amforth-installation-ja Documentation

リリース *0.1*

kuma35

2019 年 06 月 24 日

目次

Contents:

第 1 章

amforth インストール クイックスタート

1.1 用意するハードウェア

PC　今回は ubuntu 12.04LTS(amd64) をインストールしたマシンを使っています。

Arduino uno R3　オープンハードなので自作しても構いません。千石電商では 3000 円弱 (2013/3 月)。

USB A-B ケーブル　Arduino には付属してないので。今回は 50cm ぐらいのを近所のダイソーで買いました。100 円。

ATmega328P-PU　atmel 社の IC です。Arduino uno R3 に載っているのと同じものです。千石電商では 300 円ぐらい (2013/3 月)。秋月電子通商では 250 円のようです。末尾の PU はパッケージの形態を示しています。ここが違うと IC ソケットに刺せなくなるので注意。Arduino のチップを直接書き換えるなら購入しなくても構いません。

AVRISPmkII　ISP(In System Programer) と呼ばれるものです。Arduino 上の ATmega328P-PU にあらかじめ書き込まれたブートローダーを介さずに書き込みを行う為に利用します。ISP は通常のプログラマ (ROM ライタ) に比べると Arduino 上に IC を装着したまま書き換えるのと、比較的安価なのが魅力です。一方、ATmega 専用になってしまうので、既に ATmeaga にも対応したプログラマ (ROM ライタ) を持っている人はそちらを使って構いません。AVRISPmkII は USB ケーブル付属しています。秋月電子通商では 3000 円 (2013/3 月)。

IC 引き抜き工具　あれば便利。手元にあるのは aitendo で買ったもの。200 円ぐらい。

1.2 ROM ライタ＆書き込みソフト準備

ROM ライタ (プログラマ) を用意します。ここでは AVRISP mkII を使っています。パラレルプログラマ等を利用する場合でもヒューズビット、ロックビット、hex ファイルの転送は同じです。以後の説明は AVRISP mkII 前提になっているので適宜読み替えてください。

次にご利用になる ISP に対応した書き込みソフトを準備します。今回は AVRISP mkII を使ったので AVRISP mkII の場合は avrdude を準備します。aduino-1.0.3 を入手した場合は arduino-1.0.3/hardware/tools にあります。

手元の ubuntu が 64bits 版なので avrdude64 を使いました。

1.3 書き込みファイル準備

amforth をサイトから取得します。

サイトは http://amforth.sourceforge.net/ ダウンロードはサイトから DOWNLOAD リンクで辿れます (http://sourceforge.net/projects/amforth/)

ダウンロードした場合は展開すると、 appl/arduino の下に *.hex ファイルが入っています (ソースの場合は make が必要です (後述))。

1.4 準備

Arduino uno R3 の IC を IC 引き抜き工具等で引き抜き、書き込みしたい ATmega328P-PU をセットします。

ISP を接続します。 Arduino uno R3 は角にある白い丸印が 1 番ピン示していますので、これと ISP のケーブルの 1 番ピンの位置を合わせます。

AVRISP mkII の場合はターゲットのボード側にも電源供給が必要なので、 AVRISP mkII の USB ケーブルに加えて Arduino uno R3 の USB ケーブルも接続します。

1.5 ヒューズビット (fuse)、ロックビット (lock) 設定

avrdude を -P usb で使うためには root 権限が必要です (udev の設定も色々試したが分からなかった)。

ヒューズビット等設定用に edit_uno.sh を定義します。例えば以下のような感じです。

```
#!/bin/bash
HARDWARE_TOOLS=$HOME/.arduino/arduino/hardware/tools
AVRDUDE=$HARDWARE_TOOLS/avrdude64
AVRDUDE_CONF=$HARDWARE_TOOLS/avrdude.conf
sudo $AVRDUDE -C $AVRDUDE_CONF -P usb -p atmega328p -c avrisp2 -t -v
```

- -C avrdude.conf ファイルの指定。対象チップごとの情報が記述してある。
- -P アクセスするポートの指定。root で usb と指定した場合は avrdude が適切なポートを探し出して接続してくれる。
- -p 指定チップ名の指定。 Arduino uno R3 の場合は atmega328p を指定する。

- -c プログラマ (ISP) の指定。 AVRISP mkII の場合は avrisp2 と指定。
- -t プロンプトを出して入力待ちになる。
- -v verbose。

avrdude を立ち上げます。sudo のパスワードを聞かれるので適宜入力してください。

```
$ ./edit_uno.sh
```

signatrue が 0x1e950f なら正常に接続できています。0x000000 の場合は通信できていません。

lfuse, hfuse, efuse は表示されている値になっているので、指定します。

```
avrdude>w hfuse 0 0xd9
```

lock bits は表示されていないので、まず表示します。

```
avrdude>d lock
```

次に設定します。

```
avrdude>w lock 0 0x3f
```

avrdude を終了します。

```
avrdude>quit
```

1.6 書き込み

ヒューズビット、ロックビットの設定と同様に upload_uno.sh というスクリプトを作ります。例えば以下のような感じです。

```
#!/bin/bash
HARDWARE_TOOLS=$HOME/.arduino/arduino/hardware/tools
AVRDUDE=$HARDWARE_TOOLS/avrdude64
AVRDUDE_CONF=$HARDWARE_TOOLS/avrdude.conf
sudo $AVRDUDE -C $AVRDUDE_CONF -P usb -p atmega328p -c avrisp2 -U flash:w:uno.hex -U␣
↪eeprom:w:uno.eep.hex -v
```

- -C avrdude.conf ファイルの指定。対象チップごとの情報が記述してある。
- -P アクセスするポートの指定。root で usb と指定した場合は avrdude が適切なポートを探し出して接続してくれる。
- -p 指定チップ名の指定。 Arduino uno R3 の場合は atmega328p を指定する。

- -c プログラマ (ISP) の指定。 AVRISP mkII の場合は avrisp2 と指定。
- -U flashw:uno.hex カレントディレクトリの uno.hex を flash へ書き込む。
- -U eeprom:w:uno.epp.hex カレントディレクトリの uno.epp.hex を EEPROM へ書き込む。
- -v verbose。

-U で指定した順番で書き込みが行われます。それぞれ verify を行い、違いがあった場合はそこで中断します。

1.7 動作確認

ISP を取り外します。

好きなターミナルでアクセスします。ここでは byobu を使っています。

```
$ byobu /dev/ttyACM0
```

以下のとおりタイトル、プロンプトが出れば転送は出来ています (5.0 のところはバージョンによって異なる)。

```
amforth 5.0 ATmega328p ForthDuino
>
```

注釈: byobu を終了したい場合は、 プレフィックス k (CTRL+A k) で殺す。


hideo@mevius (192.168.115.203) - byobu
amforth 5.0 ATmega328P Forthduino
UNO
ARDUINO

第 2 章

amforth をソースからビルドする場合

2.1 ソースコードの入手

subversion でソースコードを入手します。

```
svn checkout svn://svn.code.sf.net/p/amforth/code/trunk amforth-code
```

ソースコードを入手したら早速ビルドと行きたいところですが、amforth はオールアセンブラで書かれていて、アセンブルには AVR マイコン用のアセンブラを使う必要があります。

Makefile にもさらりと

```
wine avrasm2.exe
```

と書いてあったりして。linux 版が無いのか探して見ましたが分かりませんでした (2013/03/21 現在)。なので、avrasm2.exe を入手します。

2.2 avrasm2.exe の入手

ググってみると、avrasm2.exe は avr studio に含まれているようです。

と言うことで最新 (2013/03/21) の avr studio 6 を入手して wine でインストールしようとしましたが上手くできませんでした。

よって、バージョンを下げて、4.19 をインストールしました (http://www.atmel.jp/ja/jp/tools/AVRSTUDIO4.aspx)

注釈: 入手時にメアドとか尋ねられるので入力してやってください。入力したメアド宛にダウンロードリンクをメールしてくるのでメアドは使えるものを入力しましょう。

2.3 Makefile 改造

amforth-code/appl/arduino/Makefile は avrasm2.exe のファイルパス等、そのままでは動かないので改造します (Makefile 全文は節末参照)。

105 行〜150 行を抜粋。

```
# AMFORTH VERSION TO USE
# 'code' for trunk and x.y for the releases (i.e 5.0)
#VERSION=5.0
VERSION=code
AMFORTH=$(HOME)/work/amforth-$(VERSION)
CORE=$(AMFORTH)/core


# directories
ATMEL="$(HOME)/.wine/drive_c/Program Files (x86)/Atmel/AVR Tools/AvrAssembler2"
# -------------------------
# PROGRAMMER CONFIGURATION
# -------------------------

PROGRAMMER=avrisp2
PORT=usb

AVRDUDE=sudo $(HOME)/.arduino/arduino/tools/avrdude64
AVRDUDE_FLAGS=-q -P $(PORT) -c $(PROGRAMMER)

# -----------------
# ASSEMBLER TO USE
# -----------------

# give to avrasm2.exe path is windows like path format :-)
#AS_INCLUDE=-I $(ATMEL)/Appnotes -I $(CORE)
AS_INCLUDE=-I "C:\\Program Files (x86)\\Atmel\\Avr Tools\\AvrAssembler2\\Appnotes" -I
↪"..\\..\\core"

ASM=wine $(ATMEL)/avrasm2.exe
# flags Specific to avrasm2.exe
#AS_FLAGS=$(AS_INCLUDE) -fI -v0
AS_FLAGS=$(AS_INCLUDE) -fI -v0

#ASM=avra $(AS_FLAGS)

#$(CORE)/devices/$(MCU)
ASM_MCU="..\\..\\core\devices\\$(MCU)"

#---------------------------
# Generic assemble patterns
```

(次のページに続く)

(前のページからの続き)

```
#--------------------------

# Assemble the target
%.hex : %.asm
     @echo "Producing Hexfiles for Arduino $*"
     @$(ASM) $(AS_FLAGS) -I $(ASM_MCU) -e $*.eep.hex -m $*.map -l $*.lst $<
```

- 105 行のコメントの通り、release バージョンでは無いので、 VERSION=code と設定。
- $(AVRDUDE) は -P usb なので hex の書き込みと同様、sudo で実行。
- $(ASM) は wine での実行なので、インクルードパスは Windows が認識できる形で相対パス指定した (/からの指定は NG だったので)。

2.3.1 ビルド

```
$ make uno.hex
```

アセンブルが終了すると、uno.hex, uno.epp.hex, uno.lst, uno.map が生成される。

エラーのあり無しは uno.lst を参照。

注釈: 2013/03/21(JST) のソースでは warnings が 2 つ。いずれも XT_NOOP への前方参照で、ソース末尾まで追ったらちゃんと定義があったので問題無しとした。

この uno.hex, uno.epp.hex をクイックスタートの手順で実機へ転送してください。

2.3.2 Makefile 全文

```
1  # Simple makefile for building the
2  # Arduino amforth vor various targets
3
4  # Examples of usage for Arduino leonardo:
5  #
6  # 1) Assemble the whole flash and eemprom files
7  #     make leonardo.hex
8  #
9  # 2) Backup the current flash & eeprom values
10 #     make leonardo.bak
11 #
12 # 3) Erase the whole MCU Flash
```

(次のページに続く)

(前のページからの続き)

```
#    make leonardo.era
#
# 4) Upload the new firmware using the hex file generated
#    make leonardo
#
# 5) Set the appropiate MCU fuses
#    make leonardo.fuse
#
# 6) Clear files (except backup)
#    make leonardo.clr


SHELL=/bin/bash

##############################
# TARGET DEPENDANT VARIABLES #
##############################

# 1) MCU should be identical to the device
#    Look at the /core/devices/ folder
# 2) PART is the device model passed to avrdude.
# 3) LFUSE, HFUSE, EFUSE are the device-specific fuses
#    there is a useful fuse calc tool at:
#    http://www.engbedded.com/fusecalc/
# ---------------------------------------
# Example fuse settings for 'leonardo'
# Low Fuse LFUSE=0xFF
#  - No Div8 prescaler,
#  - No ouptput Clock,
#  - Low Crystal mode: >=8 MHz + start-up time: 16K CK cycles + 65 ms
# High Fuse HFUSE=0xD9
# - Enable Serial Programming & Downloading
# - Bootsize 2048 words (4096 bytes)
# Extended Fuse EFUSE=0xF9
# - Brown-out detection @ 3.5V
# - no Hardware Boot Vector (=boot at $0000)
# ---------------------------------------

leonardo:         PART=m32u4
leonardo.hex:     MCU=atmega32u4
leonardo.era:     PART=m32u4
leonardo.bak:     PART=m32u4
leonardo.fuse:    PART=m32u4
leonardo.fuse:    LFUSE=0xFF
leonardo.fuse:    HFUSE=0xD9
leonardo.fuse:    EFUSE=0xE9

uno:              PART=m328p
```

(次のページに続く)

(前のページからの続き)

```
uno.hex:           MCU=atmega328p
uno.era:           PART=m328p
uno.bak:           PART=m328p
uno.fuse:          PART=m328p
uno.fuse:          LFUSE=0xFF
uno.fuse:          HFUSE=0xD9
uno.fuse:          EFUSE=0x05

mega128:         PART=m1280
mega128.hex:     MCU=atmega1280
mega128.era:     PART=m1280
mega128.bak:     PART=m1280
mega128.fuse:    PART=m1280
mega128.fuse:    LFUSE=0xFF
mega128.fuse:    HFUSE=0xD9
mega128.fuse:    EFUSE=0xF7

sanguino:          PART=m644p
sanguino.hex:      MCU=atmega644p
sanguino.era:      PART=m644p
sanguino.bak:      PART=m644p
sanguino.fuse:     PART=m644p
sanguino.fuse:     LFUSE=0xFF
sanguino.fuse:     HFUSE=0xF9
sanguino.fuse:     EFUSE=0xFD

duemilanove:       PART=m328p
duemilanove.hex:   MCU=atmega328p
duemilanove.era:   PART=m328p
duemilanove.bak:   PART=m328p
duemilanove.fuse:  PART=m328p
duemilanove.fuse:  LFUSE=0xFF
duemilanove.fuse:  HFUSE=0xD9
duemilanove.fuse:  EFUSE=0x05

diecimila:         PART=m168
diecimila.hex:     MCU=atmega168
diecimila.era:     PART=m168
diecimila.bak:     PART=m168
diecimila.fuse:    PART=m168
diecimila.fuse:    LFUSE=0xFF
diecimila.fuse:    HFUSE=0xDD
diecimila.fuse:    EFUSE=0xF9

# AMFORTH VERSION TO USE
# 'code' for trunk and x.y for the releases (i.e 5.0)
#VERSION=5.0
VERSION=code
```

(次のページに続く)

(前のページからの続き)

```
AMFORTH=$(HOME)/work/amforth-$(VERSION)
CORE=$(AMFORTH)/core


# directories
ATMEL="$(HOME)/.wine/drive_c/Program Files (x86)/Atmel/AVR Tools/AvrAssembler2"
# ------------------------
# PROGRAMMER CONFIGURATION
# ------------------------

PROGRAMMER=avrisp2
PORT=usb

AVRDUDE=sudo $(HOME)/.arduino/arduino/tools/avrdude64
AVRDUDE_FLAGS=-q -P $(PORT) -c $(PROGRAMMER)

# ----------------
# ASSEMBLER TO USE
# ----------------

# give to avrasm2.exe path is windows like path format :-)
#AS_INCLUDE=-I $(ATMEL)/Appnotes -I $(CORE)
AS_INCLUDE=-I "C:\\Program Files (x86)\\Atmel\\Avr Tools\\AvrAssembler2\\Appnotes" -I
↪"..\\..\\core"

ASM=wine $(ATMEL)/avrasm2.exe
# flags Specific to avrasm2.exe
#AS_FLAGS=$(AS_INCLUDE) -fI -v0
AS_FLAGS=$(AS_INCLUDE) -fI -v0

#ASM=avra $(AS_FLAGS)

#$(CORE)/devices/$(MCU)
ASM_MCU="..\\..\\core\devices\\$(MCU)"

#---------------------------
# Generic assemble patterns
#---------------------------

# Assemble the target
%.hex : %.asm
	@echo "Producing Hexfiles for Arduino $*"
	@$(ASM) $(AS_FLAGS) -I $(ASM_MCU) -e $*.eep.hex -m $*.map -l $*.lst $<

# Flash the target
% : %.hex
	@echo "Uploading Hexfiles to Arduino $*"
	$(AVRDUDE) $(AVRDUDE_FLAGS) -p $(PART) -e -U flash:w:$*.hex:i -U eeprom:w:$*.eep.
↪hex:i
```

(次のページに続く)

(前のページからの続き)

```
156
157 # Set the fuse bits
158 %.fuse :
159     @echo "Setting fuses to Arduino $*"
160     $(AVRDUDE) $(AVRDUDE_FLAGS) -p $(PART) -U efuse:w:$(EFUSE):m -U hfuse:w:
    ↪$(HFUSE):m -U lfuse:w:$(LFUSE):m
161
162 # Erase the whole MCU
163 %.era :
164     @echo "Erasing entire Arduino $*"
165     $(AVRDUDE) $(AVRDUDE_FLAGS) -p $(PART) -e
166
167 # Clear assembled & auxilars files
168 %.clr:
169     @echo "Cleaning all aux files"
170     @rm -f $*.hex ; rm -f $*.eep.hex ; rm -f $*.lst ; rm -f $*.map ; rm -f $*.cof ;␣
    ↪rm -f $*.obj
171
172 # Backup arduino Flash & EEPROM files
173 %.bak:
174     @echo "Backup Flash & EEPRON from Arduino $*"
175     $(AVRDUDE) $(AVRDUDE_FLAGS) -p $(PART) -U flash:r:$*.hex.bak:i -U eeprom:r:$*.eep.
    ↪hex.bak:i
176
177 # ------------------------------------------------------------
178
179 GENERIC_DEPENDECIES=*.inc words/*.asm $(CORE)/*.asm $(CORE)/words/*.asm $(CORE)/
    ↪drivers/*.asm
180
181 # Assemble all targets is the default action
182
183 TARGET = leonardo.hex uno.hex duemilanove.hex mega128.hex sanguino.hex diecimila.hex
184
185 %.asm: MCU=atmega328p
186
187 default: $(TARGET)
188
189 $(TARGET) :  $(GENERIC_DEPENDENCIES)  $(CORE)/devices/*/*.asm $(CORE)/devices/*/*.inc
190
191
192 # Cleans everything
193 clean:
194     rm -f *.hex ; rm -f *.eep.hex ; rm -f *.lst ; rm -f *.map ; rm -f *.cof ; rm -f *.
    ↪obj
195
196 # All other rules are target specific and must be typed one by one
197 # as shown in the top.
```

第 3 章

カスタマイズ (プリミティブ追加)

Users Guide から抜粋します。詳細はそちらをご覧下さい。

表 1　関連 inc ファイル

ファイル名	配置領域	備考
dict_appl.inc	フラッシュ下位領域	アドレス下位から上位方向に向かって伸びる
dict_appl.core.inc	NRWW 領域	ブートローダーセクション。uno の場合は 0x3800〜0x3FFF

NRWW 領域は dict_wl.inc を追加した時点でほぼ満杯なので、以後追加するものは dict_appl.inc に追加することになります。

3.1 ワード marker の追加

uno 用の構成ではデフォルトではワードの削除を行うワードを入れていません。つまり、定義しちゃうと消せなくて、最初のバイナリ転送からやり直しになります。フラッシュは書き込み回数制限もありますし、現在では母艦となる PC があるのが当たり前ですのでこの開発スタイルで使いつづけても構いません。

バイナリイメージの再転送無しにワード削除も行いたいと言う場合はワード MARKER を定義します。amforth では伝統的な FORGET と言うワードは準備してなくて、代わりに MARKER というワードを使います。

説明は amforth のサイトの RECIPES/Un-Doing Definitions にある通りですので詳細はそちらを見ていただくとして、ここでいくつか補足します。

説明を見れば分かる通り、dict_wl.inc を追加してアセンブルしてから所定の FORTH ワード定義を include する必要があります。

amforth-code/appl/arduino/dict_appl_core.inc の、;.include "dict_wl.inc" のコメント ; を外して make してください。

```
$ make clean ; make uno.hex
```

make したバイナリイメージを実機に入れてから、

amforth-shell.py で登録するためのファイルを用意します。たとえば extends.frt

```
#include postpone.frt
#inclue marker.frt
marker sketches
```

といった具合です。これを amforth-shell.py で取り込みます。

第 4 章

Indices and tables

- genindex
- modindex
- search