# - Makefile - myROBOT.ru

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

# ОСНОВНЫЕ ИНСТРУКЦИИ

# Тип используемого микроконтроллера

MY\_MCU = atmega8

# Частота кварцевого резонатора (Hz)

MY\_F\_CPU = 1000000

# Название файла проекта (без расширения)

MY\_TARGET = my\_test

# Используемый программатор (для AVRDUDE)

MY\_AVRDUDE\_PROGRAMMER = stk200

# Порт, к которому подключен программатор

MY\_PROGRAMMER\_PORT = lpt1

# Уровень оптимизации (может быть 0, 1, 2, 3, s)

# 0 - выключает оптимизацию, s - оптимизирует размер файла

OPTIMIZATION\_LEVEL = s

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

#----------------------------------------------------------------------------

# WinAVR Makefile Template written by Eric B. Weddington, Jцrg 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 = $(MY\_MCU)

# 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 = $(MY\_F\_CPU)

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

FORMAT = ihex

# Target file name (without extension).

TARGET = $(MY\_TARGET)

# Object files directory

OBJDIR = obj

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

SRC =$(TARGET).c

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

CPPSRC =

# 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 = $(OPTIMIZATION\_LEVEL)

# 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 = $(MY\_AVRDUDE\_PROGRAMMER)

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

AVRDUDE\_PORT = $(MY\_PROGRAMMER\_PORT)

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