OpenRTM-aist-020 を x86 アーキテクチャ以外の機器へインストール

２．ARM 系 CPU ボードへのインストール

２．１．本体に関する情報

製品名：Armadillo-240（アットマークテクノ社製）
CPU コア：ARM920T (200MHz)
SD-RAM：64MB
Flash Memory：8MB
100BASE-T/100BASE-TX/10BASE-T×1 ポート
USB2.0×2 ポート
アナログ VGA ポート
16 ビット GPIO
シリアルポート×2
OS：Armadillo Linux（カーネル 2.6 ベース）
価格：web 直販で税込み 30,000 円（開発キット，ボックス込み）
その他：ラインアップとして，Armadillo-210～230 あり．

２．２．作業の流れ

①開発環境の整備
②必要なパッケージをインストール
③OpenRTM-aist をインストール
④動作確認

２．３．開発環境の整備

Armadillo-240 ではクロス開発環境下でソフトウェア等を開発するので，別途開発用ホスト PC が必要となる．

開発用 PC に Linux がインストールされていれば，そのまま後述の開発ツールをインストールすればよいが，あとでいろいろ環境変数やシンボリックリンク等を改変するため，場合によっては他のアプリケーションに悪影響を及ぼす可能性がある．そこで，Windows に coLinux をインストールし，開発は coLinux 上で行った方が専用環境を構築しやすく，また作業中失敗したときのリカバーがしやすいということもあり（失敗直後のイメージファイルを消して，デフォルトのイメージを解凍しなおせばよいので），開発環境のベースとしてお奨めである．coLInux は製品付属の CD-ROM に収録されているので，マニュアルに記載されている手順に従ってインストールを行う．以降，coLinux 上での運用を前提に話を進める．

まず，開発用 PC に OpenRTM-aist に必要なパッケージ（ACE，boost，omniORB 関連．i686 用）をインストールする．Debian のサイトよりパッケージをダウンロードするよりも，apt コマンドを利用するした方が，依存パッケージも同時にインストールされるので，作業が楽である．

次に，付属の CD-ROM に収録されている（またはメーカーサイトのサポートページから

入手）クロス開発ツールすべてをインストールする．付属 CD-ROM に収録されていた coLinux のイメージが Debian3.1/Sarge ベースであるため，拡張子が deb のパッケージをインストールする．この場合，OpenRTM-aist に必要なパッケージ（ACE，boost，omniORB）をインストールする際，メーカーから用意された開発ツールのバージョン（3.4 系）が新しいため，バージョン依存性の問題からこれらパッケージがインストールできないという問題を抱えている．そこで，以下のパッケージについては同時に古いバージョン（3.3 系）をインストールする．これらのパッケージは http://zigzag.lvk.cs.msu.su/~nikita/debian/sarge/（2006/9/8 現在は存在していた）からダウンロード可能であるが，見つからない場合は検索サイトでサーチするとよい．

- cpp-3.3-arm-linux_3.3.5-13_i386.deb
- g++-3.3-arm-linux_3.3.5-13_i386.deb
- gcc-3.3-arm-linux_3.3.5-13_i386.deb
- libstdc++5-arm-cross_3.3.5-13_all.deb
- libstdc++5-3.3-pic-arm-cross_3.3.5-13_all.deb
- libstdc++5-3.3-dev-arm-cross_3.3.5-13_all.deb

なお，異なるバージョンのパッケージを複数同時に入れた場合，Debian 系では以下のコマンドを用いることにより，適宜バージョン切り替えを行うことが可能である．カーネルを再構築する場合は最新版（3.4）でないとコンパイルできないため，以下のコマンドですべての開発ツールのバージョンを 3.4 に切り替えること．

```
# update-alternatives --config （開発ツール名）
（例）# update-alternatives --config arm-linux-gcc
```

※root 権限でしか実行できないので，その前に su コマンドを実行すること．

続いて，ターゲット機用の ACE，boost，python 関連パッケージを用意するのだが，ホスト PC で開発を行うため，クロス開発用に変換する必要がある．

変換方法は，dpkg-cross コマンドを使って，以下の通りに行う．（software マニュアル p.7 を参照のこと）

```
# dpkg-cross --build --arch arm（パッケージ名）
（例）dpkg-cross --build --arch arm libace5.4_5.4.2.1.0-4_arm.deb
```

※dpkg-cross がインストールされてない場合は，apt-get コマンドを使ってインストールしておくこと．また，root 権限でないと変換がうまくいかないようである．

また，クロス開発用に変換するパッケージは以下の通りである．

＜ace関連＞
・libace5.4_5.4.2.1.0-4_arm.deb
・libace-dev_5.4.2.1.0-4_arm.deb
＜boost関連＞
・libboost-dev_1.32.0-6_arm.deb
・libboost-regex1.32.0_1.32.0-6_arm.deb
・libboost-regex-dev_1.32.0-6_arm.deb
（その他，好みに応じてboost関連の他のパッケージをインストールしてもよい．ただし，devパッケージも忘れずに変換，インストールすること）
＜python関連＞
・python2.3_2.3.5-3sarge1_arm.deb
・python2.3-dev_2.3.5-3sarge1_arm.deb

続いて，omniORB4 関連のクロス開発パッケージをインストールするのだが，その際，pkg-config パッケージのインストールを求められる．しかし，このパッケージは dpkg-cross ではクロス開発パッケージに変換できない．そこで，ダミーのクロス用パッケージ作成スクリプト ”mkXdummy ” を使ってダミーパッケージを作成し，そこで生成されたパッケージをインストールする．なお，mkXdummy は，アットマークテクノのホームページの以下のサイトより入手可能である．
http://download.atmark-techno.com/misc/softwaredesign/chapter5/
pkg-config ダミーパッケージの作成およびインストール方法は以下の通りである [1)]．

1)i386用equivs，pkg-configをapt-getコマンドを使ってインストール
2)mkXdummyを入手
3)$ chmod 755 mkXdummy でファイルアクセス権を変更
4)$ ./mkXdummy pkg-config と実行
5)pkg-config-arm-cross_1.0_all.debができていることを確認
6)suコマンドでroot権限に変更
7)# dpkg -i pkg-config-arm-cross_1.0_all.deb を実行

pkg-config パッケージが無事インストールできたら，以下のパッケージを入手し，クロス開発パッケージに変換してからインストールを行う．

※以下の順番でパッケージをクロス用パッケージに変換→インストールすること
・libomnithread3_4.0.6-1_arm.deb
・libomnithread3-dev_4.0.6-1_arm.deb
・libomniorb4_4.0.6-1_arm.deb
・libomniorb4-dev_4.0.6-1_arm.deb
・libcos4_4.0.6-1_arm.deb
・libcos4-dev_4.0.6-1_arm.deb

以上のパッケージのインストールが成功すると，/usr/arm-linux というディレクトリが作成され，lib 以下にクロス開発用ライブラリファイルが展開される．また，ツール関連は /usr/bin に展開され，例えば gcc であれば"arm-linux-gcc"と，ホスト用との区別をつけるため，”arm-linux”という接頭子がつく．

omniNames，omniidl などのツール類は，クロス開発パッケージに変換せず，ホスト PC にインストールされたパッケージをそのまま用いる．その際，/usr/arm-linux/bin ディレクトリにおいて，以下のシンボリックリンクを作成する．

```
$ su -
# ln -s /usr/bin/omniNames omniNames
# ln -s /usr/bin/omnicpp omnicpp
# ln -s /usr/bin/omniidl omniidl
# ln -s /usr/bin/omniidlrun.py omniidlrun.py
```

これは，configure スクリプト実行時に omniorb 関連ファイルを検索する際，ライブラリの prefix(/usr/arm-linux/lib)とツール関連の prefix(/usr/bin)が異なると，omniorb をうまく認識してくれないためである．

最後に，export コマンドを使って環境変数を以下のように設定する（root 権限で実行）．

```
# export CC=/usr/bin/arm-linux-gcc
# export CXX=/usr/bin/arm-linux-g++
# export LD=/usr/bin/arm-linux-ld
# export AR=/usr/bin/arm-linux-ar
# export AS=/usr/bin/arm-linux-as
# export RANLIB=/usr/bin/arm-linux-ranlib
# export STRIP=/usr/bin/arm-linux-strip
# export PKG_CONFIG_PATH=/usr/arm-linux/lib/pkgconfig
# export PYTHONPATH=/usr/arm-linux/lib:$PYTHONPATH
```

今後 RT-Component を独自に作成する際，以上の環境変数の設定が再び必要となるため，これらのコマンドをスクリプト化するとよい．

２．４．OpenRTM-aist のインストール

OpenRTM-aist の圧縮ファイルは，クロス開発用ライブラリの prefix と揃えるため，/usr/arm-linux 以下に展開する．続いて，「玄箱 HG 編」と同様 configure.ac ファイルの修正を行う．この修正を行わないと，omniorb オプションの認識が正当にできない．

続いて，configure スクリプトを実行する．インテル i386 系 PC では--with-omniorb と --with-python とだけ設定すればたいていインストールができるが，今回はクロス開発用に以下の通りにオプションを多数つけて実行する必要がある．

```
$ ./configure  --build=i686-pc-linux-gnu ¥
  --host=arm-linux ¥
  --with-ace-includes  =/usr/arm-linux/include ¥
  --with-ace-lib=/usr/arm-linux/lib ¥
  --with-boost-includes=/usr/arm-linux/include ¥
  --with-boost-lib=/usr/arm-linux/lib ¥
  --with-python=/usr/arm-linux/include ¥
  --with-omniorb=/usr/arm-linux
```

スクリプトの実行がうまくいくと，Makefile が生成されるので，あとはマニュアルと同様の手順で

```
$ make
$ su -
# make  install
```

を実行すればよい.

２．５．サンプルコンポーネントの動作確認方法

コンパイルに成功すると，example ディレクトリ以下に複数のサンプルファイルが生成される．このうち ”SimpleIO ”ディレクトリ以下にできた，”ConsoleInComp ”で動作確認を行う．atmark-dist を使ってターゲット機器にインストールしてもよいが，うまく動作できるかどうかまだ不明なうちから機器に書き込むのは面倒で，また書き込み回数にも限りがあるため，USB メモリを利用して動作確認を行う.

まず，Armadillo-240 の出荷状態では Recover イメージがインストールされており，USB メモリを挿すと web 上で閲覧できる便利な機能が盛り込まれているが，逆に任意のディレクトリにマウントできないようなので，あえて Base イメージに入れ替えて，この機能を消去する．Base イメージだといろいろカスタマイズができ，Telnet や ftp も使えるので，こちらの方が何かと都合がよい.

次にターゲット機器とホスト PC を RS-232 ケーブルを使って接続し，ターゲット機器のジャンパピンがすべて外れていることを確認し，任意のターミナルソフト（TeraTerm がお薦め）上でターゲットに root ログインする.

次に USB メモリを用意し（あらかじめすべて空にしておいた方がよい），ホスト PC の /usr/arm-linux/lib，/usr/arm-linux/bin，/usr/arm-linux/OpenRTM-aist/examples/SimpleIO をフォルダごとコピーする. その後 USB メモリ内に以下のディレクトリが作成されているかどうか確認する.

```
lib  bin  SimpleIO
```

続いてターゲット機器側での USB メモリのマウントポイントであるが，今回は既存の /mnt ディレクトリではなく，mkdir コマンドを使って/usr/arm-linux というディレクトリを作成し，そこをマウントポイントとする．各ファイルがコピーされた USB メモリをターゲット機器に挿入し，以下のコマンドでマウントを行う．

```
# mount -t vfat /dev/sda1 /usr/arm-linux
（※マウント先は各自の好きなところで構いません）
```

その後，ターゲット機器上でライブラリファイルのサーチパスを以下の通りに設定する

```
# export LD_LIBRARY_PATH=$LD_LIBRARY_PATH: /usr/arm-linux/lib
（※USBメモリのマウントポイントが/mntの場合は，/mnt/libとなる）
```

さらに，ターゲット機器では強固なファイヤウォールが設定されているので，以下のコマンドですべてのポート番号をスルーさせるよう設定する．なお，RT コンポーネントを動作させるためのポート番号が既に決めてあるのであれば，その番号だけ通すよう設定してもよい．

```
# iptables -F INPUT
# iptables -P INPUT ACCEPT
# iptables -F OUTPUT
# iptables -P OUTPUT ACCEPT
```

次に，ホスト PC 側で rtm-naming を立ち上げておく．なお，ホスト PC に OpenRTM-aist がインストールされていなければ，別途 OpenRTM-aist がインストールされた PC を用意する必要がある．その後ターゲット機器用の conf ファイルを作成する．ファイルの内容はマニュアル通り，以下の通りにする．

```
NameServer （rtm-namingを立ち上げたPC名またはIPアドレス）:ポート番号
（例）NameServer 192.168.1.5:9876
```

ここで，NameServer の ”S ” が大文字でないと，起動時に conf ファイルをうまく読み込めず，RtcManager がエラーを起こすので，注意を要する．

最後に，ConsoleInComp を以下のように立ち上げてみる．

```
# ./ConsoleInComp -f （confファイル名）
（例）confファイルがrtcXXX.confであれば
```

```
#   ./ConsoleInComp  -f  rtcXXX.conf
```

OpenRTM-aist が使える PC で rtc-link を立ち上げてみて, ターゲット機器側で立ち上げた ConsoleInComp のアイコンが現れたら成功である.

２．６．atmark-dist を使ってターゲット機器にインストールする

以下の手順で RT-Component をターゲット機器にインストールできる.

２．６．１．atmark-dist のデフォルトイメージ（Base 版）を作成

・一般ユーザでログインする.

・一度 su コマンドで root 特権に変更し，arm-linux-gcc,arm-linux-g++のバージョンを **3.4** に切り替える.

・root 特権から抜ける

・あとは software マニュアル p.30～33 の操作に従って，Base 版デフォルトイメージを作成する.

２．６．２．開発 PC で作成した RT-Component 入りイメージを作成

・一般ユーザでログインする.

・./atmark-distXXXXXX（XXXXXX にメーカでイメージファイルを作成した日付が入る. 2006 年 10 月現在での最新版では 060801）ディレクトリ以下に romfs ディレクトリが生成されているかどうかを確認する．イメージファイルを作成する際，実は romfs ディレクトリ以下の内容が圧縮・統合されて romfs.img.gz というユーザランドイメージファイルになり，これを本体へ転送すると，本体起動時に FlashROM へ自動展開されるという仕組みになっている.

・romfs/bin ディレクトリに開発用 PC で作成した RT-Component 実行ファイル(語尾に Comp がつく実行ファイル）のみをコピーする.

・./atmark-distXXXXXX ディレクトリに入り，make menuconfig を実行して，kernel，ユーザランドで変更したい項目があれば変更しておく.

・make を実行する．すると，RT-Component 起動に必要なライブラリファイルを自動判別し，必要なファイルを romfs/lib ディレクトリへコピーしてくれる.

・./atmark-distXXXXXX/images ディレクトリ以下にイメージファイルが生成されているかどうか確認する.

・できあがったイメージファイルを software マニュアル p.17～21 の手順に従って本体へ転送する．この際，本体 JP2 ピンをショートさせておくことを忘れないこと.

※注意

・イメージファイル生成前に ”make dep ”を実行するよう software マニュアル p.33 に記載されているが，kernel2.6 以降では行わなくてもよい．ただ，実行しても警告が出るだけで特に害はない.

・試しに ConsoleInComp を本体 ROM に書き込んだところ，FlashROM 容量が残り数%となった．RT-Component 起動時に必要なライブラリファイルの容量が大きく，FlashROM を逼

迫させているようある．したがって，追加で RT-Component を本体に転送する際には FlashROM の残り容量に気をつけること．または NAND フラッシュメモリの増設を検討すること．

２．７．その他

(a)その他の Armadillo では？

おそらく Armadillo-9 なら CF や HDD が使えるためにセルフ開発環境も構築できるということもあり，「玄箱 HG 編」とほぼ同様の手順でインストールできると思われる．

USB インタフェースを持たない Armadillo-2X0 シリーズについては，別の PC でライブラリや実行ファイルを展開し，nfs マウントを有効に使えば動作確認はできるであろう．また，220 以降であればオプションの NAND メモリを増設し，そこへ一連のファイルを展開するという手段もある．ただ，Armadillo-240 のメモリ容量が Armadillo9 と同じ 64MByte なのに対し，その他は 32MByte なので，メモリ容量不足で動かない可能性もあり，一概に断言できない（どなたか検証していただけるとありがたいです）．

(b)ターゲット機器で rtm-naming を立ち上げるには

rtm-naming スクリプトを用いず, omniNames を直接立ち上げる. まず, arm 用の omniNames パッケージを入手し，以下のコマンドで任意のディレクトリに解凍する．

```
# dpkg -x omniorb4-nameserver_4.0.6-1_arm.deb （任意の解凍先ディレクトリ）
```

次に解凍した実行ファイルを./atmark-distXXXXX/romfs/bin ディレクトリにコピーする．その後イメージファイルを作成し，できたユーザランドイメージファイルを本体へ転送しておくと，本体の/bin ディレクトリに ” omniNames ” 実行ファイルが現れるはずである．

その後は以下のコマンドでターゲット機器側のネーミングサーバを立ち上げればよい．

```
# omniNames -start（ポート番号）&
（例）omniNames -start 9876 &
```

この場合，最後に&をつけ忘れないこと．また，omniNames を何らかの理由で再起動させる場合は，/var/log/以下にできた ” omniNames-XXXX.log ” と ” omniNames-XXXX.bak ” （XXXXX には自身のホストネームが入る）を削除しておくこと．

[参考文献]

(1)Software Design2005 年 12 月号「【特集】組み込み Linux 実践講座 組み込みボード＋Linux で作る My ガジェット」，技術評論社

※アットマークテクノ社の Armadillo 開発担当者が執筆してます．今回この記事が非常に役立ちました．しかし，出版元でも完売しているそうなので，入手は困難かもしれません．