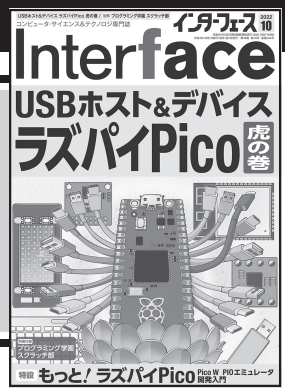


ラズパイPicoのUSB活用

第2回　USB CDCクラスを利用したFirmataライブラリ

関本 健太郎

2022年10月号特集は「USBホスト＆デバイス ラズパイPico虎の巻」でした．特集では，TinyUSBのサンプル・プログラムを詳しく解説したり，USBホスト，USBデバイスの製作事例を紹介したりしました．その中で，USB-I²Cブリッジについて，次のように解説しました．I²Cブリッジ製品には以下があります．

1，USB HIDクラスを利用したもの（2022年10月号特集第3部第1章）
2，USBベンダ・クラスを利用したもの（2022年11月号，pp.184-187）
3，USB CDCクラスを利用したもの（今回）

なお，上記1と2について，サポート・ページで公開しています．

https://interface.cqpub.co.jp/2311usb/

● 様々なプラットホームで利用できる

シリアル通信を利用したホストとデバイスの通信プロトコルとして，Firmataがあります．USBデバイスの場合には，USB CDCクラスを利用したUSBブリッジ機能を介して利用できます．

Firmataライブラリは，シリアル通信（USB CDCクラス）を利用して，ホストPC上のソフトウェアと通信するためのFirmataプロトコルを実装しています（**図1**）．Windows，Linux，macOSで動作します．これにより，使っているプログラミング環境用に独自のプロトコルやオブジェクトを作成しなくても，カスタム・ファームウェアを作成できます．Arduino IDEに特化したFirmataライブラリが公開されており，広く利用されています．そこでRP2040向けにビルドして利用したいと思います．

USBディスクリプタ

Arduino IDEのボード・マネージャで「Arduino Mbed OS RP2040 Boards」→「Raspberry Pi Pico」を選択したとき，USBディスクリプタ（**図2**）は，典型的なCDCクラスのディスクリプタ構成になっています．

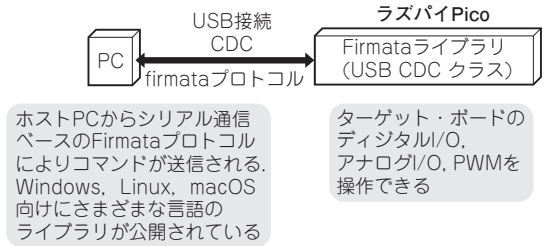


図1　Firmataライブラリの概要

Firmataプロトコル

Firmataは，コンピュータ（またはスマートフォン/タブレットなど）上のソフトウェアからマイコンと通信するためのプロトコルです．このプロトコルは任意のマイコンのファームウェアに実装できます．

Firmataはコマンド・バイトが8ビット，データ・バイトが7ビットというMIDIメッセージ・フォーマットに基づいています．Firmataのプロトコルを**表1**に示します．

Firmataメッセージ・フォーマットには，データ・メッセージ拡張（Data Messages Expansion，0xE0，0x90，0xC0，0xD0）とコントロール・メッセージ拡張（Control Messages Expansion）があります．

● データ・メッセージ拡張のフォーマットの指定方法

データ・メッセージ拡張のフォーマットは，第1バイトの上位4ビットがコマンド，下位4ビットがピン番号またはポート番号を指定します．データ・バイトが7ビットなので，8ビットのデータは，下位7ビットのデータをデータ部の第1バイトで，8ビット目のデータをデータ部の第2バイトの最下位ビットに割り当てます．例えば，

▶(1) ホストからデバイスのポート2の値を0x11001111に設定するコマンドを送信する場合

Firmataのプロトコルでは，0x92（下位4ビットが

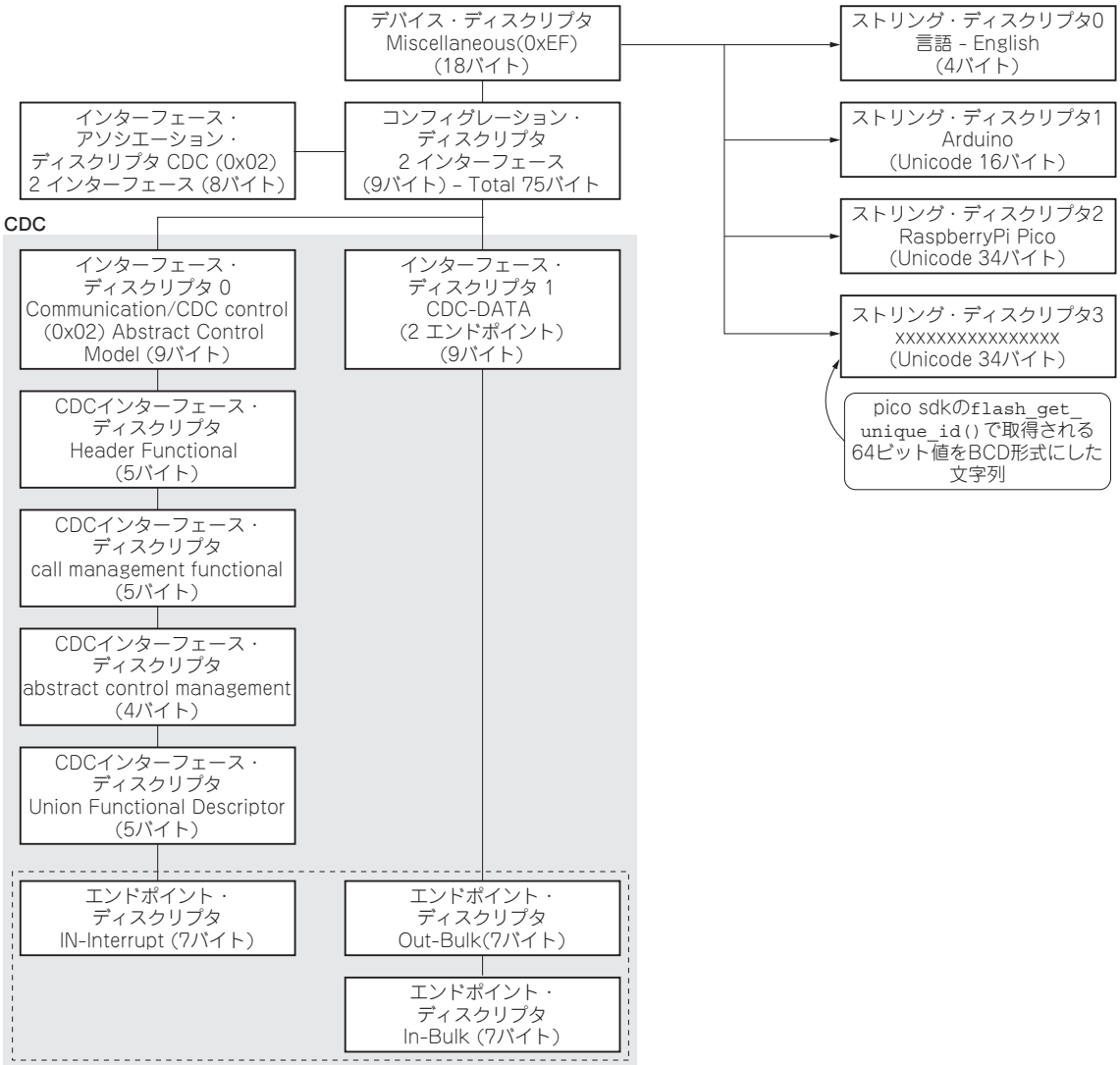


図2 Firmata USBディスクリプタ

ポート番号)，0x4F，0x01となり，この3バイトをホストからデバイスに送信します．

▶ (2) ホストからデバイスのポート2の状態を受信する場合

まず，ホストからディジタル・ポート・レポート・コマンドを送信します．Firmataのプロトコルでは，0xD2(下位4ビットがポート番号)，0x01(0：レポート無効化，1：レポート有効化)となり，この2バイトをホストからデバイスに送信します．コマンドを受け取ったデバイスは(1)で説明したフォーマットでポート2の8ビットの状態をホストに送信します．ホストでは，0x92，0x4F，0x01を受け取ることになります．

● コントール・メッセージ拡張のフォーマットの指定方法

例えばピン5番をアナログ・ピンに設定する場合のコマンドは，0xF4(ピン・モード設定コマンド)，0x05(ピン番号)，0x02(アナログ・ピン・モード値)となります(**表1**)．

● 拡張コマンド・セットの指定方法

拡張コマンド・セット(**表2**)は，Sysexベース(0xF0でSysexモードになった後)のサブコマンド(0x00 ～ 0x7F)として利用します．例えばホストからデバイスにファームウェア名とバージョンをリクエストする場合には，拡張コマンド開始，REPORT_FIRMWARE，拡張コマンド終了の手順(0xF0, 0x79,

表1　Firmataプロトコル（データ・メッセージ拡張およびコントール・メッセージ拡張）
ピン・モード（INPUT/OUTPUT/ANALOG/PWM/SERVO/I2C/ONEWIRE/STEPPER/ENCODER/SERIAL/PULLUP）は0/1/2/3/4/6/7/8/9/10/11に割り当てられている．拡張コマンド・セットはSyexeベース（0xF0でSyexeモードになった後）のサブコマンド（0x00～0x7F）として利用する．

分類	タイプ	第0バイト		第1バイト	第2バイト
		上位4ビット	下位4ビット		
データ・メッセージ拡張のフォーマット	アナログ I/O メッセージ	0xE0	ピン番号	LSB（ビット0～6）	MSB（ビット7～13）
	ディジタルI/O メッセージ	0x90	ポート	LSB（ビット0～6）	MSB（ビット7～13）
	アナログ・ピン・レポート	0xC0	ピン番号	無効/有効（0/1）	n/a
	ディジタル・ポート・レポート	0xD0	ポート	無効/有効（0/1）	n/a
コントール・メッセージ拡張のフォーマット	Sysex開始	0xF0		−	−
	ピン・モード設定（I/O）	0xF4		ピン番号（0～127）	ピン・モード
	ディジタル・ピン値設定	0xF5		ピン番号（0～127）	ピン値（0/1）
	Sysex終了	0xF7			
	プロトコル/バージョン	0xF9		メジャー・バージョン	マイナ・バージョン
	システム・リセット	0xFF		−	−
拡張コマンド・セット	文字列	0x71		文字の下位7ビット	文字の上位1ビット…文字の最後は0xF7
	ファームウェア名/バージョン	0x79		メジャー・バージョン	マイナ・バージョン

表2　Firmata拡張コマンド・セット

拡張コマンド・セット	値	内　容
EXTENDED_ID	0x00	値0x00は，次の2バイトが拡張IDを定義することを示します
RESERVED	0x01～0x0F	ID 0x01～0x0Fはユーザ定義のコマンド用に予約されています
ANALOG_MAPPING_QUERY	0x69	アナログからピン番号へのマッピングを依頼する
ANALOG_MAPPING_RESPONSE	0x6A	マッピング情報で返信
CAPABILITY_QUERY	0x6B	サポートされているモードと全てのピンの分解能を尋ねる
CAPABILITY_RESPONSE	0x6C	サポートされているモードと分解能で返信する
PIN_STATE_QUERY	0x6D	ピンの現在のモードと状態（値とは異なる）を要求する
PIN_STATE_RESPONSE	0x6E	ピンの現在のモードと状態（値とは異なる）で応答する
EXTENDED_ANALOG	0x6F	任意のピンへのアナログ書き込み（PWM，サーボなど）
STRING_DATA	0x71	1文字あたり14ビットの文字列メッセージ
REPORT_FIRMWARE	0x79	レポート名とファームウェアのバージョン
SAMPLING_INTERVAL	0x7A	アナログ入力がサンプリングされる間隔（デフォルト=19ms）
SYSEX_NON_REALTIME	0x7E	MIDI非リアルタイム・メッセージ用に予約済み
SYSEX_REALTIME	0X7F	MIDIリアルタイム・メッセージ用に予約済み

表3　ホストからデバイスへのコマンド（ファームウェア名，バージョン）

番　号	内　容	値
0	拡張コマンド開始	0xF0
1	REPORT_FIRMWARE	0x79
2	拡張コマンド終了	0xF7

表4　デバイスからホストへのレポート（ファームウェア名，バージョン）

番　号	内　容	値
0	拡張コマンド開始	0xF0
1	REPORT_FIRMWARE	0x79
2	メジャー・バージョン	0～127
3	マイナ・バージョン	0～127
4	ファームウェア名最初の文字（LSB）	??
5	ファームウェア名最初の文字（MSB）	??
6	ファームウェア名次の文字（LSB）	??
7	ファームウェア名次の文字（MSB）	??
⋮		??
N	拡張コマンド終了	0xF7

0xF7）をホストからデバイスに送信します（**表3**）．コマンドを受け取ったデバイスはホストに，**表4**の手順でレポートを送信します．

ホストPC用インターフェース・ライブラリ

Firmataには，ホストPCで動作するArduino向け

表5　Arduino Firmata向けのクライアント・ライブラリ

言　語	URL
Processing	https://github.com/firmata/processing
	http://funnel.cc
Python	https://github.com/MrYsLab/pymata4
	https://github.com/MrYsLab/pymata-express
	https://github.com/tino/pyFirmata
	https://github.com/lupeke/python-firmata
	https://github.com/firmata/pyduino
Perl	https://github.com/ntruchsess/perl-firmata
	https://github.com/rcaputo/rx-firmata
Ruby	https://github.com/hardbap/firmata
	https://github.com/PlasticLizard/rufinol
	http://funnel.cc
Clojure	https://github.com/nakkaya/clodiuno
	https://github.com/peterschwarz/clj-firmata
Javascript	https://github.com/firmata/firmata.js
	https://github.com/rwldrn/johnny-five
	http://breakoutjs.com
	https://nodered.org/docs/faq/interacting-with-arduino#firmata
Java	https://github.com/kurbatov/firmata4j
	https://github.com/4ntoine/Firmata
	https://github.com/reapzor/FiloFirmata
.NET	https://github.com/SolidSoils/Arduino
	http://www.acraigie.com/programming/firmatavb/default.html
Flash/AS3	http://funnel.cc
	http://code.google.com/p/as3glue/
Pharo	https://github.com/pharo-iot/Firmata
PHP	https://github.com/ThomasWeinert/carica-firmata
	https://github.com/oasynnoum/phpmake_firmata
Haskell	http://hackage.haskell.org/package/hArduino
iOS	https://github.com/jacobrosenthal/iosfirmata
Dart	https://github.com/nfrancois/firmata
Max/MSP	http://www.maxuino.org/
	https://github.com/NullMember/MaxFirmata
Elixir	https://github.com/kfatehi/firmata
Modelica	https://www.wolfram.com/system-modeler/libraries/model-plug/
Go	https://github.com/kraman/go-firmata
vvvv	https://vvvv.org/blog/arduino-second-service
open Frameworks	http://openframeworks.cc/documentation/communication/ofArduino/
Rust	https://github.com/zankich/rust-firmata
Pure Data	https://github.com/NullMember/PDFirmata

のさまざまな言語に対応したクライアント・ライブラリが公開されています（**表5**）．

ビルド

● Arduino IDEのインストール

ソースファイルは，GitHubのリポジトリ（https://github.com/firmata/arduino）で公開されています．RP2040向けにはArduino IDEを利用すると良いでしょう．Arudino IDEをホストPCにインストールします．Windows 10環境を想定します．Windows 10の場合には，Arduino IDEはマイクロソフト・ストアで公開されています（**図3**）．

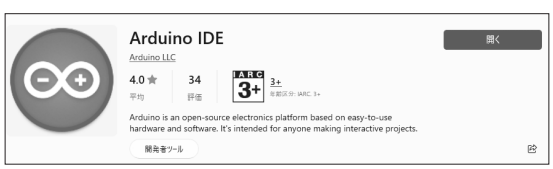


図3　Microsoft Storeから Arduino IDEのインストール

● Mbed OS RP2040 Boardsの登録

Arduino IDEを起動し，「ツール」→「ボード」→「ボードマネージャ」でボードマネージャを開き，「rp2040」で検索し，Arduino Mbed OS RP2040 Boardsの最新版（2022年5月時点で3.1.1）を登録します．

● Firmataライブラリの登録

次に，「スケッチ」→「ライブラリをインクルード」→「ライブラリを管理…」でライブラリ・マネージャを開き，「firmata」を検索し，最新版（2.5.7）を登録します．Arduino IDEにデフォルトでインストールされている場合には，この手順はスキップできます．

● RP2040向けの設定変更

2023年9月時点では，Firmataライブラリに RP2040向けの設定はデフォルトで含まれていませんでした．ConfigurableFirmataのGitHubのリポジトリ[7]には，RP2040向けの設定変更方法が記載されていま

リスト1　RP2040向けのboard.hの変更点

```
// Raspberry Pi Pico
// https://datasheets.raspberrypi.org/pico/
                                Pico-R3-A4-Pinout.pdf
#elif defined(TARGET_RP2040) || defined(TARGET_RASPBE
                                        RRY_PI_PICO)

#include <stdarg.h>

static inline void attachInterrupt(pin_size_t
      interruptNumber, voidFuncPtr callback, int mode)
{
   attachInterrupt(interruptNumber, callback,
                                    (PinStatus) mode);
}

#define TOTAL_ANALOG_PINS    4
#define TOTAL_PINS           30
#define VERSION_BLINK_PIN    LED_BUILTIN
#define IS_PIN_DIGITAL(p)    (((p) >= 0 && (p) < 23)
                               || (p) == LED_BUILTIN)
#define IS_PIN_ANALOG(p)     ((p) >= 26 && (p) < 26 +
                                    TOTAL_ANALOG_PINS)
#define IS_PIN_PWM(p)        digitalPinHasPWM(p)
#define IS_PIN_SERVO(p)      (IS_PIN_DIGITAL(p)
                                        && (p) != LED_BUILTIN)
// From the data sheet I2C-0 defaults to GP 4 (SDA)
//  & 5 (SCL) (physical pins 6 & 7)
// However, v2.3.1 of mbed_rp2040 defines WIRE_
// HOWMANY to 1 and uses the non-default GPs 6 & 7:
//#define WIRE_HOWMANY  (1)
//#define PIN_WIRE_SDA             (6u)
//#define PIN_WIRE_SCL             (7u)
#define IS_PIN_I2C(p)            ((p) == PIN_WIRE_SDA
                                || (p) == PIN_WIRE_SCL)
// SPI-0 defaults to GP 16 (RX / MISO), 17 (CSn),
// 18 (SCK) & 19 (TX / MOSI) (physical pins 21, 22,
// 24, 25)
#define IS_PIN_SPI(p)            ((p) == PIN_SPI_SCK
         || (p) == PIN_SPI_MOSI || (p) == PIN_SPI_MISO
                                  || (p) == PIN_SPI_SS)
// UART-0 defaults to GP 0 (TX) & 1 (RX)
#define IS_PIN_SERIAL(p)         ((p) == 0 || (p) == 1
       || (p) == 4 || (p) == 5 || (p) == 8 || (p) == 9
  || (p) == 12 || (p) == 13 || (p) == 16 || (p) == 17)
#define PIN_TO_DIGITAL(p)        (p)
#define PIN_TO_ANALOG(p)         ((p) - 26)
#define PIN_TO_PWM(p)            (p)
#define PIN_TO_SERVO(p)          (p)
```

す．それに従って，board.h（C:\Users\ユーザ名\Documents\Arduino\libraries\Firmata）ファイルを変更します．823行目あたりの“#else”の前に**リスト1**を追加します．

● StandardFirmataアプリケーションのビルド

Firmataのファームウェアとして，Standard Firmataプログラムを利用します．Arduino IDEを起動し，「ファイル」→「スケッチ例」→「Firmata」→「StandardFirmata」を開きます．StandardFirmataでは，デフォルトでGPIOピンが0～15までしかサポートされていません．そこでRP2040向けにcheckDigitalInputs関数中に，GPIOピンの処理を追加します（**リスト2**）．

Arduino IDEにおいて，「スケッチ」→「検証・コンパイル」および「マイコン・ボードに書き込む」で，Picoボードに書き込んでください．

PCからの操作

● Windows

WindowsからのFirmataの機能を素早く確認するには，Microsoft Storeの“Windows Remote Arduino Experience”アプリケーションを利用できます．Arduino IDEと同様の手順でインストールします．起動し，**図4**（**a**）の「Connection」メニューでConnectionとしてUSB，Baud rateとして57600を選択し，接続されているシリアル・ポートを選択し，［Connect］ボタンを押します．接続されてデフォルトでDigitalメニューが開きます．動作確認として，**図4**（**b**）にあるPicoのLEDが接続されているPin 25（GP25）を「0v」の欄をクリックしてみましょう．LEDが点灯するはずです．

● Linux

Ubuntu 22.04の環境上でPythonでFirmata機能にアクセスしてみます．ログインユーザのPython環境にpyfirmataをインストールします．

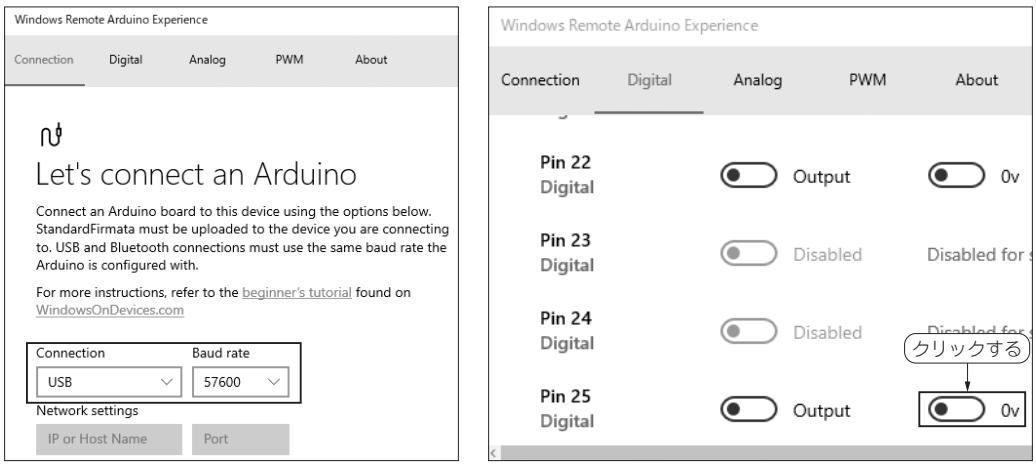
```
pip3 install pyfirmata
```

ボードの定義ファイル（~/.local/lib/python3.10/site-packages/pyfirmata/boards.py）をPico用に変更します．ここでは，ディジタル・ピンのインデックスを14から30に変更します（**リスト3**）．

Picoをホストに接続します．/dev/ttyACM0として検出されますので，デバイスを操作できるよう

リスト2　RP2040向けのStandardFirmata.inoの変更点

```
void checkDigitalInputs(void)
省略
  if (TOTAL_PORTS > 16 && reportPINs[16]) outputPort(16, readPort(16, portConfigInputs[16]), false);
  if (TOTAL_PORTS > 17 && reportPINs[17]) outputPort(17, readPort(17, portConfigInputs[17]), false);
省略
  if (TOTAL_PORTS > 28 && reportPINs[28]) outputPort(28, readPort(28, portConfigInputs[28]), false);
  if (TOTAL_PORTS > 29 && reportPINs[29]) outputPort(29, readPort(29, portConfigInputs[29]), false);
```



（a）ConnectionとしてUSB，Baud rateとして57600を選択する

（b）Pin 25の「0v」の欄をクリック

図4　Windows環境でのFirmataデバイスの制御

に，ログイン・ユーザをdialoutグループに追加します．

```
sudo usermod -a -G dialout $USER
```

firmata_gpio.py（**リスト4**）は，PicoのLEDが接続されているPin 25（GP25）を“H”に設定し，1秒スリープし，“L”に設定し，1秒スリープし，終了するプログラムです．

リスト3　ボードの定義ファイル（board.py）

```
BOARDS = {
    'arduino': {
        'digital': tuple(x for x in range(30)),
                                          # 14->30
        'analog': tuple(x for x in range(6)),
        'pwm': (3, 5, 6, 9, 10, 11),
        'use_ports': True,
        'disabled': (0, 1)  # Rx, Tx, Crystal
    },
省略
}
```


◆参考文献◆

(1) Firmata protocol.
https://github.com/firmata/protocol
(2) firmata/ConfigurableFirmata.
https://github.com/firmata/ConfigurableFirmata/blob/master/BoardSupport.md

せきもと・けんたろう

リスト4　PicoのLEDが接続されている25ピンを操作するfirmata_gpio.py

```
import time
import pyfirmata

# start connection to Arduino
#  USB: /dev/ttyUSB0 or /dev/ttyACM0
board = pyfirmata.Arduino('/dev/ttyACM0')

board.digital[25].write(1)  # LED on
time.sleep(1)               # 1s delay
board.digital[25].write(0)  # LED off
time.sleep(1)               # 1s delay
board.exit()                # exit
```