AVR GCC

Avr gcc is one of the most popular compilers for compiling c code for avr microcontrollers. Arduino IDE and platfromIO both use avr-gcc as a compiler.

It is free open source compiler and can be installed on Linux by using the following command:

sudo apt-get install gcc-avr

It is also recommended to have avr-gcc alongside the gcc, which is a C library for atmel and avr microcontrollers to be used with avr-gcc.

It can be installed through the following command on ubuntu:

sudo apt-get install avr-libc

The typical command to run to compile a code that works for most is:

avr-gcc –mmcu=<microcontroller> Os <sourcefile.c> -o <outputfile>

Where <microcontroller> is the name or architecture of the used microcontroller, for example atmega2560 or avr5.

Os is for compiling optimization

<sourcefile.c> is the main or source code file that the main code lives in

<outputfile> is an optional argument. You can specify the name of the output file with this option, otherwise the default is a.out

https://www.nongnu.org/avrdude/

Avr-gcc flags and commands

-mmcu=mcu:

- Specify Architecture or MCU type. Default is avr2. Following devices are supported:
    - avr2: 8KiB memory devices:
        - attiny22, attiny26, at90s2313, at90s2323, at90s2333, at90s2343, at90s4414, at90s4433, at90s4434, at90c8534, at90s8515, at90s8535.
    - avr25: 8KiB memory and MOVW instruction devices:
        - attiny13, attiny13a, attiny24, attiny24a, attiny25, attiny261, attiny261a, attiny2313, attiny2313a, attiny43u, attiny44, attiny44a, attiny45, attiny48, attiny441, attiny461, attiny461a, attiny4313, attiny84, attiny84a, attiny85, attiny87, attiny88, attiny828, attiny841, attiny861, attiny861a, ata5272, ata6616c, at86rf401.
    - avr3: 16KiB up to 64KiB memory devices:
        - at76c711, at43usb355.
    - avr31: 128KiB memory devices:
        - atmega103, at43usb320.
    - avr35: 16KiB up to 64KiB memory with MOVW instruction devices:
        - attiny167, attiny1634, atmega8u2, atmega16u2, atmega32u2, ata5505, ata6617c, ata664251, at90usb82, at90usb162.
    - avr4: 8KiB memory “enhanced” devices:
        - atmega48, atmega48a, atmega48p, atmega48pa, atmega48pb, atmega8, atmega8a, atmega8hva, atmega88, atmega88a, atmega88p, atmega88pa, atmega88pb, atmega8515, atmega8535, ata6285, ata6286, ata6289, ata6612c, at90pwm1, at90pwm2, at90pwm2b, at90pwm3, at90pwm3b, at90pwm81
    - avr5: 16KiB up to 64KiB memory “enhanced” devices:
        - atmega16, atmega16a, atmega16hva, atmega16hva2, atmega16hvb, atmega16hvbrevb, atmega16m1, atmega16u4, atmega161, atmega162, atmega163, atmega164a, atmega164p, atmega164pa, atmega165, atmega165a, atmega165p, atmega165pa, atmega168, atmega168a, atmega168p, atmega168pa, atmega168pb, atmega169, atmega169a, atmega169p, atmega169pa, atmega32, atmega32a, atmega32c1, atmega32hvb, atmega32hvbrevb, atmega32m1, atmega32u4, atmega32u6, atmega323, atmega324a, atmega324p, atmega324pa, atmega324pb, atmega325, atmega325a, atmega325p,

https://www.nongnu.org/avrdude/

atmega325pa, atmega328, atmega328p, atmega328pb, atmega329, atmega329a, atmega329p, atmega329pa, atmega3250, atmega3250a, atmega3250p, atmega3250pa, atmega3290, atmega3290a, atmega3290p, atmega3290pa, atmega406, atmega64, atmega64a, atmega64c1, atmega64hve, atmega64hve2, atmega64m1, atmega64rfr2, atmega640, atmega644, atmega644a, atmega644p, atmega644pa, atmega644rfr2, atmega645, atmega645a, atmega645p, atmega649, atmega649a, atmega649p, atmega6450, atmega6450a, atmega6450p, atmega6490, atmega6490a, atmega6490p, ata5795, ata5790, ata5790n, ata5791, ata6613c, ata6614q, ata5782, ata5831, ata8210, ata8510, ata5702m322, at90pwm161, at90pwm216, at90pwm316, at90can32, at90can64, at90scr100, at90usb646, at90usb647, at94k, m3000.

- avr51: 128KiB memory “enhanced” devices:
  - atmega128, atmega128a, atmega128rfa1, atmega128rfr2, atmega1280, atmega1281, atmega1284, atmega1284p, atmega1284rfr2, at90can128, at90usb1286, at90usb1287.
- avr6: 3-byte pc, more than 128KiB memory devices
  - atmega256rfr2, atmega2560, atmega2561, atmega2564rfr2.
- avrxmega2: 8KiB up to 64KiB memory XMEGA devices
  - atxmega8e5, atxmega16a4, atxmega16a4u, atxmega16c4, atxmega16d4, atxmega16e5, atxmega32a4, atxmega32a4u, atxmega32c3, atxmega32c4, atxmega32d3, atxmega32d4, atxmega32e5, avr64da28, avr64da32, avr64da48, avr64da64, avr64db28, avr64db32, avr64db48, avr64db64, avr64dd14, avr64dd20, avr64dd28, avr64dd32, avr64ea28, avr64ea32, avr64ea48.
- avrxmega3: 64KiB memory & ram XMEGA devices
  - attiny202, attiny204, attiny212, attiny214, attiny402, attiny404, attiny406, attiny412, attiny414, attiny416, attiny417, attiny424, attiny426, attiny427, attiny804, attiny806, attiny807, attiny814, attiny816, attiny817, attiny824, attiny826, attiny827, attiny1604, attiny1606, attiny1607, attiny1614, attiny1616, attiny1617, attiny1624, attiny1626, attiny1627, attiny3214, attiny3216, attiny3217, attiny3224, attiny3226, attiny3227, atmega808, atmega809, atmega1608, atmega1609, atmega3208, atmega3209, atmega4808, atmega4809, avr16dd14, avr16dd20, avr16dd28,

avr16dd32, avr16ea28, avr16ea32, avr16ea48, avr16eb14, avr16eb20, avr16eb28, avr16eb32, avr32da28, avr32da32, avr32da48, avr32db28, avr32db32, avr32db48, avr32dd14, avr32dd20, avr32dd28, avr32dd32, avr32ea28, avr32ea32, avr32ea48.

- avrxmega4: 64KiB up to 128KiB memory XMEGA devices:
  - atxmega64a3, atxmega64a3u, atxmega64a4u, atxmega64b1, atxmega64b3, atxmega64c3, atxmega64d3, atxmega64d4, avr128da28, avr128da32, avr128da48, avr128da64, avr128db28, avr128db32, avr128db48, avr128db64.
- avrxmega5: 64KiB up to 128KiB memory and more than 64KiB ram XMEGA devices:
  - atxmega64a1, atxmega64a1u.
- avrxmega6: more than 128KiB memory XMEGA devices:
  - atxmega128a3, atxmega128a3u, atxmega128b1, atxmega128b3, atxmega128c3, atxmega128d3, atxmega128d4, atxmega192a3, atxmega192a3u, atxmega192c3, atxmega192d3, atxmega256a3, atxmega256a3b, atxmega256a3bu, atxmega256a3u, atxmega256c3, atxmega256d3, atxmega384c3, atxmega384d3.
- avrxmega7: more than 128KiB memory and more than 64KiB ram XMEGA devices:
  - atxmega128a1, atxmega128a1u, atxmega128a4u.
- avrtiny:
  - attiny4, attiny5, attiny9, attiny10, attiny102, attiny104, attiny20, attiny40.
- avr1:
  - attiny11, attiny12, attiny15, attiny28, at90s1200.

-mabsdata:

- This option has only an effect on reduced Tiny devices like ATtiny40
- Assumes that all data in static storage cna be accessed by LDS/STS instructions

-maccumulate-args:

- Accumulate outgoing function arguments and acquire/release the needed stack space for outgoing function arguments once in function prologue/epilogue
- Without this option, outgoing arguments are pushed before calling a function and popped afterwards

https://www.nongnu.org/avrdude/

- Popping arguments after the function call can be expensive on AVR so that accumulating the stack space might lead to smaller executables
- Can lead to reduced code size for functions that perform several calls to functions that get their arguments on the stack like calls to printf-like functions

-mbranch-cost=cost:

- Set branch costs for conditional branch instructions to cost.
- Values for cost are small, non-negative integers
- Default branch cost is 0

-mcall-prologues:

- Functions prologues/epilogues are expanded as calls to appropriate subroutines
- Cose size is smaller

-mdouble=bits, -mlong-double=bits:

- Set the size ofthe double or long double type.
- Possible values for bits are 32 and 64
- Depends on the –with-double= and –with-long-double= configure options

-mgas-isr-prologues=

- Interrupt Service Routine may use the __gcc_isr pseudo instruction
- Activated per default if optimization is on (but not with –Og)

-mint8:

- Assumes int to be 8-bit integer
- Affects the sizes of all types:
    - Char : 1 byte
    - Int : 1 byte
    - Long : 2 bytes
    - Long long : 4 bytes
- This option does not conform to the C standars but results in smaller code size

-mmain-is-OS_task:

- Do not save registers in main
- Effect is the same like attaching attribute OS_task to main.
- Activated per default if optimization is on

https://www.nongnu.org/avrdude/

-mno-interrupt:

- Generated code is not compatible with hardware interrupts
- Makes code smaller

-mrelax:

- Try to replace CALL resp. JMP instruction by the shorter RCALL resp. RJMP instruction if applicable
- Adds the –mlink-relax option to the assembler's command line and the –relax option to the linker's command line
- Must be turned on if linker stubs are needed

-mrodata-in-ram, -mno-rodata-in-ram:

- Locate the .rodata sections for read-only data in RAM resp. in program memory.
- For most devices, there is no choice and this option acts like an assertion
- For the AVR64* and AVR128* devices, .rodata is located in flash memory per default provided the required GNU binutils support is available.
- In that case, this can be used to return to the old layout wiht .rodata in RAM

-mstrict-X:

- Use address register x in a way proposed by the hardware.
- Means that x is only used in indirect, post-increment or pre-decrement addressing
- Without this option, the x register may be used in teh same way as Y or Z which then is emulated by additional instructions.

-mtiny-stack:

- Only change the lower 8 bits of the stack pointer

-mfract-convert-truncate:

- Allow to use truncation instead of rounding towards zero for fractional fixed-point types

-nodevicelib:

- Dont link against AVR-LibC's device specify library lib<mcu>.a.

-nodevicespecs:

- Dont add –specs=device-specs/specs-mcu to the compiler driver's command line

https://www.nongnu.org/avrdude/

- User takes responsibility for supplying the sub-processes like compiler proper, assembler, and linker with appropriate command line options
- User has to supply their private device specs file by means of –specs=path-to-specs-fil thus no need for option –mmcu=mcu

-Waddr-space-convert:

- Warn about conversions between address spaces in the case where the resulting address space is not contained in the incoming address space

-Wmisspelled-isr:

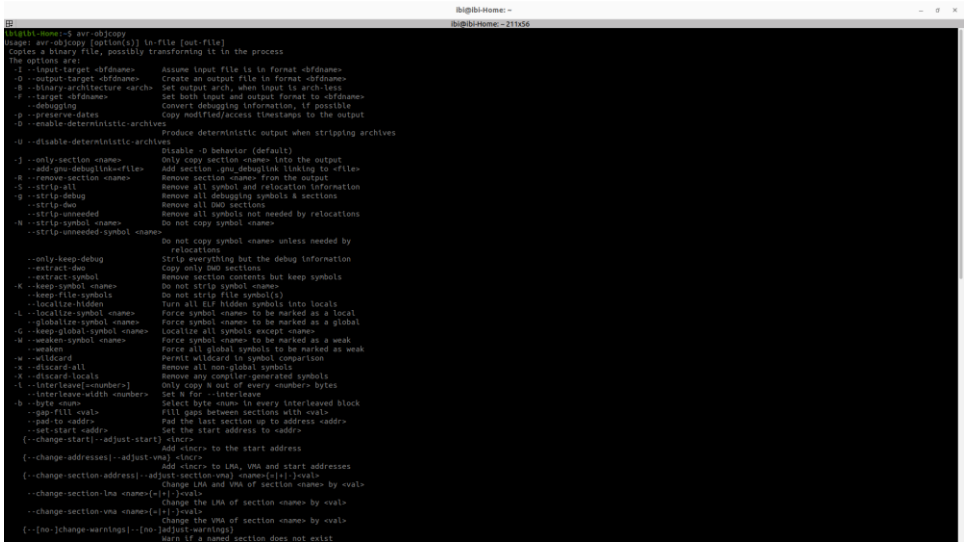
- Warn if the ISR is misspelled, i.e without __vector prefix.
- Enabled by default

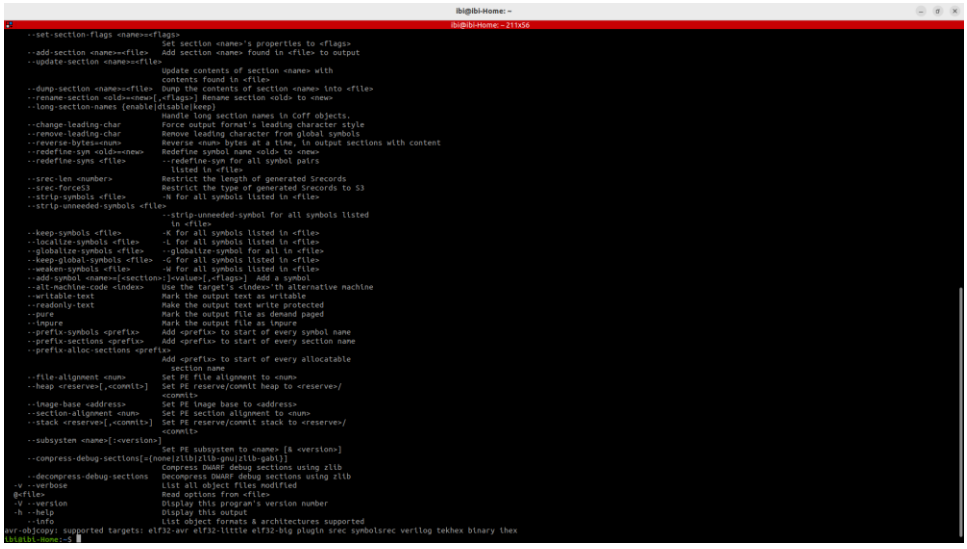
https://www.nongnu.org/avrdude/

AVR-OBJCOPY

A software that comes along automatically when installing avr-gcc. This software copies and translate object files, the a.out gotten by avr-gcc, into hex files that can be directly uploaded into the microcontroller.

Typing avr-objcopy in the command line will show the flags and commands that can be used alongside some explanation to what they do.





The typical command to run that would work most of the time:

avr-objcopy –O ihex –j .data -j .text a.out a.hext

Where: -O ihex is the file format, -j .data and –j.text so that only neccessary code is uploade, thus smaller code size, a.out name of object file, and a.hex name of hex file

https://www.nongnu.org/avrdude/

AVRDUDE

AVRDUDE commands

-p partno:

- mandatory option. Tells what type of MCU is connected.
- partno is the id listed in the configuration file.
- -p ? Lists all parts in the configuration file.
- If a part is not there, it needs to be added to the configuration file by entering the programming specifications found on the Atmel datasheet.

-b baudrate:

- Overrides the RS-232 connection baudrate specified in the respective programmers entry of the configuration file

-B bitlock:

- Specify the bit lock period for the JTAG interface (JTAG ICE only)
- The value is a floating-point number in microseconds
- The default value of the HTAG ICE results in about 1microsecond bit clock period. Suitable for targets running at 4MHz and above

-c programmer-id:

- Specify the programmer to be used
- -c ? Lists all programmer's id listed in the configuration file
- A new, unknown/undefined programmer can be added by copying an existing entry in the configuration file and changing the pin definitions to match that of the unknown programmer.

-C config-file:

- Use the specified config gile for configuration data for the programmers.
- If not specified, AVRDUDE reads the configuration file from /usr/local/etc/avrdude.conf (FreeBSD and Linux)

-D:

- Disable auto erase for flash, which is the default:

https://www.nongnu.org/avrdude/

    - When –U option with flash memory is specified, avrdude will perform a chip erase before starting any of the programming operations.
- To remain backward compatible, the –i and –m options automatically disable the auto erase feature.

-e:

- Causes a chip erase to be executed.
- Restes the contents of the flash ROM and EEPROM to the value 0xff.
- Technically a prerequisite command before the flash ROM can reprogrammed again. Only exception would be if the new contents would exclusively cause bits to be programmed from the value 1 to 0.
- Does not have to be called before programming as the MCU provides an auto-erase cycle before programming the cell

-E exitspec[,...]:

- By defaults AVRDUDE leaves the parallel port in the same state at exit as it has been found at startup. This option modifies the state of the /RESET and Vcc lines the parallel port is left at, according to the exitspec arguments provided:
    - reset: The /RESET signal will be left activated at program exit, will be held low, in order to keep the MCU in reset state afterward.
    - noreset: the /RESET line will be deactivated at program exit.
    - vcc: will leave those parallel port pins active, high, that can be used to supply Vcc power to the MCU
    - novcc: pull the Vcc pins of the parallel port down at program exit
- Multiple exitspec arguments can be seperated with commas

-F:

- Normally, AVRDUDE tries to verify that the device signature read from the part is reasonable before continuing
- This option is provided to override the check

-n:

- No-write – disables actually writing data to the MCU (useful for debugging AVRDUDE)

-P port:

- Use port to identify the device to which the programmer is attached

https://www.nongnu.org/avrdude/

- Noramlly the default parallel port is used.
- If you need to use a different parallel or serial port, use this option to specify the alternate port name.

-q:

- Disable (quell) output of the progress bar while reading or writing to the device.
- Specify it a second time for even quieter operation

-u:

- Disables the default behaviour of reading out the fuses three times before programming, then verifying at the end of programming that the fuses have not changed.
- If you want to change fuses, specify this option, as AVRDUDE will change the fuses back for you “safety”
- Designed to prevent cases of fuse bits magically changing (safemore)

-t:

- Enter the interactive “terminal” mode instead of up- or downloading files.
- See next chapter

-U memtype:op:filename[:format]:

- Perform a memory operation equivalent to specifing the –m, -i, or –o and –f options, except that multiple –U options can be specified in order to operate on multiple memories on
- The memtype fiels specifies the memory type to operate on.
- Use the –v option on the comman line or the part command from terminal mode to display all the memory types supported by a particular device.
- Typically, a device’s memoy configuration at least contains t
- Memory types currently known are:
    - calibration : one or mmore bytes of RC oscillator calubration data
    - eeprom : The EEPROM of the device
    - fuse : the fuse byte in devices that have only a single fuse byte
    - efus : the extended fuse byte
    - hfuse : the high fuse byte
    - lfuse : the low fuse byte
    - lock : the lock byte
    - signature : the three device signature bytes (device ID)

https://www.nongnu.org/avrdude/

    - Flash : typical for avr
- The op field specification:
    - r : read the specified device memory and write to the specified fil
    - w : read the specified file and write it to the specified device memory
    - v : read the specified device memory and the specified file and perform a verify operation
- The filename field indicates the name of the file to read or write.
- The format field is optional and contains the format of the file to read or write. Possible values are:
    - i : Intel
    - s : Motorola S-record
    - r : raw binary; little-endian byte order, in the case of the flash ROM data
    - m : immediate mode; actual byte values specified on the command line, seperated by commas or spaces in places of the filename field of the -i, -o, or –U options. Useful for programming fuse byes without having to create a single-byte file or enter terminal mode. If the number specified begins with 0x, it is treated as a hex value. If the number otherwise begins with a leading zero it is treated as octal. Otherwise the value is treated as decimal
    - a : auto detect; valid for input only, and only if the input is not provided at stdin.
    - The default is to use auto detection for input files, and raw binary format for output files.
    - If filename contains a colon, the format field is no longer optional since the filename part following the colon would otherwise be misinterpreted as format

-v:

- Enable ver

-V:

- Disable automatic verify check when upload

-y:

- Tells AVRDUDE to use the last four bytes of the connected parts' EEPROM memory to track the number of times the device has been erased

https://www.nongnu.org/avrdude/

- When this opotion is used and the –e flag is specified to generate a chip erase, the previous counter will be save before the chip erase, which will be incremented, and written back after the erase cycle completes.
- Typicall to track how many erase-rewrite cycles the part has undergone since the FLASH memory can only endure a finite number of erase-rewrite cycles.
- Typical limit for Atmel AVR FLASH is 1000 cycles.
- If the application needs the last four bytes of EEPROM memory, this option should not be used.
- No longer supported

-Y cycles:

- Instructs AVRDUDE to initialize the erase-rewrite cycle counter residing at the last four bytes of EEPROM memory to be the specified value.
- No longer supported

Terminal Mode Operation

Enabled by the –t option. Allows to enter interactive commands to display and modify the various device memories, perform a chip erase, display the device signature bytes and part paramteres, and to send raw programming commands.

https://www.nongnu.org/avrdude/

Commands and parameters may be abbreviated to their shortest unambiguous form.

Supports a command history so that previously entered commands can be recalled and edited.

Terminal Mode Commands

dump memtype addr nbytes:

- Read nbytes from the specified memory area, andn display them in hexadecimal and ASCII form

dump :

- Continue dumping the memory contents for another nbytes where the previous dump command left off

write memtype addr byte1 ... byteN

- Manually program the respective memory cells starting at address addr using the values byte1 throug byteN.
- Not implemented for bank-addressed memories such as the flash memory of ATMega devices.

erase :

- Perfomr a chip erase

send b1 b2 b3 b4:

- Send raw instruction codes to the AVR devices.
- If you need access to a feature of an AVR part that is not directly supported by AVRDUDE, this command allows you to use it.

sig :

- Display the device signature bytes.

part :

- Display the current part settings and parameters. Includes chip specific information including all memory types supported by the device, read/write timing, etc.

https://www.nongnu.org/avrdude/

? :

- Give a short on-line summarty of the availabe commands

quit :

- Leave the terminal mode and thus AVRDUDE

Other Commands for STK500 and JTAG ICE

vtarg voltage

- Set the targets supply voltage to voltage Volts.

varef voltage

- Set the adjustable voltage source to voltage Volts. This voltage is normally used to drive the tarftes Aref input on the STK500

fose freq [M/k] :

- Set the master oscillator to freq Hz. An optional trailing letter M multiplies by 1E6, a trailing letter k by 1E3

fosc o

- Turn the master oscillator off

sck period :

- STK500 only : Set the sck clock period to period microseconds
- JTAG ICE only : Set the JTAG ICE bit clock period to period microseconds. Will be reverted to its default value when the programming software signs off from the JTAG ICE.

parm :

- STK500 only; Display the current voltage and master oscillator parameters
- JTAG ICE only: DIsplay the current target supply voltage and JTAG bit clock rate/period

https://www.nongnu.org/avrdude/

Configuration File

AVRDUDE reads a configuration file upon startup. Any chip or programmer not supported by AVRDUDE can be added to the configration file.

AVRDUDE first looks for a system wide config file in "/usr/local/etc/avrdude.conf" on linux. The name of the file can be changed using the –C command line option.

After the system wide configuration file is parsed, AVRDUDE looks for a per-user configuration file to augment or override the system wide defaults.

The per-user file is .avrduderc within the user's home directory.

Programmer Definitions

The format of the programmer definition is as follows:

```
programmer
    id     = <id1> [, <id2> [, <id3>] ...] ;  # <idN> are quoted strings
    desc     = <description> ;                  # quoted string
    type     = par | stk500 ;                   # programmer type
    baudrate = <num> ;                          # baudrate for serial ports
    vcc      = <num1> [, <num2> ... ] ;         # pin number(s)
    reset    = <num> ;                          # pin number
    sck      = <num> ;                          # pin number
    mosi     = <num> ;                          # pin number
    miso     = <num> ;                          # pin number
    errled   = <num> ;                          # pin number
    rdyled   = <num> ;                          # pin number
    pgmled   = <num> ;                          # pin number
    vfyled   = <num> ;                          # pin number
  ;
```

Part Definitions

part

JTAG ICE: contains on-board logic to control the programming of the target device. Uses serial communication protocol. Allows both memory programming and on-chip debugging. The JTAG ICE mkII protocol can also be run on top of USB.

To access the full manual on Linux use command: man avrdude

https://www.nongnu.org/avrdude/

https://www.nongnu.org/avrdude/