



# ADK互換モジュールで遊ぶAndroid™

第4回名古屋Android勉強会

2012/2/18

愛知工業大学 本山キャンパス

@magoroku15 最底辺活動家

# このコースの目的と内容

## 目的

AndroidアプリケーションデベロッパがAndroid Open Accessory を理解する。同時にHWを理解する事が広げる世界を体感する。

## 内容

1. 互換モジュールの組み立て
2. Androidとの接続と動作確認
3. 互換モジュールの仕組み
4. Android側の仕組み

# PART0

## 事前準備

# 最新リソース

- このドキュメントを含む、最新のリソースは以下で管理しています
  - <https://github.com/magoroku15/OpenAccessoryDemo>
    - App Source CodeAndroid アプリのソースコード
    - Doc ドキュメント類
    - Firmware マイコンのソース・バイナリ
- 紹介するMicrochip Technology Inc作成のアンドロイドアプリをAndroid Marketからインストールしておいてください
  - Microchipで検索
    - 基本的なアクセサリデモ3.x
    - 基本的なアクセサリデモ2.3.x以降

# レベルごとの準備

- レベル1 手ぶらコース
  - 特に準備は不要です
- レベル2 実機に接続して動かす
  - ADKに対応したAndroid実機とACアダプタを用意
- レベル3 マイコン実行イメージのビルドまで
  - MPLABXをインストールしたPCを用意
- レベル4 Androidアプリのビルドまで
  - Android SDKをインストールしたPCを用意

# ソースコード・ドキュメント

- Githubで管理

<https://github.com/magoroku15/OpenAccessoryDemo>

- 2つの方法

- A) Githubにアカウントを作つてfork

- Linux/Macで開発する人にお勧め
    - Zipでダウンロード改変してcommit & push

- B) ZIPアーカイブをダウンロード

- Windowsで開発する人向け

<https://github.com/magoroku15/OpenAccessoryDemo>

Gitに慣れていない人は  
ここからZipを  
ダウンロード

GitHubにアカウントのある人は、ここから。Forkも歓迎

name	age	message	history
App Source Code	December 19, 2011	first commit [magoroku15]	
Doc	about 21 hours ago	構成変更 [magoroku15]	
Firmware	December 20, 2011	NA [magoroku15]	

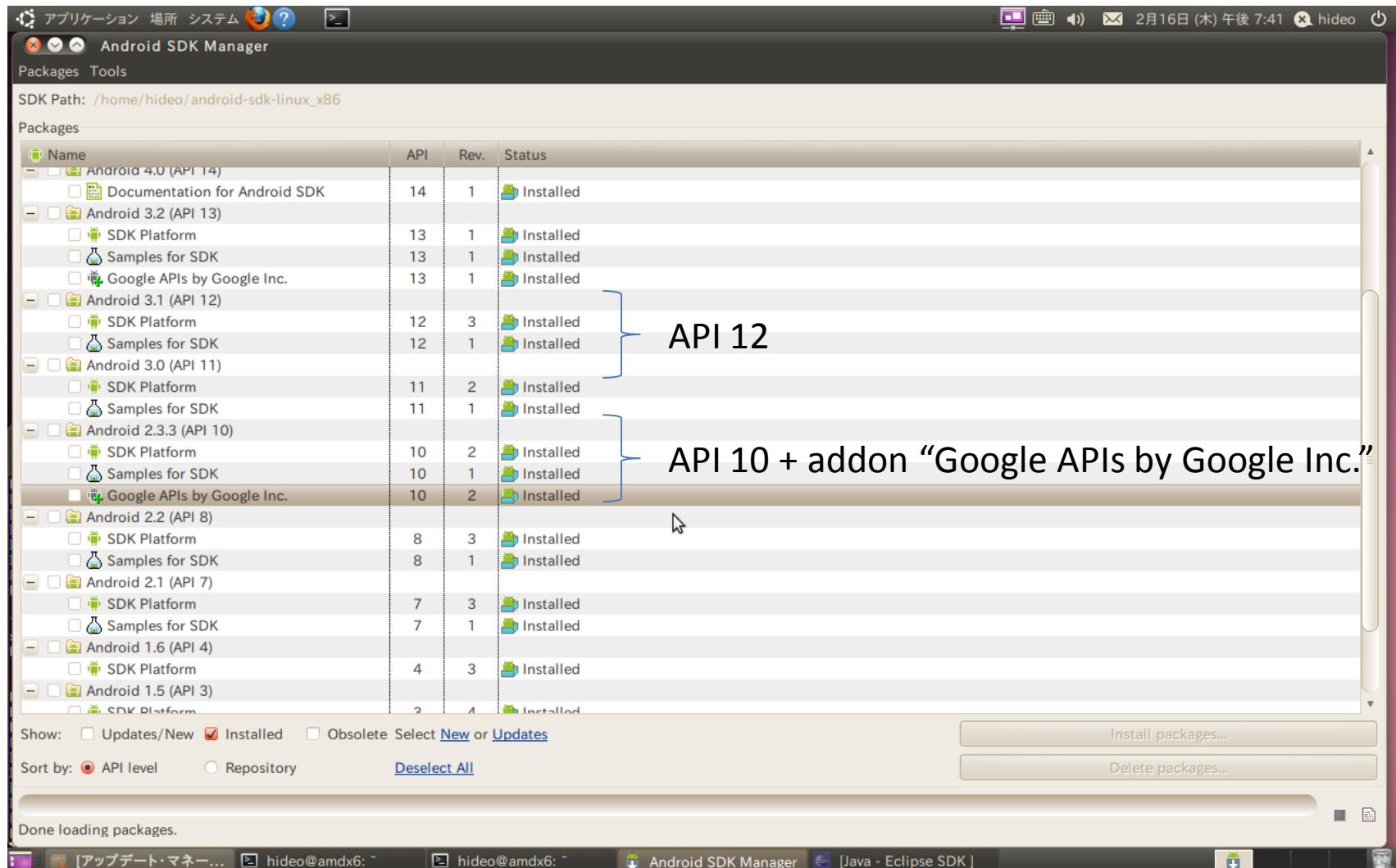
# OpenAccessoryDemoの内容

- App Source Code            Androidアプリのソース
  - v2.3.x     2.3.4以降のADKアプリのソース
  - V3.x        3.x以降のADKアプリのs-す
- Doc
  - Poorman's adk.pptx        この資料
  - OpenAccDemo.pdf          回路図 PDF形式
  - OpenAccDemo.sch          回路図 EagleCAD形式
- Firmware                    Firmwareのソース

# Android SDKのインストール

- ハンズオンでAndroidのサンプルプログラムの解説を行い、その中でAppの作成に軽く触れます
- 動作確認はマーケットからダウンロードしたアプリで行います
- サンプルソースのバージョンに合わせて各自で事前にインストールを済ませておいてください
  - Androidのバージョンが2.3.xの場合
    - API 10に加えてGoogle APIアドオンが必要
  - Androidのバージョンが3.1以降の場合
    - API12が必要

# SDKのバージョン・アドオン



# Android SDKの確認 v2.x

- v2.3.xの場合

0) SDK Managerで「Android 2.3.3 (API10)のGoogle APIsアドオン」をインストール

1) 新規のAndroidプロジェクトを作成

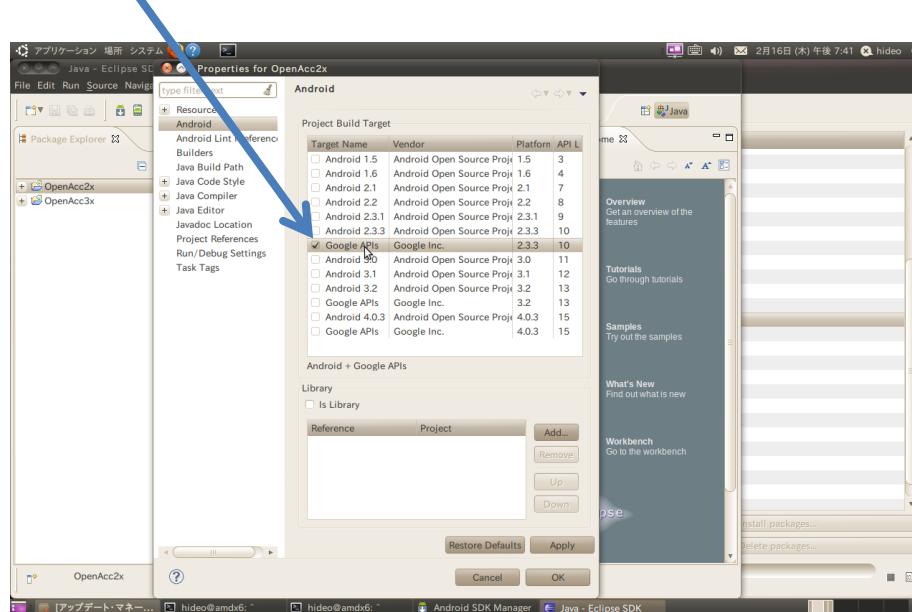
2) 「Create project from existing source」を選択

3) Locationをgithubからダウンロードしたzipを展開したフォルダの中の「OpenAccessoryDemo/App Source Code/v2.3.x」に設定

4) Nextをクリック

5) Build Targetを「Google APIs-2.3.3-10」

6) Finishをクリック



# Android SDKの確認 v3.x,v4.x

- v3.x,v4.xの場合

0) SDK Managerで「Android 3.1 (API12) SDK Platform」をインストール

1) 新規のAndroidプロジェクトを作成

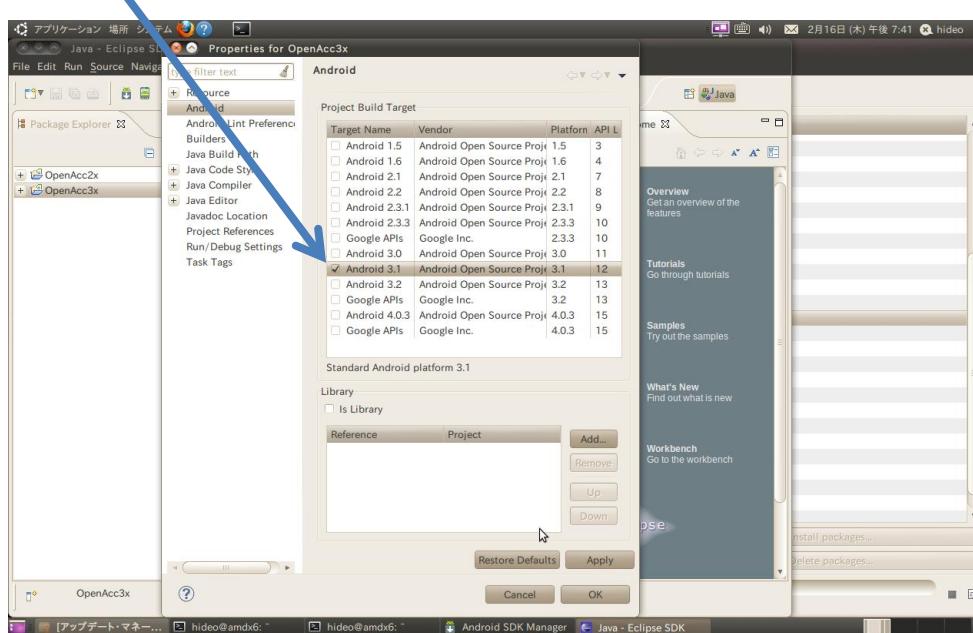
2) 「Create project from existing source」を選択

3) Locationをgithubからダウンロードしたzipを展開したフォルダの中の「OpenAccessoryDemo/App Source Code/v3.x」に設定

4) Nextをクリック

5) Build Targetを「Android 3.1 (API12)」

6) Finishをクリック



# MPLABXのインストール

- PIC用のIDE
  - 従来のMPLABはWindows専用だった
  - MPLABXはNetBeansベースでマルチプラットフォーム
  - PIC24シリーズ向けはコンパイラも無償/最適化に制限
- <http://ww1.microchip.com/downloads/mplab/X/>から
  1. Platformを選択
  2. 以下をチェックして[Download Now]
    - MPLAB IDE X v1.00
    - **MPLAB X IDE Release Notes/User' Guide (supersedes info in installer)**
    - **MPLAB C30 Lite Compiler for dsPIC DSCs and PIC24 MCUs**
  3. [Download Now]

# MPLABXのインストール

Mac  
Linux  
Windows

The screenshot shows a Firefox browser window with the URL [www.microchip.com/downloads/mplab/X/](http://www.microchip.com/downloads/mplab/X/). The page displays the MPLAB X logo and two main sections: "Select Platform:" and "Select Development Tools:". Under "Select Platform:", "Linux (32/64 bit)" is selected. Under "Select Development Tools:", several options are listed with checkboxes: "MPLAB IDE X v1.00" (checked), "MPLAB X IDE Release Notes/User' Guide (supersedes info in installer)" (checked), "MPLAB C18 Lite Compiler for PIC18 MCUs" (unchecked), "HI-TECH C Lite Compiler for PIC18 MCUs" (unchecked), "HI-TECH C Lite Compiler for PIC10/12/16 MCUs" (unchecked), "MPLAB C30 Lite Compiler for dsPIC DSCs and PIC24 MCUs" (checked), and "MPLAB C32 Lite Compiler for PIC32 MCUs" (unchecked). A blue arrow points from the "①選択" button to the "Select Platform:" dropdown. Another blue arrow points from the "②チェック" button to the checked compiler options. A third blue arrow points from the "③ダウンロード" button to the "Download Now" button. To the right of the main page, a separate window shows the Java download progress: "Downloading Java Installer" with an estimated time left of 7 sec. The Java download progress bar is shown at the top of the window.

# **PART1**

# **互換モジュールの組み立て**

# 配布部品

積層セラミックコンデンサー $10\mu\text{F}25\text{V}$

積層セラミックコンデンサー $1\mu\text{F}25\text{V}$

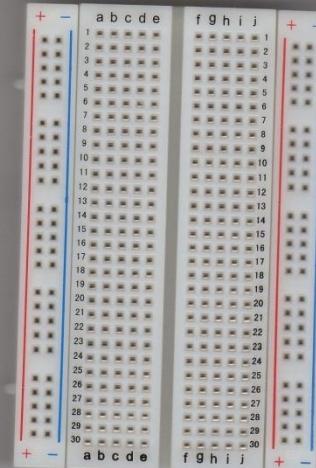
PIC24FJ64GB002

赤色LED 3mm

カーボン抵抗  $1/4\text{W}10\text{k}\Omega$

ブレッドボード・ジャンパーウイヤ

三端子レギュレーター

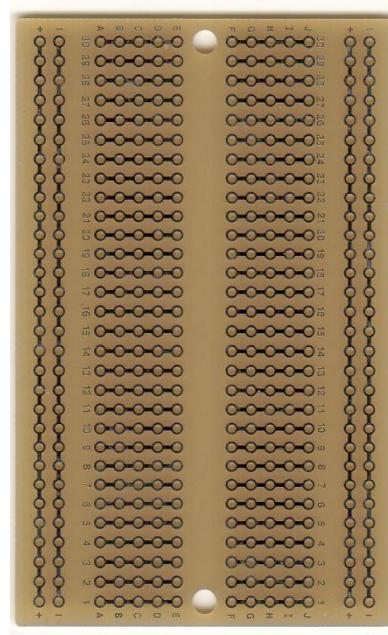
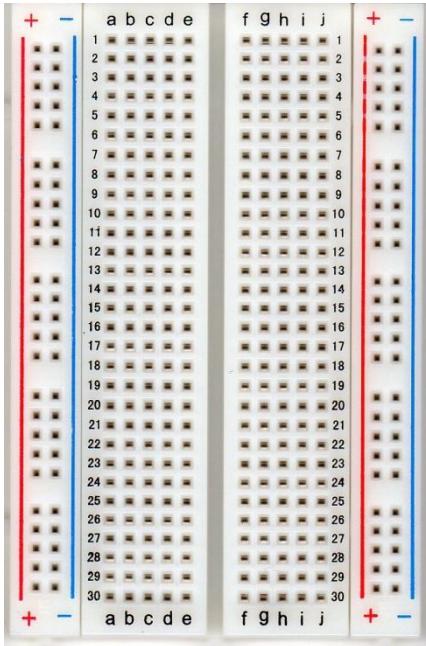


ブレッドボード EIC-801

USBケーブル(A-microB)  
Lピンヘッダ

可変抵抗

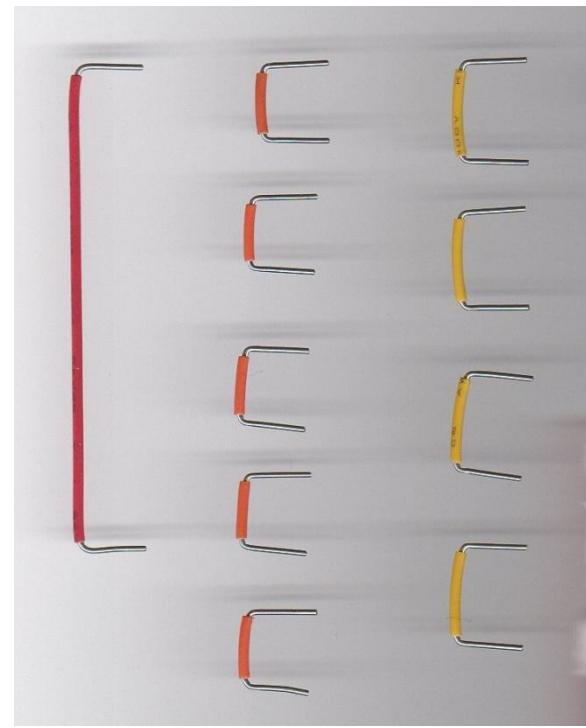
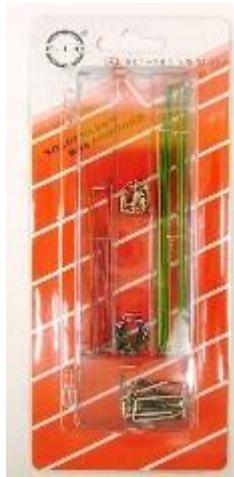
# ブレッドボード



内部の配線

# ジャンパ

- ・ ブレッドボードに刺して回路を構成する線



# マイコン PIC24FJ64GB002

- Microchip社の16bitマイコン max 32MHz
  - 64Kbyte Program Memory (Flash)、64Kbyte RAM
  - I<sup>2</sup>C, IrDA, SPI, UART/USART, USB OTG
- ハンズオン用プログラムは書き込み済で配布

The diagram shows the pin configuration for the PIC24FJ64GB002 microcontroller. The pins are numbered 1 through 28. A blue arrow points from the text "ハンズオン用プログラムは書き込み済で配布" to the top center of the pinout diagram.

MCLR	1	28	VDD
PGED3/AN0/C3INC/VREF+/ASDA1 <sup>(2)</sup> /RP5/PMD7/CTED1/V/BUSVLDN/CMPST1/CN2/RA0	2	27	VSS
PGEC3/AN1/C3IND/VREF-/ASCL1 <sup>(2)</sup> /RP6/PMD6/CTED2/SESSVLDN/CMPST2/CN3/RA1	3	26	AN9/C3INA/VBUSCHG/RP15/VBUST/CN11/RB15
PGED1/AN2/C2INB/DPH/RP0/PMD0/CN4/RB0	4	25	AN10/C3INB/CVREFV/OPCON/VBUSHON/RP14/CN12/RB14
PGEC1/AN3/C2INA/DMH/RP1/PMD1/CN5/RB1	5	24	AN11/C1INC/RP13/PMRD/REFO/SESSEND/CN13/RB13
AN4/C1INB/DPLN/SDA2/RP2/PMD2/CN6/RB2	6	23	VUSB
AN5/C1INA/DMLN/RTCC/SCL2/RP3/PMWR/CN7/RB3	7	22	PGEC2/D-/MIO/RP11/CN15/RB11
VSS	8	21	PGED2/D+/PIO/RP10/CN16/RB10
OSCI/CLKI/C1IND/PMCS1/CN30/RA2	9	20	VCAPV/DDCORE
OSCO/CLKO/PMA0/CN29/RA3	10	19	DISVREG
SOSCI/C2IND/RP4/PMBE/CN1/RB4	11	18	TDO/SDA1/RP9/PMD3/RCV/CN21/RB9
SOSCO/SCLKI/T1CK/C2INC/PMA1/CN0/RA4	12	17	TCK/USBOEN/SCL1/RP8/PMD4/CN22/RB8
VDD	13	16	TDI/RP7/PMD5/INT0/CN23/RB7
TMS/USBID/CN27/RB5	14	15	VBUS

ピン配置

# 抵抗

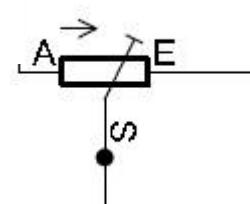
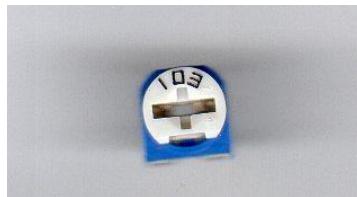
- 電流の流れを抑止する
- 単位は $\Omega$ (オーム)
  - 抵抗の値が小さいと導体
  - 抵抗の値が大きいと絶縁体に近づく
- 極性なし



回路記号

# ポテンショメータ/可変抵抗

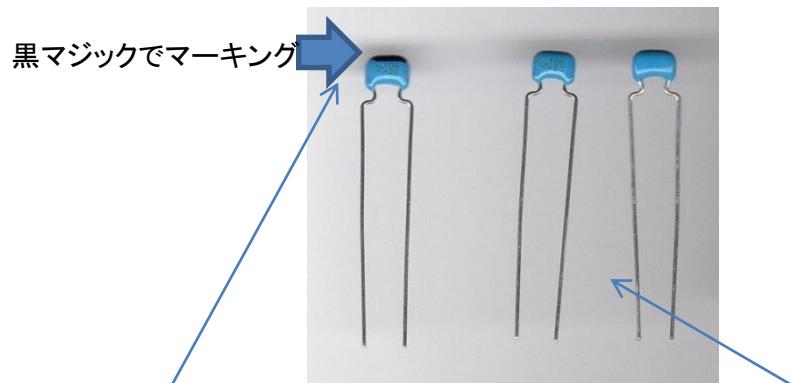
- 抵抗値を変化させる
- 単位はΩ(オーム)
  - 音響装置の「ボリューム」がこれ
  - 回転角・位地の検出にも使う



回路記号

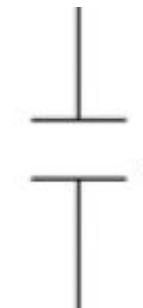
# コンデンサ

- Capacitor(キャパシタ)とも言う
- 単位はF(ファラッド)
- 微量の電気を蓄える
  - 直流は流れない
  - 交流は流れやすい
- 極性の有無に注意
  - 今回の積層セラミックコンデンサは極性なし



積層セラミックコンデンサー 10μF 25V

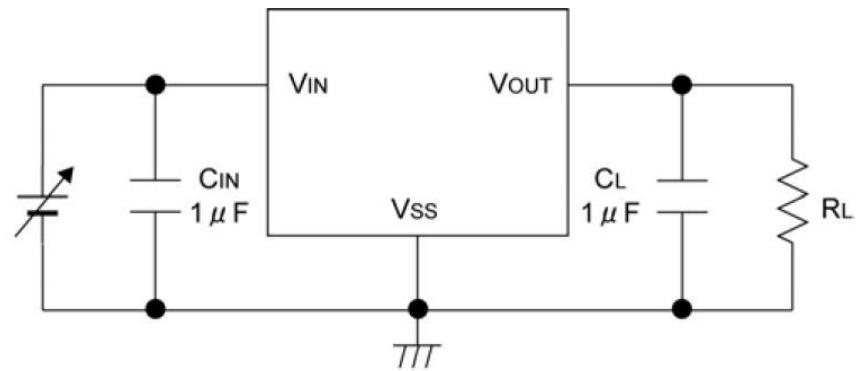
積層セラミックコンデンサー 1μF 25V



回路記号

# 3端子レギュレータ

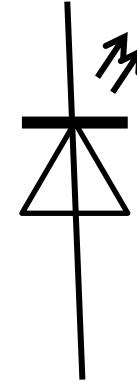
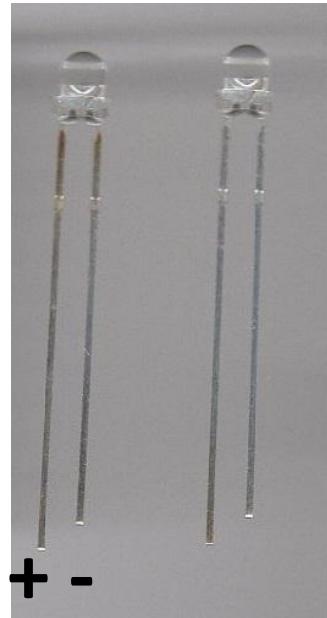
- 電圧を一定に保つ機能を備えたIC
- 5V(充電用ACアダプタ)から3.3Vを生成



回路図(周辺込み)

# LED

- 発光ダイオード
  - LED(エルレイーディー: Light Emitting Diode)
  - 極性あり、一方向にしか電流を流さない



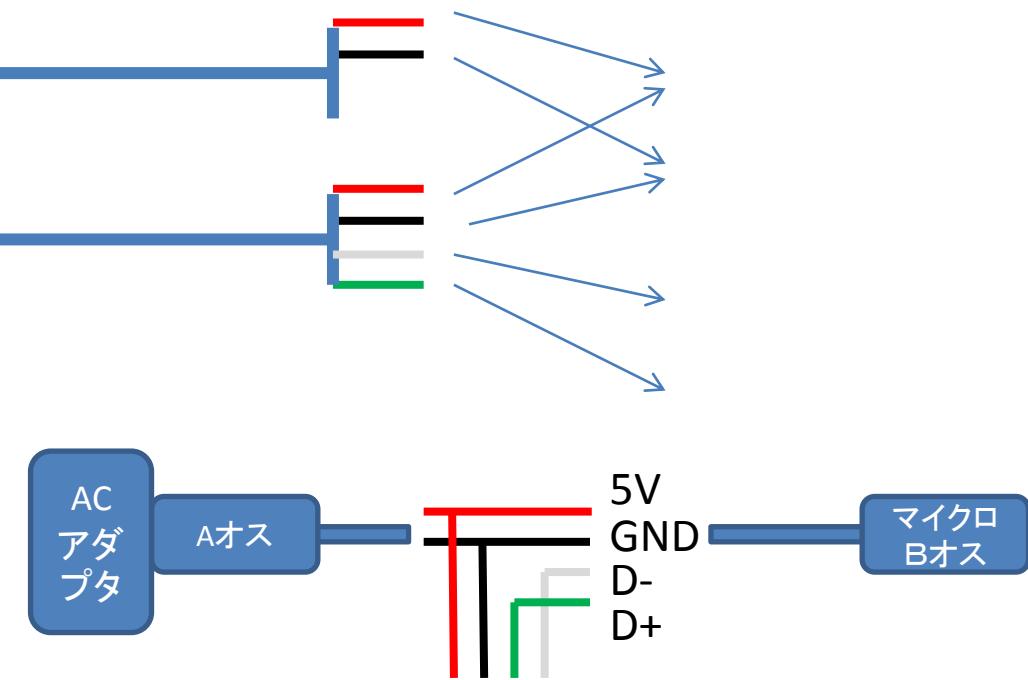
回路記号

# USBケーブル

- AオスマイクロBオスを改造(改造済で配布)
  - ACアダプタから5Vを取り、マイクロBオスに回す
  - マイクロBのデータ線を取りだす



AオスーマイクロBオス



# 互換モジュールの組み立て

- 利用する資料
  - 回路図
  - 実体配線図
  - 組み立て手順

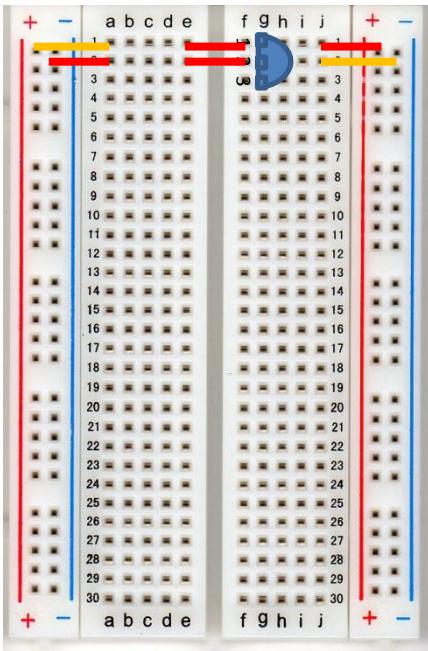
# 配布部品と入手価格

poorman's ADK 30台分の発注部品

購入先	通販型番	発注量	購入単位		1台単位		
			個数	価格	単価	個数	
秋月	C-05223	30	1	130	130	1	130 USBケーブル(A-microB)
秋月	P-02315	30	1	250	250	1	250 ブレッドボード・ジャンパーウイヤ
秋月	P-00315	30	1	250	250	1	250 ブレッドボード EIC-801
秋月	I-03421	15	2	100	50	1	50 三端子レギュレーター[3. 3V] XC6202P332TB
秋月	P-05105	60	1	15	15	2	30 積層セラミックコンデンサー1μ F50V
秋月	P-05103	30	1	30	30	1	30 積層セラミックコンデンサー10μ F25V
秋月	I-00562	1	100	350	3.5	2	7 赤色LED 3mm
秋月	R-25103	2	100	100	1	4	4 カーボン抵抗 1／4W10kΩ
秋月	C-01627	10	5	20	4	1	4 Lピンヘッダ1x20
秋月	P-02470	4	10	200	20	1	20 半固定抵抗10kΩ
秋月	P-04089	1	100	500	5	1	5 チャック袋
秋月	送料2回分	1	30	1000	33	2	67 送料500円×2回
digikey	703-7602	1	30	9821	327	1	327 PIC24FJ64GB002

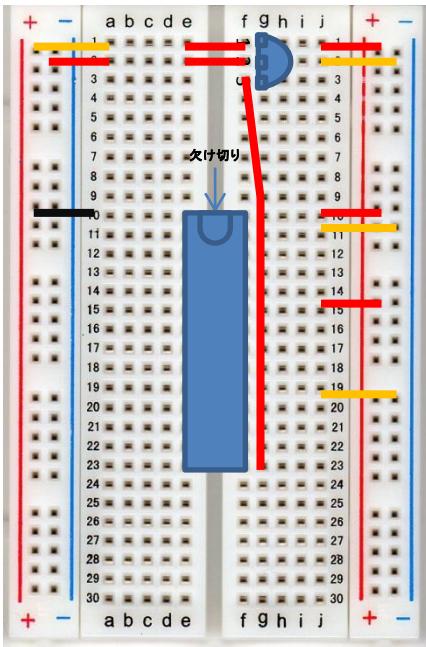
合計 1174

# Step1 電源



1. g1にレギュレータの1
2. g2にレギュレータの2
3. g3にレギュレータの3
4. —を4か所
5. —を2か所
6. 積層セラミックコンデンサー  
1μF25V2個
  - h1-h2
  - h2-h3

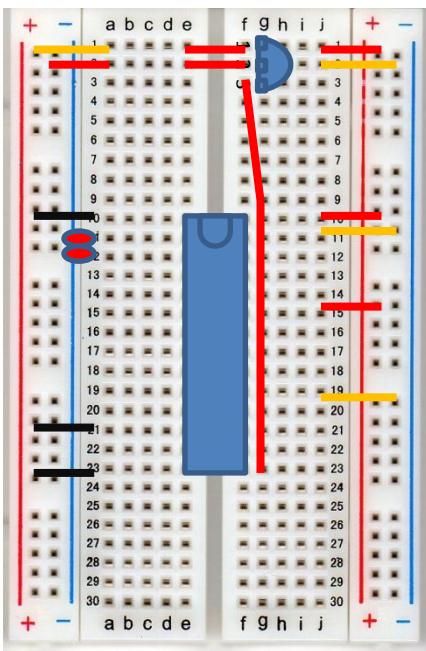
# Step2 マイコン



1. e10-e23-f10-f23'にマイコン
2. ーをf3-g23へ
3. ーをj10—へ
4. —をj11-+へ
5. —をj15—へ
6. —をj19-+へ
7. 積層セラミックコンデンサー $10\mu\text{F}$ をj18—へ
8. 抵抗 $10\text{k}\Omega$ をa10-+へ

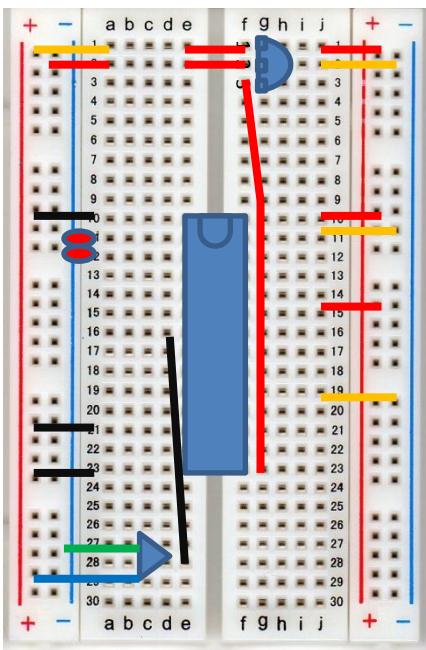
# Step3 LED/SW

- LED、足の長い方をa11,短い方を—へ
- LED、足の長い方をa12,短い方を—へ
- 抵抗 $10k\Omega$ 、a21-+へ
- 抵抗 $10k\Omega$ 、a23-+へ



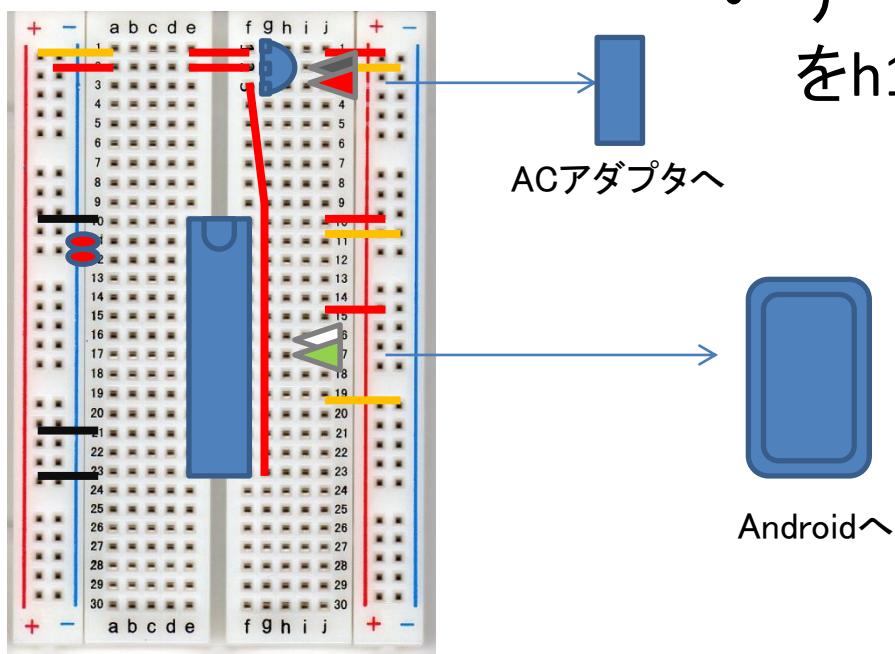
# Step4 可変抵抗

- 組み付け順注意
- d16-e28に抵抗を
- —をc27-—に
- —をc29-+に
- 可変抵抗をc27-c29-e28へ

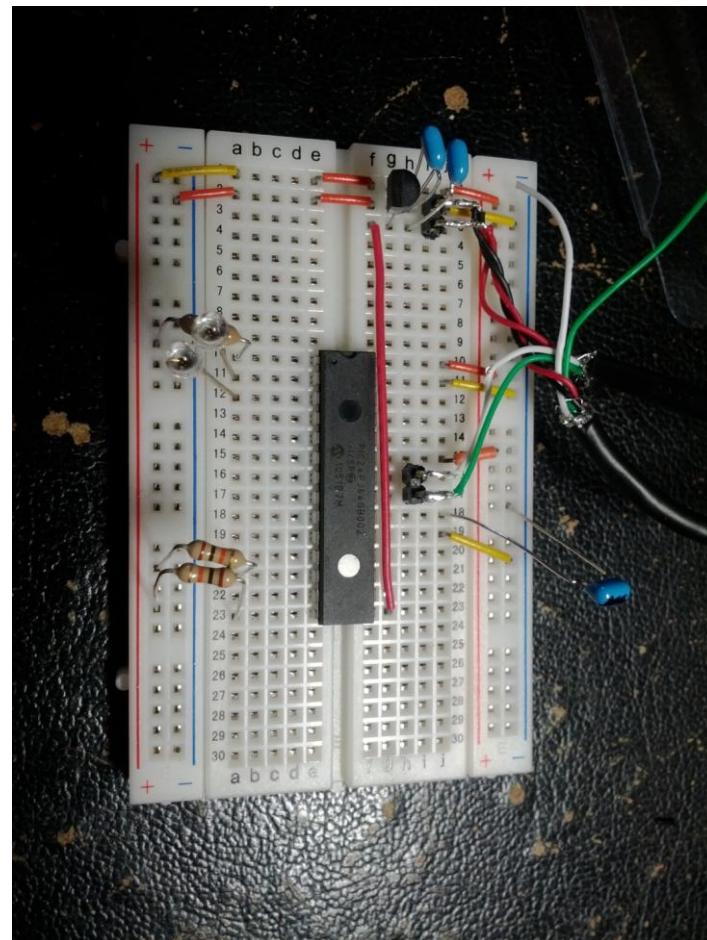
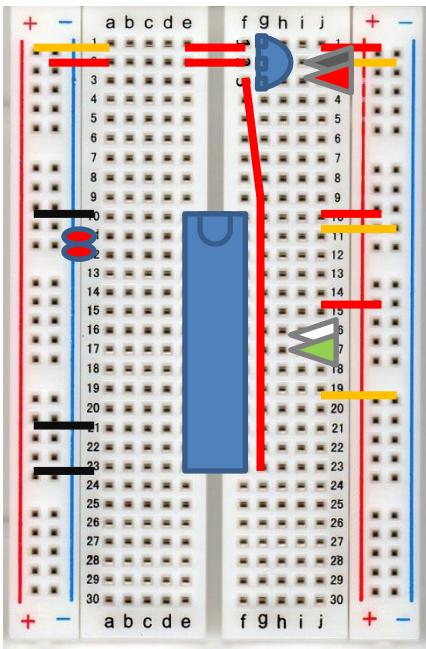


## Step5 USBケーブル

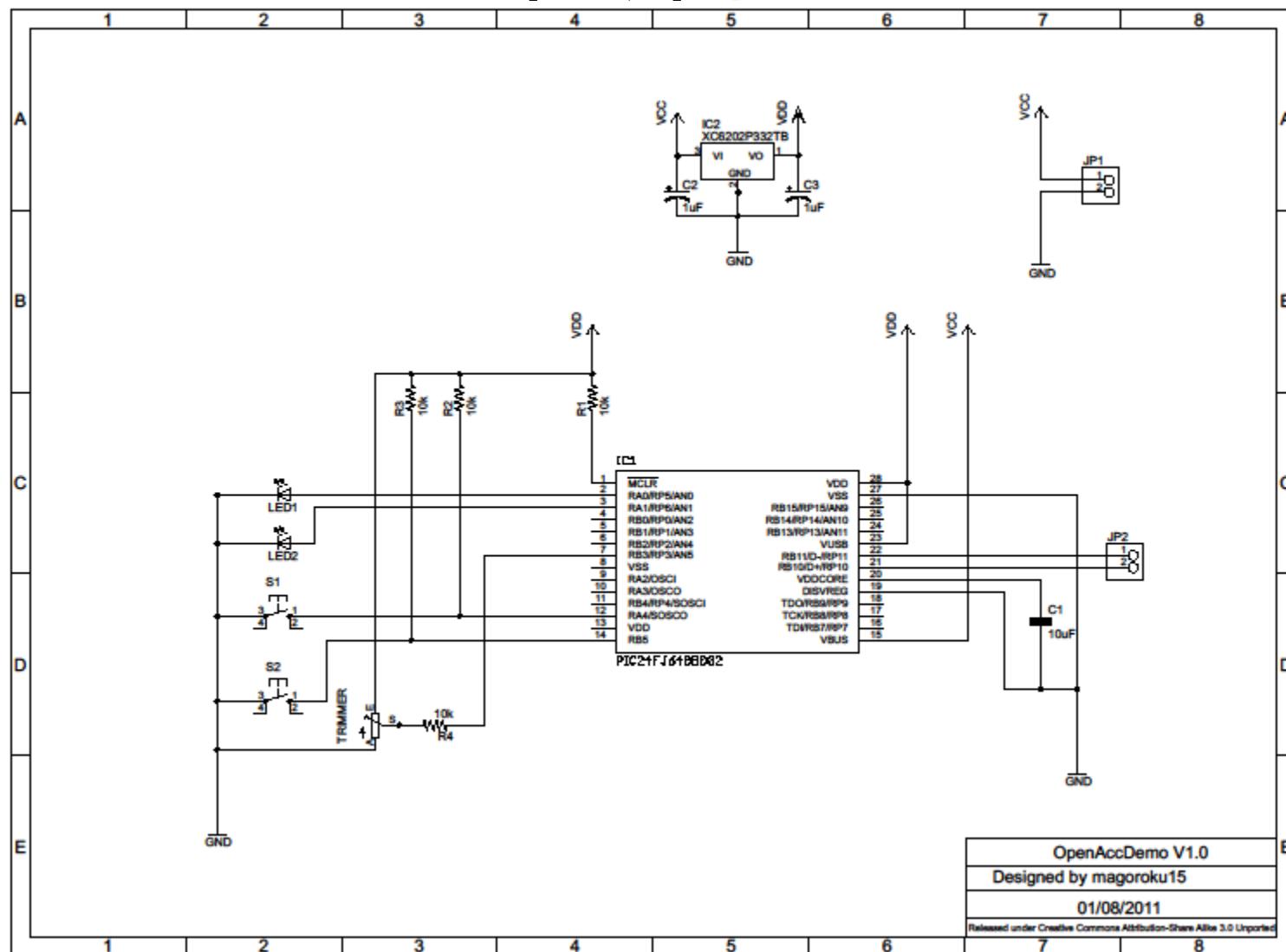
- 電源、黒をj2,赤をj3
  - データ 白(D-)をh16, 緑(D+)をh17



# 完成図と写真



# 回路図



# **PART2**

# **ANDROIDとの接続と動作確認**

# Step1 アプリのインストール

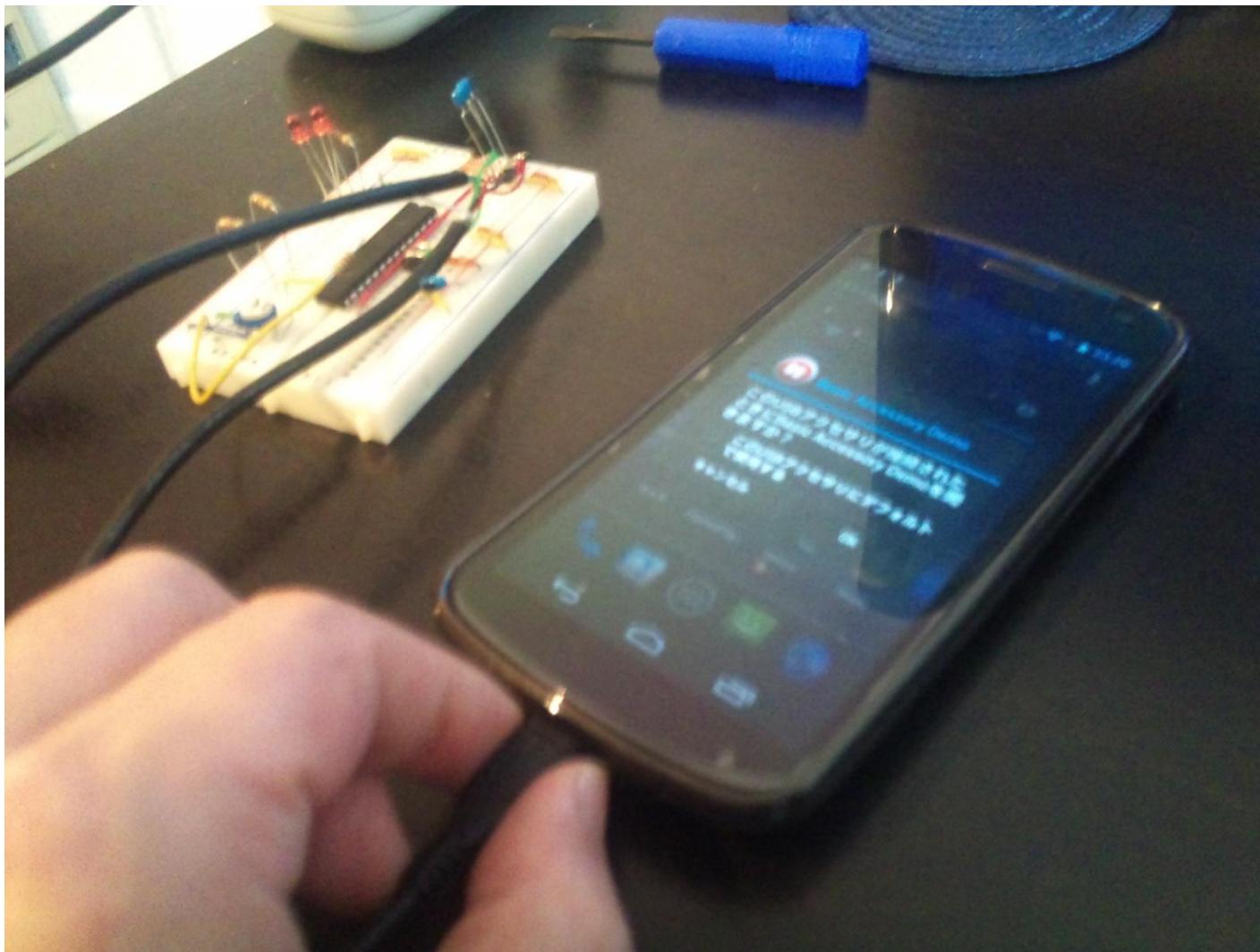
- Android Marketからインストール
  - Microchipで検索
    - 基本的なアクセサリデモ3.x
    - 基本的なアクセサリデモ2.3.x以降



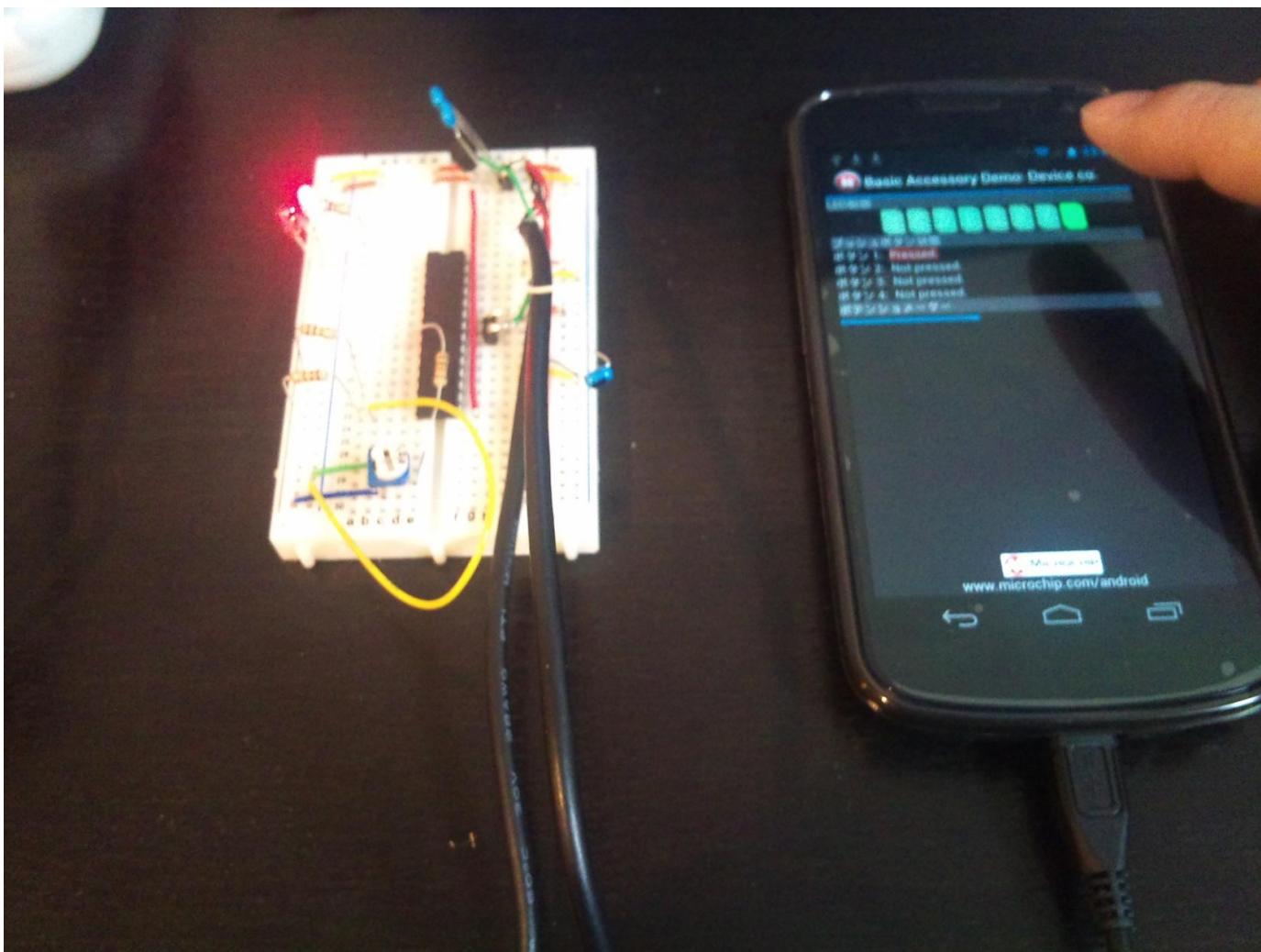
# Step2 ACアダプタに接続



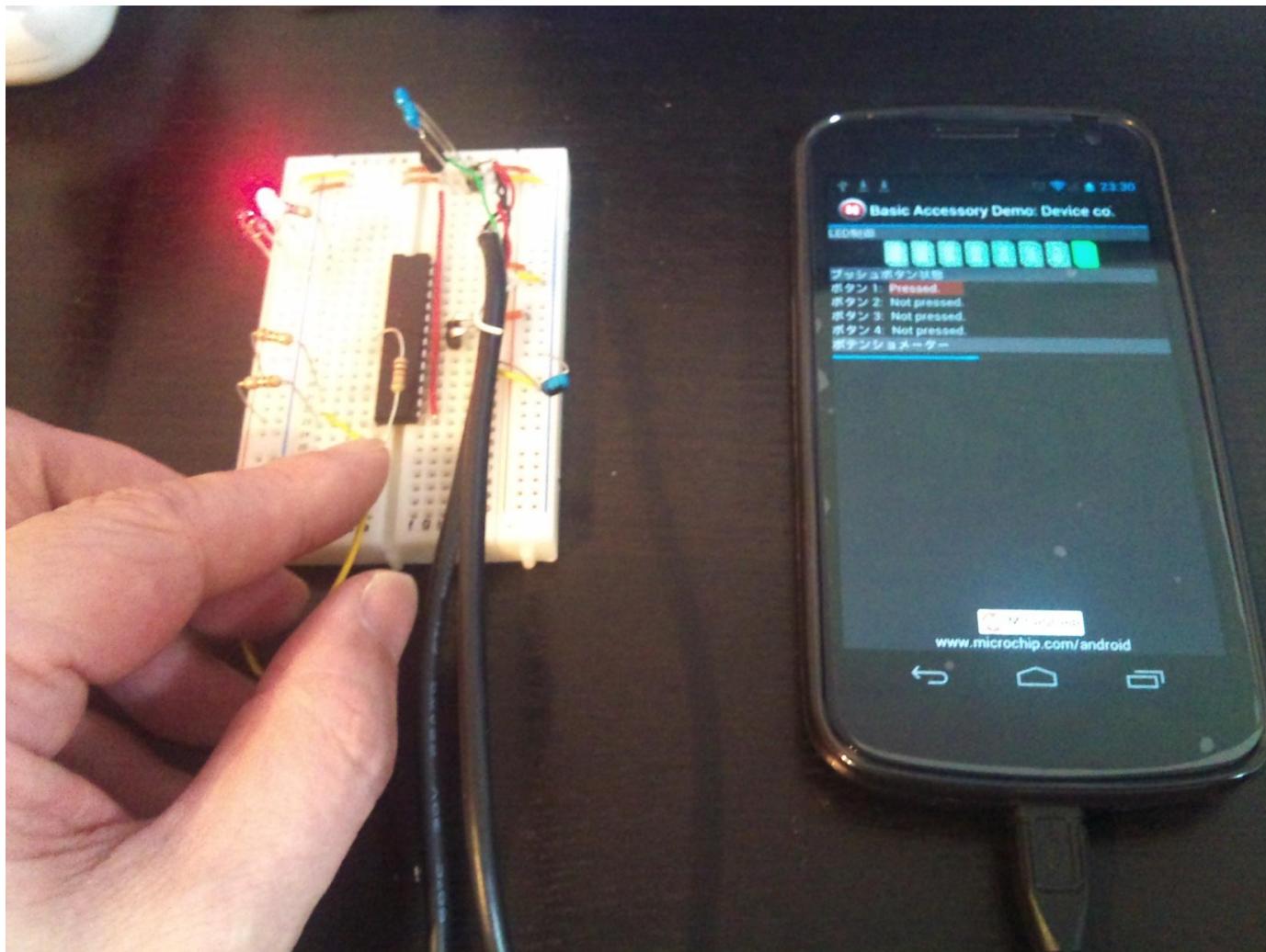
# Step3 Androidに接続



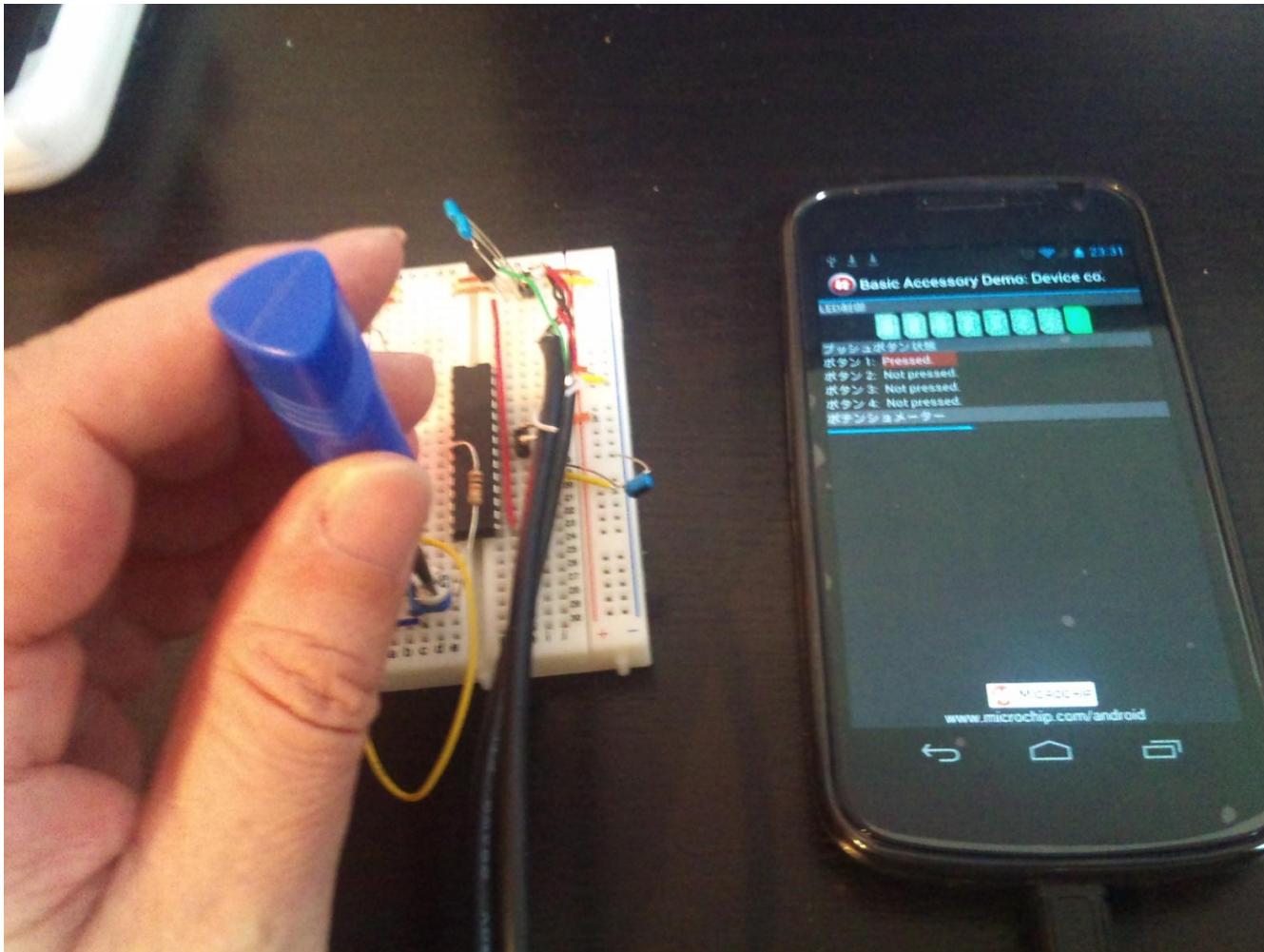
# LEDを点灯



# スイッチを操作



# ポテンションメータを操作



# **PART3**

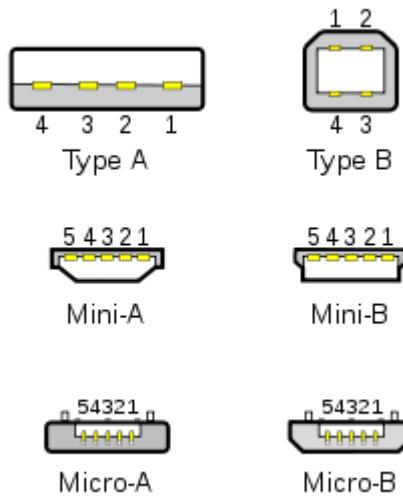
# **USBとOPEN ACCESSORY**

# USB ホスト・デバイス

- 基本は高速のシリアル通信
- 4線式
  - 通信はディファレンシャル D+,D-
  - 納電機能を持つ 5V, GND
- ホスト
  - 一般的にPC側
  - マスタとして動作
  - 複数のデバイスを収容・管理
  - 複雑なソフトウェアスタック
- デバイス
  - 一般的に装置側、マウス、KBD、プリンタ、USBメモリ
  - スレーブとして動作
  - 単純な機能

# USB コネクタ

- 4線式
  - ホスト側 1: + 2: D- 3: D+ 4: -
  - ディバイス側 Type A
  - Type B
- 5線式
  - IDでホスト、ディバイスを判別 1: + 2: D- 3: D+ 4: ID 5: -



Wikipediaより

# Androidでの利用例

- ホスト
  - システム側での対応は限定的、マウス対応(ICS)など
  - アプリ側でUSB managerと連携
  - ドライバ相当の動作をアプリとして記述・配布
- ディバイス
  - PCがホスト、Androidがディバイス
  - ADB, Mass Storage
- Open Accessory
  - PCがホスト、Androidがディバイス
  - ADBに類似
  - システムが利用していたエンドポイントをアプリに開放

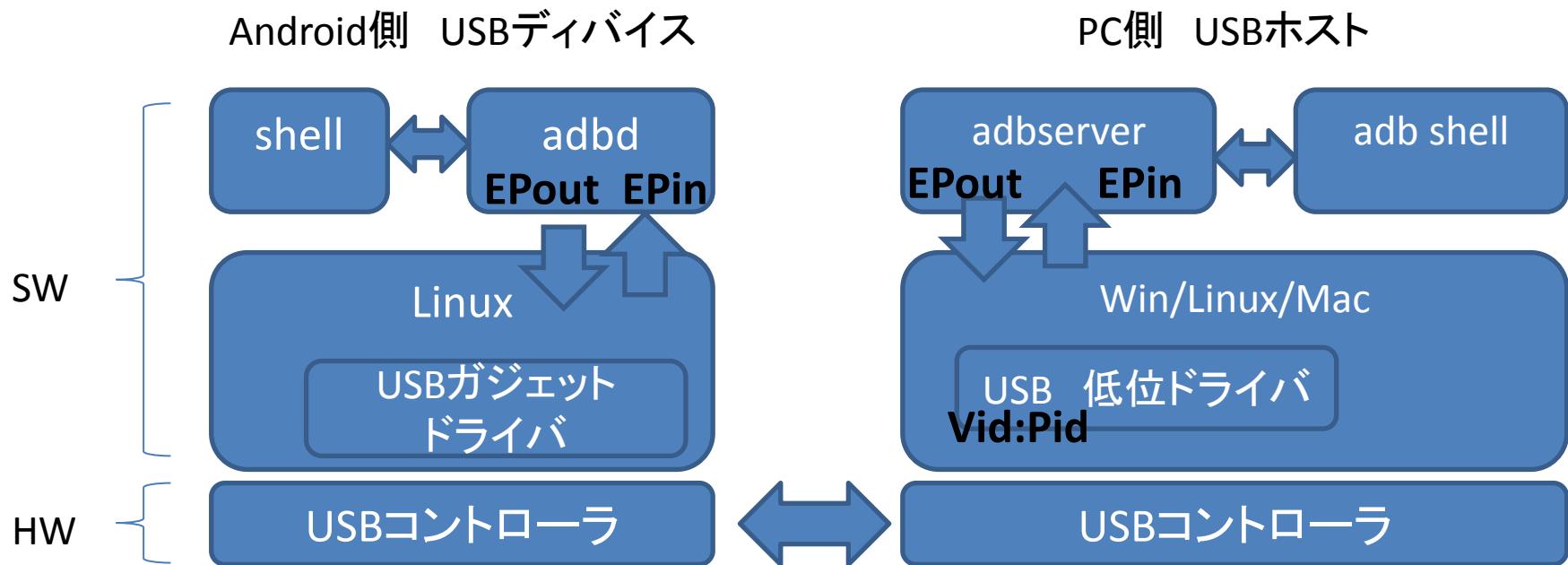
# ADBのしくみ

- Android:ドライバレイヤ
  - ADB用のVid, Pidを登録してエンドポイントを作成
- Android:アプリレイヤ
  - システム起動時にadbdコマンドを起動
- PCに接続すると
  - 特定のVid, Pidを示し、ADBの接続をPC側が認識
  - 専用Endpointがみえるのでこれをadbコマンドが握る

PC adbコマンド ⇄ PC バルク転送 ⇄ Android バルク転送 ⇄ /dev/adb ⇄ adbd

# ADBの構成

- \$adb shellを実行した場合のデータパス



# ADBのID定義

- C:\Program Files\Android\android-sdk-windows\extras\google\usb\_driver\android\_winusb.inf

[Google.NTx86]

; HTC Dream

```
%SingleAdbInterface%      = USB_Install, USB\VID_0BB4&PID_0C01
%CompositeAdbInterface%   = USB_Install, USB\VID_0BB4&PID_0C02&MI_01
%SingleBootLoaderInterface% = USB_Install, USB\VID_0BB4&PID_0FFF
```

; HTC Magic

```
%CompositeAdbInterface%   = USB_Install, USB\VID_0BB4&PID_0C03&MI_01
```

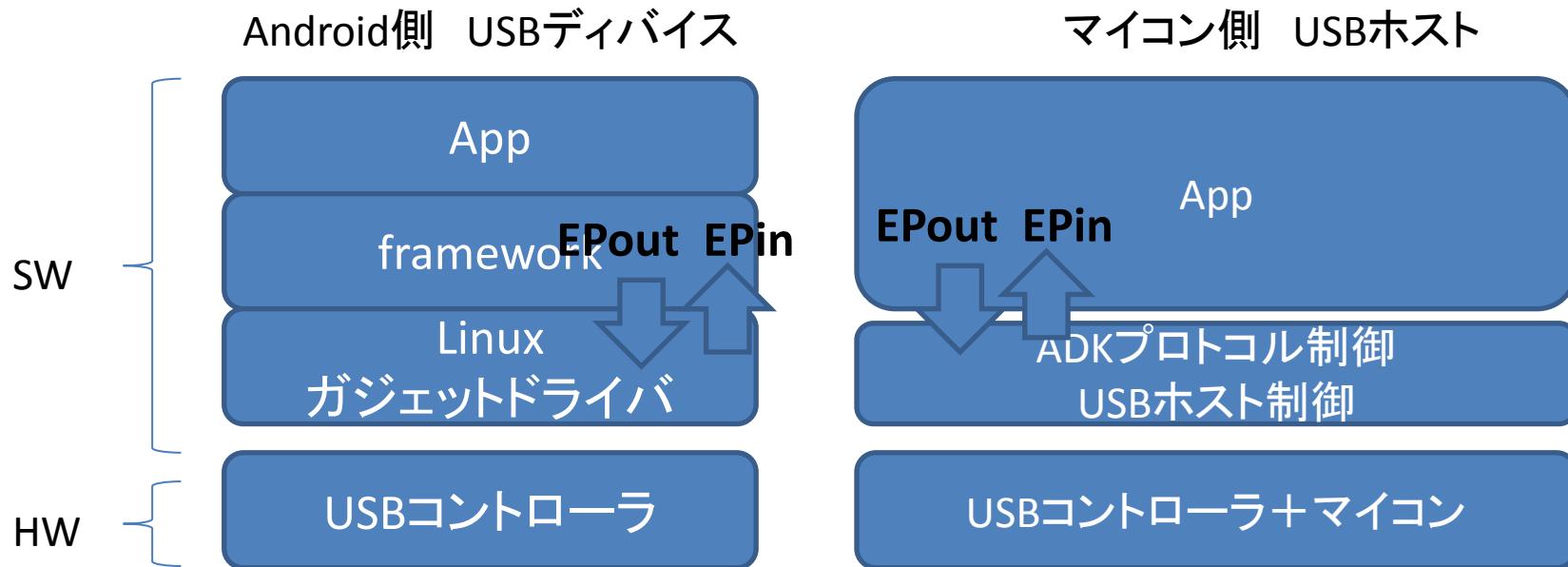
;

# Open Accessoryの構成

- 従来のAndroid側のUSBディバイスは
  - 特定の機能に限定
  - ユーザに非開放 アプリからは使えない
- 以前から、Beagleboard等の評価ボード乗りの間ではadbの拡張で自由な機能拡張が可能と認識されていた

# Open Accessory modeとは何か

- adbなどのUSBガジェットの仕組みを流用
- 専用エンドポイントを提供
- フレームワークがアプリを起動し、エンドポイントを接続



# Open Accessory protocol

1. マイコン側でUSBの接続を検出
2. Vid:Pidの0x18d1:0x2d00又は0x18d1:0x2d01かを確認できればaccessory mode 接続完了
3. ベンダ固有のIDが見えている場合があるので、'51'を送ってみる
4. 0以外が帰ってきたら、'52'で次の識別文字を送る
  1. manufacturer name: 0
  2. model name: 1
  3. description: 2
  4. version: 3
  5. URI: 4
  6. serial number: 5
5. 次に'53'を送り、2.へ



後で出でます

# 参考:adbとaccessoryは兄弟

```
$ cd Eee_PAD_TF101/Kernel_V9_4_2_7/drivers/usb/gadget  
$ diff -i f_accessory.c f_adb.c
```

```
628c499  
<     struct acc_dev           *dev = func_to_dev(f);  
---  
>     struct adb_dev           *dev = func_to_adb(f);  
632,634c503,504  
<     DBG(cdev, "acc_function_bind dev: %p\n", dev);  
<  
<     dev->start_requested = 0;  
---  
>     dev->cdev = cdev;  
>     DBG(cdev, "adb_function_bind dev: %p\n", dev);  
640c510  
<     acc_interface_desc.bInterfaceNumber = id;  
---  
>     adb_interface_desc.bInterfaceNumber = id;  
643,644c513,514  
<     ret = create_bulk_endpoints(dev, &acc_fullspeed_in_desc,  
<                               &acc_fullspeed_out_desc);  
---  
>     ret = adb_create_bulk_endpoints(dev, &adb_fullspeed_in_desc,  
>                                       &adb_fullspeed_out_desc);  
650,653c520,523  
<             acc_highspeed_in_desc.bEndpointAddress =  
<                         acc_fullspeed_in_desc.bEndpointAddress;  
<             acc_highspeed_out_desc.bEndpointAddress =  
<                         acc_fullspeed_out_desc.bEndpointAddress;  
---  
>             adb_highspeed_in_desc.bEndpointAddress =  
>                         adb_fullspeed_in_desc.bEndpointAddress;  
>             adb_highspeed_out_desc.bEndpointAddress =  
>                         adb_fullspeed_out_desc.bEndpointAddress;  
663c533  
< acc_function_unbind(struct usb_configuration *c, struct usb_function *f)  
---  
> adb_function_unbind(struct usb_configuration *c, struct usb_function *f)  
665c535  
<     struct acc_dev           *dev = func_to_dev(f);  
---  
>     struct adb_dev           *dev = func_to_adb(f);
```

# **PART4**

## **マイコン側の仕組み**

# PICマイコン

- マイコンとしては老舗
- ライバルはAtmel AVR, ARM cortex-M
- PIC10,PIC12,PIC16,PIC18,PIC24,dsPIC
  - 独自アーキ
- PIC32シリーズ
  - MIPS社からライセンスしたMIPS M4K

# マイコン(マイクロコントローラ)

- 小型、省電力のコントローラ
  - CPU
  - ROM(Flash),RAM内蔵 数K～数100Kbyte
  - 各種コントローラ内臓
    - GPIO
    - USART,I2C,AD変換,DA変換,PWM等
    - USB device, USB host
- PCでコンパイルして、ROM(Flash)に書き込んで利用

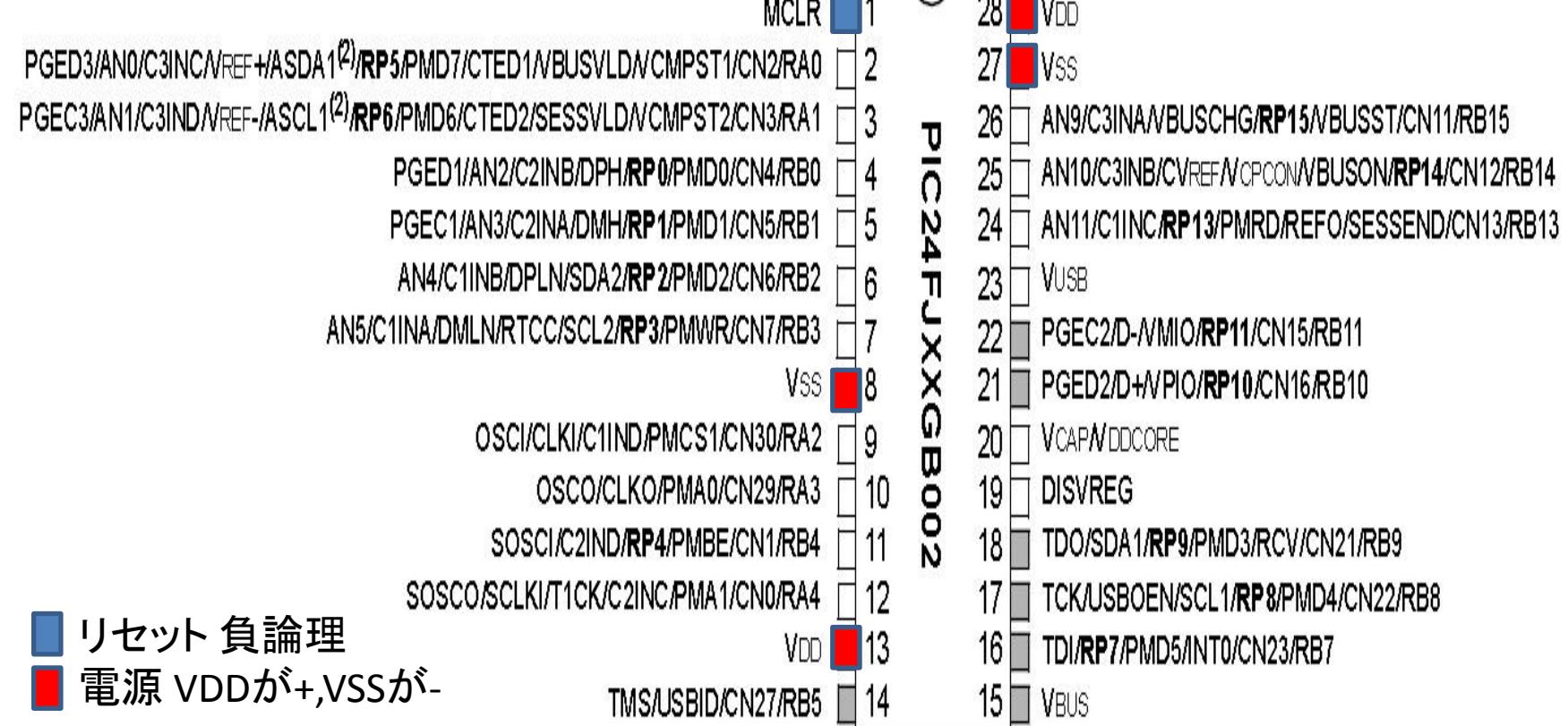
# PIC24FJ64GB002の概要

- クロック最大32MHz/16MIPS
- ROM64Kbyte/RAM 8Kbyte
- 周辺回路
  - GPIO,ADC,PWM,UART,I2CSPI

参考までに、1979年発売のPC-8001

クロック4MHz/ROM 24Kbyte/RAM 16Kbyte/168,000円

# PIC24FJ64GB002の電源・リセット



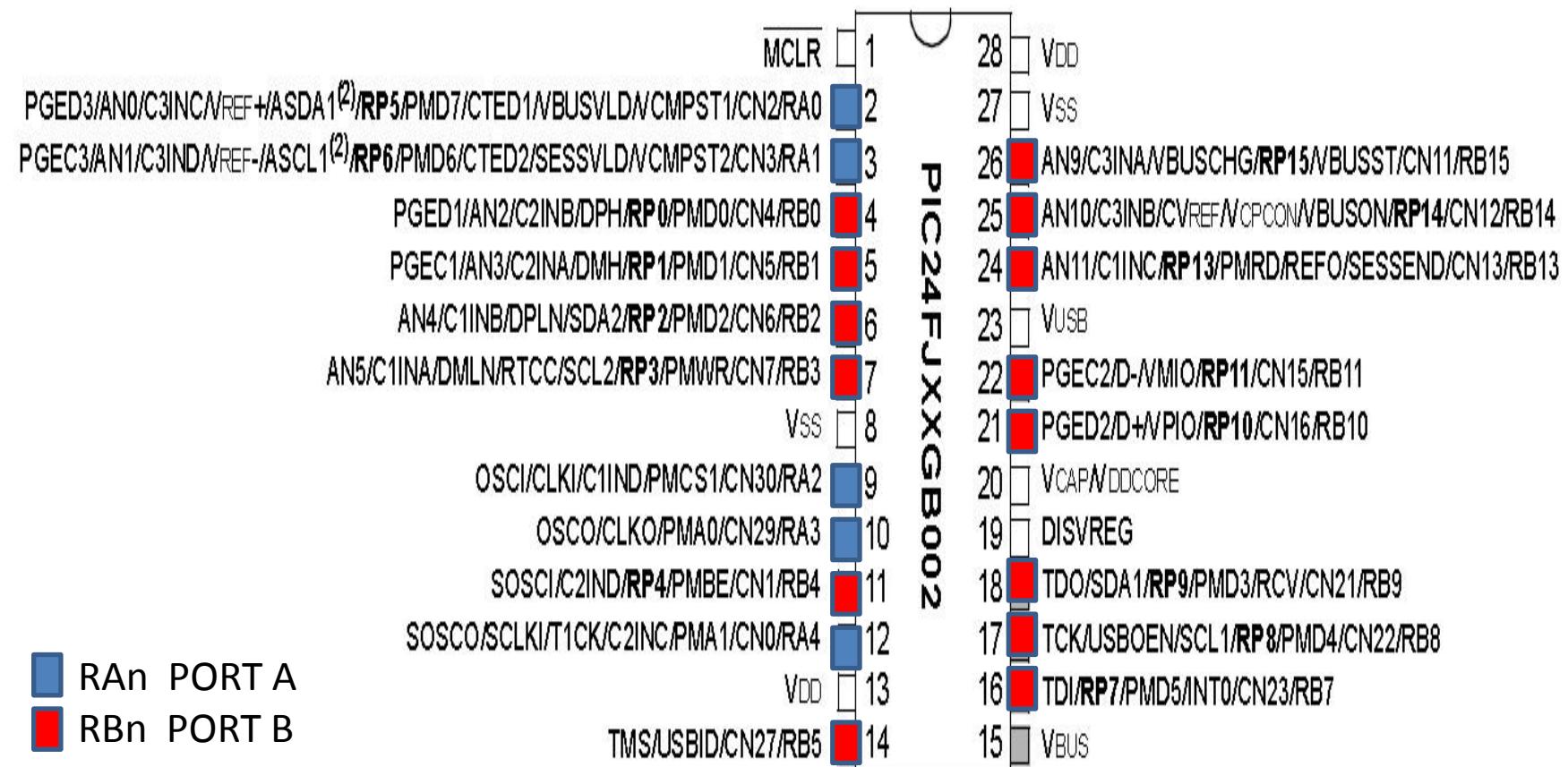
VDDとVSSに所定の電圧をかけてMCLRを軽くVDDに接続すればROMのプログラムが動作

# GPIO —汎用IO

- 出力
  - 特定のレジスタのビット操作で信号をON,OFF
- 入力
  - ピン(電極)の信号のON,OFFが特定のレジスタのビットを変更
- 1つのピンは入力・出力のどちらかを選択して利用する

レジスタのマップなどはPIC24FJ64GB004 Family Data Sheetから引用  
<http://ww1.microchip.com/downloads/en/DeviceDoc/39940d.pdf>

# PIC24FJ64GB002のGPIO



# GPIO-レジスタマップ

- TRISA,TRISB  
0:出力 1:入力  
入出力の向きを決めるレジスタ
- PORTA,PORTB  
0:ON 1:OFF  
入力で使うレジスタ
- LATA,LATB  
0:ON 1:OFF  
出力で使うレジスタ

TABLE 4-12: PORTA REGISTER MAP

File Name	Addr	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10 <sup>(1)</sup>	Bit 9 <sup>(1)</sup>	Bit 8 <sup>(1)</sup>	Bit 7 <sup>(1)</sup>	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	All Resets
TRISA	02C0	—	—	—	—	—	TRISA10	TRISA9	TRISA8	TRISA7	—	—	TRISA4	TRISA3	TRISA2	TRISA1	TRISA0	079F
PORTA	02C2	—	—	—	—	—	RA10	RA9	RA8	RA7	—	—	RA4	RA3	RA2	RA1	RA0	xxxx
LATA	02C4	—	—	—	—	—	LATA10	LATA9	LATA8	LATA7	—	—	LATA4	LATA3	LATA2	LATA1	LATA0	xxxx
ODCA	02C6	—	—	—	—	—	ODA10	ODA9	ODA8	ODA7	—	—	ODA4	ODA3	ODA2	ODA1	ODA0	0000

Legend: — = unimplemented, read as '0'. Reset values are shown in hexadecimal. Reset values shown are for 44-pin devices.

Note 1: Bits are unimplemented in 28-pin devices; read as '0'.

TABLE 4-13: PORTB REGISTER MAP

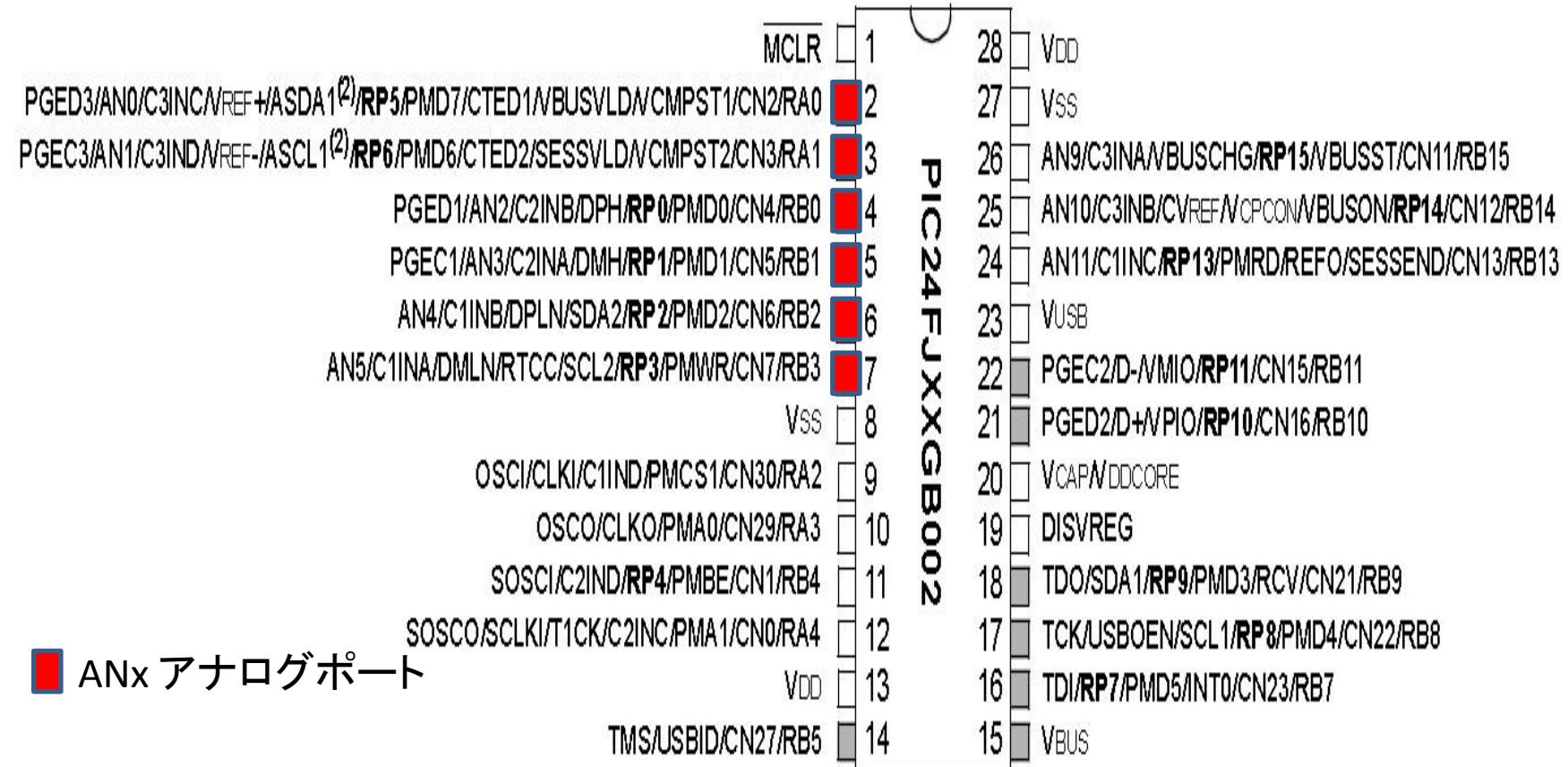
File Name	Addr	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	All Resets
TRISB	02C8	TRISB15	TRISB14	TRISB13	—	TRISB11	TRISB10	TRISB9	TRISB8	TRISB7	—	TRISB5	TRISB4	TRISB3	TRISB2	TRISB1	TRISB0	EFFF
PORTB	02CA	RB15	RB14	RB13	—	RB11	RB10	RB9	RB8	RB7	—	RB5	RB4	RB3	RB2	RB1	RB0	xxxx
LATB	02CC	LATB15	LATB14	LATB13	—	LATB11	LATB10	LATB9	LATB8	LATB7	—	LATB5	LATB4	LATB3	LATB2	LATB1	LATB0	xxxx
ODCB	02CE	ODB15	ODB14	ODB13	—	ODB11	ODB10	ODB9	ODB8	ODB7	—	ODB5	ODB4	ODB3	ODB2	ODB1	ODB0	0000

Legend: — = unimplemented, read as '0'. Reset values are shown in hexadecimal.

# ADC analog to digital converter

- 電圧で状態を出力する装置の値を読み取る
  - 温度計、湿度計、近接センサ、タッチパネル
- PIC24Fは10bit ADCなので
  - 0v～Nvまでを0～1023の解像度で読み取る
  - Nvは基準電圧で今回はVDDと同じ
- 今回のモジュールではセンサの代わりに可変抵抗を利用
  - 回転角、位置センサの読み取りとも言える

# PIC24FJ64GB002のADC



■ ANx アナログポート

# ADCのレジスタ

- 初期化
  - パラメタが多く複雑 レジスタ6個
  - 基準電圧=VDDの基本パターンを使いまわす
  - Firmwareソースのウォークスルーで紹介
- 読み取り
  - 時間を要する処理
    - 割り込み or ポーリング

# 開発環境

- MPLAB.X
  - 統合開発環境
  - 最近1.0に
  - NetBeansベース
    - MAC, Windows, Linuxの各プラットフォームで動作
- プログラマ
  - マイコン上のFlashにプログラムを転送
  - MPLAB.Xから書き込み
  - Pickit3



Pickit3 3900円  
秋月 通販コード M-03608

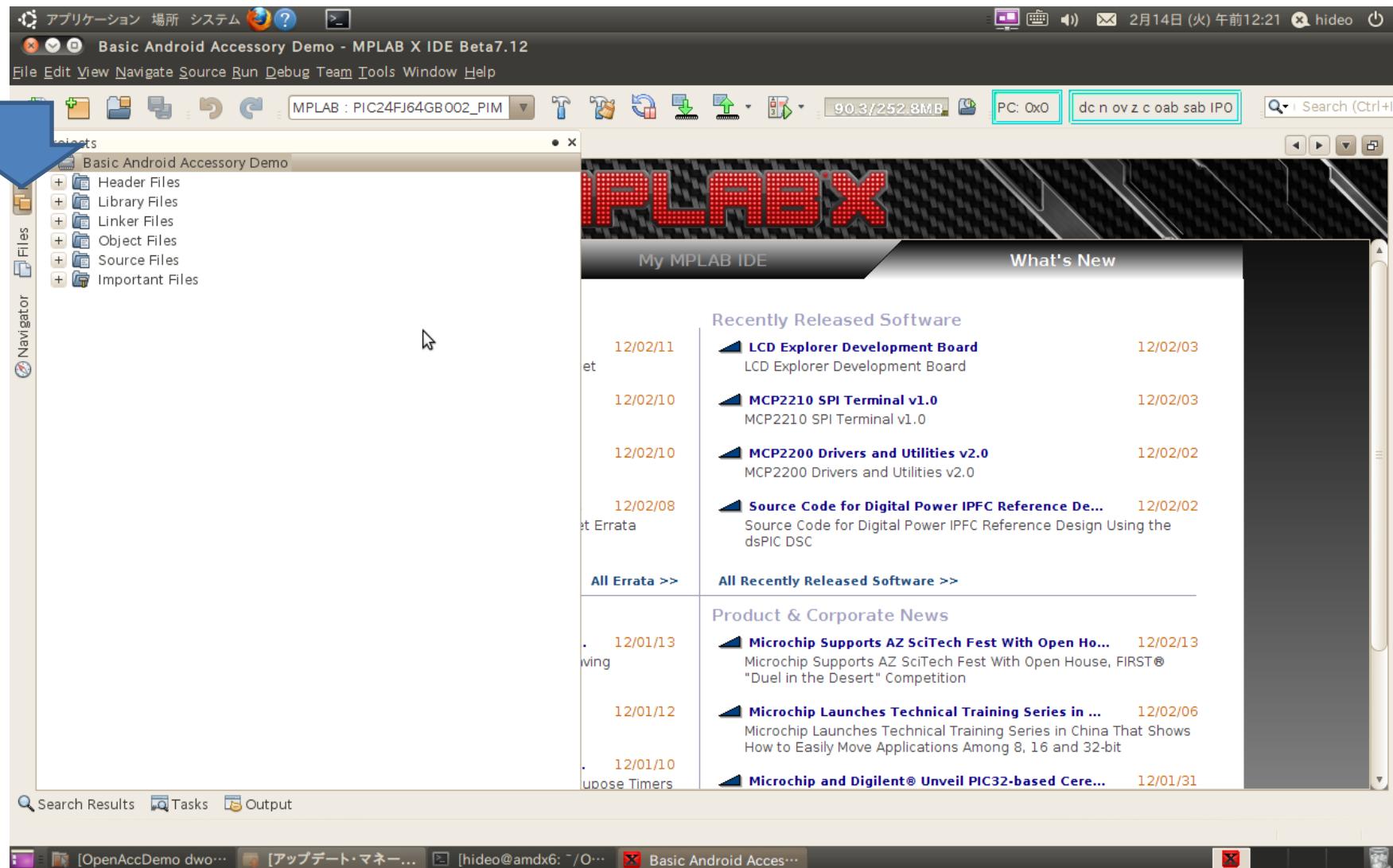
# MPLABXの使い方

- プロジェクトの手順
- ビルドの手順
- プログラムの手順

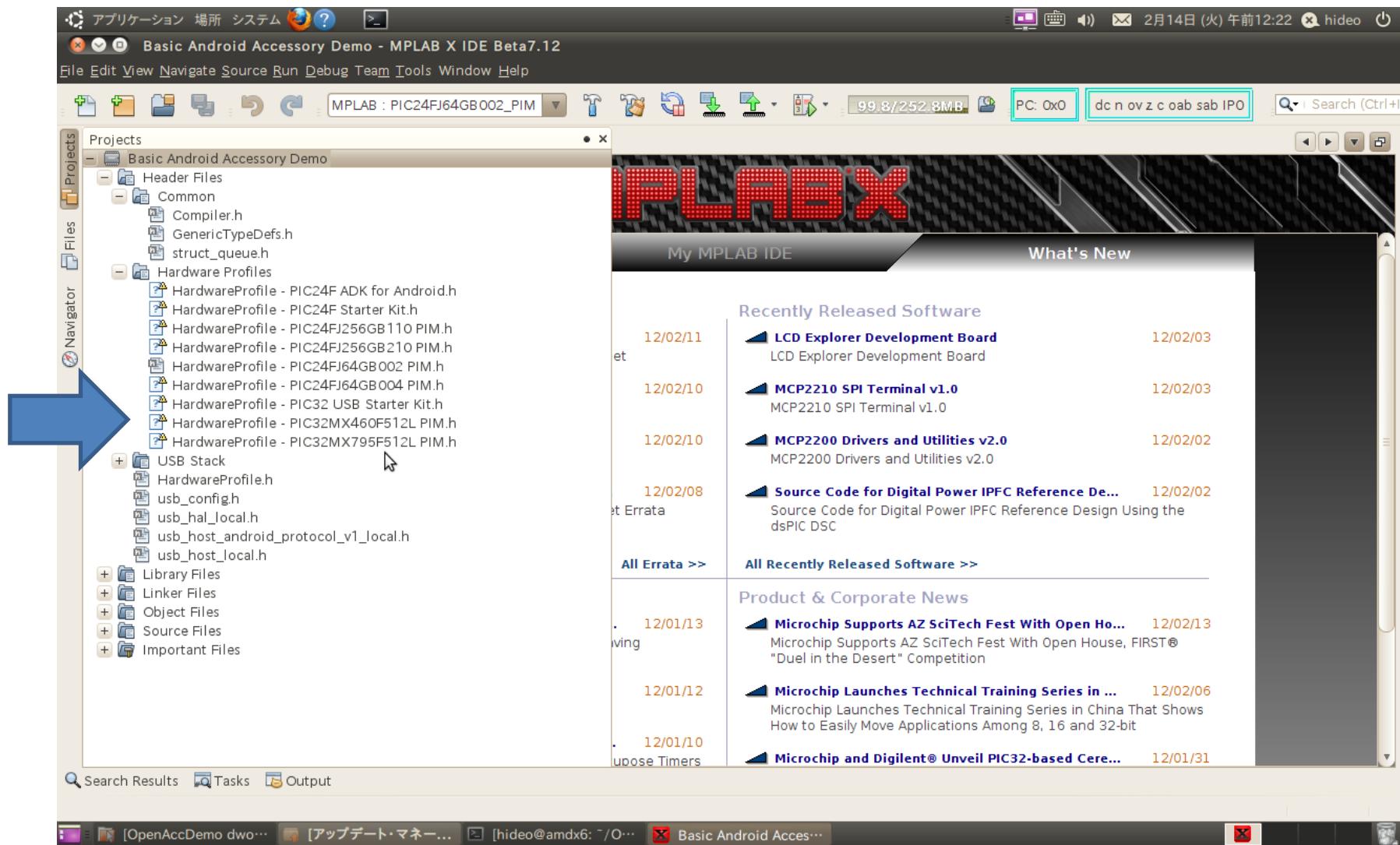
# プロジェクトのOpen

- 左上メニューからFile -> Open Project....
- プロジェクトを選択
  - OpenAccessoryDemo/Firmware/MPXLABを選択

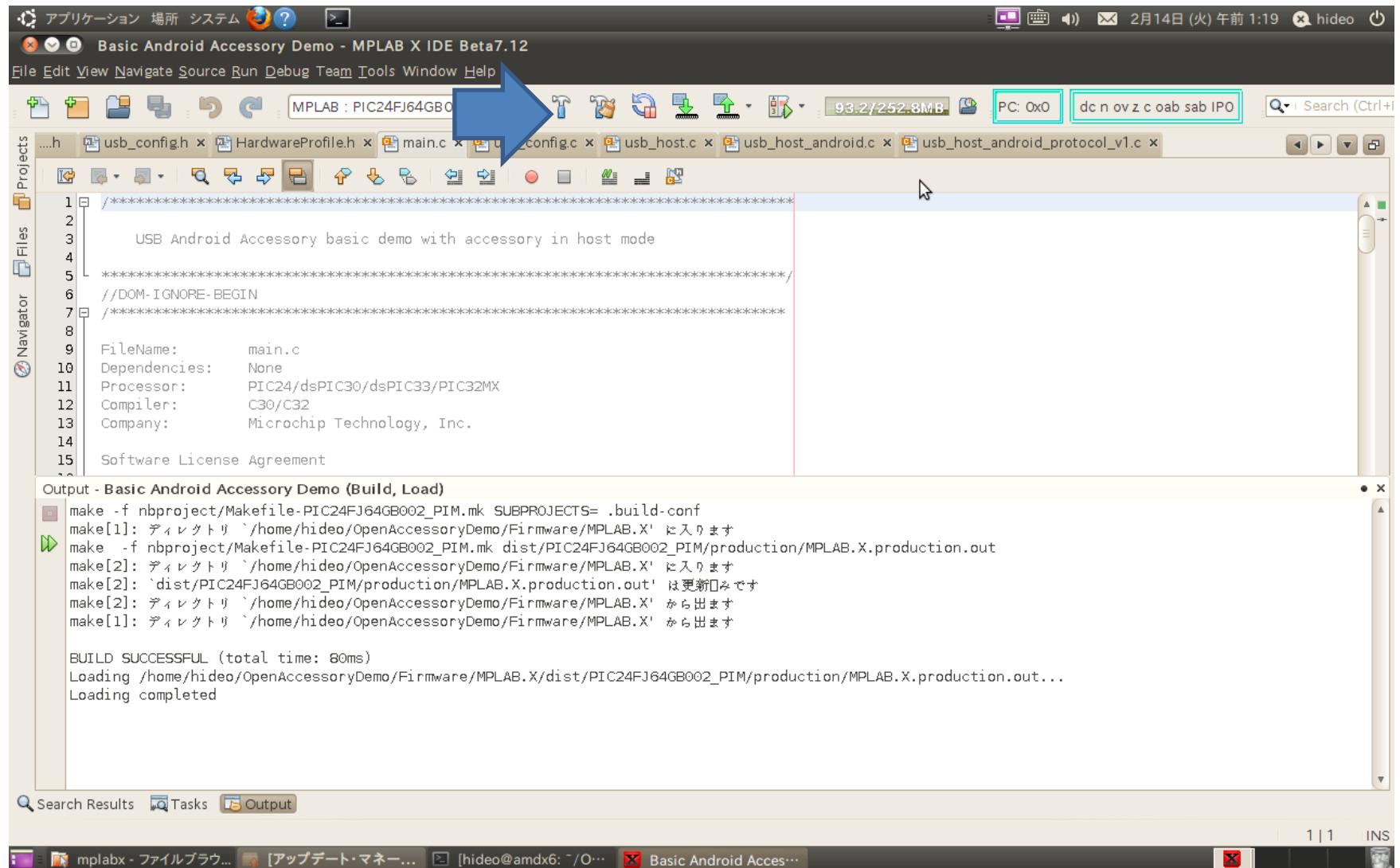
# ファイルの閲覧



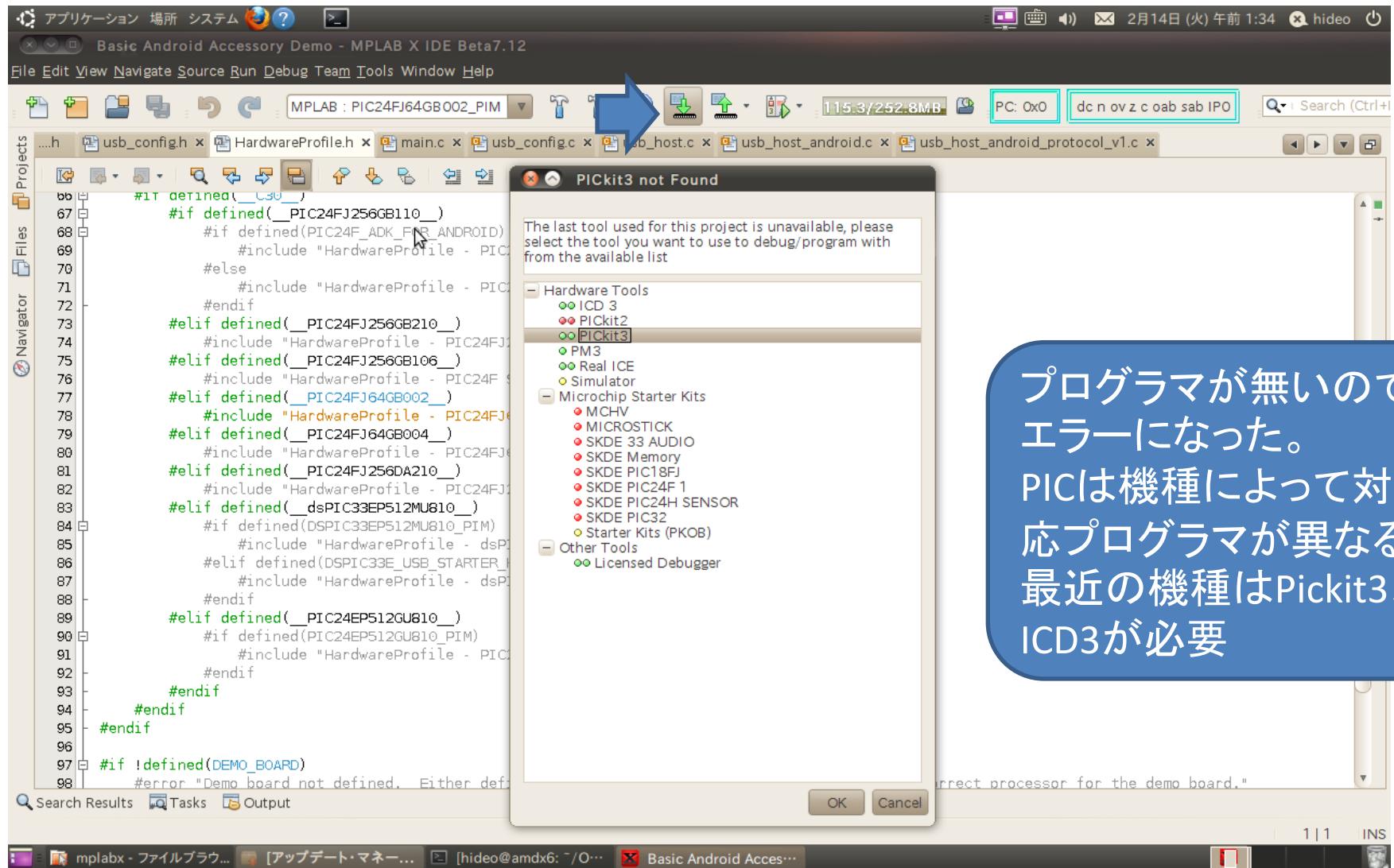
# ファイルを選択



# ビルト



# プログラム（Flashへの書き込み）

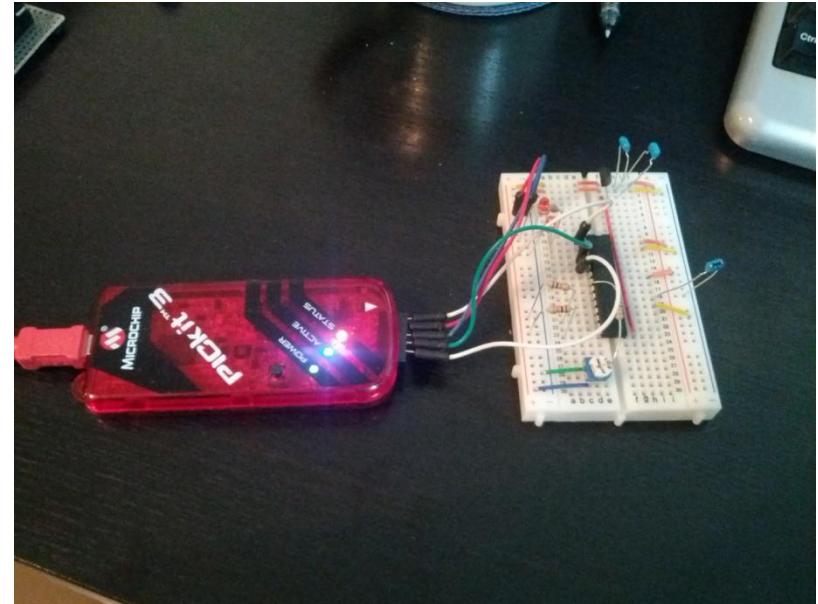


プログラマが無いので  
エラーになった。  
PICは機種によって対  
応プログラマが異なる。  
最近の機種はPickit3、  
ICD3が必要

# プログラマの接続



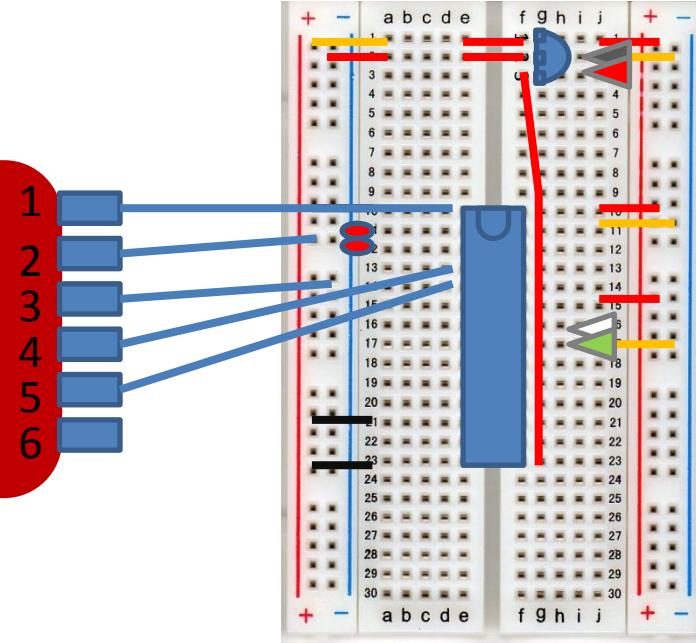
方式1)専用ソケットを用意



方式2)ICSP  
In Circuit Serial Programming  
基板上にPICをさしたまま、Flashへ書きこむ

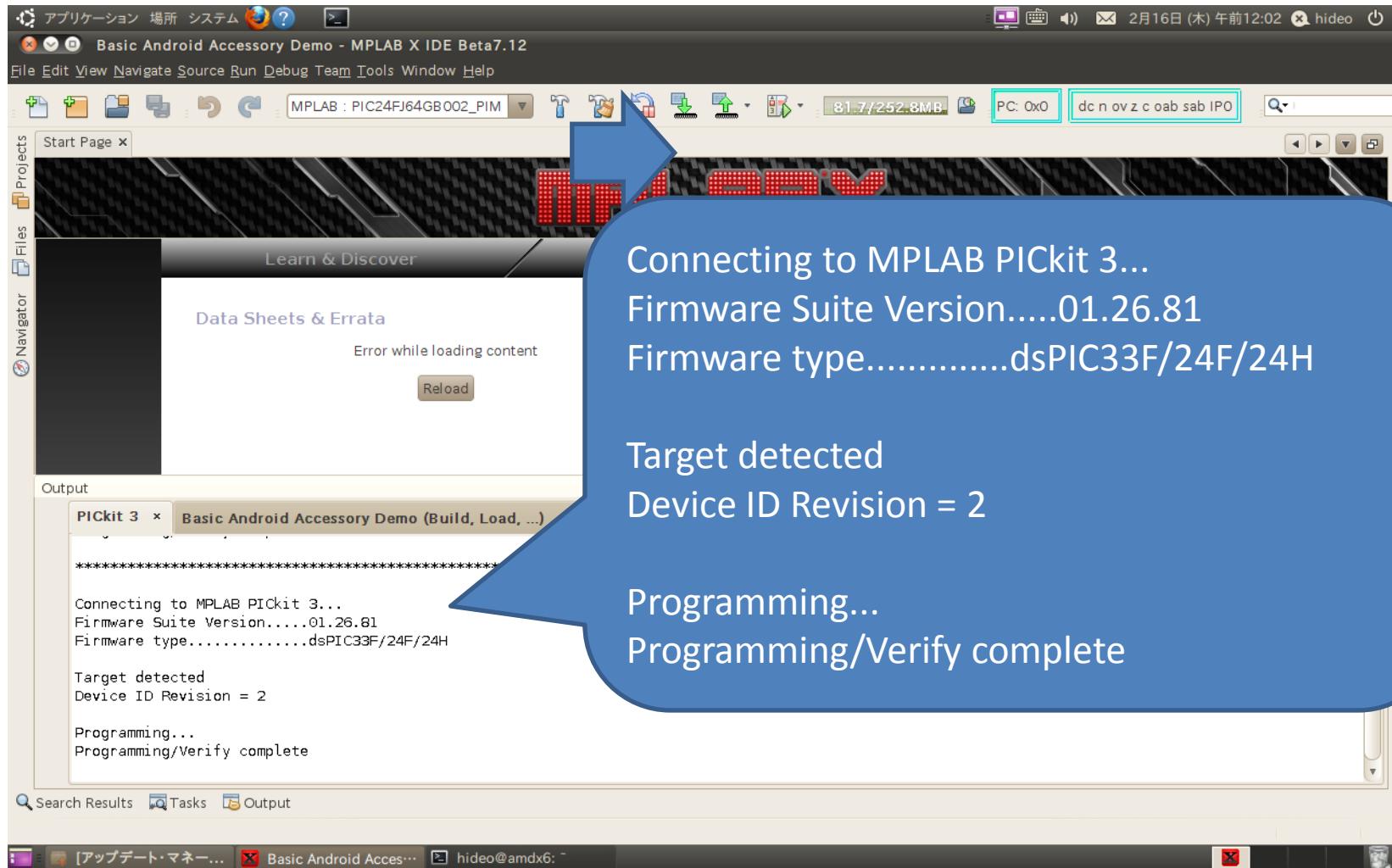
# プログラマの配線

PICKit 3



- 1をd10
- 2を+
- 3を-
- 4をd13
- 5をd14

# プログラム

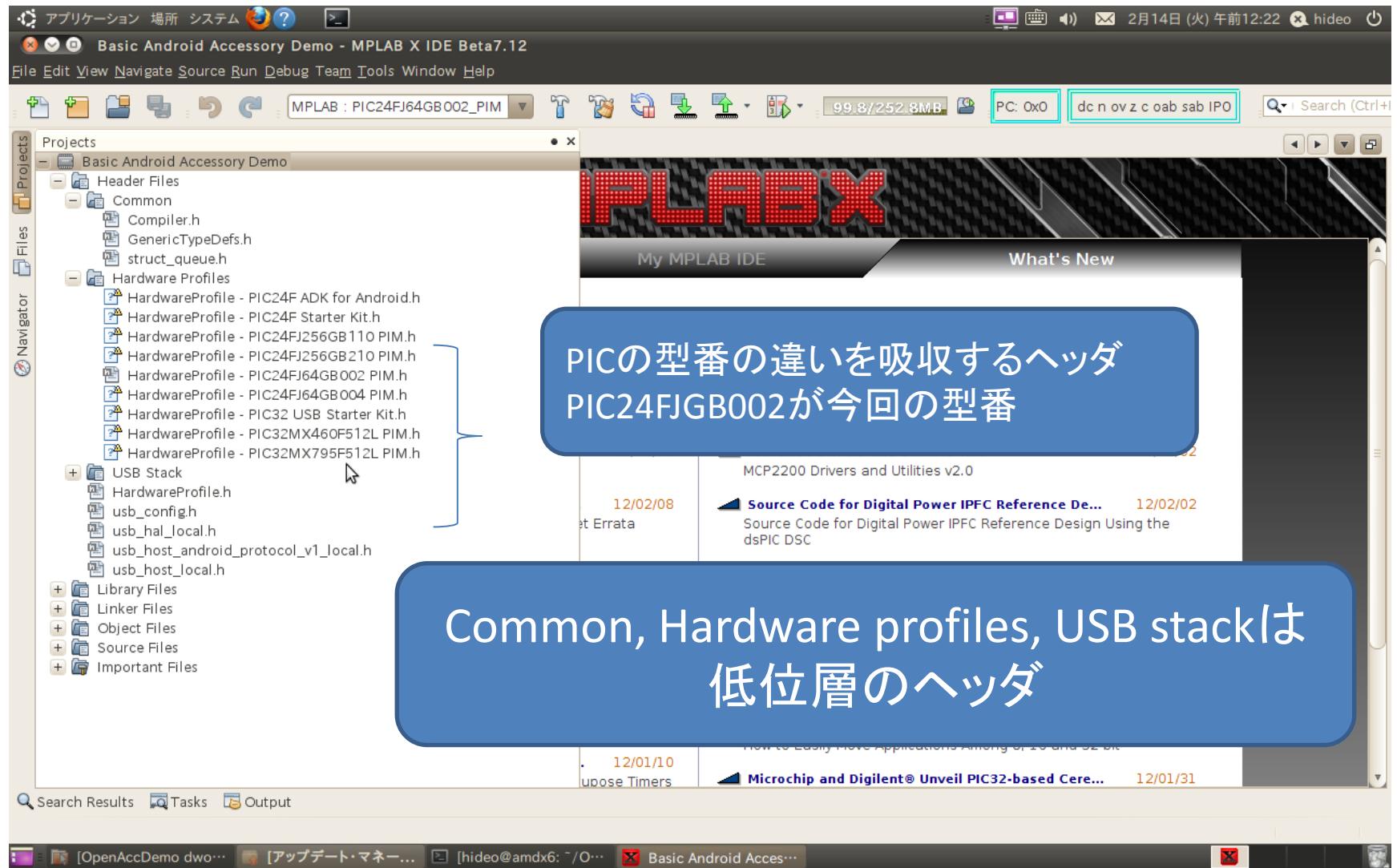


# Firmwareのウォークスルー

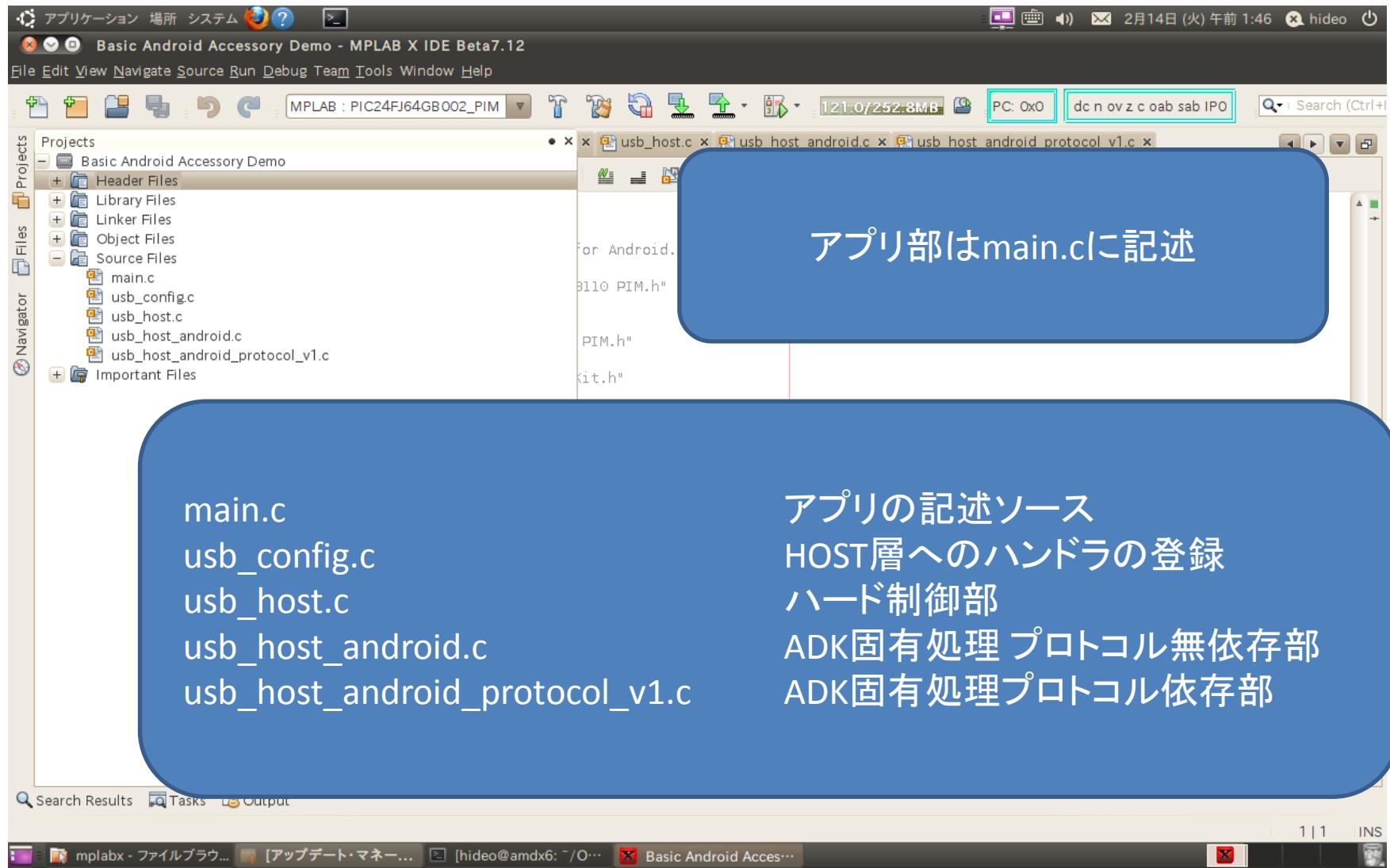
- Firmwareには、USBの低位層からアプリ層までのプログラムがソースで配置
- アプリ層のファイルは



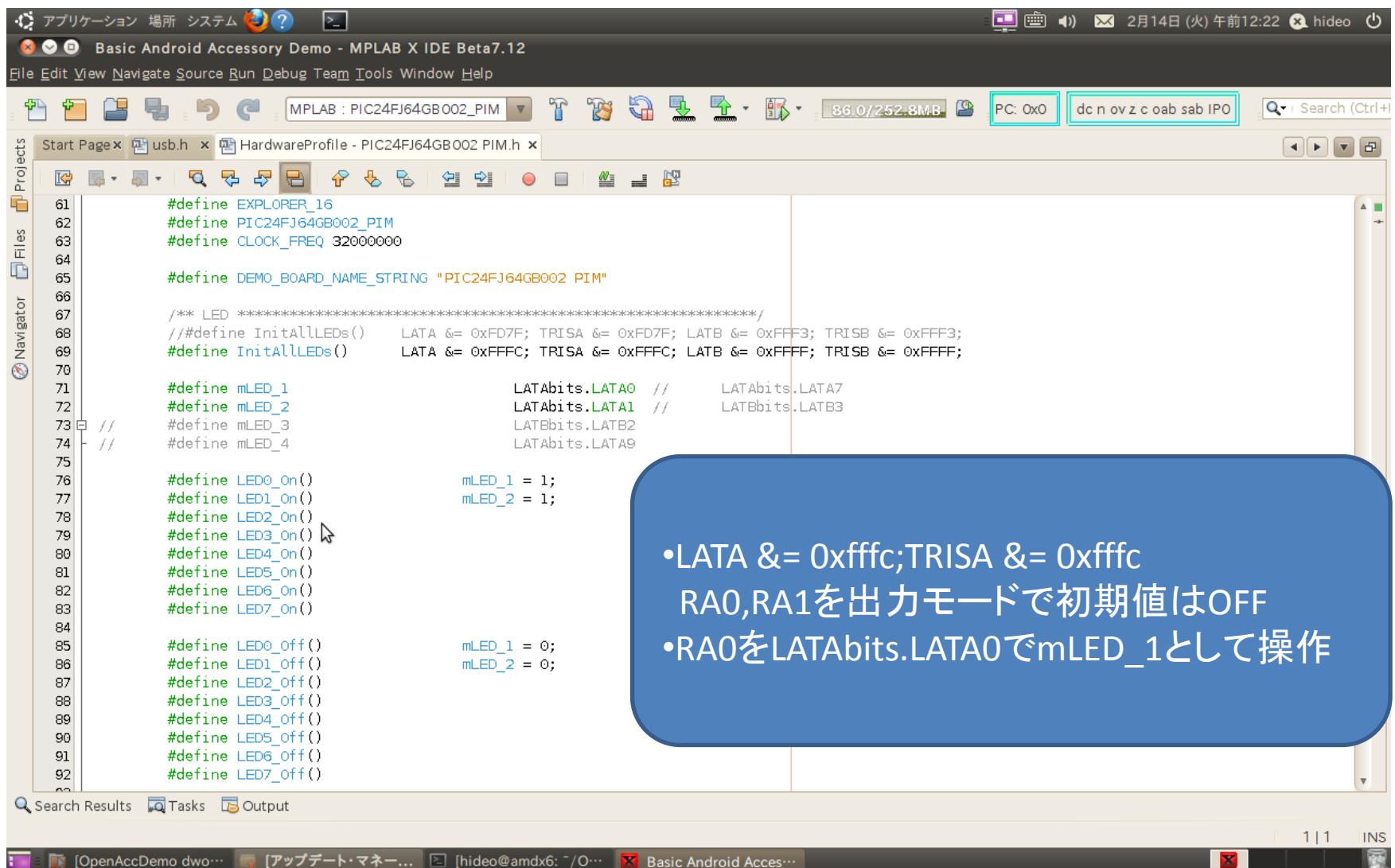
# 低位層のヘッダ



# ソースファイル



# GPIOポートの出力設定



The screenshot shows the MPLAB X IDE interface with the following details:

- Title Bar:** Basic Android Accessory Demo - MPLAB X IDE Beta7.1.2
- Toolbar:** Includes icons for file operations, build, and simulation.
- Status Bar:** Shows memory usage (86.0/252.8MB) and port status (PC: 0x0).
- Search Bar:** Contains the text "dc n ov z c oab sab IPO".
- Project Explorer:** Shows the project structure with files like Start Page, usb.h, and HardwareProfile - PIC24FJ64GB002\_PIM.h.
- Code Editor:** Displays the following C code snippet:

```
#define EXPLORER_16
#define PIC24FJ64GB002_PIM
#define CLOCK_FREQ 32000000

#define DEMO_BOARD_NAME_STRING "PIC24FJ64GB002_PIM"

/** LED *****/
//#define InitAllLEDs() LATA &= 0xFD7F; TRISA &= 0xFFD7F; LATB &= 0xFFFF3; TRISB &= 0xFFFF3;
#define InitAllLEDs() LATA &= 0xFFFFC; TRISA &= 0xFFFFC; LATB &= 0xFFFF; TRISB &= 0xFFFF;

#define mLED_1 LATAbits.LATA0 // LATAbits.LATA7
#define mLED_2 LATAbits.LATA1 // LATBbits.LATB3
#define mLED_3 LATBbits.LATB2
#define mLED_4 LATAbits.LATA9

#define LED0_On() mLED_1 = 1;
#define LED1_On() mLED_2 = 1;
#define LED2_On()
#define LED3_On() 
#define LED4_On()
#define LED5_On()
#define LED6_On()
#define LED7_On()

#define LED0_Off() mLED_1 = 0;
#define LED1_Off() mLED_2 = 0;
#define LED2_Off()
#define LED3_Off()
#define LED4_Off()
#define LED5_Off()
#define LED6_Off()
#define LED7_Off()
```

A blue callout bubble highlights the following points in the code:

- LATA &= 0xffffc; TRISA &= 0xffffc  
RA0, RA1を出力モードで初期値はOFF
- RA0をLATAbits.LATA0でmLED\_1として操作

# GPIOの入力設定

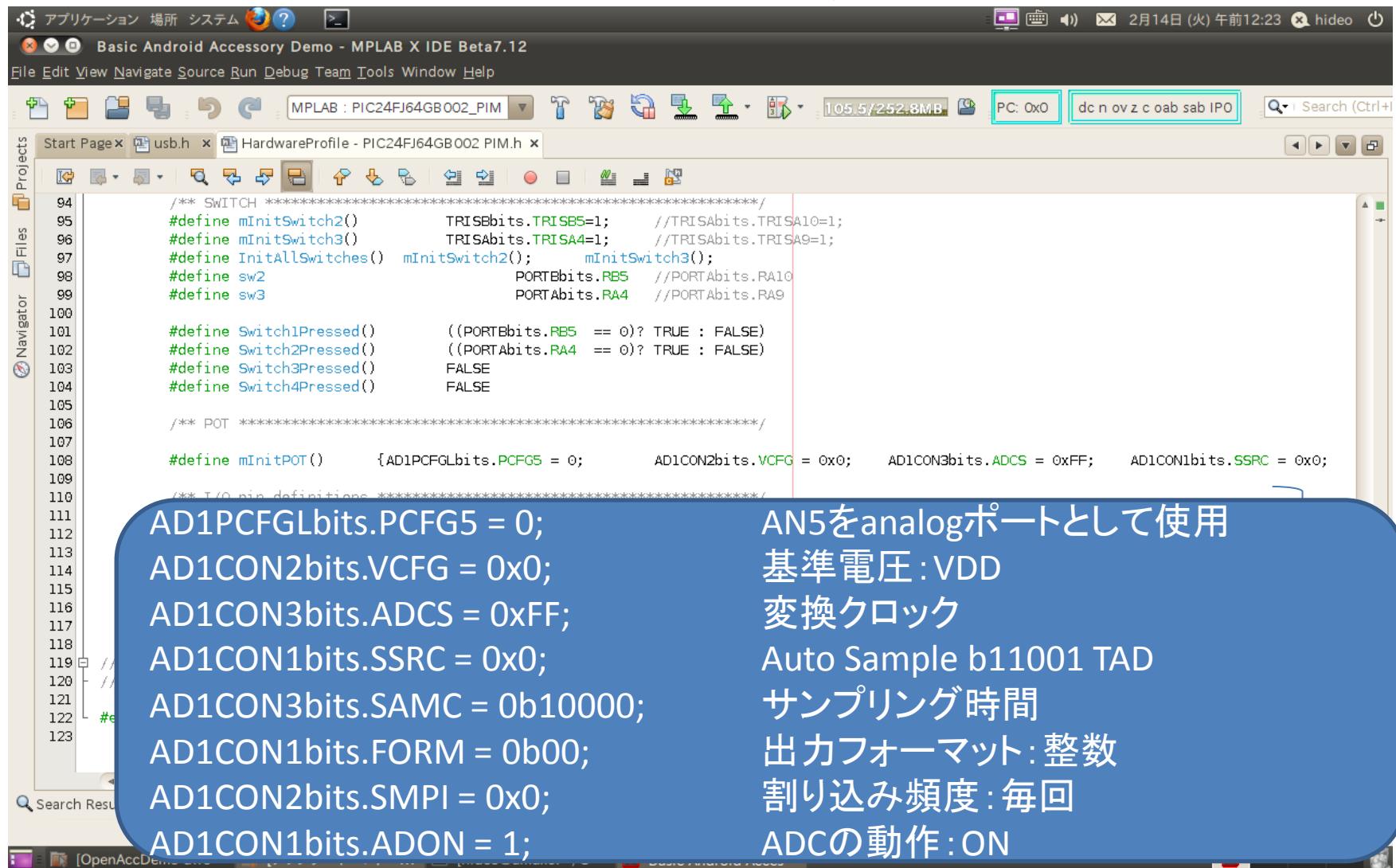
The screenshot shows the MPLAB X IDE interface with the following details:

- Title Bar:** Basic Android Accessory Demo - MPLAB X IDE Beta7.12
- Toolbar:** Includes icons for file operations, project management, and build.
- Status Bar:** Shows memory usage (105.5/252.8MB) and a search bar containing "PC: 0x0 dc n ov z c oab sab IPO".
- Code Editor:** Displays the "HardwareProfile - PIC24FJ64GB002 PIM.h" file. The code defines pin configurations and switch detection logic. A blue callout box highlights the following code snippets:
  - Line 94: `/* SWITCH *****/`
  - Line 95: `#define mInitSwitch2() TRISBbits.TRISB5=1; //TRISAbits.TRISA10=1;`
  - Line 96: `#define mInitSwitch3() TRISAbits.TRISA4=1; //TRISAbits.TRISA9=1;`
  - Line 97: `#define InitAllSwitches() mInitSwitch2(); mInitSwitch3();`
  - Line 98: `#define sw2 PORTBbits.RB5 //PORTAbits.RA10`
  - Line 99: `#define sw3 PORTAbits.RA4 //PORTAbits.RA9`
  - Line 101: `#define Switch1Pressed() ((PORTBbits.RB5 == 0)? TRUE : FALSE)`
  - Line 102: `#define Switch2Pressed() ((PORTAbits.RA4 == 0)? TRUE : FALSE)`
  - Line 103: `#define Switch3Pressed() FALSE`
  - Line 104: `#define Switch4Pressed() FALSE`
- Projects and Navigator:** Panels on the left side of the interface.
- Search Results:** Located at the bottom left.
- Task and Output:** Located at the bottom center.
- Page Number:** 1 / 1 INS

**Callout Box Content:**

- TRISBbits.TRISB5 = 1;  
RB5を入力モードで
- Switch1Pressed  
PORTBbits.RB5をテストしてTRUE/FALSE

# ADCの設定



The screenshot shows the MPLAB X IDE interface with the title bar "Basic Android Accessory Demo - MPLAB X IDE Beta7.12". The main window displays C code for initializing switches and a potentiometer (POT) on a PIC24FJ64GB002 microcontroller. A callout bubble highlights the initialization of the POT's PCFG5 bit and its associated ADC configuration bits.

```
94     /** SWITCH *****/
95     #define mInitSwitch2()          TRISBbits.TRISB5=1;      //TRISAbits.TRISA10=1;
96     #define mInitSwitch3()          TRISAbits.TRISA4=1;      //TRISAbits.TRISA9=1;
97     #define InitAllSwitches()      mInitSwitch2();           mInitSwitch3();
98     #define sw2                      PORTBbits.RB5        //PORTAbits.RA10
99     #define sw3                      PORTAbits.RA4        //PORTAbits.RA9
100
101    #define Switch1Pressed()       ((PORTBbits.RB5 == 0)? TRUE : FALSE)
102    #define Switch2Pressed()       ((PORTAbits.RA4 == 0)? TRUE : FALSE)
103    #define Switch3Pressed()       FALSE
104    #define Switch4Pressed()       FALSE
105
106    /** POT *****/
107
108    #define mInitPOT()             {AD1PCFGLbits.PCFG5 = 0;          AD1CON2bits.VCFG = 0x0;      AD1CON3bits.ADCS = 0xFF;      AD1CON1bits.SSRC = 0x0;
109
110    /** I/O pin definitions *****/
111
112    AD1PCFGLbits.PCFG5 = 0;
113    AD1CON2bits.VCFG = 0x0;
114    AD1CON3bits.ADCS = 0xFF;
115    AD1CON1bits.SSRC = 0x0;
116    AD1CON3bits.SAMC = 0b10000;
117    AD1CON1bits.FORM = 0b00;
118    AD1CON2bits.SMPI = 0x0;
119    AD1CON1bits.ADON = 1;
```

AD1PCFGLbits.PCFG5 = 0;  
AD1CON2bits.VCFG = 0x0;  
AD1CON3bits.ADCS = 0xFF;  
AD1CON1bits.SSRC = 0x0;  
AD1CON3bits.SAMC = 0b10000;  
AD1CON1bits.FORM = 0b00;  
AD1CON2bits.SMPI = 0x0;  
AD1CON1bits.ADON = 1;

AN5をanalogポートとして使用  
基準電圧:VDD  
変換クロック  
Auto Sample b11001 TAD  
サンプリング時間  
出力フォーマット:整数  
割り込み頻度:毎回  
ADCの動作:ON

# 型番依存部の切り替え

The screenshot shows the MPLAB X IDE interface with the following details:

- Title Bar:** Basic Android Accessory Demo - MPLAB X IDE Beta7.12
- Toolbar:** Includes File, Edit, View, Navigate, Source, Run, Debug, Team, Tools, Window, Help.
- Status Bar:** MPLAB : PIC24FJ64GB002\_PIM, 117.5/252.8MB, PC: 0x0, dc n ov z c oab sab IPO, Search (Ctrl+F).
- Projects View:** Shows a single project node.
- Files View:** Shows several header files: ...h, usb\_host\_local.h, usb\_host\_android\_protocol\_v1\_local.h, usb\_hal\_local.h, usb\_config.h, and HardwareProfile.h.
- Code Editor:** Displays the content of the ...h file. The code is a series of #if and #elif directives defining hardware profiles based on processor definitions. It includes profiles for PIC24FJ256GB110, PIC24FJ256GB210, PIC24FJ256GB106, PIC24FJ64GB002, PIC24FJ64GB004, PIC24FJ256DA210, dsPIC33EP512MU810, and PIC24EP512GU810. A note at the bottom states: "#if !defined(DEMO\_BOARD) #error "Demo board not defined. Either define DEMO BOARD for a custom board or select the correct processor for the demo board."
- Search Results, Tasks, Output:** Standard IDE navigation and search tools.

# 処理対象のVid,Pidの定義

The screenshot shows the MPLAB X IDE interface with the project "Basic Android Accessory Demo - MPLAB X IDE Beta7.12". The code editor displays C code for initializing a USB client driver table and defining a USB Targeted Peripheral List (TPL). A callout box highlights the VID and PID definitions.

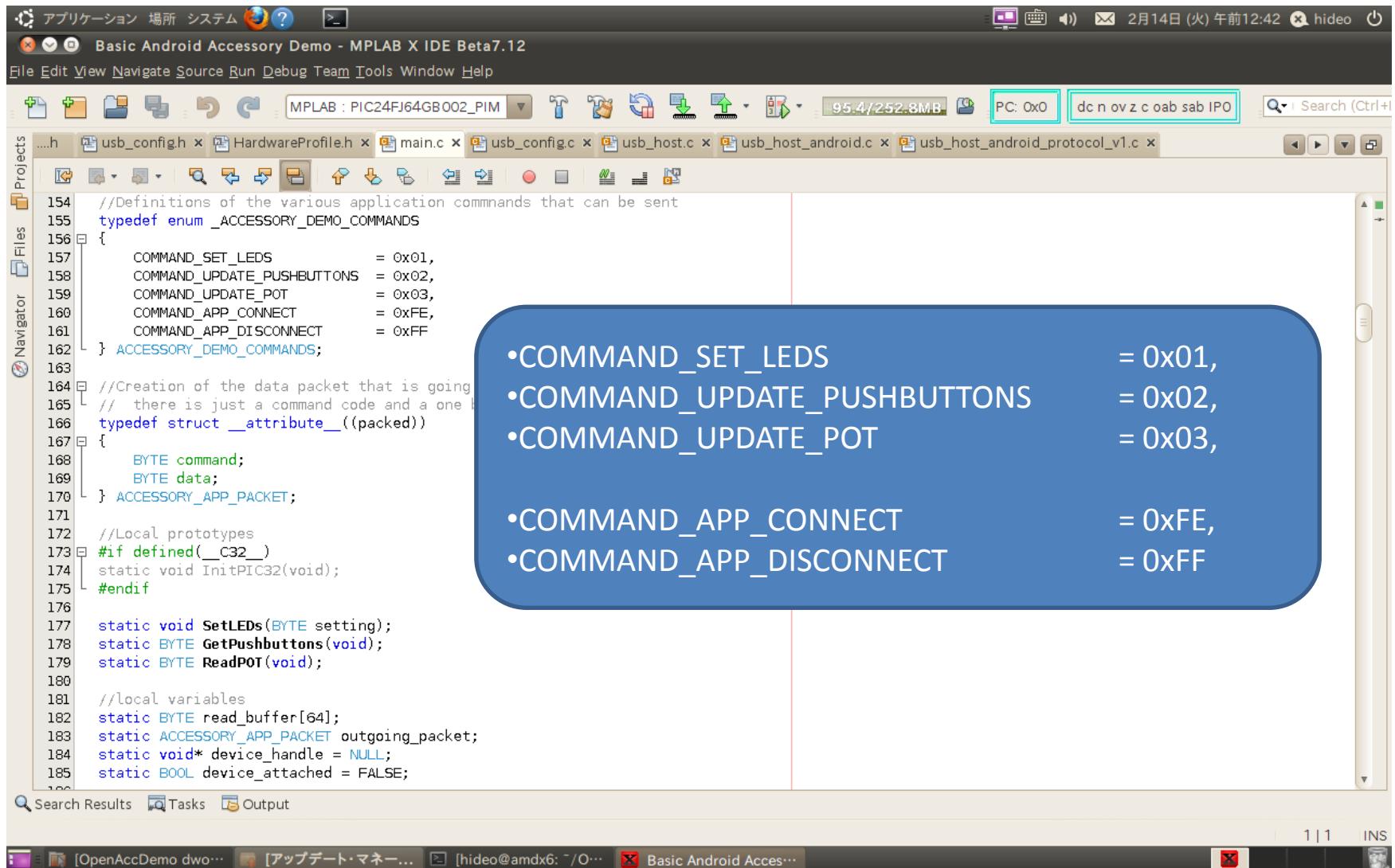
```
46 CLIENT_DRIVER_TABLE usbClientDrvTable[NUM_CLIENT_DRIVER_ENTRIES] =
47 {
48     {
49         AndroidAppInitialize,
50         AndroidAppEventHandler,
51         AndroidAppDataEventHandler,
52         0
53     },
54     {
55         AndroidAppInitialize,
56         AndroidAppEventHandler,
57         AndroidAppDataEventHandler,
58         ANDROID_INIT_FLAG_BYPASS_PROTOCOL
59     }
60 };
61
62 // *****
63 // USB Embedded Host Targeted Peripheral List (TPL)
64 // *****
65 USB_TPL usbTPL[NUM_TPL_ENTRIES] =
66 {
67     /*[1] Device identification information
68      [2] Initial USB configuration to use
69      [3] Client driver table entry
70      [4] Flags (HNP supported, client driver entry, SetConfiguration() commands allowed)
71      -----
72      [1] [2][3] [4]
73      -----
74      */
75     { INIT_VID_PID( 0x18D1ul, 0x2D00ul ), 0, 1, {0} }, // Android accessory
76     { INIT_VID_PID( 0x18D1ul, 0x2D01ul ), 0, 1, {0} }, // Android accessory
77     { INIT_VID_PID( 0xFFFFul, 0xFFFFul ), 0, 0, {0} } // Enumerates everything
78 };
79
```

Vid,PIDの定義  
0x180,0x200又は0x180,0x201

# PICのファンクション指定

- ・リセット、ディバック、Watchdog、クロック源など
- ・しばらくはコピペでよい

# Appとのプロトコル定義



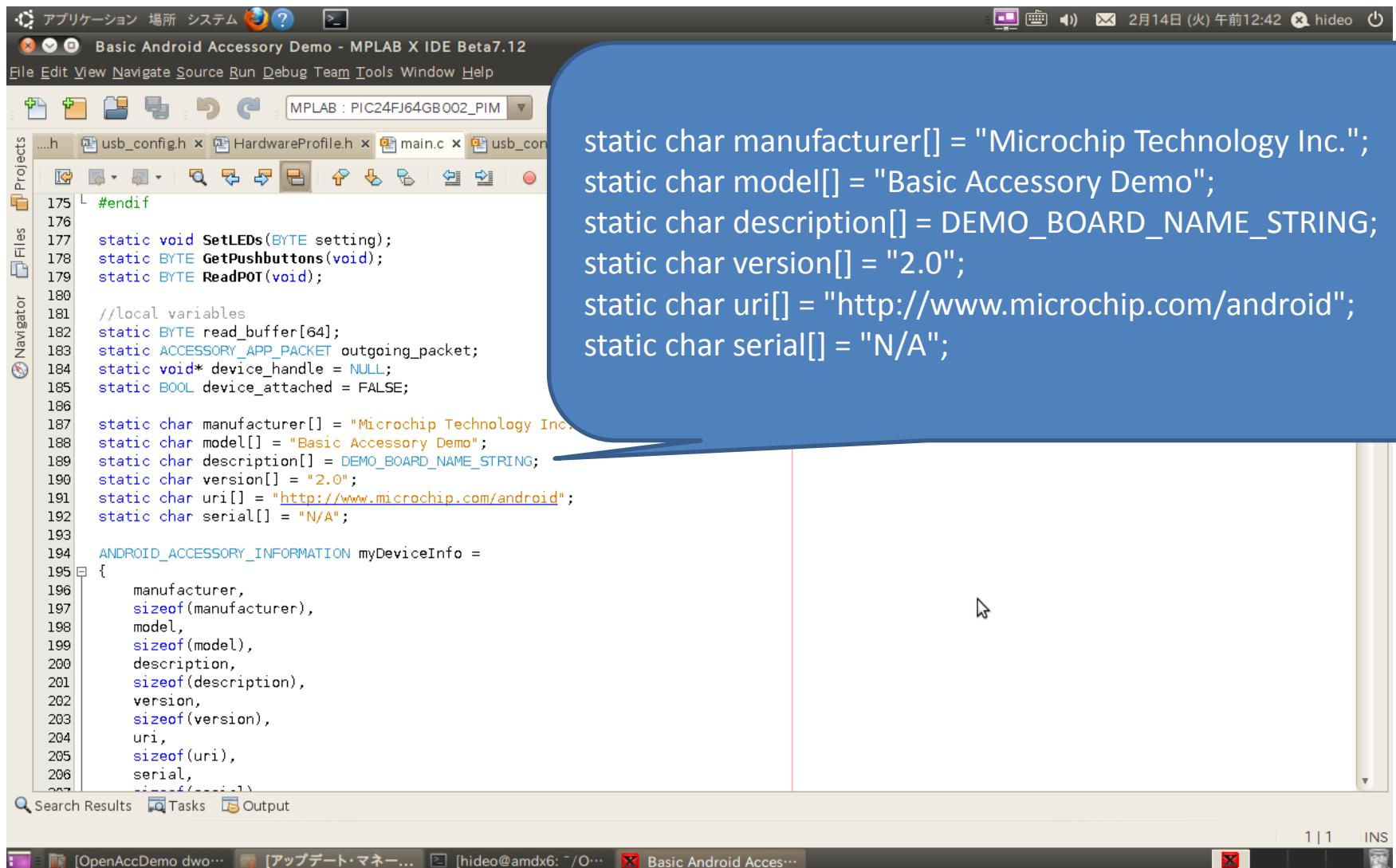
The screenshot shows the MPLAB X IDE interface with the title "Basic Android Accessory Demo - MPLAB X IDE Beta7.12". The code editor displays the file "main.c" which contains the following enum definition:

```
154 //Definitions of the various application commands that can be sent
155 typedef enum _ACCESSORY_DEMO_COMMANDS
156 {
157     COMMAND_SET_LEDS          = 0x01,
158     COMMAND_UPDATE_PUSHBUTTONS = 0x02,
159     COMMAND_UPDATE_POT        = 0x03,
160     COMMAND_APP_CONNECT       = 0xFE,
161     COMMAND_APP_DISCONNECT    = 0xFF
162 } ACCESSORY_DEMO_COMMANDS;
```

A callout box highlights the command definitions and their values:

- COMMAND\_SET\_LEDS = 0x01,
- COMMAND\_UPDATE\_PUSHBUTTONS = 0x02,
- COMMAND\_UPDATE\_POT = 0x03,
- COMMAND\_APP\_CONNECT = 0xFE,
- COMMAND\_APP\_DISCONNECT = 0xFF

# アクセサリの識別情報

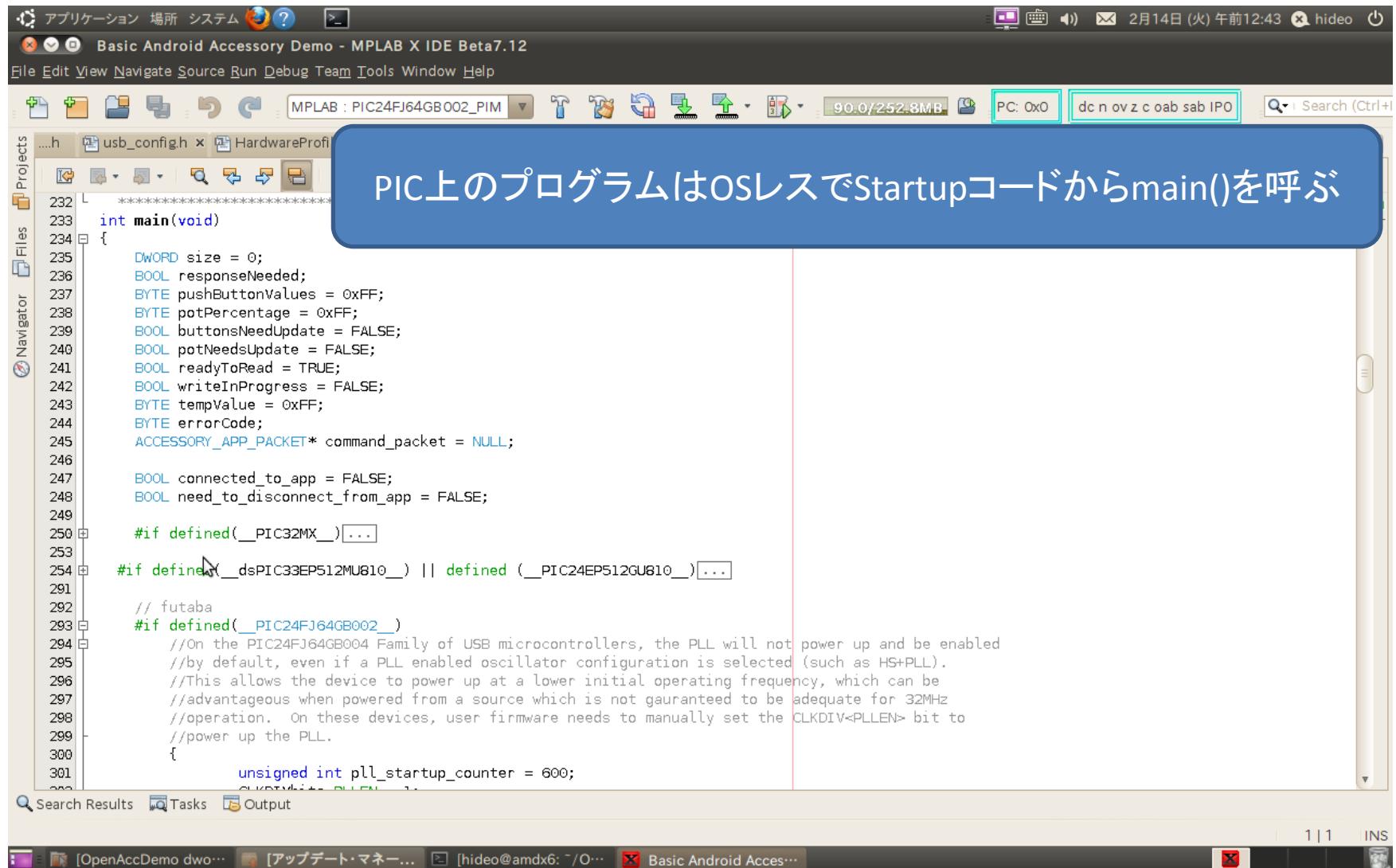


The screenshot shows the MPLAB X IDE Beta 7.12 interface. The main window displays the file `main.c` for the project `Basic Android Accessory Demo`. The code defines static character arrays for device information:

```
static char manufacturer[] = "Microchip Technology Inc.";
static char model[] = "Basic Accessory Demo";
static char description[] = DEMO_BOARD_NAME_STRING;
static char version[] = "2.0";
static char uri[] = "http://www.microchip.com/android";
static char serial[] = "N/A";
```

A callout bubble points to the line `static char manufacturer[] = "Microchip Technology Inc.";` with the text "アクセサリの識別情報". A red vertical line highlights the entire `static char manufacturer[]` line.

# main()



Basic Android Accessory Demo - MPLAB X IDE Beta7.12

File Edit View Navigate Source Run Debug Team Tools Window Help

MPLAB : PIC24FJ64GB002\_PIM

PC: 0x0 dc n ov z c oab sab IPO

Search (Ctrl+F)

Projects

Files

Navigator

...h usb\_config.h x HardwareProfile

pic24fj64gb002.c

```
232 ****
233 int main(void)
234 {
235     DWORD size = 0;
236     BOOL responseNeeded;
237     BYTE pushButtonValues = 0xFF;
238     BYTE potPercentage = 0xFF;
239     BOOL buttonsNeedUpdate = FALSE;
240     BOOL potNeedsUpdate = FALSE;
241     BOOL readyToRead = TRUE;
242     BOOL writeInProgress = FALSE;
243     BYTE tempValue = 0xFF;
244     BYTE errorCode;
245     ACCESSORY_APP_PACKET* command_packet = NULL;
246
247     BOOL connected_to_app = FALSE;
248     BOOL need_to_disconnect_from_app = FALSE;
249
250 #if defined(__PIC32MX__)
251
252 #if defined(__dsPIC33EP512MU810__) || defined(__PIC24EP512GU810__)
253
254 // futaba
255 #if defined(__PIC24FJ64GB002__)
256     //On the PIC24FJ64GB004 Family of USB microcontrollers, the PLL will not power up and be enabled
257     //by default, even if a PLL enabled oscillator configuration is selected (such as HS+PLL).
258     //This allows the device to power up at a lower initial operating frequency, which can be
259     //advantageous when powered from a source which is not guaranteed to be adequate for 32MHz
260     //operation. On these devices, user firmware needs to manually set the CLKDIV<PLLEN> bit to
261     //power up the PLL.
262     {
263         unsigned int pll_startup_counter = 600;
264         CLKDIVbits.PLLEN = 1;
265     }
266
267 #endif
268
269 #endif
270
271 #endif
272 }
```

Search Results Tasks Output

[OpenAccDemo dwo... [アップデート・マネ... [hideo@amdx6: ~/O... Basic Android Acces...

1 | 1 INS

A callout box highlights the line "int main(void)" with the text: "PIC上のプログラムはOSレスでStartupコードからmain()を呼ぶ"

# PLLの設定/ADCの初期化

内部クロックの設定

USB処理を起動

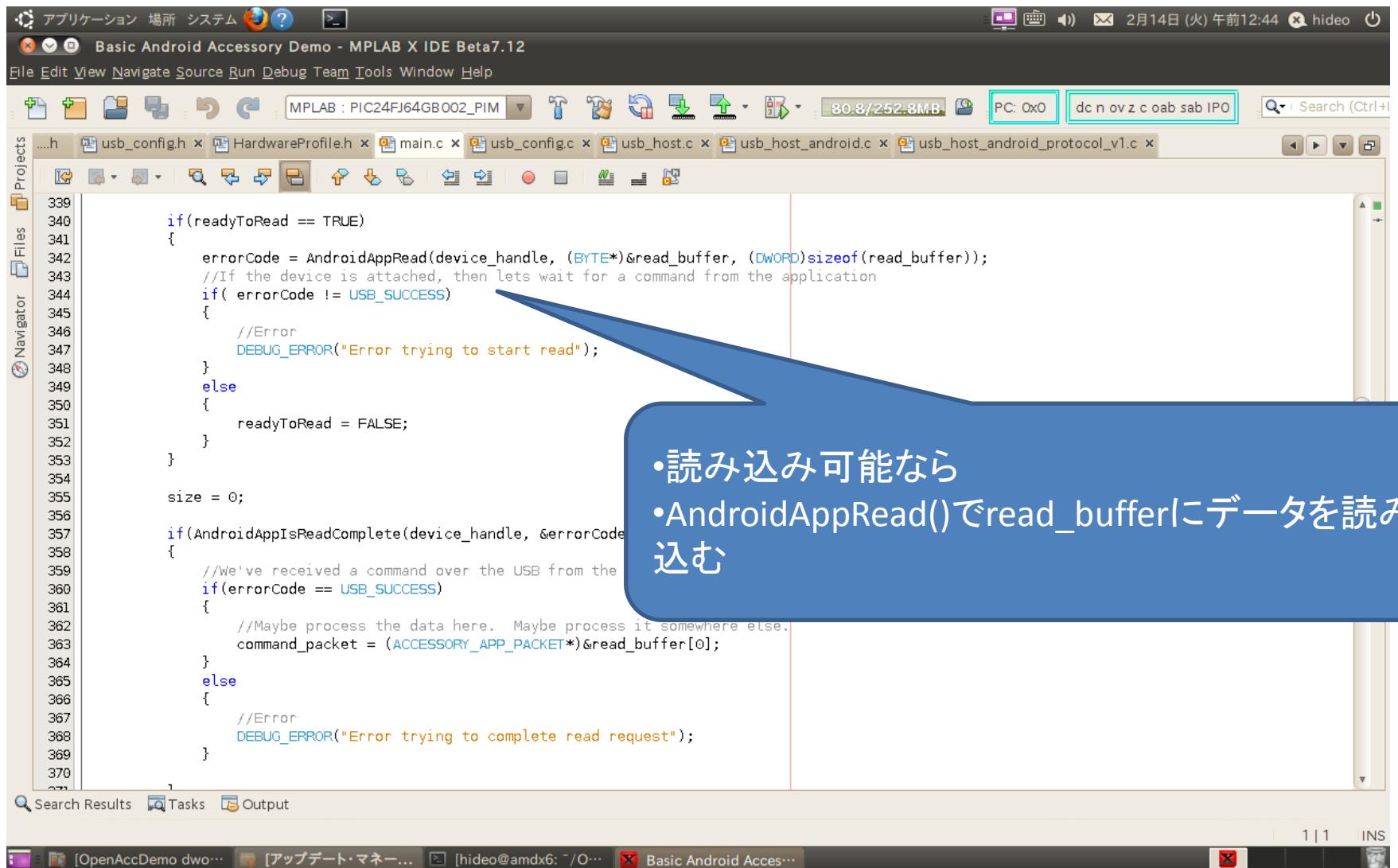
POTの使うADCの設定

```
291 // futaba
292 #if defined(__PIC24FJ64GB002__)
293 // On the PIC24FJ64GB004 Family of USB microcontrollers, the PLL will
294 // by default, even if a PLL enabled oscillator configuration is selected.
295 // This allows the device to power up at a lower initial operating frequency
296 // advantageous when powered from a source which is not guaranteed to be stable for 32MHz
297 // operation. On these devices, user firmware needs to manually set the CLKDIV<PLLEN> bit to
298 // power up the PLL.
299 {
300     unsigned int pll_startup_counter = 600;
301     CLKDIVbits.PLLEN = 1;
302     while(pll_startup_counter--);
303 }
304
305 AD1PCFG = 0xffff;
306 CLKDIV = 0x0000;      // Set PLL prescaler (1:1)
307
308 #endif
309
310 USBIInitialize();
311 AndroidAppStart(&myDeviceInfo);
312
313 responseNeeded = FALSE;
314 mInitPOT();
315
316
317 while(1)
318 {
319     //Keep the USB stack running
320     USBTasks();
321
322     //If the device isn't attached yet,
323     //don't do anything
324 }
```

# メインループの入り口

```
mInitPOT();  
315  
316  
317 while(1)  
318 {  
319     //Keep the USB stack running  
320     USBTasks();  
321  
322     //If the device isn't attached yet,  
323     if(device_attached == FALSE)  
324     {  
325         buttonsNeedUpdate = TRUE;  
326         potNeedsUpdate = TRUE;  
327         need_to_disconnect_from_app = FALSE;  
328         connected_to_app = FALSE;  
329         size = 0;  
330  
331         //Reset the accessory state variables  
332         InitAllLEDs();  
333  
334         //Continue to the top of the while loop to start the check over again.  
335         continue;  
336     }  
337  
338     //If the accessory is ready, then this is where we run all of the demo code  
339  
340     if(readyToRead == TRUE)  
341     {  
342         errorCode = AndroidAppRead(device_handle, (BYTE*)&read_buffer, (DWORD)sizeof(read_buffer));  
343         //If the device is attached, then lets wait for a command from the application  
344         if( errorCode != USB_SUCCESS)  
345         {  
346             //Error  
347             //PC: 0x0 dc n ov z c oab sab IPO  
348         }  
349     }  
350 }
```

# データの受信



The screenshot shows the MPLAB X IDE interface with the title "Basic Android Accessory Demo - MPLAB X IDE Beta7.12". The code editor displays C code for a USB accessory. A callout bubble points to the line of code where data is read from the device handle.

```
339     if(readyToRead == TRUE)
340     {
341         errorCode = AndroidAppRead(device_handle, (BYTE*)&read_buffer, (DWORD)sizeof(read_buffer));
342         //If the device is attached, then lets wait for a command from the application
343         if( errorCode != USB_SUCCESS)
344         {
345             //Error
346             DEBUG_ERROR("Error trying to start read");
347         }
348         else
349         {
350             readyToRead = FALSE;
351         }
352     }
353
354     size = 0;
355
356     if(AndroidAppIsReadComplete(device_handle, &errorCode)
357     {
358         //We've received a command over the USB from the
359         if(errorCode == USB_SUCCESS)
360         {
361             //Maybe process the data here. Maybe process it somewhere else.
362             command_packet = (ACCESSORY_APP_PACKET*)&read_buffer[0];
363         }
364         else
365         {
366             //Error
367             DEBUG_ERROR("Error trying to complete read request");
368         }
369     }
370 }
```

•読み込み可能なら  
•AndroidAppRead()でread\_bufferにデータを読み込む

# LEDの操作

Basic Android Accessory Demo - MPLAB X IDE Beta7.12

File Edit View Navigate Source Run Debug Team Tools Window Help

MPLAB : PIC24FJ64GB002\_PIM 98.5/252.8MB PC: 0x0 dc n ov z c oab sab IPO Search (Ctrl+)

Projects Files Navigator

```
372     while(size > 0)
373     {
374         if(connected_to_app == FALSE)
375         {
376             if(command_packet->command == COMMAND_APP_DISCONNECT)
377             {
378                 connected_to_app = TRUE;
379                 need_to_disconnect_from_app = FALSE;
380             }
381         }
382     }
383     else
384     {
385         switch(command_packet->command)
386         {
387             case COMMAND_SET_LEDS:
388                 SetLEDs(command_packet->data);
389                 break;
390
391             case COMMAND_APP_DISCONNECT:
392                 need_to_disconnect_from_app = TRUE;
393                 break;
394
395             default:
396                 //Error, unknown command
397                 DEBUG_ERROR("Error: unknown command received");
398                 break;
399         }
400     }
401     //All commands in this example are two bytes, so remove that from the queue
402     size -= 2;
403     //And move the pointer to the next packet (this works because
```

•読み取った情報の先頭が0x01だったら  
•後続の値はLEDのビット並びなので、SetLEDsに渡す

# スイッチの値の読みだし

The screenshot shows the MPLAB X IDE interface with the title bar "Basic Android Accessory Demo - MPLAB X IDE Beta7.12". The status bar indicates "2月14日 (火) 午前12:44 hideo". The code editor displays a portion of the main.c file:

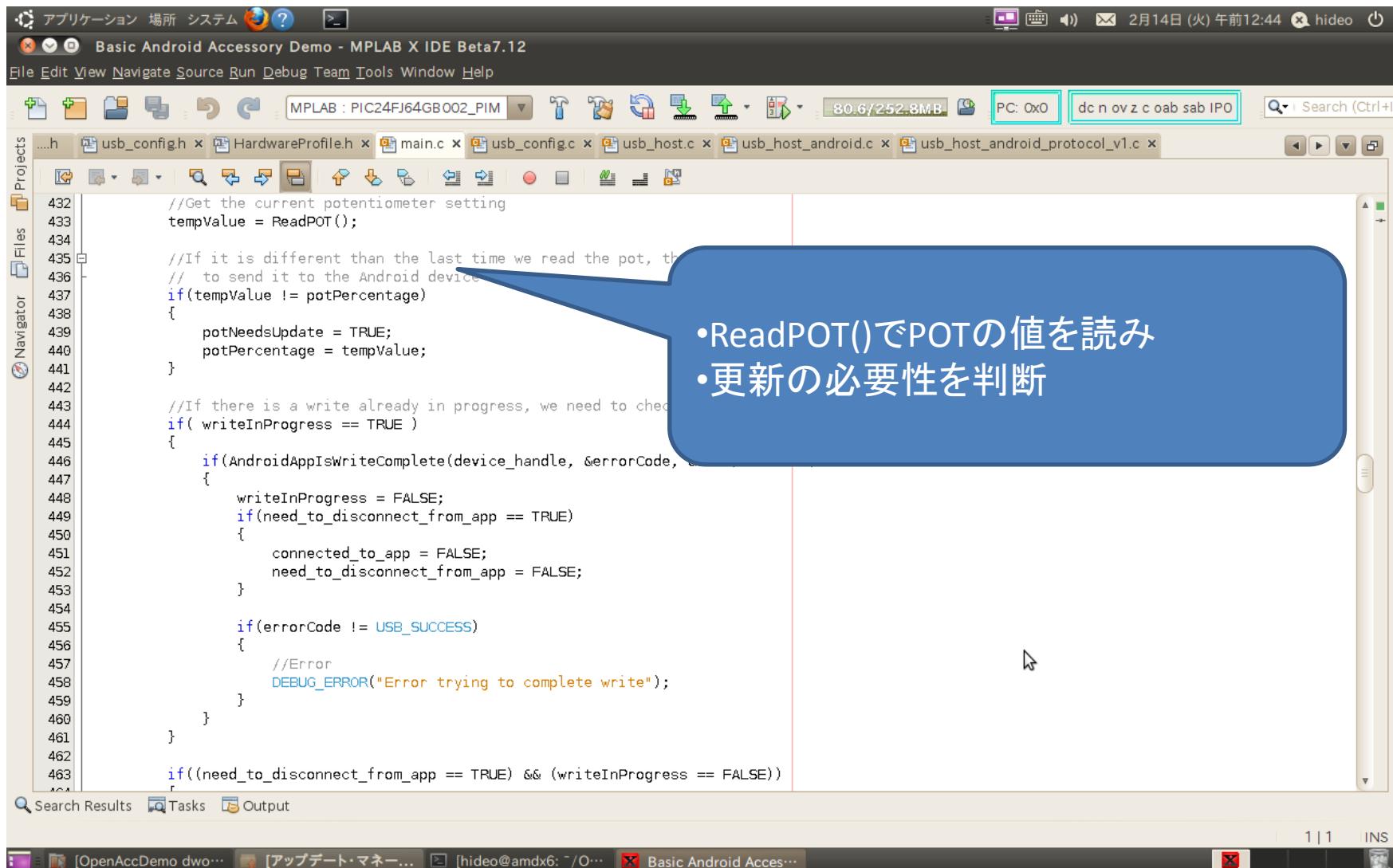
```
402     size -= 2;
403     //And move the pointer to the next packet (this works because
404     // all command packets are 2 bytes. If variable packet size
405     // then need to handle moving the pointer by the size
406     // command type that arrived.
407     command_packet++;
408
409     if(need_to_disconnect_from_app == TRUE)
410    {
411        break;
412    }
413
414
415    if(size == 0)
416    {
417        readyToRead = TRUE;
418    }
419
420    //Get the current pushbutton setting
421    tempValue = GetPushbuttons();
422
423    //If the current button settings are different than the last time
424    // we read the button values, then we need to send an update to the
425    // attached Android device
426    if(tempValue != pushButtonValues)
427    {
428        buttonsNeedUpdate = TRUE;
429        pushButtonValues = tempValue;
430    }
431
432    //Get the current potentiometer setting
433    tempValue = ReadPOT();
```

A callout bubble points to the line "tempValue = GetPushbuttons();". The bubble contains the following text:

- GetPushbuttons()でボタンの値を得て
- 最新の値から変更があったら
- 更新が必要と判断

The status bar at the bottom right shows "1 | 1 INS".

# POTの値の読みだし



The screenshot shows the MPLAB X IDE interface with the title bar "Basic Android Accessory Demo - MPLAB X IDE Beta7.12". The status bar at the top right shows the date "2月14日 (火) 午前12:44" and the user "hideo". The main window displays a C code editor for the file "main.c". A callout bubble points to the code block starting at line 432, which reads:

```
432 //Get the current potentiometer setting  
433 tempValue = ReadPOT();  
434  
435 //If it is different than the last time we read the pot, then  
// to send it to the Android device.  
436 if(tempValue != potPercentage)  
{  
    potNeedsUpdate = TRUE;  
    potPercentage = tempValue;  
}  
  
443 //If there is a write already in progress, we need to check  
444 if( writeInProgress == TRUE )  
{  
    if(AndroidAppIsWriteComplete(device_handle, &errorCode, &writeCompleted))  
    {  
        writeInProgress = FALSE;  
        if(need_to_disconnect_from_app == TRUE)  
        {  
            connected_to_app = FALSE;  
            need_to_disconnect_from_app = FALSE;  
        }  
  
        if(errorCode != USB_SUCCESS)  
        {  
            //Error  
            DEBUG_ERROR("Error trying to complete write");  
        }  
    }  
  
    if((need_to_disconnect_from_app == TRUE) && (writeInProgress == FALSE))
```

A blue callout bubble contains the following notes:

- ReadPOT()でPOTの値を読み
- 更新の必要性を判断

# ボタンデータの送出

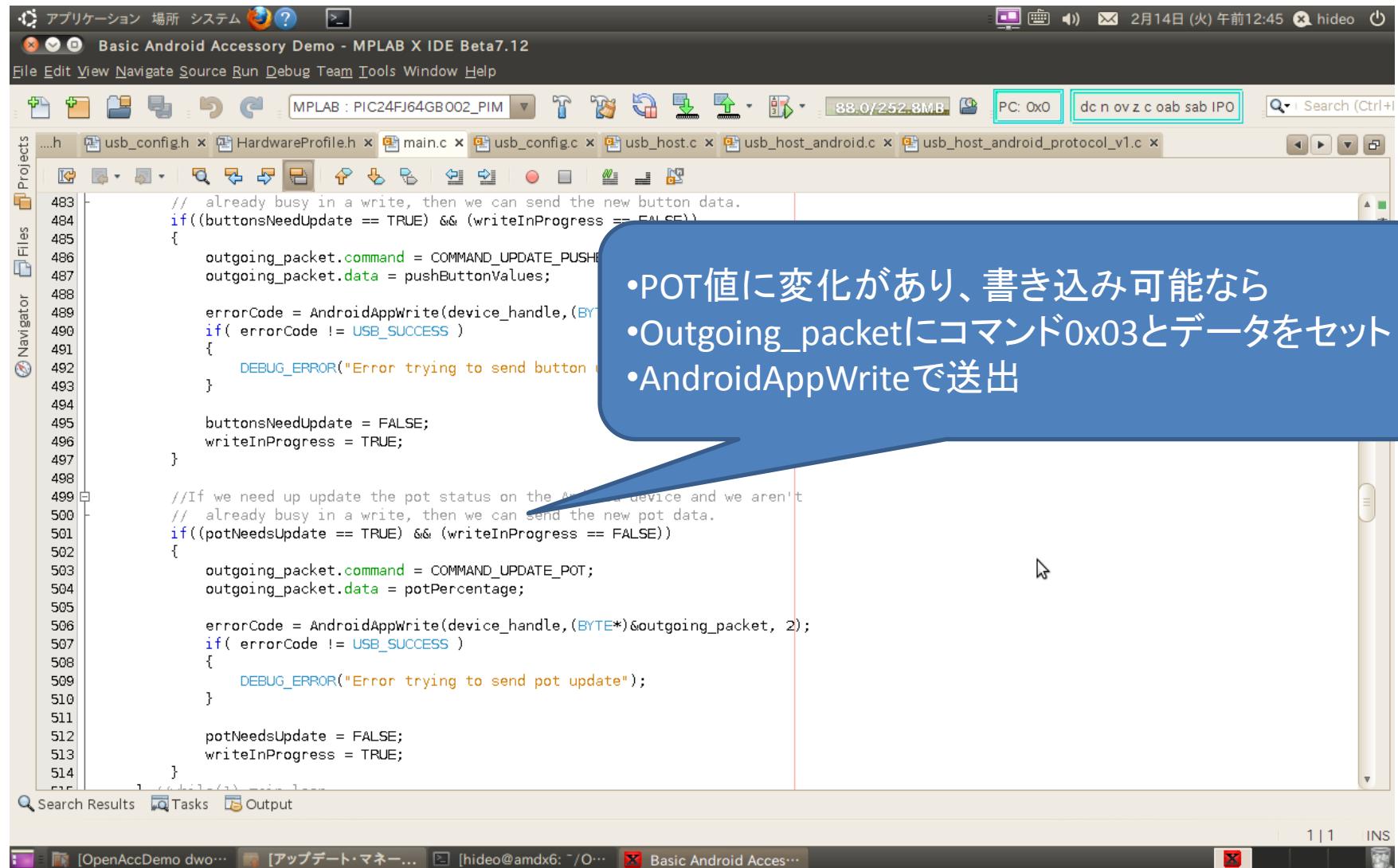
The screenshot shows the MPLAB X IDE interface with the title bar "Basic Android Accessory Demo - MPLAB X IDE Beta7.12". The main window displays a portion of the "main.c" source code. A callout bubble highlights the following points:

- ボタンの値に変化があり、書き込み可能なら
- Outgoing\_packetにコマンド0x02とデータをセット
- AndroidAppWriteで送出

```
462     if((need_to_disconnect_from_app == TRUE) && (writeInProgress == FALSE))
463     {
464         outgoing_packet.command = COMMAND_APP_DISCONNECT;
465         outgoing_packet.data = 0;
466         writeInProgress = TRUE;
467
468         errorCode = AndroidAppWrite(device_handle, (BYTE*)&outgoing_packet, 1);
469         if( errorCode != USB_SUCCESS )
470         {
471             DEBUG_ERROR("Error trying to send disconnect command");
472         }
473     }
474
475     if(connecting_to_app == FALSE)
476     {
477         //If the app hasn't told us to start yet, continue;
478     }
479
480
481     //If we need up update the button status to the android device and we aren't
482     //already busy in a write, then we can send the new button data.
483     if((buttonsNeedUpdate == TRUE) && (writeInProgress == FALSE))
484     {
485         outgoing_packet.command = COMMAND_UPDATE_PUSHBUTTONS;
486         outgoing_packet.data = pushButtonValues;
487
488         errorCode = AndroidAppWrite(device_handle, (BYTE*)&outgoing_packet, 2);
489         if( errorCode != USB_SUCCESS )
490         {
491             DEBUG_ERROR("Error trying to send button update");
492         }
493     }
494 }
```

At the bottom of the code editor, there are tabs for "Search Results", "Tasks", and "Output". The status bar at the bottom right shows "1 | 1 INS".

# POTデータの送出



The screenshot shows the MPLAB X IDE interface with the project "Basic Android Accessory Demo" open. The main window displays the "main.c" source code. A callout bubble highlights the following code snippet:

```
483     // already busy in a write, then we can send the new button data.
484     if((buttonsNeedUpdate == TRUE) && (writeInProgress == FALSE))
485     {
486         outgoing_packet.command = COMMAND_UPDATE_PUSH;
487         outgoing_packet.data = pushButtonValues;
488
489         errorCode = AndroidAppWrite(device_handle, (BYTE*)&outgoing_packet, 1);
490         if( errorCode != USB_SUCCESS )
491         {
492             DEBUG_ERROR("Error trying to send button update");
493         }
494
495         buttonsNeedUpdate = FALSE;
496         writeInProgress = TRUE;
497     }
498
499
500     //If we need up update the pot status on the Android device and we aren't
501     // already busy in a write, then we can send the new pot data.
502     if((potNeedsUpdate == TRUE) && (writeInProgress == FALSE))
503     {
504         outgoing_packet.command = COMMAND_UPDATE_POT;
505         outgoing_packet.data = potPercentage;
506
507         errorCode = AndroidAppWrite(device_handle, (BYTE*)&outgoing_packet, 2);
508         if( errorCode != USB_SUCCESS )
509         {
510             DEBUG_ERROR("Error trying to send pot update");
511         }
512
513         potNeedsUpdate = FALSE;
514         writeInProgress = TRUE;
515     }
516 }
```

The callout bubble contains the following points:

- POT値に変化があり、書き込み可能なら
- Outgoing\_packetにコマンド0x03とデータをセット
- AndroidAppWriteで送出

# SetLEDs

The screenshot shows the MPLAB X IDE Beta 7.12 interface. The main window displays the `main.c` file with the following code:

```
708 static void SetLEDs(BYTE setting)
709 {
710     if((setting & 0x01) == 0x01) { LED0_On(); } else { LED0_Off(); }
711     if((setting & 0x02) == 0x02) { LED1_On(); } else { LED1_Off(); }
712     if((setting & 0x04) == 0x04) { LED2_On(); } else { LED2_Off(); }
713     if((setting & 0x08) == 0x08) { LED3_On(); } else { LED3_Off(); }
714     if((setting & 0x10) == 0x10) { LED4_On(); } else { LED4_Off(); }
715     if((setting & 0x20) == 0x20) { LED5_On(); } else { LED5_Off(); }
716     if((setting & 0x40) == 0x40) { LED6_On(); } else { LED6_Off(); }
717     if((setting & 0x80) == 0x80) { LED7_On(); } else { LED7_Off(); }
718 }
719
720 /**
721  * Function:
722  *     BYTE GetPushbuttons(void)
723  *
724  * Summary:
725  *     Reads the current push button status.
726  *
727  * Description:
728  *     Reads the current push button status.
729  *
730  * Precondition:
731  *     None
732  *
733  * Parameters:
734  *     None
735  *
736  * Return Values:
737  *     BYTE - bitmap for button representations (1 = pressed, 0 = not pressed)
738  *         bit 0 = button 1
739  *         bit 1 = button 2
740  *         bit 2 = button 3
741  */
742
743 #endif
```

The code implements a function `SetLEDs` that takes a `BYTE` parameter `setting`. It contains an `if` block for each LED (LED0 to LED7), turning them on or off based on the value of the corresponding bit in the `setting` variable. Below the function definition, there is a detailed documentation block for the `GetPushbuttons` function, which reads the current push button status. The documentation includes sections for `Function`, `Summary`, `Description`, `Precondition`, `Parameters`, and `Return Values`.

# GetPushbuttons

The screenshot shows the MPLAB X IDE interface with the 'Basic Android Accessory Demo' project open. The code editor displays the main.c file, which contains the following function:

```
744 None
745 *****/
746 static BYTE GetPushbuttons(void)
747 {
748     BYTE toReturn;
749
750     InitAllSwitches();
751
752     toReturn = 0;
753
754     if(Switch1Pressed()){toReturn |= 0x1;}
755     if(Switch2Pressed()){toReturn |= 0x2;}
756     if(Switch3Pressed()){toReturn |= 0x4;}
757     if(Switch4Pressed()){toReturn |= 0x8;}
758
759     return toReturn;
760 }
761
762 *****/
763 Function:
764     BYTE ReadPOT(void)
765
766 Summary:
767     Reads the pot value and returns the percentage of full scale (0-100)
768
769 Description:
770     Reads the pot value and returns the percentage of full scale (0-100)
771
772 Precondition:
773     A/D is initialized properly
774
775 *****/
```

A blue speech bubble points to the line of code: `if(Switch1Pressed()){toReturn |= 0x1;}`. Inside the bubble, the Japanese text "ボタンに対応したビットを立てて復帰" (Set the bit corresponding to the button and return) is written.

# POTの読み取り

A screenshot of the MPLAB X IDE Beta 7.12 interface. The title bar shows "Basic Android Accessory Demo - MPLAB X IDE Beta7.12". The menu bar includes File, Edit, View, Navigate, Source, Run, Debug, Team, Tools, Window, Help. The toolbar has various icons for file operations. The status bar shows "103.3/252.8MB", "PC: 0x0", and "dc n ov z c oab sab IPO". The main window displays a C source code editor with the following code:

```
783 None
784 ****
785 static BYTE ReadPOT(void)
786 {
787     WORD_VAL w;
788     DWORD temp;
789
790 #if defined(__18CXX)
791     mInitPOT();
792
793     ADCON0bits.GO = 1;           // Start AD conversion
794     while(ADCON0bits.NOT_DONE); // Wait for conversion
795
796     w.v[0] = ADRESL;
797     w.v[1] = ADRESH;
798
799 #elif defined(__C30__) || defined(__C32__)
800 #if defined(PIC24FJ256GB110_PIM) || \
801     defined(PIC24FJ256DA210_DEV_BOARD) || \
802     defined(PIC24FJ256GB210_PIM)
803
804     AD1CHS = 0x5;             //MUXA uses AN5
805
806     // Get an ADC sample
807     AD1CON1bits.SAMP = 1;      //Start sampling
808     for(w.Val=0;w.Val<1000;w.Val++); //Sample delay, conversion start automatically
809     AD1CON1bits.SAMP = 0;      //Start sampling
810     for(w.Val=0;w.Val<1000;w.Val++); //Sample delay, conversion start automatically
811     while(!AD1CON1bits.DONE);   //Wait for conversion to complete
812
813     temp = (DWORD)ADC1BUFO;
814     temp = temp * 100;
815 }
```

A blue callout bubble with the text "POTの読み込み" points to the line "static BYTE ReadPOT(void)". The status bar at the bottom shows "Search Results", "Tasks", "Output", "1 | 1", and "INS".

# ADCの起動と読み込み

The screenshot shows the MPLAB X IDE interface with the title bar "Basic Android Accessory Demo - MPLAB X IDE Beta7.12". The menu bar includes File, Edit, View, Navigate, Source, Run, Debug, Team, Tools, Window, Help. The toolbar has various icons for file operations. The status bar shows "2月14日 (火) 午前12:47 hideo". The main window displays a C code editor with several tabs: ...h, usb\_config.h, HardwareProfile.h, main.c, usb\_config.c, usb\_host.c, usb\_host\_android.c, and usb\_host\_android\_protocol\_v1.c. The code editor shows the following code:

```
828     temp = temp * 100;
829     temp = temp/1023;
830
831 #elif defined(PIC24FJ64GB002_PIM)           //futaba
832     AD1CHS = 0x5;                            //MUXA uses AN5
833
834     // Get an ADC sample
835     AD1CON1bits.SAMP = 1;                   //Start sampling
836     for(w.Val=0;w.Val<1000;w.Val++); //Sample delay, conversion start automatically
837         AD1CON1bits.SAMP = 0;               //Start sampling
838     for(w.Val=0;w.Val<1000;w.Val++); //Sample delay, conversion start automatically
839         while(!AD1CON1bits.DONE);        //Wait for conversion to complete
840
841     temp = (DWORD)ADC1BUF0;
842     temp = temp * 100;
843     temp = temp/1023;
844
845 #elif defined(PIC24FJ64GB004_PIM)
846     AD1CHS = 0x7;                            //MUXA uses AN7
847
848     // Get an ADC sample
849     AD1CON1bits.SAMP = 1;                   //Start sampling
850     for(w.Val=0;w.Val<1000;w.Val++); //Sample delay, conversion start automatically
851         AD1CON1bits.SAMP = 0;               //Start sampling
852     for(w.Val=0;w.Val<1000;w.Val++); //Sample delay, conversion start automatically
853         while(!AD1CON1bits.DONE);        //Wait for conversion to complete
854
855     temp = (DWORD)ADC1BUF0;
856     temp = temp * 100;
857     temp = temp/1023;
858
859 #elif defined(PIC24F_STARTER_KIT)
```

Two callout boxes highlight specific sections of the code:

- Top callout (blue):
  - AD1CON1bits.SAMPを1に
  - しばらく待って
  - AD1CON1bits.SAMPを0に
  - しばらく待って
  - ADCON1bits.DONEが1になるのを待つ
- Bottom callout (blue):
  - ADC1BUF0に10ビットの値が格納
  - × 100して ÷ 1023
  - 0v-3.3vの範囲の値が0～100%の値に

# **PART5**

# **ANDROID側の仕組み**

# Android USB Accessory

- Android 2.3.4(API Level10)にバックポート
  - com.android.future.usb
  - add-on Google APIs Android API 10が必要
  - <http://code.google.com/intl/ja/android/add-ons/google-apis/installing.html>
- Android 3.1 (API level 12)でサポート
  - Android.hardware.usb

# Manifest

- Android 2.3.4(API Level10) + add-on
  - <uses-library>エレメント
  - com.android.feature.usb.accessory
- Android 3.1 (API level 12)
  - <uses-feature>エレメント
  - android.hardware.usb.accessory

# API

- Android 2.3.4(API Level10) + add-on
  - UsbManager manager = UsbManager.getInstance(this);
  - UsbAccessory accessory = UsbManager.getAccessory(intent);
- Android 3.1 (API level 12)
  - UsbManager manager = (UsbManager) getSystemService(Context.USB\_SERVICE);
  - UsbAccessory accessory = (UsbAccessory) intent.getParcelableExtra(UsbManager.EXTRA\_ACCESSORY);

# 接続の通知

- android.hardware.usb.action.USB\_ACCESSORY\_ATTACHED
  - メインのActivityに次の2つをペアで記述
    - <intent-filter> インテントフィルタ
    - <meta-data> アクセサリ識別のXMLファイル
- XMLファイルの記述
  - <usb-accessory>エレメントに以下を記述
    - manufacturer
    - model
    - version

# UsbAccessoryの取得

- Android 2.3.4(API Level10) + add-on  

```
UsbAccessory accessory = UsbManager.getAccessory  
y(intent);
```
- Android 3.1 (API level 12)  

```
UsbAccessory accessory = (UsbAccessory)intent.get  
ParcelableExtra(UsbManager.EXTRA_ACCESSORY);
```

# Accessoriesの列挙

- Android 2.3.4(API Level10) + add-on

```
UsbManager manager = UsbManager.getInstance(this);  
UsbAccessory[] accessoryList = manager.getAccessoryList();
```

- Android 3.1 (API level 12)

```
UsbManager manager = (UsbManager) getSystemService(Context.USB_SERVICE);  
UsbAccessory[] accessoryList = manager.getAccessoryList();
```

# 通信の許可

- broadcast receiver の作成
- Activity の onCreate() にbroadcast receiver を登録
- パーミッションダイアログの表示

UsbAccessory accessory;

...

```
mUsbManager.requestPermission(accessory, mPermissionIntent);
```

# Accessoryとの通信

- UsbManagerでfile descriptorを取得
  - InputStream/OutputStreamがEndpointに接続されている
- 通信はブロックされるの
  - スレッドを起こして、run()メソッドで通信

# APIのまとめ

- 2.x系と3-4系でAPIが異なる
  - 3-4系はframeworkに実装
  - 2.xではadd-onのlibraryで実装
- Manifestに記述する
  - 3-4系はuses-feature, 2.x uses-library
- UsbManagerとUsbAccessory を取得する
- Readは同期IOなので、別スレッドで呼ぶ

# Android Appソースのウォークスルー

- AndroidManifest.xml
- res/xml/accessory\_filter.xml
- src/com/microchip/android/BasicAccessoryDemo\_API12
  - \$ wc -l \*
  - 512 BasicAccessoryDemo.java アプリ本体
  - 138 LEDControl.java LED描画
  - 854 USBAccessoryManager.java
  - 93 USBAccessoryManagerMessage.java
  - 1597 合計

# LifeCycle handler

- onCreate  
92: accessoryManager =  
    new USBAccessoryManager(handler, USBAccessoryWhat);
- onStart  
177: this.setTitle("Basic Accessory Demo: Device not connected.");
- onResume  
185: accessoryManager.enable(this, getIntent());
- onPause  
242: accessoryManager.disable(this);



# USBAccessoryManager.java

# USBAccessoryManager.enable

- UsbManagerからUsbAccessoryを取得

141: accessories = deviceManager.getAccessoryList();

- ファイルディスクリプタを取得

159: parcelFileDescriptor = deviceManager.openAccessory(accessory);

- ReadTreadを生成

165: readThread = new ReadThread(parcelFileDescriptor);

- outputStreamを取得

178: outputStream = new  
FileOutputStream(parcelFileDescriptor.getFileDescriptor());

# enable - 1

The screenshot shows the Eclipse IDE interface with the following details:

- Title Bar:** Java - OpenAcc3x/src/com/microchip/android/BasicAccessoryDemo\_API12/USBAccessoryManager.java - Eclipse SDK
- Toolbar:** Standard Eclipse toolbar with various icons for file operations, search, and navigation.
- Left Sidebar (Package Explorer):** Shows the project structure with files like LEDControl.java, USBAccessoryManager.java, and USBAccessoryManagerMessage.java.
- Central Editor:** Displays the Java code for the `enable` method. A blue callout box labeled "初回のみ" (Only once) points to the first few lines of code where a new IntentFilter is created. Another blue callout box labeled "2つのクラス" (Two classes) points to the line where `context.registerReceiver(receiver, filter);` is called.

```
99  */
100 public RETURN_CODES enable(Context context, Intent intent) {
101     // Grab the packageName to use for an attach Intent
102     actionString = context.getPackageName() + ".action.USB_PERMISSION";
103
104     PendingIntent permissionIntent = PendingIntent.getBroadcast(context, 0,
105         new Intent(actionString), 0);
106
107     // If the USB manager isn't already enabled
108     if (enabled == false) {
109         // Create a new filter with the package name
110         // attach)
111
112         try {
113             Thread.sleep(500);
114         } catch (InterruptedException e1) {
115             // TODO Auto-generated catch block
116             e1.printStackTrace();
117         }
118
119         IntentFilter filter = new IntentFilter(actionString);
120         // Also add a few other actions to the intent...
121         filter.addAction(UsbManager.ACTION_USB_ACCESSORY_ATTACHED);
122         filter.addAction(UsbManager.ACTION_USB_ACCESSORY_DETACHED);
123         // and register the intent with the specified context
124         context.registerReceiver(receiver, filter);
125
126         UsbManager deviceManager = null;
127         UsbAccessory[] accessories = null;
128         UsbAccessory accessory = null;
129
130         // Get a UsbManager object from the specified intent (only works for
131         // v3.1+ devices)
132         deviceManager = (UsbManager) context
133             .getSystemService(Context.USB_SERVICE);
134
135         // If we were unable to get a UsbManager, return an error
136         if (deviceManager == null) {
137             return RETURN_CODES.DEVICE_MANAGER_IS_NULL;
138         }
139     }
140 }
```

- Bottom Status Bar:** Shows "Writable", "Smart Insert", and the line number "167 : 1".
- Bottom Taskbar:** Lists multiple open projects and files, including "アップデート・マ...", "hideo@amdx6: ~", "[hideo@amdx6: ~]", "[Android SDK Ma...", "Java - OpenAcc3x...", "OpenAccessoryD...", "[印刷 - localhost]", and "[...]".

# enable - 2

Accessoryの列挙

FD(EP)の取得

ReadThreadを作る

```
122     deviceManager = (UsbManager) Context.  
123             .getSystemService(Context.USB_SERVICE);  
124  
125     // If we were unable to get a UsbManager, return an error  
126     if (deviceManager == null) {  
127         return RETURN_CODES.DEVICE_MANAGER_IS_NULL;  
128     }  
129  
130     // Get a list of all of the accessories from the UsbManager  
131     accessories = deviceManager.getAccessoryList();  
132  
133     // If the list of accessories is empty, then exit  
134     if (accessories == null) {  
135         return RETURN_CODES.ACCESSORIES_LIST_IS_EMPTY;  
136     }  
137  
138     // Get the first accessory in the list (currently the Android OS  
139     // only  
140     // supports one accessory, so this is it)  
141     accessory = accessories[0];  
142  
143     // If the accessory isn't null, then let's try to attach to it  
144     if (accessory != null) {  
145         // If we have permission to access the accessory  
146         if (deviceManager.hasPermission(accessory)) {  
147             // Try to open a ParcelFileDescriptor by opening the  
148             // accessory  
149             parcelFileDescriptor = deviceManager.  
150                 .openAccessory(accessory);  
151  
152             if (parcelFileDescriptor != null) {  
153                 // Create a new read thread to handle reading data from  
154                 // the accessory  
155                 readThread = new ReadThread(parcelFileDescriptor);  
156                 readThread.start();  
157  
158                 deviceManager.requestPermission(accessory,  
159                     permissionIntent);  
160  
161             if (parcelFileDescriptor == null) {  
162                 Log.d(TAG, "USBAccessoryManager.enable(): parcelFileDescriptor == null");  
163             }  
164         }  
165     }  
166 }  
167  
168 }  
169  
170 }  
171  
172 }  
173  
174 }  
175  
176 }  
177  
178 }  
179  
180 }  
181  
182 }  
183  
184 }  
185  
186 }  
187  
188 }  
189  
190 }  
191  
192 }  
193  
194 }  
195  
196 }  
197  
198 }  
199  
200 }  
201  
202 }  
203  
204 }  
205  
206 }  
207  
208 }  
209  
210 }  
211  
212 }
```

# readThread

inputStream の設定

Thread Loop

read 同期

```
769     /*  
770     *  
771     private class ReadThread extends Thread {  
772         private boolean continueRunning = true;  
773  
774         private FileInputStream inputStream;  
775         private ParcelFileDescriptor myparcelFileDescriptor;  
776  
777         public ReadThread(ParcelFileDescriptor p) {  
778             myparcelFileDescriptor = p;  
779             inputStream = new FileInputStream(p.getFileDescriptor());  
780         }  
781  
782         @Override  
783         public void run() {  
784             byte[] buffer = new byte[1024]; // buffer for reading  
785             int bytes; // bytes returned from read()  
786  
787             while (continueRunning) {  
788                 try {  
789                     // Read from the InputStream  
790                     bytes = inputStream.read(buffer);  
791  
792                     // Send the obtained bytes to the UI Activity  
793                     byte[] data = new byte[bytes];  
794                     System.arraycopy(buffer, 0, data, 0, bytes);  
795  
796                     /*  
797                     * Synchronize to the readData object to make sure that no  
798                     * user API is accessing it at the moment  
799                     */  
800                     synchronized (readData) {  
801                         readData.add(data);  
802                     }  
803  
804                     handler.obtainMessage(  
805                         what,  
806                         bytes,  
807                         -1,  
808                         new USBAccessoryManagerMessage(  
809                             USBAccessoryManagerMessage.MessageType.READ  
810                         ));  
811                 } catch (IOException e) {  
812                     Log.e("ReadThread", "Error reading from input stream");  
813                 }  
814             }  
815         }  
816     }  
817 }
```

# enable - 3

Java - OpenAcc3x/src/com/microchip/android/BasicAccessoryDemo\_API12/USBAccessoryManager.java - Eclipse SDK

```
if (parcelFileDescriptor != null) {
    // Create a new read thread to handle reading data from
    // the accessory
    readThread = new ReadThread(parcelFileDescriptor);
    readThread.start();

    deviceManager.requestPermission(accessory,
        permissionIntent);

    if(parcelFileDescriptor == null) {
        Log.d(TAG, "USBAccessoryManager:enable() parcelFileDescriptor == null");
        return RETURN_CODES.FILE_DESCRIPTOR_WOULD_NOT_OPEN;
    }

    // Open the output file stream for writing data out to
    // the accessory
    outputStream = new FileOutputStream(
        parcelFileDescriptor.getFileDescriptor());

    if(outputStream == null) {
        Log.d(TAG, "USBAccessoryManager:enable() outputStream == null");
        try {
            parcelFileDescriptor.close();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return RETURN_CODES.FILE_DESCRIPTOR_WOULD_NOT_OPEN;
    }

    Log.d(TAG,
        "USBAccessoryManager:enable() outputStream open");

    // If the ParcelFileDescriptor was successfully opened,
    // mark the accessory as enabled and open
    enabled = true;
    open = true;

    handler.obtainMessage()
```

outputStreamの設定

# enable - 4

The screenshot shows the Eclipse IDE interface with the following details:

- Title Bar:** Java - OpenAcc3x/src/com/microchip/android/BasicAccessoryDemo\_API12/USBAccessoryManager.java - Eclipse SDK
- Toolbar:** Standard Eclipse toolbar with icons for file operations, search, and project navigation.
- Package Explorer:** Shows the project structure with packages like LED\_F\_ION, POT\_LOWER\_LIMIT, POT\_STATUS\_CHANGE, POT\_UPPER\_LIMIT, PUSHBUTTON\_STATUS\_CH, UPDATE\_LED\_SETTING, USBAccessoryWhat, accessoryManager, deviceAttached, firmwareProtocol, handler, TAG, and several methods like disconnectAccessory(), getFirmwareProtocol(), isButtonPressed(), LEDButtonEnable(), onCreate(), onPause(), onResume(), onSaveInstanceState(), onStart(), showErrorPage(), and updateButton().
- Code Editor:** Displays the Java code for `USBAccessoryManager.java`. The code is a synchronized block that reads data from a list. It checks if there is no data left, if it has read all data, or if the amount to read is larger than the buffer length. It then copies the data from the list to a buffer and removes the data from the list if the entire buffer is read.

```
482     /*
483      * Synchronize to the readData object so that the ReadThread doesn't try
484      * to add data to the list while we are accessing it.
485      */
486     synchronized (readData) {
487         /* while we still have data to read */
488         while (true) {
489             /*
490              * if there is no data left in the list, exit with the amount
491              * read already
492              */
493             if (readData.size() == 0) {
494                 return amountRead;
495             }
496
497             /*
498              * if we have read all of the data that we can store in the
499              * buffer provided, then exit.
500              */
501             if (amountRead == array.length) {
502                 return amountRead;
503             }
504
505             if ((array.length - amountRead) >= readData.get(0).length) {
506                 /*
507                  * If the amount we need to read is larger than or equal to
508                  * the data buffer for this node, then copy what is here and
509                  * remove it from the list
510                  */
511                 for (int i = 0; i < readData.get(0).length; i++) {
512                     array[amountRead++] = readData.get(0)[i];
513                 }
514                 readData.remove(0);
515             } else {
516                 /*
517                  * If the amount we need to read is less than the data
518                  * buffer for this node
519                  */
520                 int amountRemoved = 0;
521
522                 /*
523                  * then copy what we need */
524             }
525         }
526     }
527 }
```

- Bottom Status Bar:** Shows tabs for Writable, Smart Insert, and line number 167 : 1. It also includes standard Eclipse status bar icons.

# USBAccessoryManager.disable

- outputStreamをclose  
733: outputStream.close();

# BasicAccessoryDemo.java

# Handler(1)

```
private Handler handler = new Handler() {
    @Override
    public void handleMessage(Message msg) {
        byte[] commandPacket = new byte[2];

        switch(msg.what) {
            case UPDATE_LED_SETTING:
                if(accessoryManager.isConnected() == false) {
                    return;
                }

                commandPacket[0] = UPDATE_LED_SETTING;
                commandPacket[1] = 0;

                if(((LEDControl)findViewById(R.id.led_0)).getState()) {
                    commandPacket[1] |= LED_0_ON;
                }

                if(((LEDControl)findViewById(R.id.led_1)).getState()) {
                    commandPacket[1] |= LED_1_ON;
                }

                if(((LEDControl)findViewById(R.id.led_2)).getState()) {
                    commandPacket[1] |= LED_2_ON;
                }

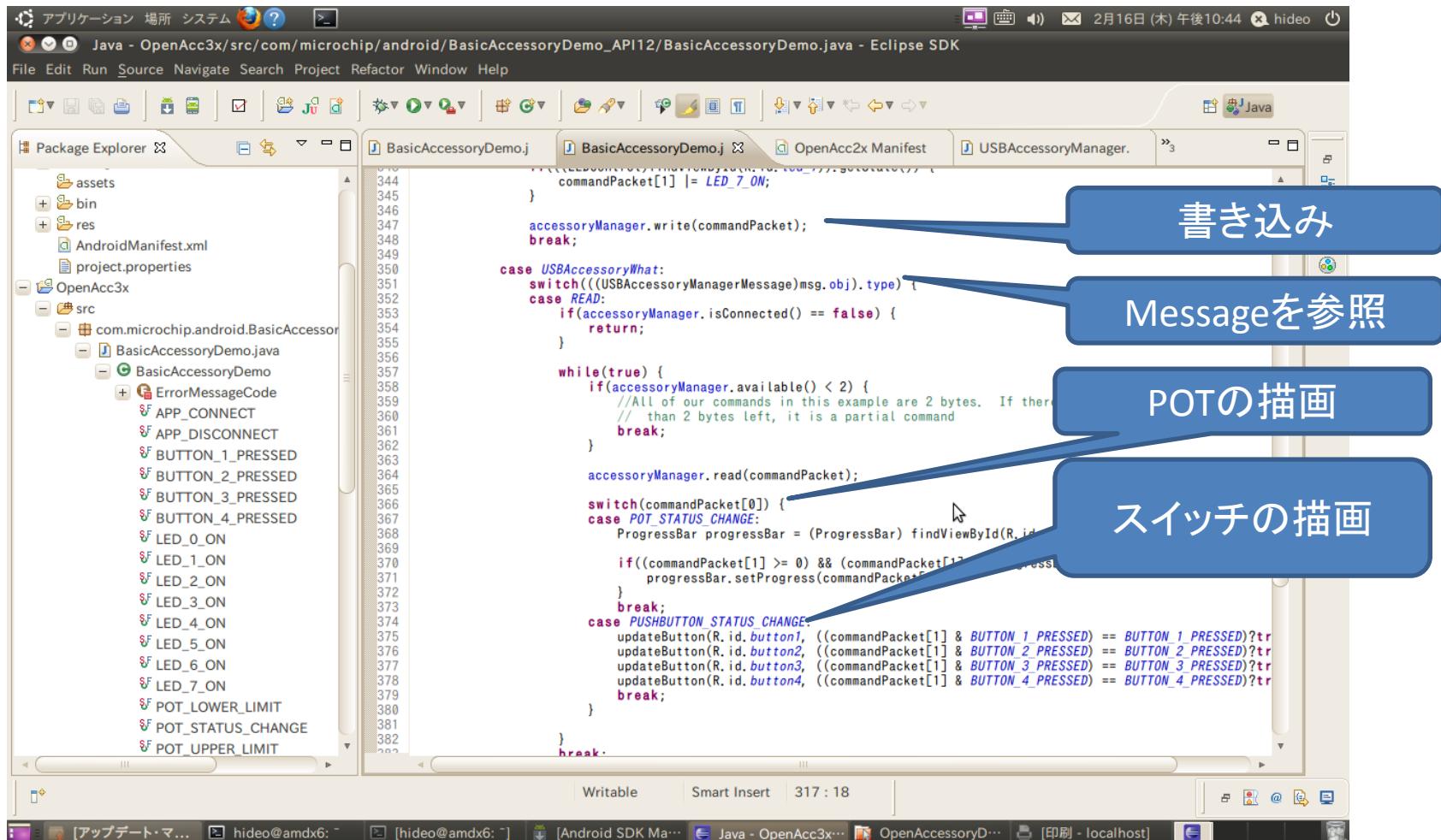
                if(((LEDControl)findViewById(R.id.led_3)).getState()) {
                    commandPacket[1] |= LED_3_ON;
                }

                if(((LEDControl)findViewById(R.id.led_4)).getState()) {
                    commandPacket[1] |= LED_4_ON;
                }

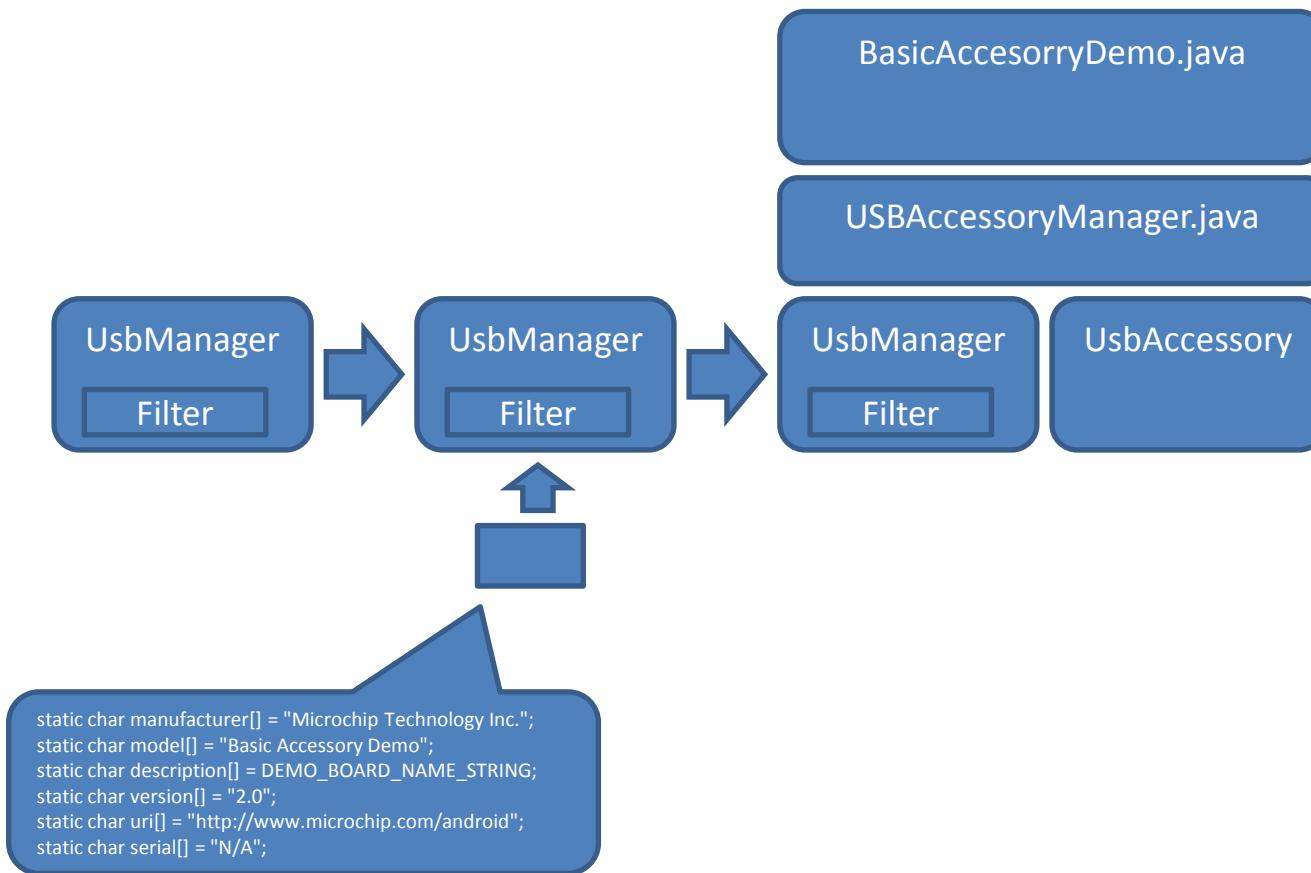
                if(((LEDControl)findViewById(R.id.led_5)).getState()) {
                    commandPacket[1] |= LED_5_ON;
                }

                if(((LEDControl)findViewById(R.id.led_6)).getState()) {
```

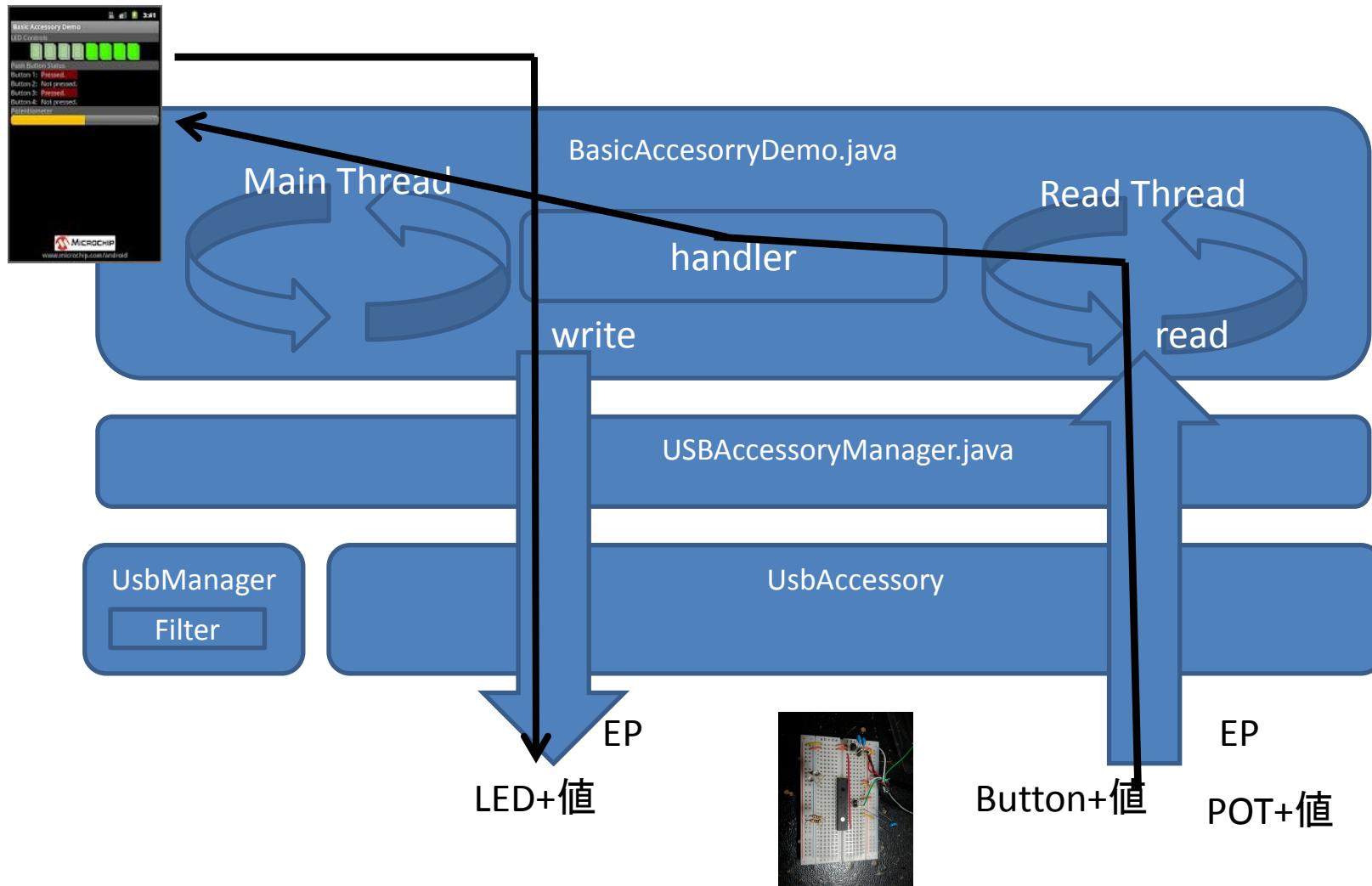
# Handler(2)



# APIの処理



# プログラム構造



# まとめ

- USB、ADKプロトコルを理解しなくても問題ない
  - Microchip社提供のソースに記述済
- プログラム変更に必要なこと
  - PICのピン・機能の割り当ての検討
  - アプリとのプロトコル{コマンド:値}を規定
  - 出力、入力、電圧値の入力の追加・削除
- このマイコンで他にできる事
  - PWM
    - モータ制御、LED輝度制御、RGB-LEDの色管理
  - SPI/I2C
    - 各種デジタルセンサ、周辺回路(CAN, RTC, 外部メモリなど)
  - UART
    - シリアル通信

おわり