# TOPPERS 活用アイデア・アプリケーション開発コンテスト

| | | |
|---|---|---|
| 部門 | ： | アプリケーション開発部門 |
| 作品のタイトル | ： | Raspberry Pi を使った Cortex-M 開発環境の構築 |
| 作成者 | ： | 石岡之也（個人） |
| 共同作業者 | ： | |
| 対象者 | ： | 組込みソフト開発初心者 |
| 使用する開発成果物 | ： | TOPPERS/ASP 1.9.3<br>ARM Cortex-M4 アーキテクチャ・GCC 依存部パッケージ コンフィギュレータ Release 1.9.6 |

目的・狙い

個人的に WinPC 上に VirtualBox と Linux で Cortex-M 版 TOPPERS/ASP の開発環境を構築してプログラム開発を行っていたが､WinPC のメモリや HDD の消費が大きくなることと会社支給の PC では業務外のアプリのインストールができない場合がある。
昨今、Raspberry Pi が広く行き渡っていること、小型ボードで他者への受け渡しが容易なことから TOPPERS/ASP の利用拡大の観点から Raspberry Pi 上に開発環境を構築しようと考えた。

アイデア/アプリケーションの概要

Raspberry Pi 3 or 4＋ubuntu20.04-ARM64 上に開発用ミドルウェアと Cortex-M 用クロスコンパイラなどのツールチェーンをインストールし、コンフィギュレータ 1.9.6、TOPPERS/ASP 1.9.3 を展開してビルドするための構築手順を明示する。
また、64 ビットの Linux 環境でコンフィギュレータ Release 1.9.6 や Cortex-M 版 TOPPERS/ASP 1.9.3 がビルドできない問題があるため、これを回避する修正方法を提示し、プログラムの作成からバイナリの作成までを Raspberry Pi 上だけで行える環境を提供する。

１．開発の背景

過去に TOPPERS コンテストに応募する作品の製作のため、WindowsPC に仮想化ソフト VirtualBox を導入し、そこに Linux をインストールして Cortex-M 版の TOPPERS/ASP カーネルやアプリケーションをビルドするためのクロス開発環境を構築して利用していた。これにより開発に別途 LinuxPC を準備することなく開発を行え、友人へもビルド環境を容易に展開することができた。

また、構築手順などを Qiita へ投稿して開発環境を公開してきた。

| |
|---|
| TOPPERS/ASP をビルドして動かしてみる<br><https://qiita.com/Yukiya_Ishioka/items/9ecbe080939600c323c6> |

| |
|---|
| TOPPERS/ASP(2020-03-10 版)をビルドして動かす<br><https://qiita.com/Yukiya_Ishioka/items/13b642eeffca1c266641> |

ただし、この方式だと WindowsPC へ VirtualBox のインストールや、VirtualBox 上で Linux を動作させるため WindowsPC の CPU パワー、メモリやＨＤＤの容量が消費させられ、リソースが少ない WindowsPC だと PC への負荷が高くなる。

また、会社支給のＰＣなどでは会社指定外のアプリケーションのインストールが制限されるなど VirtualBox や Linux を利用できない場合もある。

このため、WindowsPC へ負荷をかけずに開発を行えるよう利用者や保有者が非常に多いと考えられる Raspberry Pi 上に開発環境を構築し、開発に利用しようと考えた。

また、Raspberry Pi が非常に小型のマイコンボードで持ち運びや設置が楽で、さらに起動や運用はマイクロＳＤカードを用いることから環境を貸したり、コピーすることも容易であることから多くの人に利用してもらえると考えた。

２．利用機器・ソフトウェア

| | |
|---|---|
| Linux ボード | Raspberry Pi 3 / 3++ / 4 |
| Linux | Raspberry Pi 用 64bit Ubuntu 20.04.1 LTS<br><https://ubuntu.com/download/raspberry-pi><br><https://ubuntu.com/download/raspberry-pi/thank-you?version=20.04.1&architecture=arm64+raspi> |
| 追加ミドルウェア | g++、libboost-all-dev、libxerces-c-dev、make<br>net-tools、minicom、samba |
| ツールチェーン | GNU Arm Embedded Toolchain: 10-2020-q2-preview June 29, 2020<br>gcc-arm-none-eabi-10-2020-q2-preview-aarch64-linux.tar.bz2<br><https://developer.arm.com/-/media/Files/downloads/gnu-rm/10-2020q2/gcc-arm-none-eabi-10-2020-q2-preview-aarch64-linux.tar.bz2?revision=a7134e5c-fad8-490c-a62b-9200acca15ef&la=en&hash=12954B763712885C1DDE3318D10DF96B1606463A> |
| TOPPERS<br>コンフィギュレーター | コンフィギュレータ Release 1.9.6<br><https://www.toppers.jp/download.cgi/cfg-1.9.6.tar.gz> |
| TOPPERS/ASP | TOPPERS/ASP 1.9.3<br><https://www.toppers.jp/download.cgi/asp-1.9.3.tar.gz><br>ARM Cortex-M4 アーキテクチャ・GCC 依存部パッケージ<br><https://www.toppers.jp/download.cgi/asp_arch_arm_m4_gcc-1.9.6.tar.gz> |
| ターゲットボード | STM32F401 Nucleo-64<br><http://akizukidenshi.com/catalog/g/gM-07723/> |

３．Linux 環境の構築

・Linux のインストール

Raspberry Pi 用の Linux は複数のＳＤカードのイメージがインターネット上に公開されているが、後で説明するＡＲＭのクロスツールチェーンの新しい版は６４ビット環境用しか提供されていないため ARM64 用の Ubuntu 20.04.1 LTS を用いることとした。(2020 年 08 月 31 日現在最新)

このため、使用する Raspberry Pi は Raspberry Pi 3、Raspberry Pi 4 を用いることとする。

ＳＤカードのイメージは以下のダウンロードページの「Ubuntu 20.04.1 LTS」「Download 64-bit」からダウンロードします。xz 形式のファイルがダウンロードできるので、これを展開してできたイメージファイルをイメージ書込みソフトを使ってマイクロＳＤカードへ書き込みます。

・ネットワーク設定

ubuntu 20.04 では有線 LAN の eth0 を DHCP で IP アドレスが取得できるよになっています。DHCP サーバが動作しているネットワーク環境で LAN ケーブルを接続すればネットワークで通信を行うことができます。

また、/etc/netplan 配下に設定ファイルを追加して IP アドレスなどを定義することで固定の IP アドレスでの運用も可能となります。

以下は私の自宅環境で使っている設定ファイルです。

/etc/netplan/99_config.yaml

```
network:
  version: 2
  renderer: networkd
  ethernets:
    eth0:
      dhcp4: false
      dhcp6: false
      addresses: [192.168.0.140/24]
      gateway4: 192.168.0.1
      nameservers:
        addresses: [192.168.0.1, 8.8.8.8, 8.8.4.4]
```

・追加ミドルウェアのインストール

ネットワークで通信ができるようになったら、「apt-get install」コマンドを使って以下の追加ミドルウェアをインターネットからダウンロードしてインストールします。なお、「apt-get install」前に「apt-get update」コマンドでダウンロード用の情報を最新の情報へ更新しておいてください。

```
net-tools
g++
libboost-all-dev
libxerces-c-dev
make
minicom
samba
```

４．クロスツールチェーンの構築

GNU Arm Embedded Toolchain Downloads のページからクロスコンパイラなどのツールチェーンをダウンロードし、Raspberry Pi 上に展開します。今回は GCC10 用のツールチェーンを用いましたが、安定版の「9-2020-q2-update June 30, 2020」をダウンロードして用いても問題ありません。

ダウンロードするのは Linux AArch64 Tarball です。

tar.bz2 形式のファイルをダウンロード後、tar コマンドで「-C」で指定したインストール先のディレクトリへ展開します。

以下は /home/ubuntu/bin 配下へ展開するコマンドの例です。

```
mkdir ~/bin
tar xjf gcc-arm-none-eabi-10-2020-q2-preview-aarch64-linux.tar.bz2 -C ~/bin
```

今回展開したツールチェーンを利用するには、TOPPERS/ASP のビルド前に
環境変数 PATH へ実行ファイルのパスを設定します。
私の自宅環境では、以下の定義を「~/.bashrc」へ追加し、
ログイン後にいつでも使える状態にしてあります。

```
export PATH=~/bin/gcc-arm-none-eabi-10-2020-q2-preview/bin:$PATH
```

５．コンフィグレータの変更・ビルド

コンフィギュレータ Release 1.9.6 を使ってコンフィギュレータのビルドを行い、TOPPERS/ASP カーネルのビルドができるようにします。ただし、コンフィギュレータのページには以下の注記があることなどからダウンロードしたままではビルドすることができませんでした。

今回はこの問題を回避する修正なども含めてあります。

```
64bit の Linux 用の boost ライブラリに現状問題があるため、64bit の Linux でビルドした cfg は正しく動作しません。
64bit の Linux で cfg を使用する場合は、下記の 32bit Linux 用バイナリを使用するか、32bit の Linux でソースからビルドしてスタティックリンクしたバイナリを使用下さい。
```

まず、コンフィギュレータのページから以下のファイルをダウンロードして任意のディレクトリへ配置します。
以後の説明では以下のディレクトリに配置した想定でコマンド入力などを記します。

```
/home/ubuntu/toppers
```

・ソースの展開

ダウンロードした tar.gz ファイルを展開します。
以下のコマンドで展開すると cfg ディレクトリが生成され、配下にソースファイルなどが展開されます。

```
tar zxf cfg-1.9.6.tar.gz
```

・ディレクトリの移動

cfg ディレクトリへ移動します。

```
cd cfg
```

・libboost 用ヘッダファイルの追加

コンフィギュレータをビルドするとヘッダファイルのインクルード不足によるエラーで止まってしまうため、以下のように toppers/text.hpp ファイルと toppers/cpp.hpp ファイルへインクルードの定義を追加します。

toppers/text.hpp

```
54 #include "toppers/text_line.hpp"
55 #include "toppers/misc.hpp"
56 #include <boost/next_prior.hpp>  /* ※ 追加 */
57 #include <boost/utility.hpp>
58 #include <boost/iterator/iterator_facade.hpp>
```

toppers/cpp.hpp

```
44 #include "toppers/codeset.hpp"
45 #include "toppers/diagnostics.hpp"
46 #include <boost/next_prior.hpp>  /* ※ 追加 */
47 #include <boost/utility.hpp>
48 #include <boost/filesystem/path.hpp>
```

・コンフィギュレーション実行

コンフィギュレータのビルド前の環境設定を行います。ただし、展開直後の configure スクリプトは改行コードに「0x0d」が含まれているためこのままではエラーとなってしまいます。

このため以下のコマンドで不要なコードの削除などを行います。

```
mv configure configure.org
tr -d '¥r' < configure.org > configure
chmod u+x configure
```

修正後、以下のコマンドでコンフィグレーションを実行します。

```
./configure --with-libraries=/usr/lib/aarch64-linux-gnu
```

configure スクリプトの実行が成功したら以下のコマンドでコンフィギュレータをビルドします。

２０分弱程度かかります。

```
make OPTIONS=-std=c++11
```

cfg/cfg コマンドができていたら成功です。

```
$ cfg/cfg -v
TOPPERS Kernel Configurator version 1.9.6
```

６．ＡＳＰカーネルの変更・コンフィグレーション

TOPPERS/ASP カーネル ターゲット非依存部パッケージ asp-1.9.3.tar.gz と ARM Cortex-M4 アーキテクチャ・GCC 依存部パッケージ asp_arch_arm_m4_gcc-1.9.6.tar.gz を/home/ubuntu/toppers ディレクトリへダウンロードして ASP カーネルのビルドします。

・ソースの展開

ダウンロードした tar.gz ファイルを展開します。

以下のコマンドで展開すると asp ディレクトリが生成され、配下にソースファイルなどが展開されます。

```
cd /home/ubuntu/toppers
tar xzf asp-1.9.3.tar.gz
tar xzf asp_arch_arm_m4_gcc-1.9.6.tar.gz
```

・コンフィギュレータのコピー

ビルドして作成したコンフィギュレータを asp は以下にコピーします。

以下のコマンドで必要なディレクトリを作成してコピーすることができます。

```
mkdir -p asp/cfg/cfg
cp cfg/cfg/cfg asp/cfg/cfg
```

以下のコマンドを実行し、バージョン情報が出力されたらコピー成功です。

```
$ asp/cfg/cfg/cfg -v
TOPPERS Kernel Configurator version 1.9.6
```

・コンフィギュレーションエラーの回避

展開された ASP カーネルでは、GNU Arm Embedded Toolchain の GCC7 以降でカーネルビルド時に使われるリンカスクリプトで使われている「PROVIDE()」が意図した動作をせず、常に記述されている「hardware_init_hook = 0」が有効となり hardware_init_hook 関数のアドレスが常に 0x0 番地でバイナリが生成されます。

このため ASP カーネルをボード上で実行すると 0x0 番地へジャンプし、リセットを繰り返すことになります。このため以下のように hardware_init_hook 用のコードをコメントアウトします。

以下は stm32f401nucleo_gcc 用のリンカスクリプトの修正内容です。

asp/target/stm32f401nucleo_gcc/stm32f4xx_rom.ld

```
 10 PROVIDE(hardware_init_hook = 0);
 11 PROVIDE(software_init_hook = 0);
 12 PROVIDE(software_term_hook = 0);
 13 STARTUP(start.o)
```

↓

```
 10 /* PROVIDE(hardware_init_hook = 0); */
 11 PROVIDE(software_init_hook = 0);
 12 PROVIDE(software_term_hook = 0);
 13 STARTUP(start.o)
```

※今回のアプリケーションの構築方法では software_init_hook、software_term_hook は 0x0 番地で問題ありません。

このままではビルド中の cfg1_out の生成でエラーとなるため、「PROVIDE(hardware_init_hook = 0)」の代替処理を start.S へ追加します。

以下のように start.S の終端行へ以下の処理を追加します。

asp/arch/arm_m_gcc/common/start.S

```
.weak  hardware_init_hook
       bx  lr
```

・ＡＳＰカーネルのコンフィギュレーション

ASP カーネルのコンフィギュレーションはアプリケーションをビルドするディレクトリで実行します。今回は STM32F401 Nucleo-64 ボードをターゲットとして、/home/ubuntu/toppers/f401/obj ディレクトリにアプリケーションのビルド環境を構築します。

以下のコマンドで f401/obj ディレクトリを作成します。

```
cd /home/ubuntu/toppers
mkdir -p f401/obj
```

以下のコマンドで f401/obj ディレクトリへ移動し、コンフィギュレーションを実行します。

```
cd f401/obj
../../asp/configure -T stm32f401nucleo_gcc -dROM
```

以下のファイルが生成されたら成功です。

```
-rw-rw-r-- 1 ubuntu ubuntu 15335 Aug 30 18:22 Makefile
-rw-rw-r-- 1 ubuntu ubuntu 15652 Aug 30 18:22 sample1.c
-rw-rw-r-- 1 ubuntu ubuntu   918 Aug 30 18:22 sample1.cfg
-rw-rw-r-- 1 ubuntu ubuntu  3246 Aug 30 18:22 sample1.h
```

７．アプリケーションのビルド

Nucleo ボードは PC へ USB で接続することでディスクドライブとして見えるようになり、このディレクトリへビルドして生成されたバイナリコードをコピーすることでプログラムを書き込むことができます。

アプリケーションのビルドでバイナリコードが生成されるよう Makefile の以下の箇所に .bin ファイル生成の記述を追加します。

Makefile

```
327 $(OBJFILE): $(APPL_CFG) kernel_cfg.timestamp $(ALL_OBJS) $(LIBS_DEP)
328         $(LINK) $(CFLAGS) $(LDFLAGS) -o $(OBJFILE) $(START_OBJS) ¥
329                         $(APPL_OBJS) $(SYSSVC_OBJS) $(CFG_OBJS) $(ALL_LIBS) $(END_OBJS)
330         $(NM) -n $(OBJFILE) > $(OBJNAME).syms
331         $(OBJCOPY) -O srec -S $(OBJFILE) $(OBJNAME).srec
332         $(CFG) --pass 3 --kernel asp $(INCLUDES) ¥
```

↓

```
327 $(OBJFILE): $(APPL_CFG) kernel_cfg.timestamp $(ALL_OBJS) $(LIBS_DEP)
328         $(LINK) $(CFLAGS) $(LDFLAGS) -o $(OBJFILE) $(START_OBJS) ¥
329                         $(APPL_OBJS) $(SYSSVC_OBJS) $(CFG_OBJS) $(ALL_LIBS) $(END_OBJS)
330         $(NM) -n $(OBJFILE) > $(OBJNAME).syms
331         $(OBJCOPY) -O srec -S $(OBJFILE) $(OBJNAME).srec
332         $(OBJCOPY) -O binary -S $(OBJFILE) $(OBJNAME).bin      ←★追加
333         $(CFG) --pass 3 --kernel asp $(INCLUDES) ¥
```

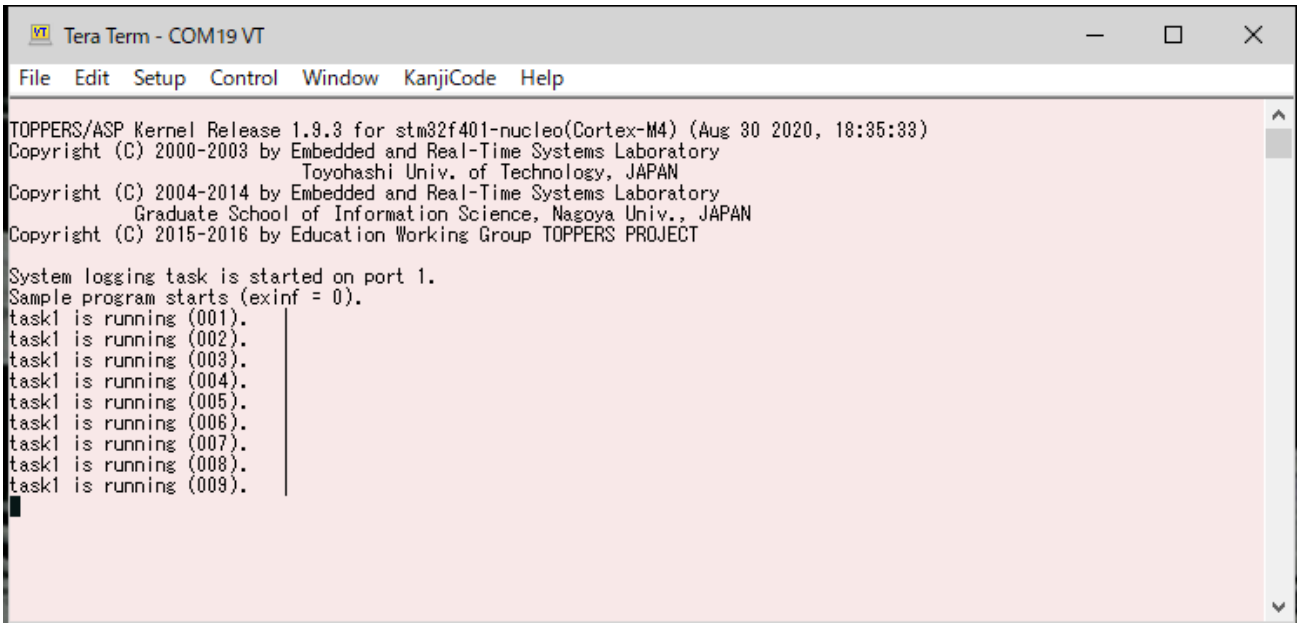
追加後、f401/obj ディレクトリで make を実行することでアプリケーションのビルドを実行できます。

```
cd f401/obj
make
```

以下のファイルが生成されたらビルド成功です。

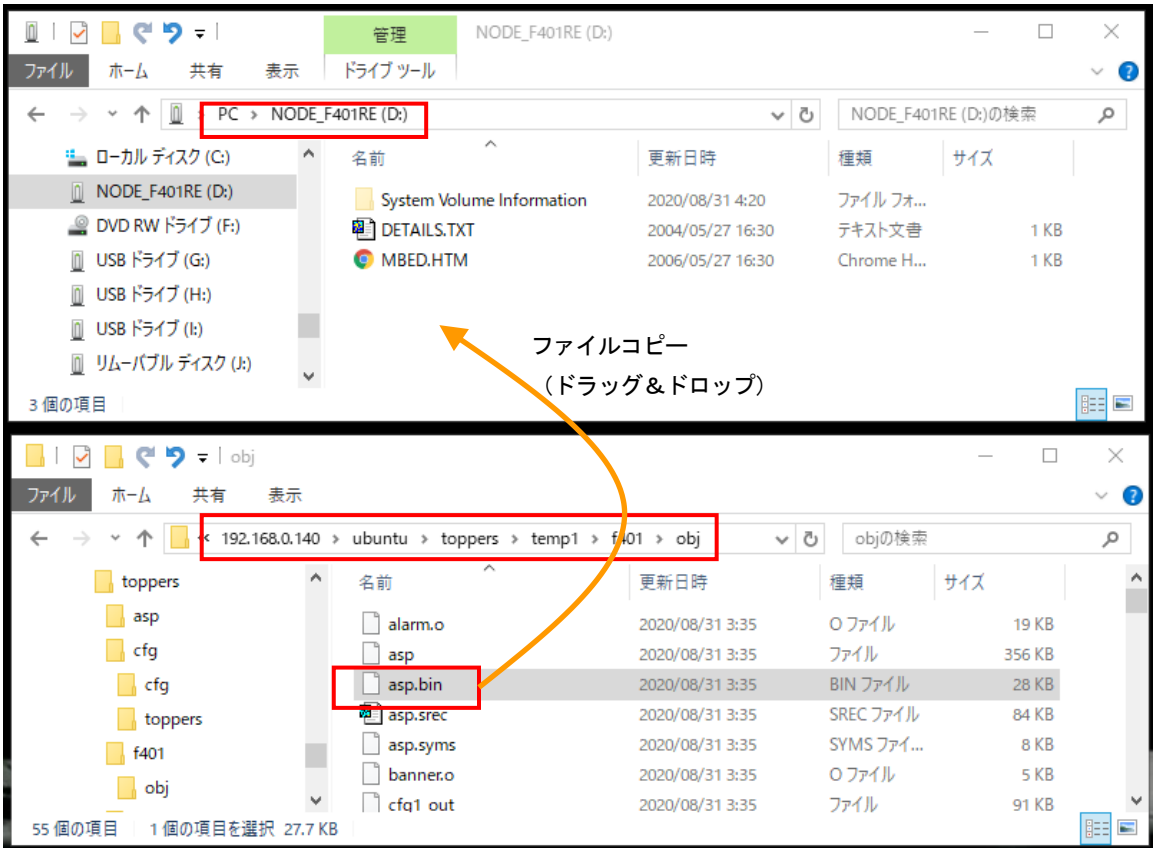
```
----------------------------------------
-rwxrwxr-x 1 ubuntu ubuntu 363656 Aug 30 18:35 asp*
-rwxrwxr-x 1 ubuntu ubuntu  28442 Aug 30 18:35 asp.bin*
-rwxrwxr-x 1 ubuntu ubuntu  85392 Aug 30 18:35 asp.srec*
-rw-rw-r-- 1 ubuntu ubuntu   8109 Aug 30 18:35 asp.syms
```

STM32F401 Nucleo-64 ボードへ asp.bin をコピーしてプログラムを実行すると STM32F401 Nucleo-64 ボードの仮想 COM ポートへ以下のメッセージが出力されます。



・WindowsPC 上での asp.bin ファイルのコピー

Raspberry Pi に Samba をインストール・設定することで、Raspberry Pi 上の asp.bin ファイルを Windows のエクスプローラー上でドラッグ＆ドロップ操作で STM32F401-Nucleo へ書き込むことができる。

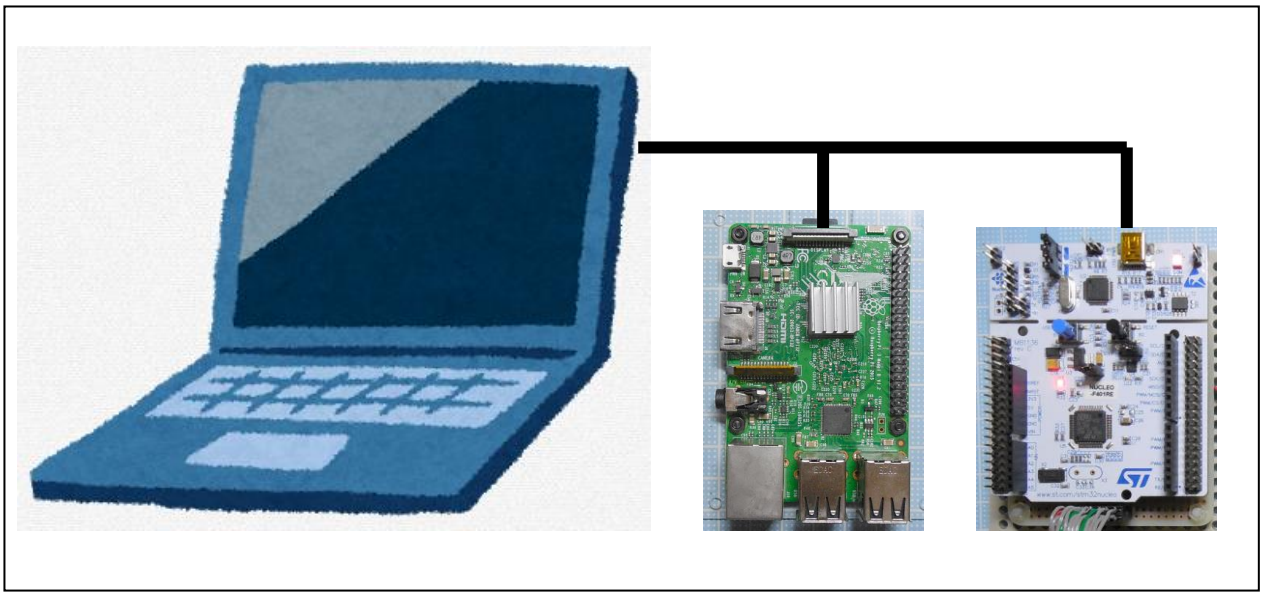


８．運用構成

Raspberry Pi を使った開発では、WindowsPC を介して Raspberry Pi 上で生成さたバイナリファイルをターゲットボードへ書き換える方法のほか、Raspberry Pi へモニタやキーボード、ターゲットボードを接続した Raspberry Pi 単体の運用構成をとることも可能となる。

STM32F401-Nucleo ボードは Linux からも USB 接続で仮想 COM ポートの/dev/ttyACM、マスストレージの /dev/sda1 としてアクセスできることから、ターゲットボードへの書き込みから実行後の syslog()出力も Raspberry Pi 単体で運用することができる。

●WindowsPC を介した運用構成



●Raspberry Pi 単体の運用構成