SoC Linux 道場【其ノ参】

クロス・コンパイラのインストールとコンパイル、GDB デバッグ

Ver.13.1

2017 年 8 月 Rev.2

目次

1. はじめに

前回の SoC Linux 道場【其ノ弐】では、クロス・コンパイル環境として使用する Linux® マシンを準備しました。

今回は、プログラムをコンパイルするためのクロス・コンパイラ環境として、 Linaro から提供されている 32-bit ARMv7 Cortex-A 向け Linux GNU クロス・ツールチェーンを Vine Linux 上にインストール・環境設定して使用する方法を解説します。

クロス・コンパイラを使用することで、例えば Helio に搭載されている ARM® Cortex™-A9 プロセッサをターゲットとした実行イメージを PC で生成することができます。

また、簡単な“Hello World”アプリケーション・プログラムをクロス・コンパイルして、Helio に FTP 転送してプログラムを実行します。

さらに、GDB サーバを使用して“Hello World”アプリケーションをリモート・デバッグする方法についても解説します。

尚、この資料の説明で使用している主な開発環境は以下の通りです。

【表 1.1】この資料の説明で使用している主な開発環境

項番	項目	内容
1	ホスト OS	Microsoft Windows 7 Professional sp1 日本語版(64 bit)
2	ゲスト OS	Vine Linux 6.5 x86_64 この資料では、Linux 開発環境として、Vine Linux ディストリビューションを使用します。 詳細については、SoC Linux 道場【其ノ弐】を参照ください。
3	仮想 OS 実行環境	OS を仮想的に実行するための環境です。 この説明では、「VMware Player for Windows」と呼ばれるフリーウェア・ソフトを使用しています。 詳細については、SoC Linux 道場【其ノ弐】を参照ください。
4	クロス・コンパイラ	ターゲット(Helio)向けの実行イメージを生成するためのコンパイラです。 この説明では、Linaro から提供されている 32-bit ARMv7 Cortex-A 向け Linux GNU クロス・ツールチェーン gcc-linaro-4.9.4-2017.01-x86_64_arm-linux-gnueabihf を使用しています。 ■ Linaro ツールチェーンのダウンロード URL https://www.linaro.org/downloads/
5	Python	GDB (arm-linux-gnueabihf-gdb) の拡張により、 Python-2.7 が必要となります。 この説明では、 Python-2.7.13 を使用しています。 ■ Python のダウンロード URL https://www.python.org/downloads/
6	Helio ボード	動作確認でターゲット・ボードとして使用する、 アルテラ Cyclone V SoC を搭載したマクニカ Helio ボードです。 Helio には複数のリビジョンが存在しますが、この資料では Rev1.2 または Rev1.3 を使用して動作確認を行っています。 ■ Helio ボード Rev1.2 http://www.rocketboards.org/foswiki/Documentation/HelioResourcesForRev12 ■ Helio ボード Rev1.3 http://www.rocketboards.org/foswiki/Documentation/HelioResourcesForRev13

2. クロス・コンパイラのインストールとコンパイル、GDB デバッグ

2-1. Linaro ARMv7 Cortex-A 向け Linux GNU クロス・ツールチェーン のダウンロードとインストール

プログラムをコンパイルするためのクロス・コンパイラとしては、アルテラ社提供の SoC 開発ツールに含まれるコンパイラはもちろんあります。

しかし搭載されている CPU は ARM の Cortex-A9 なので、この CPU 用のコンパイラでクロス・コンパイルできます。

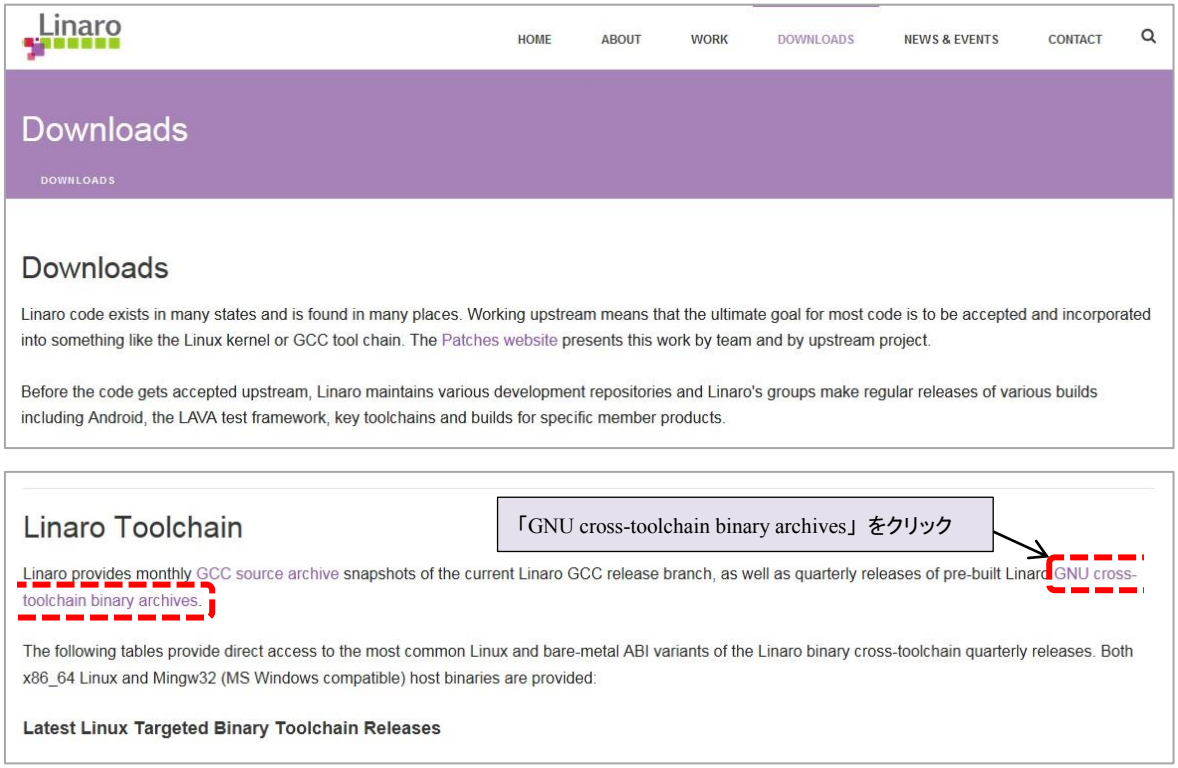
そこで今回は Linaro から提供されている、32-bit ARMv7 Cortex-A 向け Linux GNU クロス・ツールチェーン gcc-linaro-4.9.4-2017.01-x86_64_arm-linux-gnueabihf を使用することにします。

以降にダウンロードとインストールの手順を説明します。

(1) 以下の Linaro のダウンロード・ページに移動します。

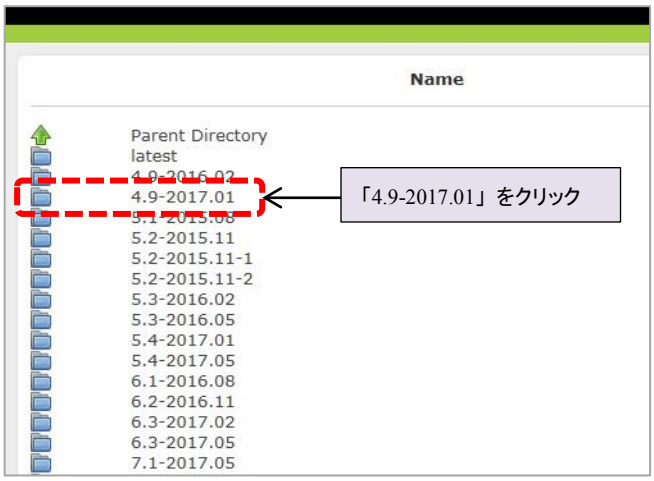
https://www.linaro.org/downloads/

(2) 下図のように、「Linaro Toolchain」の「GNU cross-toolchain binary archives」をクリックします。



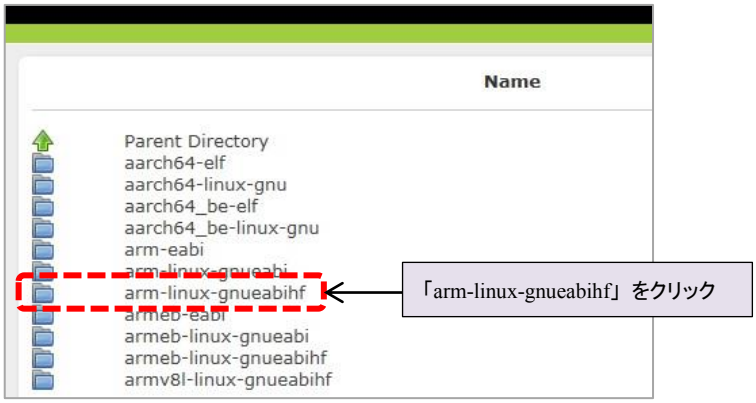
【図 2-1.1】Linaro のダウンロード・ページ

(3) 下図のように、Linaro Releases のダウンロード・ページに移動するので、「4.9-2017.01」をクリックします。



【図 2-1.2】Linaro Releases のダウンロード・ページ

(4)「arm-linux-gnueabihf」をクリックします。

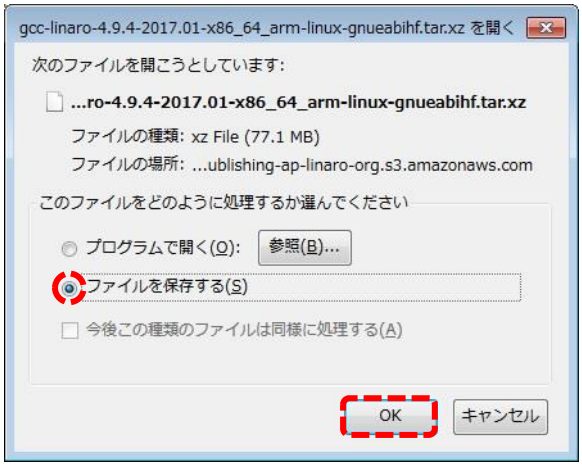


【図 2-1.3】「arm-linux-gnueabihf」をクリック

(5)「gcc-linaro-4.9.4-2017.01-x86_64_arm-linux-gnueabihf.tar.xz」をクリックします。

【図 2-1.4】「gcc-linaro-4.9.4-2017.01-x86_64_arm-linux-gnueabihf.tar.xz」をクリック

(6) 下図のように、ファイルをダウンロードして保存します。



【図 2-1.5】ファイルをダウンロードして保存

(7) 以下の Python のダウンロード・ページに移動し、「Linux/UNIX」をクリックします。

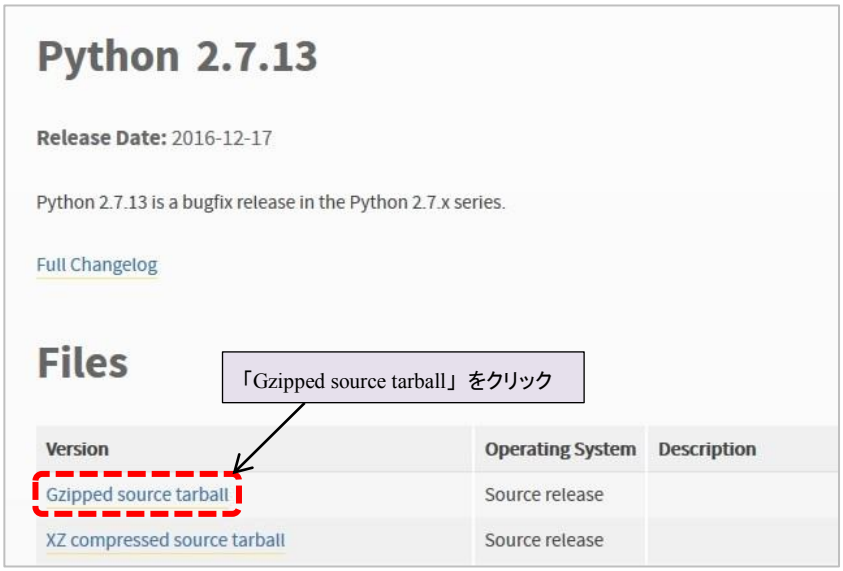
https://www.python.org/downloads/

【図 2-1.6】Python のダウンロード・ページ

(8) Python Source Releases のページから、「Latest Python 2 Release – Python 2.7.13」をクリックします。

【図 2-1.7】Python Source Releases のページ

(9) Python 2.7.13 のページから、「Gzipped source tarball」をクリックします。



【図 2-1.8】Python 2.7.13 のページ

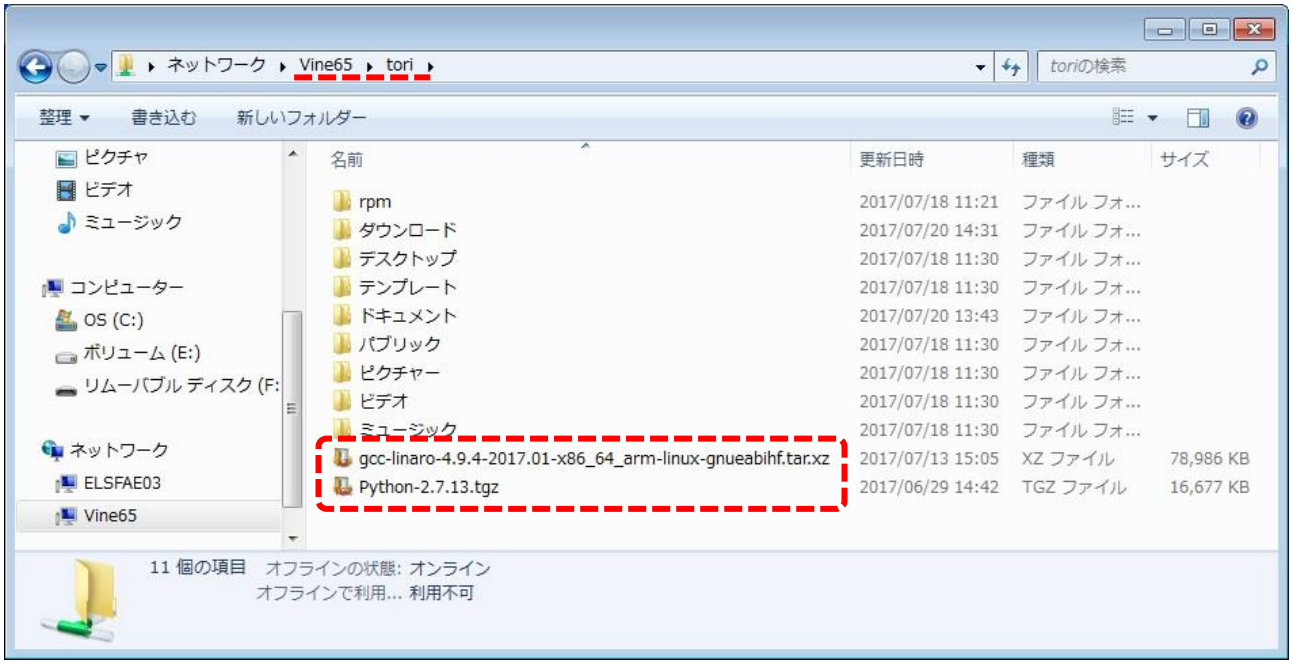
(10) 下図のように、ファイルをダウンロードして保存します。

【図 2-1.9】ファイルをダウンロードして保存

(11) ダウンロードした 2 つのファイル「gcc-linaro-4.9.4-2017.01-x86_64_arm-linux-gnueabihf.tar.xz」、「Python-2.7.13.tgz」を Samba サーバ経由で、Windows 上のエクスプローラから Vine Linux 上（この例では /home/tori ）に運びます。

【注記】

以降の説明は、SoC Linux 道場【其ノ弐】で説明した、Linux マシンと Samba サーバが既に用意・設定されていて、使用できることを前提としています。



【図 2-1.10】Samba サーバ経由で Windows エクスプローラから Vine Linux 上にファイルを運ぶ

(12) **Vine Linux** からルート権限で、以下のコマンドを実行します。

```
[tori@Vine65 ~]$ su -
パスワード:
[root@Vine65 ~]# cd /home/tori
[root@Vine65 tori]# tar -C /opt -Jxvf gcc-linaro-4.9.4-2017.01-x86_64_arm-linux-gnueabihf.tar.xz
```

su – コマンドで root 権限になる
root のパスワード（この例では、toriumi）を入力

(13) インストールが完了すると、

/opt/gcc-linaro-4.9.4-2017.01-x86_64_arm-linux-gnueabihf/

の下にクロス・コンパイラがインストールされます。

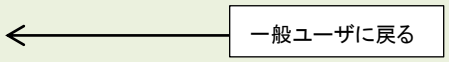
2-2. コマンド・パスの設定

クロス・コンパイラが使用できるように、以降の手順でコマンド・パスに追加します。

(1) 一般ユーザになって、以下の export コマンドを実行してクロス・コンパイラをコマンド・パスに追加し、which コマンドで追加したコマンド・パスを表示します。

うまくコマンドのフルパスが表示できない場合は、export コマンドのパス名などが間違っていないか確認して下さい。

```
[root@Vine65 tori]# cd ~
[root@Vine65 ~]# su - tori
[tori@Vine65 ~]$ export PATH=/opt/gcc-linaro-4.9.4-2017.01-x86_64_arm-linux-gnueabihf/bin/:$PATH
[tori@Vine65 ~]$ which arm-linux-gnueabihf-gcc
/opt/gcc-linaro-4.9.4-2017.01-x86_64_arm-linux-gnueabihf/bin/arm-linux-gnueabihf-gcc
```



(2) コマンド・パスが正しく設定できたら、シェル（コマンド・ライン端末）が起動するたびにこのパス設定が読み込まれて、クロス・コンパイラが使用できるように、ホーム・ディレクトリの下にある .bashrc ファイルを編集してパス定義を追加します。

まず以下のコマンドで、Leafpad を使用して、一般ユーザで自身の .bashrc を開きます。

以下の「~」はコマンドを実行したユーザのホーム・ディレクトリ（この例では、/home/tori）を表します。

```
[tori@Vine65 ~]$ leafpad ~/.bashrc
```

(3) .bashrc ファイルが開いたらファイルの末尾に、

```
export PATH=/opt/gcc-linaro-4.9.4-2017.01-x86_64_arm-linux-gnueabihf/bin/:$PATH
```

を追加します。

【リスト 2-2.1】に .bashrc ファイルの中身を示します。

```
# .bashrc

# User specific aliases and functions

# Source global definitions
if [ -f /etc/bashrc ]; then
 . /etc/bashrc
fi

#stty -ixon

# unlimit stacksize for large aray in user mode
#ulimit -s unlimited

# set aliases
alias ls='ls -F --color=auto'
alias ll='ls -la --color=auto'
alias la='ls -a --color=auto'
alias eng='LANG=C LANGUAGE=C LC_ALL=C'

# user file-creation mask
umask 022

export PATH=/opt/gcc-linaro-4.9.4-2017.01-x86_64_arm-linux-gnueabihf/bin/:$PATH
```

ファイルの末尾にこの行を追加する

【リスト 2-2.1】.bashrc ファイルの中身

上記の追加が完了したら Leafpad で保存します。

【注記】

.bashrc ファイルの編集作業は慎重に行ってください。コマンドやパス名などを打ち間違うと、通常の Linux のコマンドも利かなくなります。

念のため **Leafpad のエディタは終了させずに、上書き保存することをお勧めします。**編集して保存を行うたびに下記 (4) の方法で確認すると良いでしょう。うまく行ってから Leafpad を閉じましょう。

(4) 新たに別のシェル(コマンド・ライン端末)を立ち上げて、以下の which コマンドを実行することで、上記で追加したパスが表示されていれば OK です。

```
[tori@Vine65 ~]$ which arm-linux-gnueabihf-gcc
/opt/gcc-linaro-4.9.4-2017.01-x86_64_arm-linux-gnueabihf/bin/arm-linux-gnueabihf-gcc
```

このパスが表示されていれば OK

【参考】

上記 (4) では、新たに別のシェルを立ち上げることで .bashrc を読み込んで、追加したパスを確認する方法を紹介しましたが、もうひとつの方法としては、現在のシェルから下記の source コマンドを実行して .bashrc を読み込んでから which コマンドでパスを確認する方法もあります。

```
[tori@Vine65 ~]$ source ./.bashrc
[tori@Vine65 ~]$ which arm-linux-gnueabihf-gcc
/opt/gcc-linaro-4.9.4-2017.01-x86_64_arm-linux-gnueabihf/bin/arm-linux-gnueabihf-gcc
```

現在のシェルから source コマンドを実行して .bashrc を再読み込みしてから、which コマンドで確認

(5) 次に root でも同様に .bashrc ファイルの末尾に、export のコマンドを追加しておきます。

```
[tori@Vine65 ~]$ su -
パスワード:
[root@Vine65 ~]# leafpad ~/.bashrc
```

su – コマンドで root 権限になる
root のパスワード(この例では、toriumi)を入力

```
# .bashrc

# User specific aliases and functions

# Source global definitions
if [ -f /etc/bashrc ]; then
 . /etc/bashrc
fi

#stty -ixon

# unlimit stacksize for large aray in user mode
#ulimit -s unlimited

# set aliases
alias ls='ls -F --color=auto'
alias ll='ls -la --color=auto'
alias la='ls -a --color=auto'
alias eng='LANG=C LANGUAGE=C LC_ALL=C'

# user file-creation mask
umask 022

export PATH=/opt/gcc-linaro-4.9.4-2017.01-x86_64_arm-linux-gnueabihf/bin/:$PATH
```

ファイルの末尾にこの行を追加する。

【リスト 2-2.2】.bashrc ファイルの中身

上記の追加が完了したら Leafpad で保存します。

(6) 新たに別のシェル(コマンド・ライン端末)を立ち上げて、以下のコマンドで root 権限になってからパスが通っているかを確かめます。 上記で追加したパスが表示されていれば OK です。

※ または、前出の【参考】で説明した、現在のシェルからの source コマンドによる方法でも確認することができます。

```
[tori@Vine65 ~]$ su -
パスワード:
[root@Vine65 ~]# which arm-linux-gnueabihf-gcc
/opt/gcc-linaro-4.9.4-2017.01-x86_64_arm-linux-gnueabihf/bin/arm-linux-gnueabihf-gcc
```

このパスが表示されていれば OK

これでどこのディレクトリにいても、クロス・コンパイラを起動することができるようになります。

開いている Leafpad があれば閉じてください。

2-3. Hello World のクロス・コンパイル

arm-linux-gnueabihf-gcc のコマンドでクロス・コンパイルしてみましょう。

やはり最初は【リスト 2-3.1】の言わずと知れた「Hello World」のプログラム・ソースから始めましょう。

(1) 以下のようなソース・ファイルを作成します(ファイル名は hello.c)。この作成は、ホスト・マシンである **Vine Linux 上**で Leafpad を使用して行います。

```
[tori@Vine65 ~]$ leafpad hello.c
```

```
#include <stdio.h>

int main(int argc, char *argv[]){
 printf("Hello Helio World!!¥n");
 return 244;
}
```

これらを記述する

【リスト 2-3.1】hello.c ソース・ファイルの中身

でき上がったら、Leafpad でファイルを保存して閉じます。

(2) 以下のコマンドで、hello.c ファイルをクロス・コンパイルします。
コンパイルの結果、出力ファイルとして hello-helio ファイルが生成されます。
エラーが無ければ OK です。エラーになった場合にはソース・ファイルなどを良く見直してください。

```
[tori@Vine65 ~]$ arm-linux-gnueabihf-gcc hello.c -o hello-helio
```

(3) Helio(ターゲット)で実行する前に、file コマンドを使用して生成したファイルを確かめます。

```
[tori@Vine65 ~]$ file hello-helio
hello-helio: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), dynamically linked, interpreter
/lib/ld-linux-armhf.so.3, for GNU/Linux 2.6.31, [illegible]8,
not stripped
```

この出力結果を見ると、ARM CPU で実行可能なことがわかります

(4) FTP サーバを使って hello-helio ファイルを Helio に運びます。
以下のコマンドを **Helio 側** で実行します。

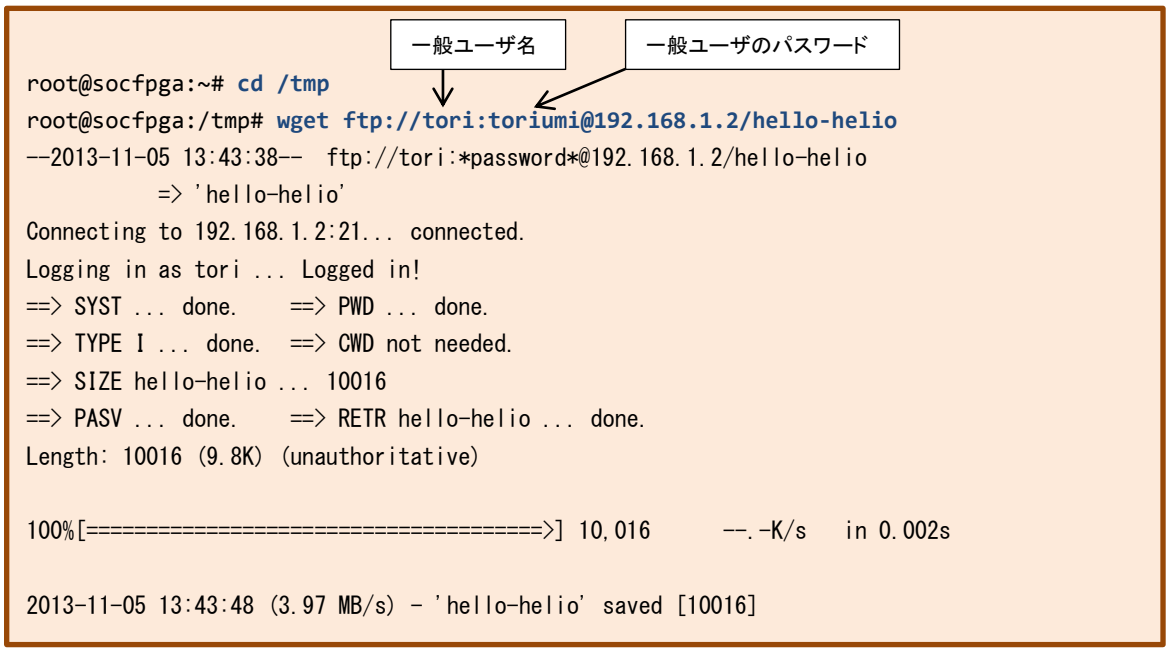
【注記】

以降の説明は、既に SoC Linux 道場【其ノ弐】で説明した、Linux マシンが用意され、DHCP サーバ および FTP サーバ が設定されていて、使用できることを前提としています。

特に、DHCP サーバから Helio に対して IP アドレスが付与されていて、Helio と Vine Linux 間の通信が行える状態にしておいてください。

以降の説明では、Linux マシン(Vine Linux)の IP アドレスを、SoC Linux 道場【其ノ弐】で設定した 192.168.1.2 として説明しています。IP アドレスがわからない場合は、Vine Linux 上から「ifconfig eth0」コマンドを実行することで設定されている IP アドレスを確認することができます。

Helio と Vine Linux 間の通信確認方法としては、Helio 側 から「ping 192.168.1.2」を実行して Vine Linux との接続を確認し、接続できない場合は、Helio 側 から「udhcpc」コマンドを実行して IP アドレスが付与されることを確認してから、再度 ping を試してみてください。ping で接続できたら下記のコマンドを実行してください。




```
root@socfpga:~# cd /tmp
root@socfpga:/tmp# wget ftp://tori:toriumi@192.168.1.2/hello-helio
--2013-11-05 13:43:38--  ftp://tori:*password*@192.168.1.2/hello-helio
           => 'hello-helio'
Connecting to 192.168.1.2:21... connected.
Logging in as tori ... Logged in!
==> SYST ... done.    ==> PWD ... done.
==> TYPE I ... done.  ==> CWD not needed.
==> SIZE hello-helio ... 10016
==> PASV ... done.    ==> RETR hello-helio ... done.
Length: 10016 (9.8K) (unauthoritative)

100%[======================================>] 10,016      --.-K/s   in 0.002s

2013-11-05 13:43:48 (3.97 MB/s) - 'hello-helio' saved [10016]
```

(5) 以下のように、hello-helio ファイルに chmod コマンドで実行権限を与えてから実行します。
実行の結果 “Hello Helio World” が表示されるはずです。

```
root@socfpga:/tmp# chmod 755 hello-helio
root@socfpga:/tmp# ./hello-helio
Hello Helio World
```

上記のように実行できれば、Helio 上でユーザ・プログラムが実行できたことになります。

2-4. GDB によるリモート・デバッグ

Helio 上で動作している Linux では GDB サーバ(gdbserver)を実行することができます。

GDB は Gnu DeBugger の略で、gcc(または互換コンパイラ)でコンパイルしたプログラムのデバッグを支援するデバッガです。

先ほどの Hello Helio World のソースを、GDB を使って操作してみましょう。

(1) まず、GDB でデバッグできるように、**ホスト・マシン(Vine Linux)側**で以下のように「-g」オプションを追加してクロス・コンパイルします(出力ファイルは、hello-helio-gdb)。

「-g」オプションは、ファイルにデバッグ情報を付加します。これがないとデバッグ時に変数名や行番号が表示されません。

```
[tori@Vine65 ~]$ arm-linux-gnueabihf-gcc -g hello.c -o hello-helio-gdb
```

(2) クロス・コンパイルに成功したら、**ターゲット・ボード(Helio)側**で、/tmp ディレクトリから ftp を用いて、ホスト・マシン(この例では、IP アドレス 192.168.1.2)から hello-helio-gdb 実行ファイルを取得します。

```
root@socfpga:/tmp# wget ftp://tori:toriumi@192.168.1.2/hello-helio-gdb
--2013-11-05 07:03:56--  ftp://tori:*password*@192.168.1.2/hello-helio-gdb
           => 'hello-helio-gdb'
Connecting to 192.168.1.2:21... connected.
Logging in as tori ... Logged in!
==> SYST ... done.    ==> PWD ... done.
==> TYPE I ... done.  ==> CWD not needed.
==> SIZE hello-helio-gdb ... 10582
==> PASV ... done.    ==> RETR hello-helio-gdb ... done.
Length: 10582 (10K) (unauthoritative)

100%[======================================>] 10,582      --.-K/s   in 0.002s

2013-11-05 07:03:56 (5.79 MB/s) - 'hello-helio-gdb' saved [10582]
```

(3) 転送したファイルを実行可能にします。

```
root@socfpga:/tmp# chmod 755 hello-helio-gdb
```

(4) 以下のコマンドで、gdbserver を起動します。ポート番号はとりあえず使用されていない 1500 番を使用することにします。

なお、この例では DHCP サーバから付与された Helio の IP アドレスは「192.168.1.237」としています。

```
root@socfpga:/tmp# gdbserver 192.168.1.237:1500 ./hello-helio-gdb
Process ./hello-helio-gdb created; pid = 694
Listening on port 1500
```

DHCP サーバから付与された Helio の IP アドレスを指定
(この例では、192.168.1.237)

(5) 次にホスト側で gdb を立ち上げます。

```
[tori@Vine65 ~]$ arm-linux-gnueabihf-gdb hello-helio-gdb
arm-linux-gnueabihf-gdb: error while loading shared libraries: libpython2.7.so.1.0: cannot open
shared object file: No such file or directory
```

← このエラーが出たら Python 2.7 をインストール

上記のようなエラー・メッセージが現れた場合は、以下の手順で Python 2.7 をインストールしてから、gdb を立ち上げます。

① which コマンドで python のパスを表示します。また、 ldd コマンドで共有ライブラリの依存関係を表示すると、 libpython2.6.so.1.0 から Python 2.6 がインストールされていることがわかります。

```
[tori@Vine65 ~]$ which python
/usr/bin/python
[tori@Vine65 ~]$ ldd /usr/bin/python
 linux-vdso.so.1 (0x00007ffd87b87000)
 libpython2.6.so.1.0 => /usr/lib64/libpython2.6.so.1.0 (0x00007fcaf981d000)
 libpthread.so.0 => /lib64/libpthread.so.0 (0x00007fcaf9600000)
 libdl.so.2 => /lib64/libdl.so.2 (0x00007fcaf93fc000)
 libutil.so.1 => /lib64/libutil.so.1 (0x00007fcaf91f9000)
 libm.so.6 => /lib64/libm.so.6 (0x00007fcaf8eef000)
 libc.so.6 => /lib64/libc.so.6 (0x00007fcaf8b2e000)
 /lib64/ld-linux-x86-64.so.2 (0x00007fcaf9bcf000)
```

libpython2.6.so.1.0 から Python 2.6 がインストールされている

② ダウンロードしておいた Python-2.7.13.tgz ファイルを tar コマンドで解凍してインストールします。

```
[tori@Vine65 ~]$ tar xvzf Python-2.7.13.tgz
[tori@Vine65 ~]$ cd Python-2.7.13
[tori@Vine65 Python-2.7.13]$ ./configure --enable-shared --enable-ipv6 --enable-unicode=ucs4
--with-system-ffi --with-threads --prefix=/usr/lib64/python2.7
[tori@Vine65 Python-2.7.13]$ make
[tori@Vine65 Python-2.7.13]$ sudo make install
[sudo] tori のパスワード:
[tori@Vine65 Python-2.7.13]$ sudo cp -rf /usr/lib64/python2.7/lib/libpython2.7.so.1.0 /usr/lib64/
[tori@Vine65 Python-2.7.13]$ sudo ln -s /usr/lib64/libpython2.7.so.1.0 /usr/lib64/libpython2.7.so
```

③ ldconfig コマンドで共有ライブラリの依存関係情報が格納されたライブラリ・キャッシュを更新します。

```
[tori@Vine65 Python-2.7.13]$ sudo ldconfig
[sudo] tori のパスワード:

[tori@Vine65 Python-2.7.13]$ sudo cp -rf /usr/lib64/python2.7/bin/python* /usr/bin/
[tori@Vine65 Python-2.7.13]$ sudo ln -fs /usr/lib64/python2.7/bin/python2.7 /usr/bin/python2
[tori@Vine65 Python-2.7.13]$ sudo ln -fs /usr/lib64/python2.7/bin/python2.7-config
/usr/bin/python2-config
[tori@Vine65 Python-2.7.13]$ sudo ln -fs /usr/bin/python2 /usr/bin/python
[tori@Vine65 Python-2.7.13]$ sudo ln -fs /usr/bin/python2-config /usr/bin/python-config
```

④ 再度、ldd コマンドで共有ライブラリの依存関係を表示すると、libpython2.7.so.1.0 から Python 2.7 がインストールされたことがわかります。

```
[tori@Vine65 Python-2.7.13]$ ldd /usr/bin/python
 linux-vdso.so.1 (0x00007ffdef79e000)
 libpython2.7.so.1.0 => /usr/lib64/libpython2.7.so.1.0 (0x00007f3a7ef66000)
 libpthread.so.0 => /lib64/libpthread.so.0 (0x00007f3a7ed49000)
 libdl.so.2 => /lib64/libdl.so.2 (0x00007f3a7eb45000)
 libutil.so.1 => /lib64/libutil.so.1 (0x00007f3a7e942000)
 libm.so.6 => /lib64/libm.so.6 (0x00007f3a7e638000)
 libc.so.6 => /lib64/libc.so.6 (0x00007f3a7e277000)
 /lib64/ld-linux-x86-64.so.2 (0x00007f3a7f386000)
```

libpython2.7.so.1.0 から Python 2.7 がインストールされている

⑤ 再度、**ホスト側**で gdb を立ち上げます。

```
[tori@Vine65 Python-2.7.13]$ cd ..
[tori@Vine65 ~]$ arm-linux-gnueabihf-gdb hello-helio-gdb
GNU gdb (Linaro_GDB-2017.01) 7.10.1.20160210-cvs
Copyright (C) 2015 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=x86_64-unknown-linux-gnu --target=arm-linux-gnueabihf".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from hello-helio-gdb...done.
```

(6) ホスト側に gdb のプロンプトが出るので、以下のコマンドを実行します。

```
(gdb) target remote 192.168.1.237:1500
Remote debugging using 192.168.1.237:1500
Reading /lib/ld-linux-armhf.so.3 from remote target...
warning: File transfers from remote targets can be slow. Use "set sysroot" to access files locally instead.
Reading /lib/ld-linux-armhf.so.3 from remote target...
Reading symbols from target:/lib/ld-linux-armhf.so.3...Reading /lib/ld-2.15.so from remote target...
Reading /lib/.debug/ld-2.15.so from remote target...
(no debugging symbols found)...done.
0x76fe0c80 in ?? () from target:/lib/ld-linux-armhf.so.3
```

DHCP サーバから付与された Helio の IP アドレスを指定（この例では、192.168.1.237）

(7) ここまで実行すると、**ターゲット側**に以下のメッセージが出力されます。

```
Remote debugging from host 192.168.1.2
```

(8) 次に**ホスト側**で以下のコマンドを実行します。b コマンドで main 関数内の最初の命令がある 4 行目にブレイク・ポイントを設定し、cont コマンドでブレイク・ポイントを設定した 4 行目まで実行します。

```
(gdb) list
1  #include <stdio.h>
2
3  int main(int argc, char *argv[]){
4      printf("Hello Helio World!!¥n");
5      return 244;
6  }
(gdb) b main
Breakpoint 1 at 0x103da: file hello.c, line 4.
(gdb) cont
Continuing.
Reading /lib/libc.so.6 from remote target...
Reading /lib/libc-2.15.so from remote target...
Reading /lib/.debug/libc-2.15.so from remote target...

Breakpoint 1, main (argc=1, argv=0x7efffe14) at hello.c:4
4      printf("Hello Helio World!!¥n");
```

4 行目まで実行

(9) ここで以下のように next コマンドで、1 行(printf 命令の行)だけ実行します。

```
(gdb) next
5    return 244;
```

(10) この時**ターゲット側**では、以下のように printf の出力が現れます。

```
Hello Helio Wrold!!
```

(11) 最後に cont コマンドですべて実行して、quit コマンドで gdb を抜けます。

```
(gdb) cont
Continuing.
[Inferior 1 (process 694) exited with code 0364]
(gdb) quit
[tori@Vine65 ~]$
```

(12) この時、以下のようにターゲット側も gdbserver を終了しています。

```
Child exited with status 244
GDBserver exiting
root@socfpga:/tmp#
```

このように、GNU デバッガも使うことができます。

次回【其ノ四】では、GNU アプリケーションのコンパイルとインストールについて解説します。

Busybox と呼ばれる、主に組み込み Linux で使われるオープン・ソースの“コマンド集”アプリケーションと、 thttpd という組み込み機器では割と使用されている Web サーバ・ソフトをコンパイルしてインストールします。

改版履歴

Revision	年月	概要
1	2015 年 1 月	新規作成
2	2017 年 8 月	① Sourcery CodeBench Lite Edition for ARM GNU/Linux の配布終了に伴い、クロス・コンパイル環境として Linaro Toolchain を使用する説明に変更 ② ゲスト OS を Vine Linux 6.2.1 i686 から Vine Linux 6.5 x86_64 に変更