# GR-SAKURAのUSBホスト機能で遊ぼう

OSC 2012 Kansai@Kyoto

2012/8/4

SAKURAボードユーザ会

Yuuichi Akagawa

# 自己紹介

- Yuuichi Akagawa (あかがわ　ゆういち)
- がじぇるねプロジェクトのベータテストに参画
- 仕組みを知ることが大好きなので、成果物がほとんどないのが特徴。
- 本職は某SI企業のインフラ担当。ツッコミ厳禁。

# USBホストと私

Android ADKをハックしていたら、
USBホスト遊びが楽しくなってしまった。


STM32F4 DiscoveryでADK

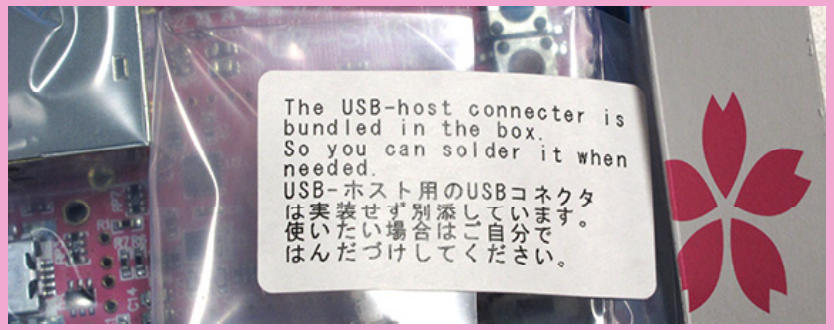

LM3S3748でADK
CQ出版「超入門!付属ARMマイコンで始めるロボット製作」基板


USB MIDI to レガシーMIDI変換
Harpy nano(Arduino互換)利用
→ USBH_MIDIライブラリ
https://github.com/YuuichiAkagawa/USBH_MIDI

★Arduino用ADKライブラリ Yaoadkも絶賛公開中。https://github.com/ADKstudyGroupTokyo/Yaoadk

# GR-SAKURAのUSBホスト(ハード)

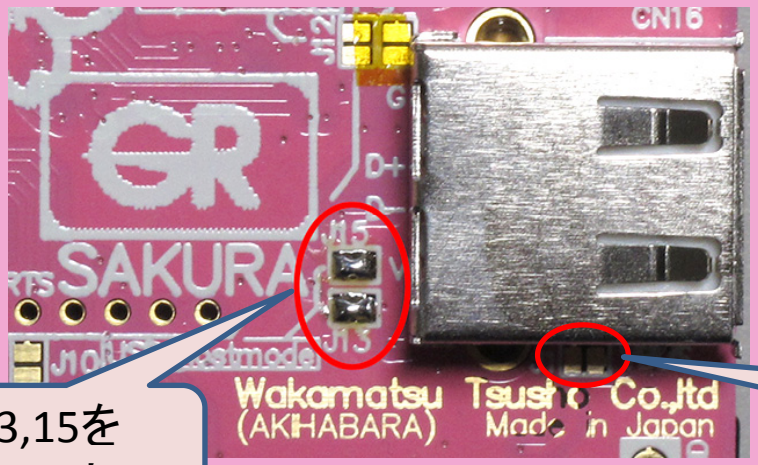
- Functionと共用のため排他利用(RX63NのUSB0)
  → ホスト時は仮想COMポートが使えない。
- GR-SAKURA-FULLを買ってもUSB-Aコネクタは未接続。




- GR-SAKURAでもUSB-Aコネクタを接続すれば利用可能。

# GR-SAKURAのUSBホスト(ハード)

- USBホストを使うには、まずハンダ付けから。
  1. 基板裏面にUSB-Aコネクタをハンダ付け
  2. J13,J15をハンダでショート
  3. J1をショートすると保護ダイオードがバイパスされる。必要に応じて。

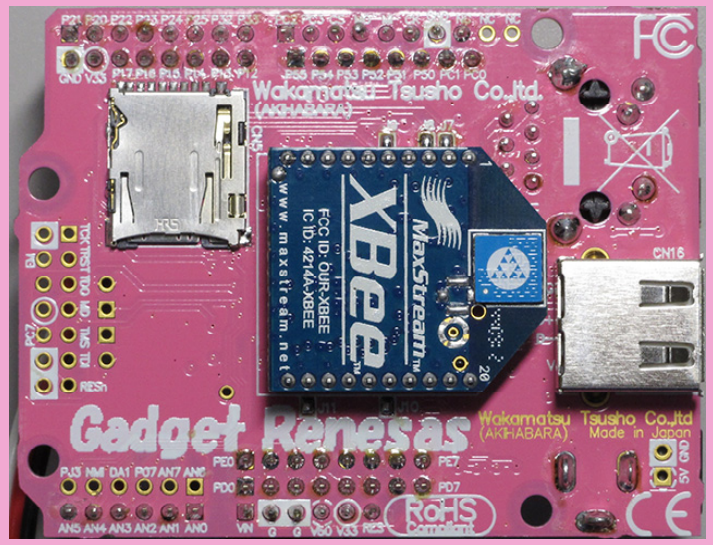



J13,15を
ショート

J1はお好みで。

# GR-SAKURAのUSBホスト(ハード)

- printfデバッグはXBee接続がオススメ




```
void setup()
{
   Serial.begin(9600,SCI_SCI2B); //XBee
   Serial.setDefault(); // printfをSerialに出力

    if(tkusbh_init() != TKUSBH_OK){
      printf("!!! USB Host Initialization ERROR !!!¥n");
      while(1);
   }
}
```

※写真は評価版基板

# GRのUSBホストライブラリ

- よくあるステートマシンを回すタイプではなく、同期処理っぽい作り。
- そのため、あんまり詳しくない人でも割と簡単に使える(超お手軽)。
- 詳しい人には痒いところに手が届かない。

# GRのUSBホストライブラリ(続)

## 関数一覧

| 関数名 | 機能 |
|---|---|
| tkusbh_init() | USBホストモジュールを初期化する |
| tkusbh_is_connected() | ターゲットが接続されているかどうかを調べる |
| tkusbh_connect() | ターゲットに接続する |
| tkusbh_disconnect() | ターゲットを切断する。切断されるまで待つ。 |
| tkusbh_get_descriptor() | 接続されているターゲットのディスクリプタを取得する |
| tkusbh_get_string() | 接続されているターゲットのストリングディスクリプタを取得する |
| tkusbh_control_msg() | コントロールトランザクションを発行する |
| tkusbh_set_configuration() | SET CONFIGURATIONを実行する |
| tkusbh_bulk_write() | バルクOUT転送を実行する |
| tkusbh_bulk_read() | バルクIN転送を実行する |
| tkusbh_start_interrupt_trans() | インタラプト転送を行う(開始する) |
| tkusbh_interrupt_write() | インタラプトOUT転送で送信されるデータを登録する |
| tkusbh_interrupt_read() | インタラプトIN転送で受信したデータを取得する |

# GRのUSBホストライブラリ(続)

## サポート状況

| 大分類 | 中分類 | ｻﾎﾟｰﾄ | 備考 |
|---|---|---|---|
| 転送タイプ | コントロール転送 | ○ | |
| | バルク転送 | ○ | エンドポイントはIN/OUT 1つずつ。 |
| | インタラプト転送 | ○ | エンドポイントはIN/OUT 1つずつ。<br>受信割り込み無し |
| | アイソクロナス転送 | × | RX63Nには機能あり。 |
| Speed | - | LS<br>FS | HSは未対応 |
| クラスドライバ | - | × | 何にも無いよ。 |
| HUB対応 | - | × | |
| その他 | ディスクリプタのパース | × | 支援なし。自力でがんばる。 |

# GRのUSBホストライブラリ(続)

- たったこれだけで、USBの接続処理が完了。

```
tkusbh_init();
tkusbh_connect();
```

- あとは、bulk_readなりbulk_writeするだけ。
- でも、エンドポイント番号は自分でディスクリプタをパースしないとわからない。USB詳しくない人はここでつまづくはず。
- ディスクリプタをパースしてデバイスを確認するのが定石だけど、しないのもアリなのかも。

# GRのUSBホストライブラリ(続)

- tkusbh_connect()を実行すると…

Get Device Descriptor(先頭8バイト)
Get Device Descriptor(ALL)
Set Address
Get String Descriptor(0)
Get Configuration Descriptor(先頭9バイト)
Get Configuration Descriptor(ALL)
Get String Descriptor(1)
Get String Descriptor(2)
Get String Descriptor(3)

ここまでやってくれる。(介入はできない)

※でも、Get Configuration Descriptorの結果は残らない。

# GRのUSBホストライブラリ(続)

- 自分は以下の感じで使うようにしてみた。

```
bool connected = false;
void setup()
{
   tkusbh_init();
}

void loop()
{
   if( isConnected() == false ){
      delay(100);
      return;
   }
   //接続済みなので、送受信する
   if(tkusbh_bulk_read(…) > 0 ){
      //受信データを処理
   }
   //何か送信
   if(tkusbh_bulk_write(…) < 0 ){
     //送信エラー
   }
   delay(); //デバイス側に指定があれば。
}
```

```
bool isConnected ()
{
   TKUSBH_RESULT rcode;
   //Check the disconnect state
   if(connected && tkusbh_is_connected() == TKUSBH_DISCONNECT){
      tkusbh_disconnect(TIMEOUT_INFINITE);
      isPollEnable = false;
      return false;
   }
   if(connected) //Already connected
      return true;
   if(tkusbh_connect(500) != TKUSBH_OK){//Start Enumeration
       return false;
    }
    if(!parseConfigDescr()){
      return false;
     }
    connected = true;
    return true;
}
bool parseConfigDescr()
{エンドポイント情報をパース}
```

# GRのUSBホストライブラリ(続)

| Configuration Descriptor | | |
|---|---|---|
| Configuration Descriptor | bLength | |
| | bDescriptorType | |
| | wTotalLength | 全体のサイズ |
| | bNumInterfaces | Interface数 |
| | bConfigurationValue | |
| | iConfiguration | |
| | bmAttributes | |
| | bMaxPower | |
| Interface Descriptor | bLength | |
| | bDescriptorType | |
| | bInterfaceNumber | |
| | bAlternateSetting | |
| | bNumEndpoints | Endpoint数 |
| | bInterfaceClass | USBクラス |
| | bInterfaceSubClass | USBサブクラス |
| | bInterfaceProtocol | |
| Endpoint Descriptor | bLength | |
| | bDescriptorType | |
| | bEndpointAddress | エンドポイント番号 |
| | bmAttributes | |
| | wMaxPacketSize | パケットサイズ |
| | bInterval | ポーリングインターバル |

コンフィギュレーションディスクリプタは

tkusbh_get_descriptor(CONFIG_DESCRIPTOR_TYPE, 0, buf, 128);

で取得できる。

これをパースしてエンドポイント番号なりを取得する。

デバイスで決め打ち出来るのであればデバイスディスクリプタの
・idVendor
・idProduct
の値を見れば良い。
tkusbh_connect()が成功していれば、
・USBHostInfo.DeviceDesc.idVendor
・USBHostInfo.DeviceDesc.idProduct
に代入されている。

# GRのUSBホストライブラリ(続)

- 2012/8/1現在のUSBホストライブラリの制約
  - USBホストを有効にすると、IO2,IO5が使えない
    →USB0_DPRPD, USB0_DRPDに割り当てられてしまう
    回避方法は

```
tkusbh_init() の後で以下を実行
    PORT2.PMR.BIT.B2 = 0;
    PORT2.PMR.BIT.B5 = 0;
```

※今後は、gUsbHostGpioPulldownという変数で制御できるようになるらしい…。

# 作例

# もちろんGR-SAKURAでもADK

6/12に待望のUSBホストライブラリが公開され、6/16のプロデューサミーティングに間に合わせるべくADKの実装に着手。

USBホストライブラリの不具合修正を依頼しつつ、なんとか当日動かせた。

(6/19のプレスに間に合った)

ソースコードはRulzに公開済み。

GR-SAKURAでADK。

# が、事件発生。

- Google I/O 2012にて、ADK2012発表。
- 従来のものはAOA1.0となり、新たにAOA2.0が登場。
- 大化けした。こんなに作れないよorz
  - ADK connection over Bluetooth
  - USB Audio / Bluetooth Audio(A2DP)
  - HID

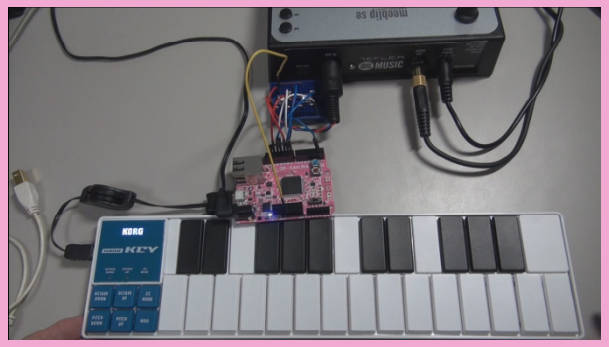
# AOA2.0の対応について

- USBオーディオって必要？
  - 実物見たときは、「おー。音出る」と思ったけど…
  - アイソクロナス転送が必要だね。
- Bluetooth経由
  - できれば実装したい。
  - BTドングルでも動かせるのかな？
- HID
  - やってみたけど、何故か動かない。困った。

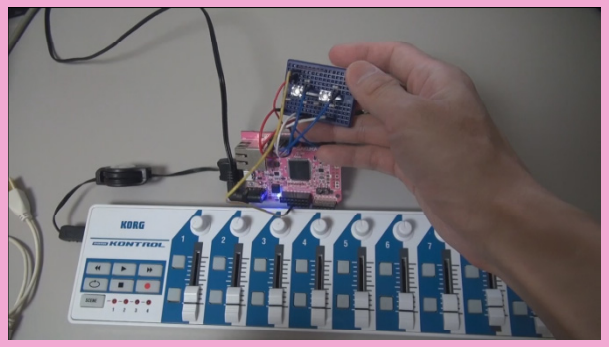
時間と技術力の許す限りやってみたいと思います。

# USBH_MIDI

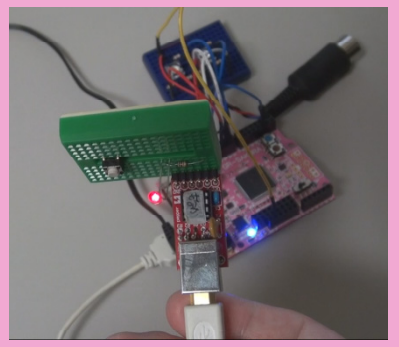
- Arduino用に作ったUSB-MIDIクラスドライバをGR-SAKURAにも移植。
- 音楽用ではないのでタイミング制御的なものは無く、MIDIメッセージの送受信が出来るというもの。



USB-MIDI to シリアルMIDI変換

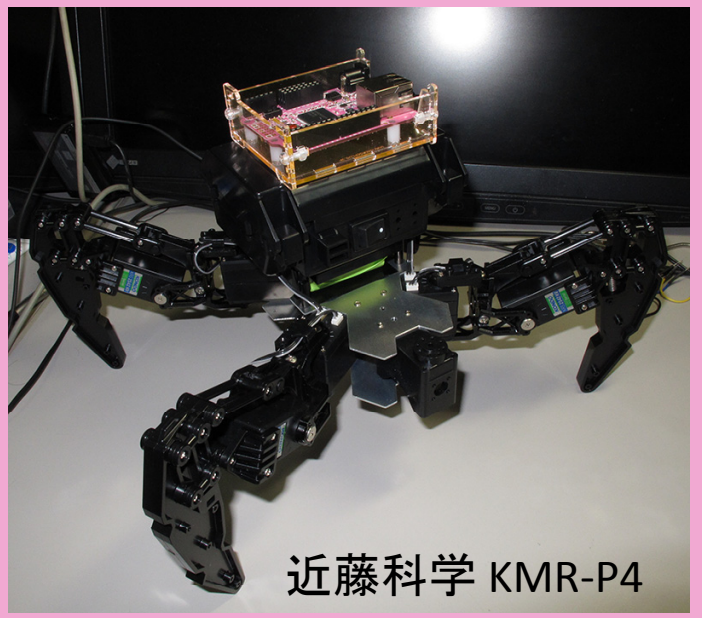


USB-MIDIコントローラでLED操作


Monakaと接続

# USBゲームコントローラ

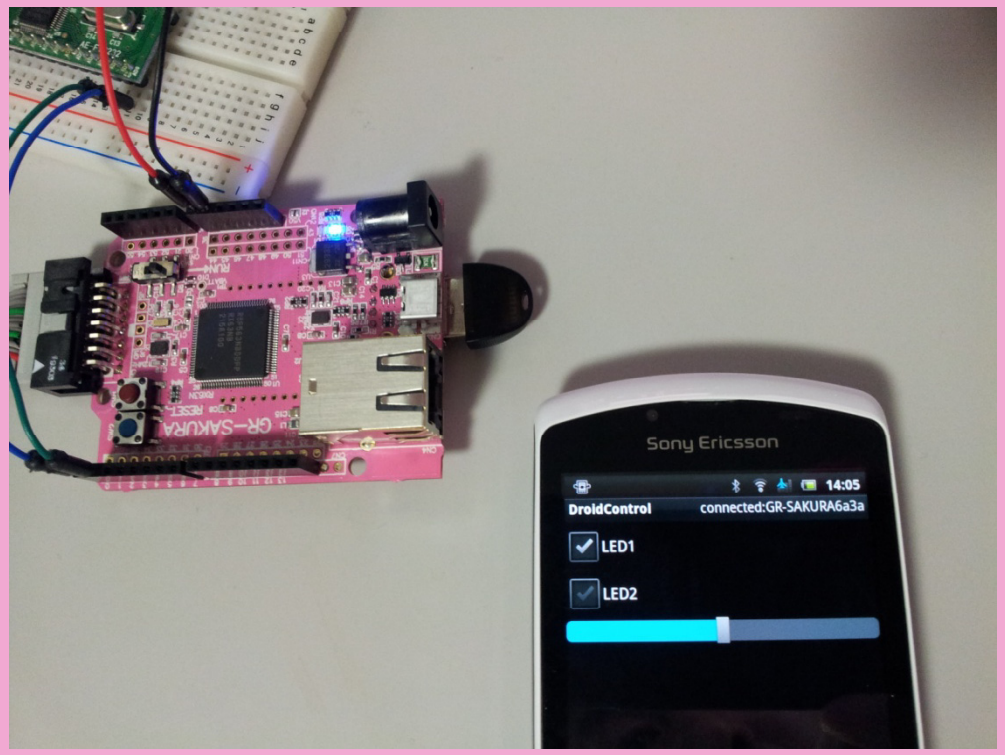
- Bluetooth使う人が多いけど、送信だけならこれで十分。
- これでラジコンしてみた。

# 他のメンバーの作例

# USB Bluetoothアダプタ

- @hrdakinoriさんのBluetooth(btstackの移植)

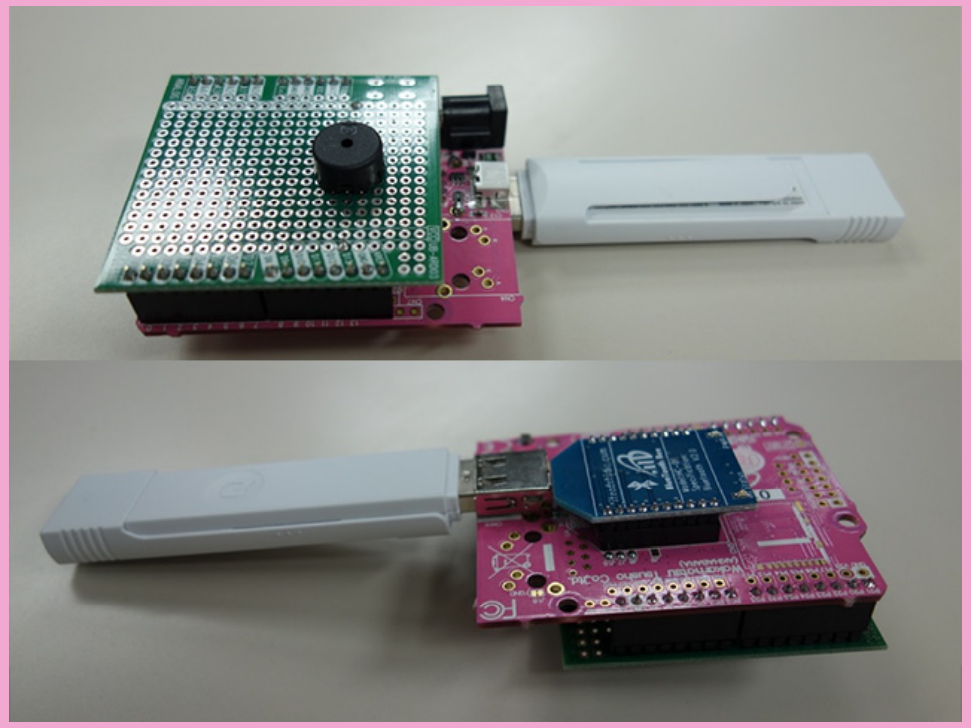



http://d.hatena.ne.jp/hrdakinori/20120729/1343542201

# USB NFCリーダ

- @KazuyukiEguchiさんのNFCリーダ

  読み取ったデータをBluetooth Bee経由でAndroid端末に送信



おしまい