RELEASE NOTES

GNU Toolchain for Atmel AVR8 Embedded Processors

Introduction

The Atmel AVR 8-bit GNU Toolchain (3.5.0.1662) supports all AVR 8-bit devices. The AVR 8-bit Toolchain is based on the free and open-source GCC compiler. The toolchain includes compiler, assembler, linker and binutils (GCC and Binutils), Standard C library (AVR-libc) and GNU Debugger (GDB).

Table of Contents

Introduction 1

1. Installation Instructions 3
   - 1.1. System requirements 3
     - 1.1.1. Hardware requirements 3
     - 1.1.2. Software Requirements 3
   - 1.2. Downloading, Installing and Upgrading 3
     - 1.2.1. Downloading/Installing on Windows 3
     - 1.2.2. Downloading/Installing on Linux 3
     - 1.2.3. Upgrading from previous versions 3
   - 1.3. Layout 3

2. Toolset Background 5
   - 2.1. Component Versions 5
   - 2.2. Compiler 5
   - 2.3. Assembler, Linker, Librarian and More 5
   - 2.4. C Library 6
   - 2.5. Debugging 6
   - 2.6. Source Code 6

3. Bugs and New Features 7
   - 3.1. New Features 7
   - 3.2. Notable Bugs Fixed 8
   - 3.3. Known Issues 9

4. Supported Devices 10

5. Contact Information and Disclaimer 12
   - 5.1. Disclaimer 12

1. Installation Instructions

1.1 System requirements

1.1.1 Hardware requirements

- Minimum processor Pentium 4, 1GHz
- Minimum 512 MB RAM
- Minimum 500 MB free disk space

AVR 8-bit GNU Toolchain has not been tested on computers with less resources, but may run satisfactorily depending on the number and size of the projects and the user's patience.

1.1.2 Software Requirements

- Windows 2000, Windows XP, Windows Vista, Windows 7 (x86 or x86-64) or Windows 8 (x86 or x86-64)
- AVR 8-bit GNU Toolchain is not supported on Windows 98, NT or ME.
- The toolchain should work on the Linux distributions Fedora, RedHat Enterprise, Arch Linux and Ubuntu for both 32-bits and 64-bits architecture. AVR 8-bit GNU Toolchain may very well work on other distributions. However those are untested and unsupported.

1.2 Downloading, Installing and Upgrading

The AVR8 GNU toolchain provided by Atmel is available for download and install in one of the following ways.

1.2.1 Downloading/Installing on Windows

- If you want to try the Atmel AVR8 GNU toolchain alone, you can download it from here[1]
- If you want to try the Atmel AVR8 GNU Toolchain along with Atmel studio, you can download and install Atmel studio 6.0 or (newer) which will also install the Atmel AVR8 GNU toolchain. See Atmel studio release notes for more details.

1.2.2 Downloading/Installing on Linux

For Linux, the Atmel AVR8 GNU Toolchain is available as a tar.gz archive which can be extracted using the tar utility. In order to install, simply extract to the location from where you want to run it from. Linux builds are available from here[2].

1.2.3 Upgrading from previous versions

If the Atmel AVR8 GNU Toolchain is installed by Atmel studio installation, refer Atmel Studio documentation to upgrade.

If the toolchain is installed separately using one of the (Windows, Linux, Mac) installers, upgrading is not supported. You can install the new package side-by-side of the old package and use it.

1.3 Layout

Listed below are some directories you might want to know about.

`<install_dir>` = The directory where you installed AVR 8-bit GNU Toolchain.

- <install_dir>\bin
  The AVR software development programs. This directory should be in your `PATH` environment variable. This includes:
  - GNU Binutils
  - GCC

[1] http://www.atmel.com/tools/ATMELAVRTOOLCHAINFORWINDOWS.aspx
[2] http://www.atmel.com/tools/ATMELAVRTOOLCHAINFORLINUX.aspx

    - GDB

- <install_dir>\avr\lib
  avr-libc libraries, startup files, linker scripts,and stuff.

- <install_dir>\avr\include
  avr-libc header files for AVR 8-bit.

- <install_dir>\avr\include\avr
  header files specific to the AVR 8-bit MCU. This is where, for example, #include <avr/io.h> comes from.

- <install_dir>\lib
  GCC libraries, other libraries, headers and stuff.

- <install_dir>\libexec
  GCC program components

- <install_dir>\doc
  Various documentation.

2. Toolset Background

AVR 8-bit GNU Toolchain is a collection of executable, open source software development tools for the Atmel AVR 8-bit series of microcontrollers. It includes the GNU GCC compiler for C and C++.

2.1 Component Versions

GCC: 4.9.2

binutils: 2.25

avr-libc: "1.8.0svn"

gdb: 7.8 7.8

2.2 Compiler

The compiler is the GNU Compiler Collection, or GCC. This compiler is incredibly flexible and can be hosted on many platforms, it can target many different processors/operating systems (back-ends), and can be configured for multiple different languages (front-ends).

The GCC included in AVR 8-bit GNU Toolchain is targeted for the AVR 8-bit microcontroller and is configured to compile C or C++.

CAUTION: There are caveats on using C++. See the avr-libc FAQ. C++ language is not fully supported and has some limitations. libstdc++ is unsupported.

Because this GCC is targeted for the AVR 8-bit MCUs, the main executable that is created is prefixed with the target name: `avr-gcc` (with '.exe' extension on MS Windows). It is also referred to as AVR GCC.

`avr-gcc` is just a "driver" program only. The compiler itself is called `cc1.exe` for C, or `cc1plus.exe` for C++. Also, the preprocessor `cpp.exe` will usually automatically be prepended with the target name: `avr-cpp`. The actual set of component programs called is usually derived from the suffix of each source code file being processed.

GCC compiles a high-level computer language into assembly, and that is all. It cannot work alone. GCC is coupled with another project, GNU Binutils, which provides the assembler, linker, librarian and more. Since 'gcc' is just a "driver" program, it can automatically call the assembler and linker directly to build the final program.

2.3 Assembler, Linker, Librarian and More

GNU Binutils is a collection of binary utilities. This also includes the assembler, as. Sometimes you will see it referenced as GNU as or gas. Binutils includes the linker, ld; the librarian or archiver, ar. There are many other programs included that provide various functionality.

Note that while the assembler uses the same mnemonics as proposed by Atmel, the "glue" (pseudo-ops, operators, expression syntax) is derived from the common assembler syntax used in Unix assemblers, so it is not directly compatible to Atmel assembler source files.

Binutils is configured for the AVR target and each of the programs is prefixed with the target name. So you have programs such as:

- *avr-as:* The Assembler.
- *avr-ld:* The Linker.
- *avr-ar:* Create, modify, and extract from archives (libraries).
- *avr-ranlib:* Generate index to archive (library) contents.
- *avr-objcopy:* Copy and translate object files.
- *avr-objdump:* Display information from object files including disassembly.
- *avr-size:* List section sizes and total size.
- *avr-nm:* List symbols from object files.
- *avr-strings:* List printable strings from files.
- *avr-strip:* Discard symbols.

- *avr-readelf:* Display the contents of ELF format files.
- *avr-addr2line:* Convert addresses to file and line.
- *avr-c++filt:* Filter to demangle encoded C++ symbols.
- *avr-gdb:* GDB, the GNU debugger, allows you to see what is going on `inside' another program targeted to AVR, while it executes.

See the binutils user manual for more information on what each program can do.

2.4 C Library

avr-libc is the Standard C Library for AVR 8-bit GCC. It contains many of the standard C routines, and many non-standard routines that are specific and useful for the AVR 8-bit MCUs.

In addition to avr-libc libraries, Host IO library (libhostio.a) is integrated to this toolchain. This Host IO library allows allows the target to use the host's file system and console I/O to perform various avr I/O operations.

NOTE: The actual library is currently split into two main parts, libc.a and libm.a, where the latter contains mathematical functions (everything mentioned in <math.h>, and a bit more). Also, there are additional libraries which allow a customization of the printf and scanf function families. avr-libc contains documentation on how to use (and build) the entire toolset, including code examples. The avr-libc user manual also contains the FAQ on using the toolset.

2.5 Debugging

Atmel Studio provides a debugger and also provides simulators for the parts that can be used for debugging as well. Note that `Atmel Studio` is currently free to the public, but it is not Open Source. The GNU debugger is now shipped along with the toolchain.

2.6 Source Code

Atmel AVR 8-bit GNU Toolchain uses modified source code from GCC, Binutils and AVR-LibC. The source code and the build scripts used for building the packaged binaries are available here[1].

Please refer to the README for the instructions on how to use the supplied script to build the toolchain.

[1] http://distribute.atmel.no/tools/opensource/Atmel-AVR-GNU-Toolchain/3.5.0

3. Bugs and New Features

3.1 New Features

Issue #AVRTC-714:

Optimize wdt_enable expression by avoiding un-necessary loads

Issue #AVRTC-726:

The compiler no more supports individual devices like ATmega8. Specifying, say, -mmcu=atmega8 triggers the usage of the device-specific spec file specs-atmega8 which is part of the installation and describes options for the sub-processes like compiler proper, assembler and linker. You can add support for a new device -mmcu=mydevice as follows:

# In an empty directory /someplace, create a new directory device-specs.

# Copy a device spec file from the installed device-specs folder, follow the comments in that file and then save it as /someplace/device-specs/specs-mydevice.

# Add -B /someplace -mmcu=mydevice to the compiler's command-line options. Notice that /someplace must specify an absolute path and that mydevice must not start with "avr".

# Provided you have a device-specific library libmydevice.a available, you can put it at /someplace, dito for a device-specific startup file crtmydevice.o.

The contents of the device spec files depend on the compiler's configuration, in particular on --with-avrlibc=no and whether or not it is configured for RTEMS.

Issue #AVRTCDEV-626:

Add python scripting enabled gdb (avr-gdb-py)

Issue #AVRTCDEV-653:

Add 32-bit PC relative relocation is added to support diff expressions for symbols from different sections. AVR assembler now can generate dwarf-debug-sections.

Issue #AVRTCDEV-704:

Move device specific functions from standard library to new device library

Issue #AVRTCDEV-719:

Upgrade gcc to version 4.9.2

Issue #AVRTCDEV-741:

Allow symbols in MEMORY region specification of Linker script

Issue #AVRTCDEV-743:

Add device memory details in a note section of device startup file

Issue #AVRTCDEV-744:

Rewritten wdt_enable/disable macros so that it doesn't require device name macro

Issue #AVRTCDEV-745:

Remove device specific macro in sleep.h, Instead use sleep mode conditions from device header file

Issue #AVRTCDEV-747:

Remove device specific macro in power.h, Instead use power reduction conditions from device header file.

Issue #AVRTCDEV-748:

Let io.h identify the device header file without needing to hard code the device name macro

Issue #AVRTCDEV-796:

Remove device specific macro definition in power.h. Instead use power reduction conditions that will be available from device header files.

Issue #AVRTCDEV-825:

Implemented fopen and fclose functions using avr-libc call backs to FILE IO

Issue #AVRTCDEV-826:

Implemented a serial protocol using which the IO operations information are sent via UART

Issue #AVRTCDEV-828:

Remove device specific information from standard library and headers

Issue #AVRTCDEV-847:

Add Host IO library that emulates IO operations in host machine.

(Use Host IO Server application (Atmel Studio extension) to handle the serial data sent by device for IO operations)

Issue #AVRTCDEV-850:

Make avr-libc backward compatible with gcc < 5.1.0

Issue #AVRTCDEV-861:

Let linker remove all debug sections for a function if that function is garbage collected

Issue #AVRTCDEV-866:

Device library (lib<device>.a) and startup object files (crt<device>.o) are installed in multilib location

Issue #AVRTCDEV-868:

Binutils upgraded to version 2.25

Issue #AVRTCDEV-888:

Added object file wise memory usage details to map file. This shall be enabled using '--detailed-mem-usage' linker option.

3.2 Notable Bugs Fixed

Issue #AVRTC-365:

RJMP and RCALL in avr-libc assembly functions are changed to XJMP/XCALL macros that will expand to jmp/call if the selected device has jmp instruction. When optimization/ relax enabled Linker can relax jmp instruction to rjmp if possible.

Issue #AVRTC-708:

Incorrect values for label diff expression in assembly code. Assembler now emits a DIFF reloc, which subsequently gets resolved to the correct value at link time.

Issue #AVRTC-735:

Incorrect constraint in wdt_enable/disable macros corrected

Issue #AVRTC-737:

wdt_enable/disable: Update constraints for inline assembler arguments. Mask wdt value to avoid overwritting un-intended bits.

Issue #AVRTC-738:

Avoid unintended reset when using wdt_disable

Issue #AVRTC-741:

Backport: Device specs changes in gcc-5.1

Issue #AVRTC-742:

Update library build option -mtiny-stack to -msp8

Issue #AVRTC-743:

Update Documentation for memory sections that require __attribute__ ((used))

Issue #AVRTC-746:

Fix incorrect register clobber when reading a __memx parameter.

Issue #AVRTC-747:

Backport: Alias entry to sqrt function

Issue #AVRTC-748:

PR 43011 (iom128rfa1.h): Removed SPI mode only bits from struct definition of __reg_UCSR1C and __reg_UCSR0C. Removed duplicate defines of UCPHA0, UDORD0, UCPHA1 and UDORD1.

Issue #AVRTC-751:

Remove reserved IO address defines for ATmega32U4 (iom32u4.h) (PR 45539)

Issue #AVRTC-752:

Define SLEEP_MODE_PWR_SAVE in iotn167.h (ATtiny167) for power-save sleep mode (PR 45551)

Issue #AVRTC-753:

Fix documentation typo in wdt.h

Issue #AVRTC-782:

Fix ICE when using attributs 'address' and 'io_low' (PR 65210)

Issue #AVRTC-757:

avr-gdb: Fix memory write failure for restore command

Issue #AVRTC-784:

Fix bit addressable instruction generation for invalid memory address

3.3 Known Issues

Issue #AVRTC-731:

For AVRTINY architecture, libgcc implementation has some known limitations.

Standard C / Math library implementation is very limited or not present.

Issue #AVRTC-732:

Program memory images beyond 128KBytes are supported by the toolchain, subject to the limitations mentioned in "3.17.4.1 EIND and Devices with more than 128 Ki Bytes of Flash" at http://gcc.gnu.org/onlinedocs/gcc/AVR-Options.html

Issue #AVRTC-733:

Named address spaces are supported by the toolchain, subject to the limitations mentioned in "6.16.1 AVR Named Address Spaces" at http://gcc.gnu.org/onlinedocs/gcc/Named-Address-Spaces.html#AVR%20Named%20Address%20Spaces

4. Supported Devices

avr2

at90s2313	at90s2343	at90s4414	at90s8515
at90s2323	attiny22	at90s4433	at90c8534
at90s2333	attiny26	at90s4434	at90s8535

avr25

ata5272	attiny4313	attiny85	attiny87
ata6616c	attiny44	attiny261	attiny48
attiny13	attiny44a	attiny261a	attiny88
attiny13a	attiny441	attiny461	attiny828
attiny2313	attiny84	attiny461a	attiny841
attiny2313a	attiny84a	attiny861	at86rf401
attiny24	attiny25	attiny861a	
attiny24a	attiny45	attiny43u	

avr3

at43usb355	at76c711

avr31

atmega103	at43usb320

avr35

ata5505	at90usb82	atmega16u2	attiny1634
ata6617c	at90usb162	atmega32u2	
ata664251	atmega8u2	attiny167	

avr4

ata6285	atmega48a	atmega88pa	at90pwm2b
ata6286	atmega48p	atmega88pb	at90pwm3
ata6289	atmega48pa	atmega8515	at90pwm3b
ata6612c	atmega48pb	atmega8535	at90pwm81
atmega8	atmega88	atmega8hva	
atmega8a	atmega88a	at90pwm1	
atmega48	atmega88p	at90pwm2	

avr5

ata5702m322	atmega168pa	atmega329	atmega649
ata5782	atmega168pb	atmega329a	atmega649a
ata5790	atmega169	atmega329p	atmega649p
ata5790n	atmega169a	atmega329pa	atmega6490
ata5791	atmega169p	atmega3290	atmega16hva
ata5795	atmega169pa	atmega3290a	atmega16hva2
ata5831	atmega16hvb	atmega3290p	atmega32hvb
ata6613c	atmega16hvbrevb	atmega3290pa	atmega6490a
ata6614q	atmega16m1	atmega32c1	atmega6490p
ata8210	atmega16u4	atmega32m1	atmega64c1
ata8510	atmega32a	atmega32u4	atmega64m1
atmega16	atmega32	atmega32u6	atmega64hve
atmega16a	atmega323	atmega406	atmega64hve2
atmega161	atmega324a	atmega64	atmega64rfr2
atmega162	atmega324p	atmega64a	atmega644rfr2
atmega163	atmega324pa	atmega640	atmega32hvbrevb
atmega164a	atmega325	atmega644	at90can32
atmega164p	atmega325a	atmega644a	at90can64
atmega164pa	atmega325p	atmega644p	at90pwm161
atmega165	atmega325pa	atmega644pa	at90pwm216
atmega165a	atmega3250	atmega645	at90pwm316
atmega165p	atmega3250a	atmega645a	at90scr100
atmega165pa	atmega3250p	atmega645p	at90usb646
atmega168	atmega3250pa	atmega6450	at90usb647
atmega168a	atmega328	atmega6450a	at94k
atmega168p	atmega328p	atmega6450p	m3000

avr51

atmega128	atmega1281	atmega128rfa1	at90can128
atmega128a	atmega1284	atmega128rfr2	at90usb1286
atmega1280	atmega1284p	atmega1284rfr2	at90usb1287

avr6

atmega2560	atmega2561	atmega256rfr2	atmega2564rfr2

avrxmega2

atxmega8e5	atxmega32a4	atxmega16a4u	atxmega32e5
atxmega16a4	atxmega32c3	atxmega16c4	
atxmega16d4	atxmega32d3	atxmega32a4u	
atxmega16e5	atxmega32d4	atxmega32c4	

avrxmega4

atxmega64a3	atxmega64a3u	atxmega64b1	atxmega64c3
atxmega64d3	atxmega64a4u	atxmega64b3	atxmega64d4

avrxmega5

atxmega64a1	atxmega64a1u

avrxmega6

atxmega128a3	atxmega128d3	atxmega192d3	atxmega256c3
atxmega128a3u	atxmega128d4	atxmega256a3	atxmega256d3
atxmega128b1	atxmega192a3	atxmega256a3b	atxmega384c3
atxmega128b3	atxmega192a3u	atxmega256a3bu	atxmega384d3
atxmega128c3	atxmega192c3	atxmega256a3u	

avrxmega7

atxmega128a1	atxmega128a1u	atxmega128a4u

avrtiny

attiny4	attiny9	attiny20
attiny5	attiny10	attiny40

avr1

at90s1200	attiny12	attiny28
attiny11	attiny15	

5. Contact Information and Disclaimer

For support on Atmel AVR 8-bit GNU Toolchain, visit design support[1].

Users of AVR 8-bit GNU Toolchain are also welcome to discuss on the AVRFreaks website forum for AVR Software Tools.

5.1 Disclaimer

AVR 8-bit GNU Toolchain is distributed free of charge for the purpose of developing applications for Atmel AVR processors. AVR 8-bit GNU Toolchain comes without any warranty.

[1] http://www.atmel.com/design-support/