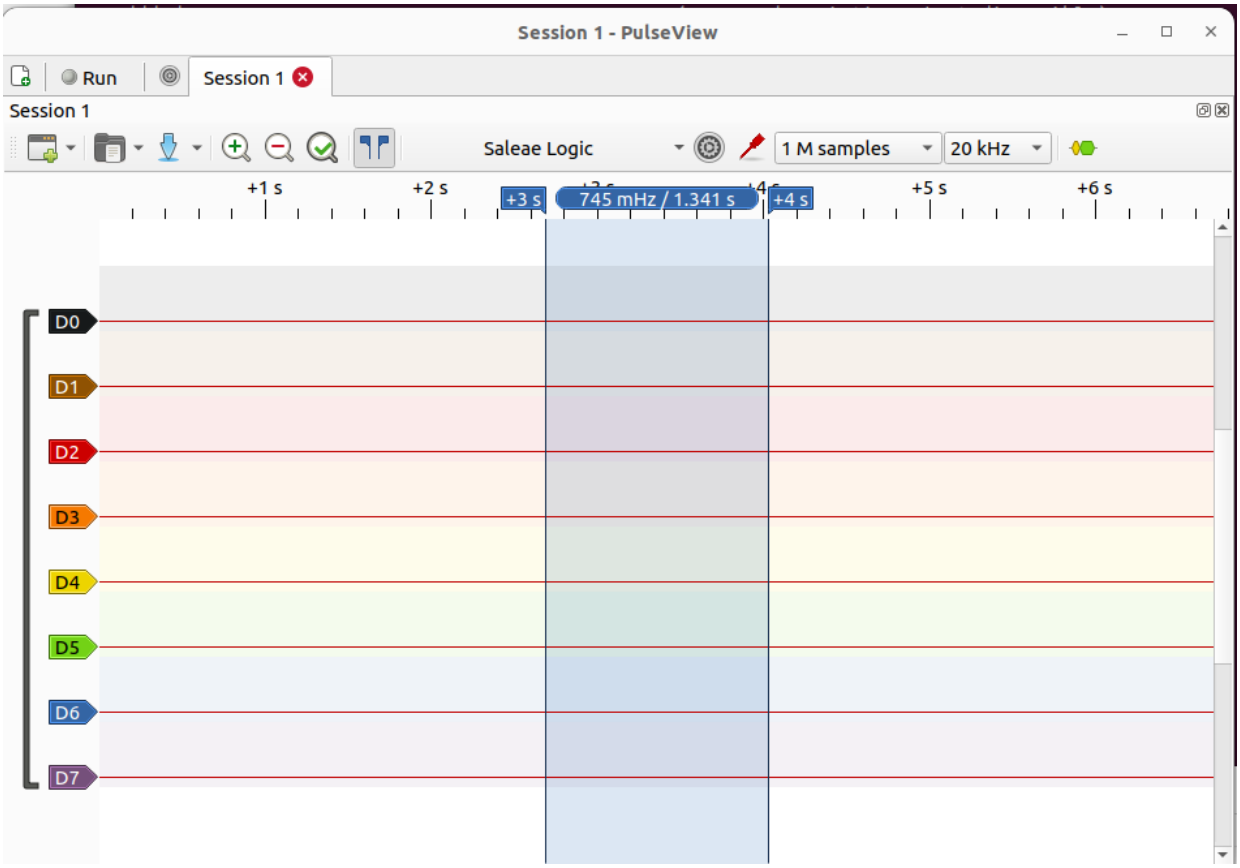
Rear button selects oscilloscope or logic analyzer modes.
- Depressed: logic analyzer

After selecting a mode, you must off/on the equipment to register the new mode.

lsusb: ID 0925:3881 Lakeview Research Saleae Logic

```
user@system : Pulseview
```

If no device detected, select Saleae logic



https://this-page-intentionally-left-blank.org

Choosing components

Weinberg's Second Law : If builders built buildings the way programmers wrote programs, then the first woodpecker that came along would destroy civilization.
Gerald Weinberg

1 Introduction

You will find below some some examples of common components for heating and solar energy applications. This chapter will be completed later with examples of real applications

2 Clock generators

2.1 Quartz

EuroQuartz Quartz crystal QUARZ HC49/US HC49/4H 14.7456 MHz 18 pF
UART baudrate friendly frequency: 14.7456 MHz.

https://www.conrad.com/p/euroquartz-quartz-crystal-quarz-hc49us-hc494h-147456-mhz-18-pf-l-x-w-x-h-368-x-1026-x-35-mm-1-pcs-155130

Type	QUARTZ HC49/US
Type	HC49/4H
Nominal frequency	14.7456 MHz
Frequency stability	-30 ppm, +30 ppm
Charging capacity	18 pF
Width	10.26 mm
Height	3.5 mm
Length	3.68 mm
Series	HC49/4H
Operating temperature [min.]	-10 °C
Max. operating temperature	+60 °C
Mounting type	Through-hole
Dim	[L x W x H] 3.68 x 10.26 x 3.5 mm
Content	1 pc[s]
Tolerance	± 30 ppm
Type	14,7456MHZ HC49/4H 30/50/40/18
Temperature range	-40 - +85 °C
Product type	Quartz crystal

3 Current sensor

3.1 SCT013

3.1.1 Introduction

https://en.yhdc.com/product/SCT013-401.html

Version 100A:50mA

https://domoticx.com/sct013-current-sensor

3.1.2 Application

https://learn.openenergymonitor.org/electricity-monitoring/ct-sensors/interface-with-arduino

4 Humidity sensor

4.1 DHT22 / AM2302

4.1.1 Introduction

http://www.aosong.com/en/products-22.html

https://www.kandrsmith.org/RJS/Misc/Hygrometers/calib_many.html

5 Temperature sensors

5.1 DS18B20

Available as stadard case PCB component or steel enclosure for industrial use.

5.1.1 Application

https://learn.openenergymonitor.org/electricity-monitoring/temperature/DS18B20-temperature-sensing

5.2 Thermocouple K3M

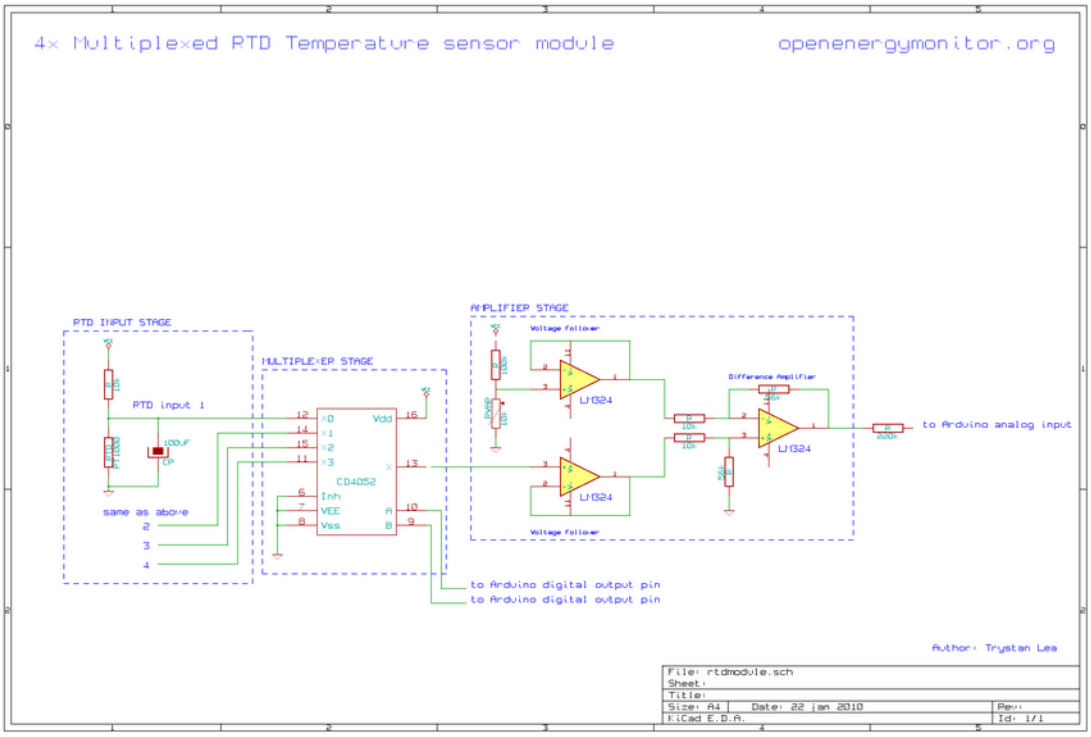
5.2.1 Introduction

https://instrumentationtools.com/how-to-calculate-thermocouple-temperature-by-measuring-the-output-millivoltage

ANSI LETTER	LEG*	METALLIC COMPOSITION	MELTING POINT		USABLE TEMPERATURE RANGE	TOLERANCES (THE GREATER OF BASE OR % OF READING)	
			° F	° C	(LONG TERM)	** STANDARD	PREMIUM
B	P N	PLATINUM +30% RHODIUM PLATINUM +6% RHODIUM	3320	1825	400 TO 3050 ° F 200 TO 1680 ° C	± 0.5%	NOT SET
C ***	P N	(TUNGSTEN +5% RHENIUM) (TUNGSTEN +26% RHENIUM)	4500	2480	30 TO 4200 ° F 0 TO 2300 ° C	NOT ESTABLISHED SEE IPTS-90	N.A.
E	P N	CHROMEL®, CONSTANTAN	2230	1220	-300 TO 850° F -200 TO +450° C	± 1.7° C o r± 0.5%	± 1.0 ° C or ± 0.4%
J	P N	IRON CONSTANTAN	2230	1220	30 TO 700° F 0 TO 400° C	± 2.2° C or ± 0.75%	± 1.1 ° C or 0.4%
K	P N	CHROMEL, ALUMEL,	2550	1400	-300 TO 1800° F -200 TO 1000° C	± 2.2° C or ± 0.75%	± 1.1° C or ± 0.4%
N	P N	NICROSIL**** NISIL	2440	1340	30 TO 1800° F 0 TO 1000° C	± 2.2° C or 0.75%	± 1.1° C or ± 0.4%
R	P N	PLATINUM +13% RHODIUM PURE PLATINUM	3215	1770	400 TO 2700° F 200 TO 1500° C	± 1.5° C or ± 0.25%	± 0.6° C or ± 0.1%
S	P N	PLATINUM +10% RHODIUM PURE PLATINUM	3215	1770	400 TO 2700° F 200 TO 1500° C	± 1.5° C or 0.25%	± 0.6° C or ± 0.1%
T	P N	COPPER CONSTANTAN,	1980	1080	-450 TO 660° F -270 TO 350° C	± 1.0° or 0.75%	± 0.5° C or ± 0.4%

5.3 Application

https://learn.openenergymonitor.org/electricity-monitoring/temperature/rtd-temperature-sensing



6 Pulse sensing

https://learn.openenergymonitor.org/electricity-monitoring/ct-sensors/interface-with-arduino

7 LCD displays

7.1 2 x 8 - types 0802 ou 0820

7.2 2 x 16 - types standard Hitachi

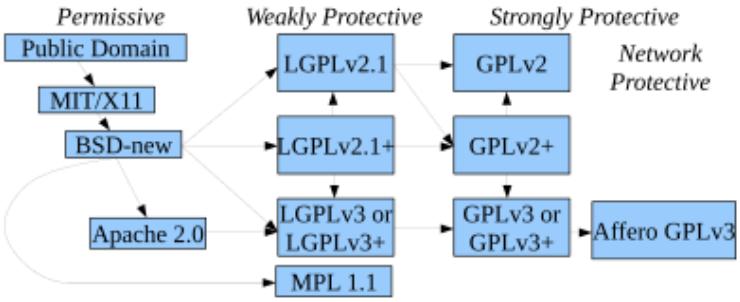
7.3 4 x 20 - types 2004A

Appendices

1 Copyrights & credits

1.1 Library Licence

v20 is copyright Sowebio under GPL v3 license.



1.1.1 GPL v3 compatibility with others licenses

https://en.wikipedia.org/wiki/License_compatibility: MIT licence is compatible with GPL and can be re-licensed as GPL. European Union Public Licence (EUPL) is *explicitly compatible* with GPL v2 v3, OSL v2.1 v 3, CPL v1, EPL v1, CeCILL v2 v2.1, MPL v2, LGPL v2.1 v3, LiLIQ R R+ AGPL v3.

1.2 Manual license

2 Links

2.1 Ada

https://en.wikibooks.org/wiki/Ada_Programming

2.2 Hardware

2.2.1 Seeduino

If you are considering buying an Arduino Mega, look at the Seeeduino Mega which owns more I/O (in fact, a lot more) and comes with other goodies.

Browse http://www.seeedstudio.com for details.

2.2.2 Ethernet

You may use board with the so-called Wiz5100. This chip comes with a full built-in IP/TCP/UDP layer inside ! Tero has wrote Ada code for it.

2.3 Compiling GNU/Linux toolchain from sources

https://sourceforge.net/p/avr-ada/wiki/Setup

http://arduino.ada-language.com/building-avr-gnat-for-avr-ada.html
http://arduino.ada-language.com/avr-ada-package-for-ubuntu-1404.html
https://bitbucket.org/tkoskine

2.4 Programming and debugging

- Toolchain compilation and use

  http://uracoli.nongnu.org/avrtools.html

- AVR Debugging on GNU/Linux (with debugWire)

  http://www.luniks.net/avr-debug.jsp

- Detailled JTAGICE mkII fuse setting, programming and debugging session :

  http://ardupilot.org/dev/docs/jtag.html

- Variables not updated in debug session (test it to validate debugging with AVR)

  https://groups.google.com/forum/#!topic/comp.lang.ada/Z3FPId6nKMY

2.5 Goodies

Fuse calculator :
http://www.engbedded.com/fusecalc

Useful tips & trics :
http://www.ladyada.net/resources/ucannoyances.html

2.6 Others

Avr Freaks : http://www.avrfreaks.net

Adacore Github : https://github.com/AdaCore
Adacore Papers : https://www.adacore.com/papers
Adacore Gems : https://www.adacore.com/gems
Adacore Books : https://www.adacore.com/books
Adacore GPL : https://www.adacore.com/community

http://arduino.ada-language.com/measuring-the-accuracy-of-delays-in-avr-ada.html

2.7 People

2.7.1 Rolf Ebert

The AVR-Ada project leader, since the beginning.

https://sourceforge.net/projects/avr-ada

2.7.2 Tero Koskinen

Tero is deeply involved in AVR-Ada development.

Home site : http://tkoskine.me
Ada related site : http://ada-language.com
Ada related site :http://stronglytyped.org
Sources repository : https://bitbucket.org/tkoskine

Ada, « it's stronger than you ».
Tribute to Daniel Feneuille, a legendary french Ada teacher (and much more)[14]

[14] http://d.feneuille.free.fr