組込み機器用RTコンポーネント開発環境
ATDE for OpenRTM-aist

産業技術総合研究所 ○安藤慶昭

ATDE for OpenRTM-aist: RT-Component development environment for embedded devices

*Noriaki ANDO, AIST

Abstract— RT-Component development for embedded devices is complex and difficult because of its hardware and software constraints and embedded system specific knowledge that are required to develop. ATDE for OpenRTM-aist provides various tools and system configurations for target board Armadillo series. This development environment promotes use of embedded device based RT-Component for practical applications.

Key Words: RT-Middleware, embedded systems, development environment

1. はじめに

RT ミドルウエアをより実用的なシステムに導入する際、スペース、価格、安定性、消費電力など様々な制約により、組込みデバイスや組込みボードを利用する必要性に迫られるケースがある。一方、著者らが開発する RT ミドルウエア OpenRTM-aist[1, 2] は通信に CORBA[3] を利用していることから、組込み機器への導入は容易ではない。

今回開発した ATDE for OpenRTM-aist は、Linux ベースの組込みボード Armadillo をターゲットとしたクロス開発において、より開発を容易にするツール群を提供することで、組込み開発の効率化を促進するものである。

開発に当たりコンパイルから実行・テストに至るまでの一連の開発作業を見直し、組込み機器開発経験はあまりないが、PC 上での開発経験がある開発者にとって利用しやすいツールとなるよう工夫した。なお、ベースとなった開発環境は組込み Linux 開発で一般に用いられる μClinux-dist[6] であり、今回開発したツールなどは他の組込み Linux 開発にも容易に応用できるものとなっている。

2. 組込みシステムと RT ミドルウエア

RT ルーム [7] のような既存の機器や設備を RT コンポーネント化する場合や、分散センサシステムのような多数の小型のデバイスをネットワーク化する場合は、様々な制約から、組込みデバイスに RT コンポーネントを搭載する必要がある。

RT ミドルウエア、RT コンポーネントを組込み機器上に搭載するため、これまで様々な試みが行われてきた。RTC-Lite[4] は、PIC ベースのボード上の簡易 RTC フレームワークと PC 上のプロキシコンポーネントを利用することで、PIC のような非常に小型の CPU であっても RT ミドルウエアのネットワークに参加できることを示した。また、ARM を搭載した組込み Linux ボードを利用することで、同一ソースのコンポーネントをクロスコンパイルするのみで、そのまま組込み CPU 上で動作させたものもある [5]。組込み向け OS として日本で人気のある μITRON に、OpenRTM-aist を移植することにより Linux を実行することが難しいデバイスにも RT ミドルウエアが搭載可能であることが示された [8]。また μITRON の後継とも言われる T-Kernel に移植した例もある [9]。

CANopen は CAN バス上のプロトコル、および RTC のモデルに類似したデバイスプロファイルを含む標準である。RTC-CANopen[10] は両者のマッピングを定めることで RTC-CANopen デバイスをあたかも RTC のように扱うことができる。なお、RTC-CANopen 自体もすでに CiA(CAN in Automation) において標準化されている。また、miniRTC[11], microRTC[12] では通信路に CAN や Zigbee を用い、小型のマイクロコントローラ上に RTC フレームワークを実装している。AVR マイコンを用いたオープンハードである Arduino 上に軽量の RTC フレームワークを搭載し、PC 上のプロキシと組み合わせた RT-no[13] なども開発された。

このように、RT ミドルウエアを組込みデバイス上で動作させる需要は非常に多く、移植も数多く行われてきた。一方で、組込みデバイス上で RTC を動作させることは、特に組込み機器開発に精通した者でなければ難しいという問題があった。

2·1 クロス開発

Armadillo をはじめとする組込み CPU ボードは特定の用途向けのアプリケーションのみを実行することを前提としているため、一般的な PC とは異なり十分な容量のハードディスクやメモリを搭載していない。 従って、アプリケーション・プログラムの開発もターゲットボード上で行うのではなく、PC 上の開発環境内で行い、コンパイルしたバイナリを何らかの方法でターゲットに転送して実行する。

一般的な PC の CPU は Intel の x86 系アーキテクチャであるが、組込み CPU ボードのアーキテクチャは ARM, SH, PowerPC 等、PC の CPU とは異なるケースが多い。x86 用の実行ファイルは ARM その他の CPU 上では実行することはできず、その逆も不可能である。PC 上の開発環境では ARM 等 PC とは異なるアーキテクチャ用の実行ファイルをコンパイルできるコンパイラを動作させ、アプリケーションを開発する。こうした開発方法をクロス開発と呼ぶ (図 1)。

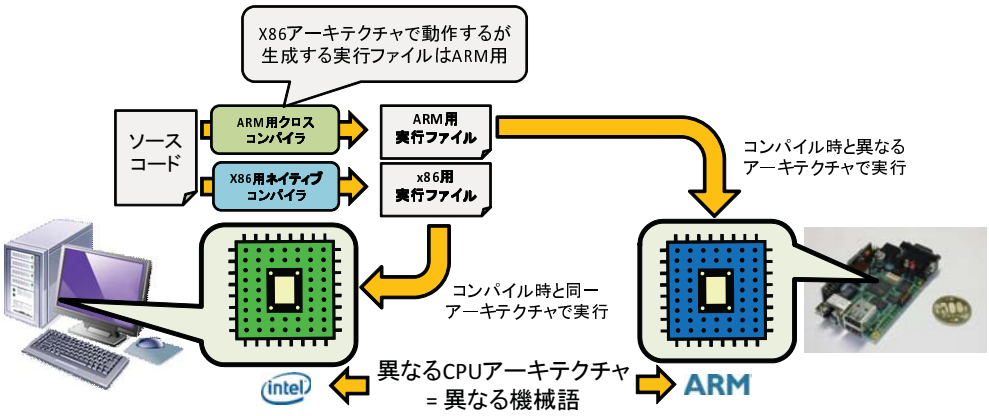

Fig.1 Cross development between host computer and target computer.

2·2 開発・運用の実際

クロス開発では、ホスト PC 上で実行ファイルをコンパイル後、ターゲットに実行ファイルを移して実行・テストを行う必要がある。ターゲットと PC をシリアルケーブル等で接続して、ターゲット上のフラッシュ ROM などに書き込む方法が一般的である。シリアルケーブル経由でのフラッシュROM への書き込みは一般に遅く、また Linux の場合、対象とする実行ファイル以外の他の多数のファイルを含む大きなファイルシステムイメージを書き込むため時間がかかる[1]。こうした作業が組込みシステムの開発効率の低下につながっている。開発効率の向上には、このコンパイル、実行、テスト、修正のサイクルをいかに早めるかが重要なカギとなる。

また、開発・テストが終了した RT コンポーネントを、実際に運用システムに配置 (デプロイ) するプロセスについても考慮する必要がある。組込みボードは一般にキーボードやモニタを持たないため、設置場所に行き PC を接続して設定する方法が考えられる。しかし、すでに設置済みのデバイスには物理的にアクセスが難しいケースも考えられ、ネットワーク経由ですべて設定可能であることが望ましい。また、分散センサシステムなどでは、セットアップするデバイスの数が数十個単位に上る場合もあるため、一つ一つのデバイスを開発ターゲットのようにセットアップすることは現実的ではない。したがって、多数のシステムおよび RT コンポーネントの設定を容易に行えることが望ましい。

3. Armadillo と開発環境

3·1 Armadillo

Armadillo はアットマークテクノ社から発売されている ARM CPU を搭載した名刺サイズの小型 CPU ボードである。μCLinux ではなく通常の Linux を搭載可能であり、内蔵のフラッシュメモリからブート可能である。

ATDE for OpenRTM-aist は Armadillo 200 および 400 シリーズをターゲット CPU ボードとする。Armadillo-200 シリーズおよび 400 シリーズの中で、OpenRTM-aist を動作させるのに適したものとして特に Armadillo-240 と Armadillo-440 をターゲットとした。それぞれ外観を図 2 に、仕様を表 1 に示す。

[1]Armadillo では、カーネルイメージの書き込みに数分、ユーザランドイメージの書き込みに十数分程度要する。

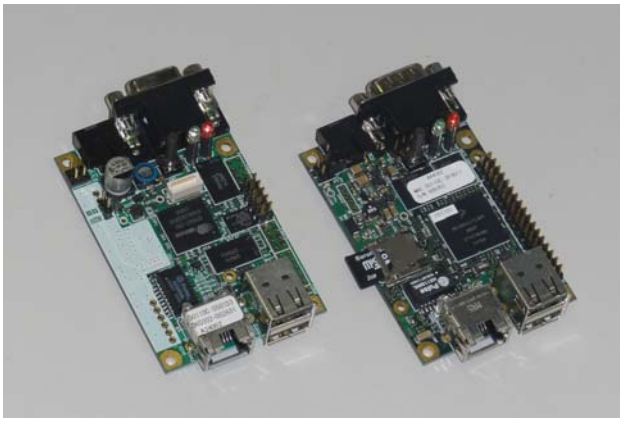

Fig.2 Armadillo-240 (left) and Armadillo-440 (right).

3·2 ATDE

ATDE (Atmark Techno Development Environment) は、 Debian GNU/Linux をベースとした Armadillo シリーズ用のクロス開発環境である。アットマークテクノ社から VMware 上で動作する VM イメージとして配布されている。VMware は Windows, Linux, Mac OS X 上で動作する仮想 PC 環境用のソフトウエアであり、無料版の VMware Player は現在、Windows 版、Linux 版が利用可能である。

Armadillo-200 シリーズ用には ATDE2、Armadillo-400 シリーズには ATDE3 といった異なるバージョンが用意されている。ATDE2 for OpenRTM-aist および ATDE3 for OpenRTM-aist は Armadillo 上で動作する RT コンポーネントを開発するため、ATDE2 および ATDE3 に OpenRTM-aist および依存ライブラリ、各種ツールを事前にインストールした開発環境として、OpenRTM-aist 公式サイト [2] で配布されている。

3·3 Atmark Dist

Atmark Dist はアットマークテクノ社がリリースしている Armadillo 用のカーネルおよびユーザランドの開発環境である。組込み Linux では、メモリやディスクの容量が限られているため、アプリケーションに合わせて必要最小限のシステム構成にする必要がある。これを実現するのが、μClinux で利用される μClinux-dist であり、Atmark Dist はこれを基にしている。

Atmark Dist および μClinux-dist では、デバイス固

Table 1 Armadillo-240,440 specification.

	Armadillo-240	Armadiilo-440
プロセッサ	Cirrus Logic EP9307	Freescale i.MX257
CPU コア	ARM920T	ARM926EJ-S
CPU コアクロック	200MHz	400MHz
バスクロック	100MHz	133MHz
RAM	64MB (SDRAM)	128MB (LPDDR SDRAM)
フラッシュメモリ	8MB (NOR 型)	32MB (NOR 型)
LAN(Ethernet)	RJ45 × 1 (100BASE-TX/10BASE-T)	
シリアル (UART)	3.3V CMOS × 2	RS232C × 1, 3.3V CMOS × 2
GPIO	16bit (コネクタ非搭載)	18bit (最大 24bit) LCD I/F
USB	USB 2.0(Host) × 2 (Full Speed)	
電源電圧	DC5V ± 5 または PoE(Type A/B) 対応	DC3.1〜5.25V
消費電力	1.5W (Typ.)	1.2W (Typ.)

有のカーネルの設定のほかに、必要に応じてカスタマイズできるユーザランドを作成する機能がある。基本コマンドのほとんどはディスクサイズ節約のため BusyBox を用い、組み込むサービスおよび起動スクリプトもメニューから選択したもののみが自動的に組み込まれるようになっている。組込み機器では一般的に電源投入後ただちにシステムが起動し、必要なアプリケーションプログラムやサービスを自動起動させる必要がある。また、利用するサービスやコマンドの種類を吟味し必要最小限に制限することは、メモリや起動時間の節約において重要である。

4. ATDE for OpenRTM-aist

ATDE (ATDE2 および ATDE3) にはそれぞれ事前にクロス開発環境がインストールされている。ATDE for OpenRTM-aist では、これに加えて ARM 用にコンパイルされた omniORB および OpenRTM-aist が事前にインストール済みである。

なお、ATDE のベースとなっている Debian のパッケージリポジトリ内の omniORB のパッケージにはバグが含まれており、自分でインストールする場合は注意が必要である。ATDE for OpenRTM-aist を利用すればこうした心配をする必要はない。

また、OpenRTM バージョン 1.1 以降、RTC のビルドには CMake を利用しているため、Debian のリポジトリで公開されているのとは別に、最新の CMake がインストールされている。

このほか、RT コンポーネントをコンパイルする際に使用するクロスビルドツール、コンパイルした RT コンポーネントを依存ライブラリを自動で探索し USB メモリや SD メモリカードにコピーするファイルセットを作るためのツールなどが含まれる。以下、これらのツールについて使い方を解説する。

4·1 OpenRTM-aist 用プロダクトファイル

Armadillo-240 では USB メモリ、Armadillo-440 では USB メモリまたは SD メモリがストレージとして利用可能である。RT コンポーネントの実行ファイルとライブラリ、設定ファイル、起動スクリプトを USB/SD メモリに配置し、RT コンポーネントを起動すれば、ユーザランドイメージの書き換えなど時間のかかる作業を行わなくてよいので開発効率が向上すると考えた。また、ユーザが介入しなくても USB/SD メモリ上の RT コンポーネントを自動起動できれば、運用時に非常に便利となる。

そこで、RT コンポーネントの開発と Armadillo 上での実行・テストに関わるプロセスを以下のように仮定した。

1) RT コンポーネントを ATDE 上でクロスコンパイルし作成する。2) RTC および OpenRTM-aist および依存ライブラリを USB/SD メモリにコピーする。3) USB/SD メモリが Armadillo に挿入されたら、自動的にマウントし RT コンポーネントを自動起動させる。4) Armadillo にのタクトボタンを押すことで実行中の RT コンポーネントを全停止し、USB/SD メモリをアンマウントする。

1) については、4·3 節のクロスビルドツール、2) については 4·4 節のファイルパッケージ作成ツールで実現する。3) および 4) については、Armadillo 上に搭載された Linux システムの挙動であるため、Atmark Dist のプロダクトファイルにて実現する。以下にその方法を述べる。

Linux には USB や SD メモリの挿抜をフックし、ディスクデバイスの自動マウントやコマンドに実行などを自動的に行う udev というメカニズムがある。これにより、USB/SD メモリの挿入を検知して、1) メモリデバイスのチェック、2) USB/SD メモリのマウント、3) USB/SD メモリ上の boot.sh を実行、RTC を起動、4) タクトスイッチをフックするコマンドを設定、を行う。一方、終了処理として、タクトスイッチが押された際には、RTC のシャットダウン後メモリをアンマウントするといった処理を行う。

Atmark Dist では、製品固有の設定や起動スクリプトをまとめたものをプロダクトと呼び、が特定のディレクトリに格納されている。RT コンポーネントの自動起動を実現するために、ATDE for OpenRTM-aist では以上のように設定した OpenRTM-aist 用にカスタマイズしたプロダクトファイルを用意した。

4·2 イメージのダウンロード

Web ブラウザからユーザランドおよびカーネルイメージを OpenRTM-aist 用のものに書き換え、その後 ATDE 上でコンパイルした RT コンポーネントを USB/SD メモリにコピーし Armadillo に挿入することで RT コンポーネントを実行する図 3 に示す流れを実現する。

上記の OpenRTM-aist 用プロダクトから、Armadillo に搭載するユーザランド (起動スクリプトや各種コマンド類を含む root ファイルシステム) ファイルイメージおよびカーネルイメージを作成したものを OpenRTM-aist のオフィシャルサイト [2] において提供している。

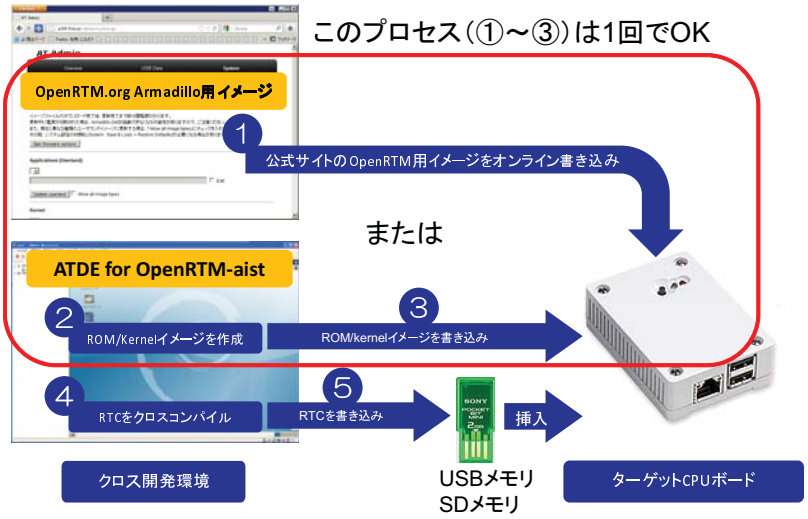

Fig.3 Cross development process using ATDE for OpenRTM-aist.

Fig.4 Downloading romfs from openrtm.org by AT Admin.

Armadillo では AT admin という Web 管理システム (図 4) が利用可能であるが、これを利用して、OpenRTM-aist 用ユーザランドイメージやカーネルイメージをオンラインで書き換えることが可能である。ネットワーク経由でイメージをダウンロードしフラッシュメモリを書き換えるため、シリアルケーブル経由で書き込むよりも非常に早く書き込むことが可能である[2]。

4·3 クロスビルドツール

通常のコンパイルとは異なり、クロスコンパイルでは、様々な事前設定を行わなければならない。通常の Makefile を利用したコンパイルでは、

```
$ CXX=/usr/bin/arm-linux-gnu-g++ make
```

のように、環境変数 CXX に使用するクロスコンパイラを与えることでコンパイルできる。ただし、C 言語と C++言語が混在している場合、環境変数 CC も設定しなければならないなど、ビルド中に何が行われるか理解している必要がある。

一方、CMake を利用したクロスコンパイルは、通常以下のようなツールチェーンファイルを作成したうえで、CMake に与える必要がある。

```
SET(CMAKE_SYSTEM_NAME Linux)
SET(CMAKE_SYSTEM_VERSION 1)
SET(CMAKE_C_COMPILER
    /usr/bin/arm-linux-gnu-gcc)
SET(CMAKE_CXX_COMPILER
    /usr/bin/arm-linux-gnu-g++)
SET(CMAKE_FIND_ROOT_PATH /usr/arm-linux-gnu)
 :略
```

ATDE for OpenRTM-aist にはクロスコンパイルを容易に行うための、make/cmake/ccmake のラッパーコマンド、rtc-make-cross, rtc-cmake-cross, rtc-cmake-cross が用意されている。これらのコマンドは以下のように利用する。

```
$ rtc-make-cross -a arm
```

“-a” オプションでターゲットとするアーキテクチャを指定する。この場合、ARM 用のクロスコンパイルを行うので arm となっている。このコマンドは、利用可能なクロスコンパイル環境を自動で認識し、1 種類しか環境が存在しない場合は “-a” オプションは不要である。利用可能はターゲットアーキテクチャは “-h” オプションで表示される　(下の例では ARM と PowerPC がターゲットアーキテクチャとして指定可能)。

```
$ rtc-make-cross -h
Usage: rtc-make-cross -a <arch_name> [options]

     -a     target architecture name
     -h     print this help

 Available architectures:
    arm powerpc
```

[2]カーネルイメージで 1 分以内、ユーザランドイメージで 2,3 分程度。

なお、“-a”, “-h” 以外のオプションはすべて make/cmake/ccmake にそのまま渡される。[3]

4·4 ファイルパッケージ作成ツール

コンパイルした RT コンポーネントの実行ファイルを、ターゲット上で実行するためには、実行ファイルと共に、依存ライブラリが必要となる。これらを間違いなくコピーすることは非常に手間がかかる作業である。このため、ATDE for OpenRTM-aist では対象とする RT コンポーネント、依存ライブラリファイル、起動スクリプトおよび設定ファイル rtc.conf を自動でターゲットディレクトリにコピーするツール: rtc2usbmem が用意されている。

以下のように、“-r” オプションで対象とする RTC の実行ファイルを、“-d” オプションでコピーするターゲットディレクトリを指定し実行する。

```
$ rtc2usbmem -r SeqOutComp -d ~/seqout

 Now Armadillo's USB memory image is created.
 Target architecture:  arm
 Target directory:     /home/atmark/seqout
 Target RTCs:          SeqOutComp
 Library search path:  /usr/arm-linux-gnu
 : 略
 NOTE:
   Edit /home/atmark/seqout/rtc.conf for your
        environment.

 Target binaries have been copied to:
        /home/atmark/seqout
```

rtc2usbmem は指定された実行ファイルのアーキテクチャを調べ、さらに利用しているライブラリのアーキテクチャをチェック、依存関係を調べる。依存するライブラリを特定のディレクトリおよび “-l” オプションで指定されたディレクトリから探し出し、ターゲットディレクトリ内の rtc ディレクトリに RT コンポーネントの実行ファイル、lib ディレクトリに依存ライブラリをコピーする。このほか、ターゲットディレクトリ直下にコンポーネントの起動スクリプト boot.sh とコンポーネントの設定ファイル rtc.conf という 2 つのファイルを作成する。

なお、rtc2usbmem コマンドのオプションは以下のとおりである。その環境で利用可能なターゲットアーキテクチャは rtc-make-cross 同様 “-h” オプションでヘルプを表示すると表示される。

```
 -a              ターゲットアーキテクチャ
 -d <dir>        ターゲットディレクトリ
 -r <rtc>        ターゲット RTC 実行ファイル
 -l <libdir>     ライブラリサーチパス
 -x              RTM 関連ライブラリをコピーしない
 -h              ヘルプの表示
```

以上で作成されたターゲットディレクトリの中身を USB メモリ (Armadillo-240 の場合)、SD メモリ (Armadillo-440 の場合) にコピーし Armadillo に挿入することで RT コンポーネントを自動的に起動することができる。

[3] “-a” オプションは make/cmake/ccmake のいずれにも使用されていないオプションである。

5. おわりに

組込み CPU ボード Armadillo 上で動作する RT コンポーネント開発環境 ATDE for OpenRTM-aist を開発した。組込みシステム開発特にクロス開発の効率向上のため、実行ファイルおよび依存ライブラリを USB メモリまたは SD メモリ上に置き、挿入と共に自動実行する仕組みを提供することで、コンパイル、実行、テストのサイクルがより素早く実行可能となった。また、ターゲットボードのイメージを Web 上で提供することで、CPU ボード自体のセットアップも効率的に行えるものとなっている。今後は、RaspberryPi 等他のターゲットに対しても今回開発した枠組みを適用する予定である。

参考文献

[1] Noriaki ANDO, Takashi SUEHIRO, Kosei KITAGAKI, Tetsuo KOTOKU, Woo-Keun Yoon, ”RT-Middleware: Distributed Component Middleware for RT (Robot Technology)”, 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS2005), pp.3555-3560, 2005.08

[2] OpenRTM-aist official site, http://openrtm.org

[3] Common Object Request Broker Architecture (CORBA) Specification, Version 3.1, OMG Specification formal/2008-01-04,07,08, http://www.omg.org/spec/CORBA/3.1/

[4] 安藤 慶昭, 鈴木 喬, 大原 賢一, 大場 光太郎, 谷江 和雄, ”組込機器のための軽量 RT コンポーネント: RTComponent-Lite”, 計測自動制御学会 システムインテグレーション部門 講演会 2005 (SI2005), p.3C2-2, 2005.12

[5] 安藤 慶昭, 神徳 徹雄, 佐々木 毅, 橋本 秀紀, ”組込み Linux のための RT ミドルウエアと開発環境”, 日本機械学会 ロボティクス・メカトロニクス講演会 2009, p.2A1-C13, 2009.05

[6] uClinux, http://www.uclinux.org/

[7] Tamio Tanikawa, Kenichi Ohara, Hiroyuki Nakamoto, Masato Iijima, Noriaki Ando, Takeshi Sakamoto, Tohru Takahashi, Masaki Nagatsuka, Tetsuo Kotoku, Kohtaro Ohba, Tatsuo Arai: “Smart home for security and low power consumption with common network modules based on RT middleware”, The 5th International Conference on Advanced Mechatronics(ICAM2010), pp.551-556, 2012.10

[8] 安藤 慶昭, 原 功, 大場 光太郎, ”μ ITRON ベース組込みシステムのための RT ミドルウエア”, 日本機械学会 ロボティクス・メカトロニクス講演会 2009, p.2A1-C14, 2009.05

[9] 青木利憲, 高瀬弘勝, “OpenRTM on T-Kernel の開発〜RTOS の T-Kernel(TRON) で動作する RT ミドルウェア〜”, ロボット学会学術講演会 2012, pp.2B1-5, 2012.9

[10] 藤田 恒彦, 水川 真, 安藤 吉伸, 吉見 卓, 田中 基雅, “CANopen を用いた組み込み系 RTC : RTC-CANopen の開発”, ROBOMECH2010, pp.1A1-G06, 2010.6

[11] 豊田 光弘, 池添 明宏, 中本 啓之, 長瀬 雅之, “miniRTC: 省資源マイコンで動作し複数の通信プロトコルに対応した RT ミドルウェア”, SI2009, 3D4-4, 2009.12

[12] 豊田 光弘, 伊藤 裕一, 荒田 耕造, 大和田 資, 池添 明宏, 中本 啓之, “組み込み向け軽量 RT ミドルウェアによるプラグアンドプレイ機能を有した RT システムの構築”, ROBOMEC2012

[13] 菅 佑樹, “RT コンポーネント対応デバイスを開発するためのマイコン用ライブラリ＆ツール「RTno」の開発”, 計測自動制御学会 システムインテグレーション部門 講演会 2011 (SI2011), p.1K4-4, 2011.12