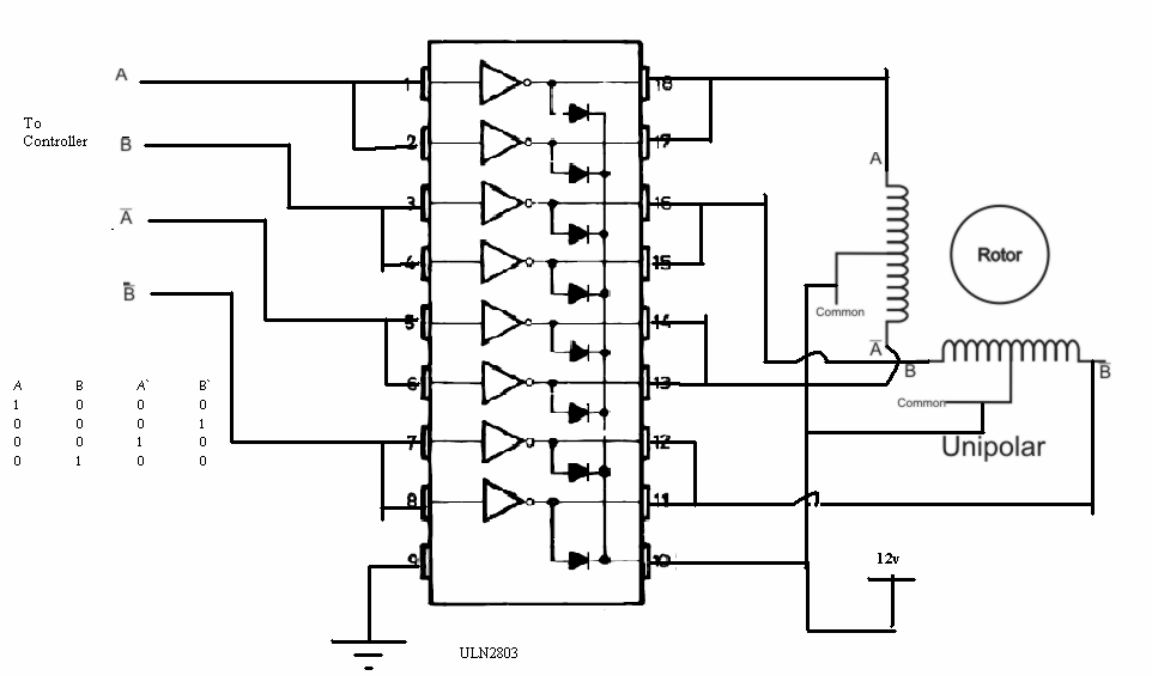
Stepper motor driving with ULN2803 and ATmega16/32:

Please read http://www.triindia.co.in/resources/?p=40 if you are handling the stepper motor for the first time.



Example describes stepper motor driver using ULN2803. In this example code two stepper motors are driven by a single timer. Code is written for ATmega16/32 running at 8MHz, for serial baudrate of 19200 and winavr platform.
Circuit description:
It is a simple unipolar drive. Circuit for a single stepper is shown same circuit is used for other stepper also. A B A' B' of first motor are connected to PORTB0-PORTB3. and other motor is connected to PORTD4-PORTD7.

Program description:
Include uart.h and makefile in your project folder and name your main program as main.c.
Stepper motors run by interrupt of Timer 0. Timer 0 is configured in CTC mode(refer datasheet) with pre-scalar of 1024. Every time interrupt is executed motors move by one step. Delay between the steps can be controlled by OCR0. To increase the speed of motor lower the value of OCR0 and do reverse to decrease the speed.

I use serial port of PC to configure speed and number of steps to rotate, this program runs without serial connection but motors will rotate only 100 steps each time it is switched ON. Connect a RS232 to TTL level converter like MAX 232 between TX and RX pin of uC and PC serial port. Please read http://www.triindia.co.in/resources/?p=35 to configure your hyperterminal to be able to communicate with your robot through PC serial port.
Uart.h enables us to use C functions printf , scanf with controller. printf ("string"); writes specified string on hyperterminal screen and scanf can be used to take input from hyperterminal.

I have tested this circuit and program and it works fine but, There may be some bug introduced while writing this article so please post your comments to find the bug if any.

Downlaod main.h and uart.h from http://www.swapniljariwala.co.nr/files/stepper.zip

main.h:

```
#define UART_BAUD 19200  // change this to select different baudrate.

#include <ctype.h>
#include <stdint.h>
#include <stdio.h>
#include <string.h>
#include <avr/io.h>
#include <avr/pgmspace.h>
#include <avr/interrupt.h>

#include <util/delay.h>

#include "uart.h"
#define F_CPU 8000000    // Crystal frequency.
#include <avr/eeprom.h>


// required for serial communication.
// Copy as it is and place before main() to use in other projects
FILE uart_str = FDEV_SETUP_STREAM(uart_putchar, uart_getchar, _FDEV_SETUP_RW);

//globals
//const seq[]={8,4,2,1};
const seq[]={8,12,4,6,2,3,1,9}; // half stepping sequence.
//const seq[]={5,9,10,6};
//const seq[]={5,1,9,8,10,2,6,4};


volatile int stp,stptop;
volatile char ;

void mydelay();
void timer_init();
void main()
{
   unsigned char a;
   DDRD=0xf0;
   DDRB=0xff;
   DDRC=0xff;

   timer_init();  // timer initialization in CTC mode and 1024 prescalar
                  // and timer interrupt enable.
   uart_init();  // uart initialisation
   UCSRB=UCSRB|(1<<RXCIE);        // enable serial receive interrupt.
```

```
   // copy inside the main to use serial in other project.
   stdout = stdin = &uart_str; // required to be able to use
                               // printf scanf function as in c
                               // for serial communication.
   sei();         // enable interrupt
   escape(CLR);
   OCR0=30;
   stptop=100; //  motor will stop after 100 steps.
   stp=0;
   printf(" Device ready : \n");
   while(1);


  /* while(1)
   {
  for(a=0;a<4;a++)
   {
            PORTC=seq[a];
            PORTA=seq[a];
      _delay_ms(50);
      }
      }
      */
}


ISR(TIMER0_COMP_vect)
{
    static unsigned char cnt;//stepper motor sequence count
      static unsigned int stpcnt; // number of steps to count distance.
      cnt++;


      if(stp<stptop) //motors stops when number of steps stp exceeds stptop
     {
            PORTD=seq[cnt]<<4;  // rotate motor 1 clockwise
            PORTB=seq[7-cnt];   // rotate motor 2 anticlockwise so that mouse
//moves straight
       }

      stp++;

      if(cnt>=7)
    cnt=0;

}


ISR(USART_RXC_vect)
{
  char ch;
  cli();
  PORTB=0;
  PORTD=0;
  escape(CLR);
  //uart_getchar(stdin);
```

```
  printf(" \n Speed (1) Steps (2) :");
  scanf("%c",&ch);
  if(ch=='1')
    {

            scanf("%d",&OCR0);
            printf("\n received OCR0 %d",OCR0);
            printf(" pulse rate %d",4000/(1+OCR0));


      }
   else if(ch=='2')
    {
       printf("%d Enter number of steps :",stp);
         scanf("%d",&stptop);
         stp=0;
       while((PIND&0x8)>0);
       mydelay();
      //   printf("hi");
      }
  sei();
  //printf("sei");
}

void timer_init()
{
      TCCR0=(1<<WGM01)|(5<<CS00); //set WGM01 bit ie CTC mode and 1024 prescalar
      TIMSK=(1<<OCIE0);         // set Timer/Counter0 Output Compare Match Interrupt Enable

}
void mydelay()
{
   unsigned int i,j;
   //printf("ma delay");
   for(i=0;i<10000;i++)
   for(j=0;j<10000;j++);
   //printf("tata");
}
```

uart.h: lifted from winavr examples.

```
/*
 * ----------------------------------------------------------------------------
 * "THE BEER-WARE LICENSE" (Revision 42):
 * <joerg@FreeBSD.ORG> wrote this file.  As long as you retain this notice you
 * can do whatever you want with this stuff. If we meet some day, and you think
 * this stuff is worth it, you can buy me a beer in return.        Joerg Wunsch
 * ----------------------------------------------------------------------------
 *
 * Stdio demo, UART declarations
 *
 * $Id: uart.h,v 1.1.2.1 2005/12/28 22:35:08 joerg_wunsch Exp $
 */

/*
 * Perform UART startup initialization.
 */
 #define CLR 45
void  uart_init(void);

/*
 * Send one character to the UART.
 */
int   uart_putchar(char c, FILE *stream);

/*
 * Size of internal line buffer used by uart_getchar().
 */
#define RX_BUFSIZE 80

/*
 * Receive one character from the UART.  The actual reception is
 * line-buffered, and one character is returned from the buffer at
 * each invokation.
 */
int   uart_getchar(FILE *stream);

void escape(char ch)
{
    loop_until_bit_is_set(UCSRA, UDRE);
    UDR = ch;
}



/*
 * Initialize the UART to 9600 Bd, tx/rx, 8N1.
 */
void
uart_init(void)
{
#if F_CPU < 2000000UL && defined(U2X)
```

```
  UCSRA = _BV(U2X);             /* improve baud rate error by using 2x clk */
  UBRRL = (F_CPU / (8UL * UART_BAUD)) - 1;
#else
  UBRRL = (F_CPU / (16UL * UART_BAUD)) - 1;
#endif
  UCSRB = _BV(TXEN) | _BV(RXEN); /* tx/rx enable */
}




/*
 * Send character c down the UART Tx, wait until tx holding register
 * is empty.
 */
int
uart_putchar(char c, FILE *stream)
{

  if (c == '\a')
    {
      fputs("*ring*\n", stderr);
      return 0;
    }

  if (c == '\n')
    uart_putchar('\r', stream);
  loop_until_bit_is_set(UCSRA, UDRE);
  UDR = c;

  return 0;
}

/*
 * Receive a character from the UART Rx.
 *
 * This features a simple line-editor that allows to delete and
 * re-edit the characters entered, until either CR or NL is entered.
 * Printable characters entered will be echoed using uart_putchar().
 *
 * Editing characters:
 *
 * . \b (BS) or \177 (DEL) delete the previous character
 * . ^u kills the entire input buffer
 * . ^w deletes the previous word
 * . ^r sends a CR, and then reprints the buffer
 * . \t will be replaced by a single space
 *
 * All other control characters will be ignored.
 *
 * The internal line buffer is RX_BUFSIZE (80) characters long, which
 * includes the terminating \n (but no terminating \0).  If the buffer
 * is full (i. e., at RX_BUFSIZE-1 characters in order to keep space for
 * the trailing \n), any further input attempts will send a \a to
 * uart_putchar() (BEL character), although line editing is still
```

```
 * allowed.
 *
 * Input errors while talking to the UART will cause an immediate
 * return of -1 (error indication).  Notably, this will be caused by a
 * framing error (e. g. serial line "break" condition), by an input
 * overrun, and by a parity error (if parity was enabled and automatic
 * parity recognition is supported by hardware).
 *
 * Successive calls to uart_getchar() will be satisfied from the
 * internal buffer until that buffer is emptied again.
 */
int
uart_getchar(FILE *stream)
{
  uint8_t c;
  char *cp, *cp2;
  static char b[RX_BUFSIZE];
  static char *rxp;

  if (rxp == 0)
    for (cp = b;;)
      {
      loop_until_bit_is_set(UCSRA, RXC);
      if (UCSRA & _BV(FE))
        return _FDEV_EOF;
      if (UCSRA & _BV(DOR))
        return _FDEV_ERR;
      c = UDR;
      /* behaviour similar to Unix stty ICRNL */
      if (c == '\r')
        c = '\n';
      if (c == '\n')
        {
          *cp = c;
          uart_putchar(c, stream);
          rxp = b;
          break;
        }
      else if (c == '\t')
        c = ' ';

      if ((c >= (uint8_t)' ' && c <= (uint8_t)'\x7e') ||
          c >= (uint8_t)'\xa0')
        {
          if (cp == b + RX_BUFSIZE - 1)
            uart_putchar('\a', stream);
          else
            {
            *cp++ = c;
            uart_putchar(c, stream);
            }
          continue;
        }

      switch (c)
        {
        case 'c' & 0x1f:
```

```
          return -1;

        case '\b':
        case '\x7f':
          if (cp > b)
            {
            uart_putchar('\b', stream);
            uart_putchar(' ', stream);
            uart_putchar('\b', stream);
            cp--;
            }
          break;

        case 'r' & 0x1f:
          uart_putchar('\r', stream);
          for (cp2 = b; cp2 < cp; cp2++)
            uart_putchar(*cp2, stream);
          break;

        case 'u' & 0x1f:
          while (cp > b)
            {
            uart_putchar('\b', stream);
            uart_putchar(' ', stream);
            uart_putchar('\b', stream);
            cp--;
            }
          break;

        case 'w' & 0x1f:
          while (cp > b && cp[-1] != ' ')
            {
            uart_putchar('\b', stream);
            uart_putchar(' ', stream);
            uart_putchar('\b', stream);
            cp--;
            }
          break;
        }
      }

  c = *rxp++;
  if (c == '\n')
    rxp = 0;

  return c;
}
```

makefile : copy content and save it as makefile don’t give any extension to it.

```
# Hey Emacs, this is a -*- makefile -*-
#----------------------------------------------------------------------------
# WinAVR Makefile Template written by Eric B. Weddington, Jörg Wunsch, et al.
#
# Released to the Public Domain
#
# Additional material for this makefile was written by:
# Peter Fleury
# Tim Henigan
# Colin O'Flynn
# Reiner Patommel
# Markus Pfaff
# Sander Pool
# Frederik Rouleau
# Carlos Lamas
#
#----------------------------------------------------------------------------
# On command line:
#
# make all = Make software.
#
# make clean = Clean out built project files.
#
# make coff = Convert ELF to AVR COFF.
#
# make extcoff = Convert ELF to AVR Extended COFF.
#
# make program = Download the hex file to the device, using avrdude.
#                Please customize the avrdude settings below first!
#
# make debug = Start either simulavr or avarice as specified for debugging,
#              with avr-gdb or avr-insight as the front end for debugging.
#
# make filename.s = Just compile filename.c into the assembler code only.
#
# make filename.i = Create a preprocessed source file for use in submitting
#                   bug reports to the GCC project.
#
# To rebuild project do "make clean" then "make all".
#----------------------------------------------------------------------------


# MCU name
MCU = atmega32


# Processor frequency.
#     This will define a symbol, F_CPU, in all source code files equal to the
#     processor frequency. You can then use this symbol in your source code to
#     calculate timings. Do NOT tack on a 'UL' at the end, this will be done
#     automatically to create a 32-bit value in your source code.
```

```
#     Typical values are:
#         F_CPU =  1000000
#         F_CPU =  1843200
#         F_CPU =  2000000
#         F_CPU =  3686400
#         F_CPU =  4000000
#         F_CPU =  7372800
#         F_CPU =  8000000
#         F_CPU = 11059200
#         F_CPU = 14745600
#         F_CPU = 16000000
#         F_CPU = 18432000
#         F_CPU = 20000000
F_CPU = 8000000


# Output format. (can be srec, ihex, binary)
FORMAT = ihex


# Target file name (without extension).
TARGET = testpp


# Object files directory
OBJDIR = obj


# List C source files here. (C dependencies are automatically generated.)
SRC =


# List C++ source files here. (C dependencies are automatically generated.)
CPPSRC = main.cpp


# List Assembler source files here.
#     Make them always end in a capital .S.  Files ending in a lowercase .s
#     will not be considered source files but generated files (assembler
#     output from the compiler), and will be deleted upon "make clean"!
#     Even though the DOS/Win* filesystem matches both .s and .S the same,
#     it will preserve the spelling of the filenames, and gcc itself does
#     care about how the name is spelled on its command-line.
ASRC =


# Optimization level, can be [0, 1, 2, 3, s].
#     0 = turn off optimization. s = optimize for size.
#     (Note: 3 is not always the best optimization level. See avr-libc FAQ.)
OPT = s


# Debugging format.
#     Native formats for AVR-GCC's -g are dwarf-2 [default] or stabs.
#     AVR Studio 4.10 requires dwarf-2.
#     AVR [Extended] COFF format requires stabs, plus an avr-objcopy run.
DEBUG = dwarf-2
```

```
# List any extra directories to look for include files here.
#     Each directory must be seperated by a space.
#     Use forward slashes for directory separators.
#     For a directory that has spaces, enclose it in quotes.
EXTRAINCDIRS =


# Compiler flag to set the C Standard level.
#     c89   = "ANSI" C
#     gnu89 = c89 plus GCC extensions
#     c99   = ISO C99 standard (not yet fully implemented)
#     gnu99 = c99 plus GCC extensions
CSTANDARD = -std=gnu99


# Place -D or -U options here for C sources
CDEFS = -DF_CPU=$(F_CPU)UL


# Place -D or -U options here for C++ sources
CPPDEFS = -DF_CPU=$(F_CPU)UL
#CPPDEFS += -D__STDC_LIMIT_MACROS
#CPPDEFS += -D__STDC_CONSTANT_MACROS



#---------------- Compiler Options C ----------------
#  -g*:          generate debugging information
#  -O*:          optimization level
#  -f...:        tuning, see GCC manual and avr-libc documentation
#  -Wall...:     warning level
#  -Wa,...:      tell GCC to pass this to the assembler.
#    -adhlns...: create assembler listing
CFLAGS = -g$(DEBUG)
CFLAGS += $(CDEFS)
CFLAGS += -O$(OPT)
#CFLAGS += -mint8
#CFLAGS += -mshort-calls
CFLAGS += -funsigned-char
CFLAGS += -funsigned-bitfields
CFLAGS += -fpack-struct
CFLAGS += -fshort-enums
#CFLAGS += -fno-unit-at-a-time
CFLAGS += -Wall
CFLAGS += -Wstrict-prototypes
CFLAGS += -Wundef
#CFLAGS += -Wunreachable-code
#CFLAGS += -Wsign-compare
CFLAGS += -Wa,-adhlns=$(<:%.c=$(OBJDIR)/%.lst)
CFLAGS += $(patsubst %,-I%,$(EXTRAINCDIRS))
CFLAGS += $(CSTANDARD)


#---------------- Compiler Options C++ ----------------
#  -g*:          generate debugging information
```

```
#  -O*:          optimization level
#  -f...:        tuning, see GCC manual and avr-libc documentation
#  -Wall...:     warning level
#  -Wa,...:      tell GCC to pass this to the assembler.
#    -adhlns...: create assembler listing
CPPFLAGS = -g$(DEBUG)
CPPFLAGS += $(CPPDEFS)
CPPFLAGS += -O$(OPT)
#CPPFLAGS += -mint8
#CPPFLAGS += -mshort-calls
CPPFLAGS += -funsigned-char
CPPFLAGS += -funsigned-bitfields
CPPFLAGS += -fpack-struct
CPPFLAGS += -fshort-enums
CPPFLAGS += -fno-exceptions
#CPPFLAGS += -fno-unit-at-a-time
CPPFLAGS += -Wall
#CPPFLAGS += -Wstrict-prototypes
CFLAGS += -Wundef
#CPPFLAGS += -Wunreachable-code
#CPPFLAGS += -Wsign-compare
CPPFLAGS += -Wa,-adhlns=$(<:%.cpp=$(OBJDIR)/%.lst)
CPPFLAGS += $(patsubst %,-I%,$(EXTRAINCDIRS))
#CPPFLAGS += $(CSTANDARD)


#---------------- Assembler Options ----------------
#  -Wa,...:   tell GCC to pass this to the assembler.
#  -ahlms:    create listing
#  -gstabs:   have the assembler create line number information; note that
#             for use in COFF files, additional information about filenames
#             and function names needs to be present in the assembler source
#             files -- see avr-libc docs [FIXME: not yet described there]
ASFLAGS = -Wa,-adhlns=$(<:%.S=$(OBJDIR)/%.lst),-gstabs


#---------------- Library Options ----------------
# Minimalistic printf version
PRINTF_LIB_MIN = -Wl,-u,vfprintf -lprintf_min

# Floating point printf version (requires MATH_LIB = -lm below)
PRINTF_LIB_FLOAT = -Wl,-u,vfprintf -lprintf_flt

# If this is left blank, then it will use the Standard printf version.
PRINTF_LIB =
#PRINTF_LIB = $(PRINTF_LIB_MIN)
#PRINTF_LIB = $(PRINTF_LIB_FLOAT)


# Minimalistic scanf version
SCANF_LIB_MIN = -Wl,-u,vfscanf -lscanf_min

# Floating point + %[ scanf version (requires MATH_LIB = -lm below)
SCANF_LIB_FLOAT = -Wl,-u,vfscanf -lscanf_flt

# If this is left blank, then it will use the Standard scanf version.
SCANF_LIB =
```

```
#SCANF_LIB = $(SCANF_LIB_MIN)
#SCANF_LIB = $(SCANF_LIB_FLOAT)


MATH_LIB = -lm



#---------------- External Memory Options ----------------

# 64 KB of external RAM, starting after internal RAM (ATmega128!),
# used for variables (.data/.bss) and heap (malloc()).
#EXTMEMOPTS = -Wl,-Tdata=0x801100,--defsym=__heap_end=0x80ffff

# 64 KB of external RAM, starting after internal RAM (ATmega128!),
# only used for heap (malloc()).
#EXTMEMOPTS = -Wl,--defsym=__heap_start=0x801100,--defsym=__heap_end=0x80ffff

EXTMEMOPTS =



#---------------- Linker Options ----------------
#  -Wl,...:     tell GCC to pass this to linker.
#    -Map:      create map file
#    --cref:    add cross reference to  map file
LDFLAGS = -Wl,-Map=$(TARGET).map,--cref
LDFLAGS += $(EXTMEMOPTS)
LDFLAGS += $(PRINTF_LIB) $(SCANF_LIB) $(MATH_LIB)
#LDFLAGS += -T linker_script.x



#---------------- Programming Options (avrdude) ----------------

# Programming hardware: alf avr910 avrisp bascom bsd
# dt006 pavr picoweb pony-stk200 sp12 stk200 stk500
#
# Type: avrdude -c ?
# to get a full listing.
#
AVRDUDE_PROGRAMMER = pony-stk200

# com1 = serial port. Use lpt1 to connect to parallel port.
AVRDUDE_PORT = lpt1

AVRDUDE_WRITE_FLASH = -U flash:w:$(TARGET).hex
#AVRDUDE_WRITE_EEPROM = -U eeprom:w:$(TARGET).eep


# Uncomment the following if you want avrdude's erase cycle counter.
# Note that this counter needs to be initialized first using -Yn,
# see avrdude manual.
#AVRDUDE_ERASE_COUNTER = -y

# Uncomment the following if you do /not/ wish a verification to be
# performed after programming the device.
```

```
#AVRDUDE_NO_VERIFY = -V

# Increase verbosity level.  Please use this when submitting bug
# reports about avrdude. See <http://savannah.nongnu.org/projects/avrdude>
# to submit bug reports.
#AVRDUDE_VERBOSE = -v -v

AVRDUDE_FLAGS = -p $(MCU) -P $(AVRDUDE_PORT) -c $(AVRDUDE_PROGRAMMER)
AVRDUDE_FLAGS += $(AVRDUDE_NO_VERIFY)
AVRDUDE_FLAGS += $(AVRDUDE_VERBOSE)
AVRDUDE_FLAGS += $(AVRDUDE_ERASE_COUNTER)



#---------------- Debugging Options ----------------

# For simulavr only - target MCU frequency.
DEBUG_MFREQ = $(F_CPU)

# Set the DEBUG_UI to either gdb or insight.
# DEBUG_UI = gdb
DEBUG_UI = insight

# Set the debugging back-end to either avarice, simulavr.
DEBUG_BACKEND = avarice
#DEBUG_BACKEND = simulavr

# GDB Init Filename.
GDBINIT_FILE = __avr_gdbinit

# When using avarice settings for the JTAG
JTAG_DEV = /dev/com1

# Debugging port used to communicate between GDB / avarice / simulavr.
DEBUG_PORT = 4242

# Debugging host used to communicate between GDB / avarice / simulavr, normally
#     just set to localhost unless doing some sort of crazy debugging when
#     avarice is running on a different computer.
DEBUG_HOST = localhost



#============================================================================


# Define programs and commands.
SHELL = sh
CC = avr-gcc
OBJCOPY = avr-objcopy
OBJDUMP = avr-objdump
SIZE = avr-size
NM = avr-nm
AVRDUDE = avrdude
REMOVE = rm -f
REMOVEDIR = rm -rf
COPY = cp
```

```
WINSHELL = cmd


# Define Messages
# English
MSG_ERRORS_NONE = Errors: none
MSG_BEGIN = -------- begin --------
MSG_END = --------  end  --------
MSG_SIZE_BEFORE = Size before:
MSG_SIZE_AFTER = Size after:
MSG_COFF = Converting to AVR COFF:
MSG_EXTENDED_COFF = Converting to AVR Extended COFF:
MSG_FLASH = Creating load file for Flash:
MSG_EEPROM = Creating load file for EEPROM:
MSG_EXTENDED_LISTING = Creating Extended Listing:
MSG_SYMBOL_TABLE = Creating Symbol Table:
MSG_LINKING = Linking:
MSG_COMPILING = Compiling C:
MSG_COMPILING_CPP = Compiling C++:
MSG_ASSEMBLING = Assembling:
MSG_CLEANING = Cleaning project:
MSG_CREATING_LIBRARY = Creating library:




# Define all object files.
OBJ = $(SRC:%.c=$(OBJDIR)/%.o) $(CPPSRC:%.cpp=$(OBJDIR)/%.o)
$(ASRC:%.S=$(OBJDIR)/%.o)

# Define all listing files.
LST = $(SRC:%.c=$(OBJDIR)/%.lst) $(CPPSRC:%.cpp=$(OBJDIR)/%.lst)
$(ASRC:%.S=$(OBJDIR)/%.lst)


# Compiler flags to generate dependency files.
GENDEPFLAGS = -MD -MP -MF .dep/$(@F).d


# Combine all necessary flags and optional flags.
# Add target processor to flags.
ALL_CFLAGS = -mmcu=$(MCU) -I. $(CFLAGS) $(GENDEPFLAGS)
ALL_CPPFLAGS = -mmcu=$(MCU) -I. -x c++ $(CPPFLAGS) $(GENDEPFLAGS)
ALL_ASFLAGS = -mmcu=$(MCU) -I. -x assembler-with-cpp $(ASFLAGS)




# Default target.
all: begin gccversion sizebefore build sizeafter end

# Change the build target to build a HEX file or a library.
build: elf hex eep lss sym
#build: lib
```

```
elf: $(TARGET).elf
hex: $(TARGET).hex
eep: $(TARGET).eep
lss: $(TARGET).lss
sym: $(TARGET).sym
LIBNAME=lib$(TARGET).a
lib: $(LIBNAME)



# Eye candy.
# AVR Studio 3.x does not check make's exit code but relies on
# the following magic strings to be generated by the compile job.
begin:
	@echo
	@echo $(MSG_BEGIN)

end:
	@echo $(MSG_END)
	@echo


# Display size of file.
HEXSIZE = $(SIZE) --target=$(FORMAT) $(TARGET).hex
ELFSIZE = $(SIZE) -A $(TARGET).elf
AVRMEM = avr-mem.sh $(TARGET).elf $(MCU)

sizebefore:
	@if test -f $(TARGET).elf; then echo; echo $(MSG_SIZE_BEFORE); $(ELFSIZE);
\
	$(AVRMEM) 2>/dev/null; echo; fi

sizeafter:
	@if test -f $(TARGET).elf; then echo; echo $(MSG_SIZE_AFTER); $(ELFSIZE);
\
	$(AVRMEM) 2>/dev/null; echo; fi



# Display compiler version information.
gccversion :
	@$(CC) --version



# Program the device.
program: $(TARGET).hex $(TARGET).eep
	$(AVRDUDE) $(AVRDUDE_FLAGS) $(AVRDUDE_WRITE_FLASH) $(AVRDUDE_WRITE_EEPROM)


# Generate avr-gdb config/init file which does the following:
#     define the reset signal, load the target file, connect to target, and set
#     a breakpoint at main().
gdb-config:
	@$(REMOVE) $(GDBINIT_FILE)
	@echo define reset >> $(GDBINIT_FILE)
	@echo SIGNAL SIGHUP >> $(GDBINIT_FILE)
```

```
	@echo end >> $(GDBINIT_FILE)
	@echo file $(TARGET).elf >> $(GDBINIT_FILE)
	@echo target remote $(DEBUG_HOST):$(DEBUG_PORT)  >> $(GDBINIT_FILE)
ifeq ($(DEBUG_BACKEND),simulavr)
	@echo load  >> $(GDBINIT_FILE)
endif
	@echo break main >> $(GDBINIT_FILE)

debug: gdb-config $(TARGET).elf
ifeq ($(DEBUG_BACKEND), avarice)
	@echo Starting AVaRICE - Press enter when "waiting to connect" message
displays.
	@$(WINSHELL) /c start avarice --jtag $(JTAG_DEV) --erase --program --file
\
	$(TARGET).elf $(DEBUG_HOST):$(DEBUG_PORT)
	@$(WINSHELL) /c pause

else
	@$(WINSHELL) /c start simulavr --gdbserver --device $(MCU) --clock-freq \
	$(DEBUG_MFREQ) --port $(DEBUG_PORT)
endif
	@$(WINSHELL) /c start avr-$(DEBUG_UI) --command=$(GDBINIT_FILE)



# Convert ELF to COFF for use in debugging / simulating in AVR Studio or VMLAB.
COFFCONVERT = $(OBJCOPY) --debugging
COFFCONVERT += --change-section-address .data-0x800000
COFFCONVERT += --change-section-address .bss-0x800000
COFFCONVERT += --change-section-address .noinit-0x800000
COFFCONVERT += --change-section-address .eeprom-0x810000



coff: $(TARGET).elf
	@echo
	@echo $(MSG_COFF) $(TARGET).cof
	$(COFFCONVERT) -O coff-avr $< $(TARGET).cof


extcoff: $(TARGET).elf
	@echo
	@echo $(MSG_EXTENDED_COFF) $(TARGET).cof
	$(COFFCONVERT) -O coff-ext-avr $< $(TARGET).cof



# Create final output files (.hex, .eep) from ELF output file.
%.hex: %.elf
	@echo
	@echo $(MSG_FLASH) $@
	$(OBJCOPY) -O $(FORMAT) -R .eeprom $< $@

%.eep: %.elf
	@echo
	@echo $(MSG_EEPROM) $@
```

```
	-$(OBJCOPY) -j .eeprom --set-section-flags=.eeprom="alloc,load" \
	--change-section-lma .eeprom=0 -O $(FORMAT) $< $@

# Create extended listing file from ELF output file.
%.lss: %.elf
	@echo
	@echo $(MSG_EXTENDED_LISTING) $@
	$(OBJDUMP) -h -S $< > $@

# Create a symbol table from ELF output file.
%.sym: %.elf
	@echo
	@echo $(MSG_SYMBOL_TABLE) $@
	$(NM) -n $< > $@



# Create library from object files.
.SECONDARY : $(TARGET).a
.PRECIOUS : $(OBJ)
%.a: $(OBJ)
	@echo
	@echo $(MSG_CREATING_LIBRARY) $@
	$(AR) $@ $(OBJ)


# Link: create ELF output file from object files.
.SECONDARY : $(TARGET).elf
.PRECIOUS : $(OBJ)
%.elf: $(OBJ)
	@echo
	@echo $(MSG_LINKING) $@
	$(CC) $(ALL_CFLAGS) $^ --output $@ $(LDFLAGS)


# Compile: create object files from C source files.
$(OBJDIR)/%.o : %.c
	@echo
	@echo $(MSG_COMPILING) $<
	$(CC) -c $(ALL_CFLAGS) $< -o $@


# Compile: create object files from C++ source files.
$(OBJDIR)/%.o : %.cpp
	@echo
	@echo $(MSG_COMPILING_CPP) $<
	$(CC) -c $(ALL_CPPFLAGS) $< -o $@


# Compile: create assembler files from C source files.
%.s : %.c
	$(CC) -S $(ALL_CFLAGS) $< -o $@


# Compile: create assembler files from C++ source files.
%.s : %.cpp
	$(CC) -S $(ALL_CPPFLAGS) $< -o $@
```

```
# Assemble: create object files from assembler source files.
$(OBJDIR)/%.o : %.S
	@echo
	@echo $(MSG_ASSEMBLING) $<
	$(CC) -c $(ALL_ASFLAGS) $< -o $@


# Create preprocessed source for use in sending a bug report.
%.i : %.c
	$(CC) -E -mmcu=$(MCU) -I. $(CFLAGS) $< -o $@


# Target: clean project.
clean: begin clean_list end

clean_list :
	@echo
	@echo $(MSG_CLEANING)
	$(REMOVE) $(TARGET).hex
	$(REMOVE) $(TARGET).eep
	$(REMOVE) $(TARGET).cof
	$(REMOVE) $(TARGET).elf
	$(REMOVE) $(TARGET).map
	$(REMOVE) $(TARGET).sym
	$(REMOVE) $(TARGET).lss
	$(REMOVEDIR) $(OBJDIR)
	$(REMOVE) $(SRC:.c=.s)
	$(REMOVE) $(SRC:.c=.d)
	$(REMOVEDIR) .dep


# Create object files directory
$(shell mkdir $(OBJDIR) 2>/dev/null)


# Include the dependency files.
-include $(shell mkdir .dep 2>/dev/null) $(wildcard .dep/*)


# Listing of phony targets.
.PHONY : all begin finish end sizebefore sizeafter gccversion \
build elf hex eep lss sym coff extcoff \
clean clean_list program debug gdb-config
```