

RT コンポーネント操作マニュアル

RT ミドルウェア環境への 電子デバイスの導入を容易にするための **USB-GPIO, SPI, I²C**変換コンポーネント

2022 年 7 月 26 日版

芝浦工業大学

デザイン工学部デザイン工学科

佐々木 毅

更新履歴

2021 年 11 月 12 日版 第 1 版。

2022 年 7 月 26 日版 OpenRTM-2.0.0 での動作確認。連絡先の変更。

目次

1	本コンポーネントの概要	1
1.1	開発の背景	1
1.2	開発環境	1
1.3	本稿で利用しているコンポーネント群	2
2	コンポーネントの説明	2
2.1	USB-GPIO, SPI, I ² C 変換コンポーネント (Adafruit_FT232H_Breakout)	2
2.2	GPIO の利用例で使用するコンポーネント群	5
2.2.1	入力データ変化カウントコンポーネント (CountChange)	5
2.2.2	Lookup Table コンポーネント (ConfigLUT)	6
2.3	シリアル通信の利用例で使用するコンポーネント群	7
2.3.1	OctetSeq データ出力コンポーネント (SendOctetSeq)	7
2.3.2	気圧・温湿度センサ計算コンポーネント (BME280Decode)	7
3	Adafruit_FT232H_Breakout の使用方法	8
3.1	使用前の準備	8
3.2	RTSystemEditor によるシステム構築の手順	9
3.3	Adafruit_FT232H_Breakout の使用手順	9
3.3.1	GPIO の利用	10
3.3.2	シリアル通信の利用	12
4	Adafruit_FT232H_Breakout の利用例	13
4.1	GPIO の利用例	13
4.1.1	ポート D を用いたスイッチ情報の取得と LED の点消灯	13
4.1.2	ポート C を用いた LED アレイの点消灯	15
4.2	シリアル通信の利用例	17
5	お問い合わせ	21

1 本コンポーネントの概要

1.1 開発の背景

ロボットシステムや RT システムでは、センサや LED といった電子部品や、モータなどのアクチュエータを利用した開発が行われます。このような場合には GPIO (General Purpose Input/Output) ポートや、SPI や I²C 通信といったシリアル通信インタフェースが必要となるため、PC に加えて Raspberry Pi や Arduino といったワンボードコンピュータやワンボードマイコンを利用して RT コンポーネントの開発を行う必要がありました。そこで、USB ポートを介して GPIO やシリアル通信機能を利用することができる USB-GPIO, SPI, I²C 変換ボードをコンポーネント化することとしました。

開発した USB-GPIO, SPI, I²C 変換コンポーネントには以下のような機能があります。

- デジタル入出力機能（入力・出力はピンごとに選択可能）
- シリアル通信（I²C, SPI）機能

いずれの機能についても設定は Configuration から行うことができ、またデータは全てコンポーネントの入出力ポートを通してやり取りできるため、プログラムの記述やマイコンへの書き込みを行うことなくこれらの機能を利用することができます。また、様々な OS (Windows, Mac OSX, Linux) で利用することが可能です。

コンポーネントの設計指針等につきましては、

佐々木毅, “RT ミドルウェア環境への電子デバイスの導入を容易にするための USB-GPIO, SPI, I²C 変換コンポーネント”, 第 23 回計測自動制御学会システムインテグレーション部門講演会, 2022.

に詳細がありますので、そちらもご参照いただけましたら幸いです。

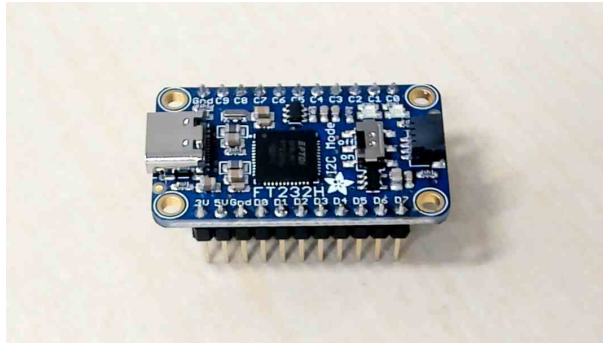
1.2 開発環境

本コンポーネント群は Linux 及び Windows にて動作確認を行っています。ソフトウェア開発環境は以下の通りです。

- Windows 10
- RT ミドルウェア: OpenRTM-aist-2.0.0-RELEASE (Python 版) / OpenRTM-aist-1.2.1-RELEASE (Python 版)
- Python: Python 3.10.5 / Python 3.7.5
- OpenRTP: OpenRTP 2.0.0-RELEASE / OpenRTP 1.2.1-RELEASE

開発したコンポーネントのうち、USB-GPIO, SPI, I²C 変換コンポーネント (Adafruit_FT232H_Breakout) については以下のライブラリとハードウェアを利用しています。

- 依存ライブラリ
 - pyusb (Windows の場合) , libusb (Mac OSX, Linux の場合)
 - pyftdi
 - Adafruit Blinka
- 利用ハードウェア
 - USB-GPIO, SPI, I²C 変換ボード Adafruit FT232H Breakout ボード [1]



1.3 本稿で利用しているコンポーネント群

USB-GPIO, SPI, I²C 変換コンポーネントとその操作例を示すものとして、以下のコンポーネント群を開発しました。

- USB-GPIO, SPI, I²C 変換コンポーネント (Adafruit_FT232H_Breakout)

Adafruit FT232H Breakout ボードを PC に USB 接続することで GPIO や SPI, I²C 通信を使用可能にするコンポーネント。

- 入力データ変化カウントコンポーネント (CountChange)

入力値が 1 つ前の入力値と比べ指定の変化をするたびに出力値が増加するコンポーネント。

- Lookup Table コンポーネント (ConfigLUT)

Configuration に列挙した値のうちの 1 つを入力値に応じて出力する 1 次元の Lookup Table (LUT) コンポーネント。

- OctetSeq データ出力コンポーネント (SendOctetSeq)

コンソールから入力した値を TimedOctetSeq 型のデータとして出力するコンポーネント。

- 気圧・温湿度センサ計算コンポーネント (BME280Decode)

気圧・温湿度センサ BME280 からシリアル通信で得られた計測値 (バイト列) を、気圧、温度、湿度の値に変換して出力するコンポーネント。

それぞれのコンポーネントの詳細については 2 章を、使用方法については 3 章を参照してください。

2 コンポーネントの説明

2.1 USB-GPIO, SPI, I²C 変換コンポーネント (Adafruit_FT232H_Breakout)

Adafruit_FT232H_Breakout は、Adafruit FT232H Breakout ボード [1] を使用した USB-GPIO, SPI, I²C 変換コンポーネントです。Configuration で GPIO やシリアル通信の設定を行うことができます。変数名が GPIO から始まる変数は GPIO の設定に関するもの、SER から始まる変数はシリアル通信の設定に関するもの (SPI, I²C 共通の設定)、I²C から始まる変数は I²C 通信の設定に関するもの、SPI から始まるものは SPI 通信の設定に関するものになっています。

GPIO に関しては、まず非アクティブ状態の時に Configuration から各ピンをデジタル入力として使うか、デジタル出力として使うかを選択します。アクティブ状態では、出力として設定したピンに対しては、対応する InPort に値が入力されると、その値に応じて High または Low を出力します。入力として設定したピンに対しては、High または Low を読み込み、それに応じた値を対応する OutPort から出力します。I²C/SPI についても、ま

ず非アクティブ状態の時に Configuration からいずれかを利用するか、利用するならばそのパラメータを設定します。送信するデータや受信バイト数はアクティブ状態の時に InPort から入力し、受信したデータは OutPort から出力されます。

非アクティブ化すると、出力ピンに設定された全ての GPIO から Low が出力されます。

本コンポーネントを利用するには Adafruit FT232H Breakout ボードと、ライブラリとして pyusb (Windows の場合) もしくは libusb (Mac OSX, Linux の場合), pyftdi, Adafruit Blinka のインストールが必要です。

● InPort

名称	型	説明
C7_0out	TimedLong	C7～C0 のうち、出力ピンとしているピンからの出力に対応した値。C7 から順に、High とする場合は 1、Low とする場合は 0 とした 8 ビットの値を入力する。入力ピンとして指定されているピンに対してはどちらを入力しても影響はない。例えば、C7, C5, C4 を High、他を Low とする場合には 10110000、つまり 176 を入力する。
D*out	TimedLong	D*を出力ピンとしている場合のピンからの出力に対応した値。High とする場合は 1 を、Low とする場合は 0 を入力する。入力ピンとして指定されている場合はどちらを入力しても影響はない。
I2C_SPLwcommand_rbytes	TimedOctetSeq	I ² C もしくは SPI 通信で送信するデータ・受信するデータのバイト数。最終要素の前までが送信データ、最終要素が受信バイト数となる。例えば、1, 2, 3 というデータの場合は 1, 2 という 2 バイトの値をデバイスに送信し、その後 3 バイトの情報をデバイスから受信する。受信バイト数が 0 の場合は送信だけを行い、受信を行わない (例:1, 2, 0)。受信バイト数のみが指定されている場合 (要素が 1 つの場合) は送信を行わず受信だけを行う (例:3)。

● OutPort

名称	型	説明
C7_0in	TimedLong	C7～C0 のうち、入力ピンとしているピンへの入力に対応する値。C7 から順に、High の場合は 1、Low もしくは出力ピンとして指定している場合は 0 とした 8 ビットの値が出力される。例えば、C7, C5, C4 が High、他が Low もしくは出力ピンである場合には 10110000、つまり 176 が出力される。全てのピンが出力ピンに指定されている場合は出力を行わない。
D*in	TimedLong	D*を入力ピンとしている場合のピンへの入力に対応する値。High の場合は 1 が、Low の場合は 0 が出力される。出力ピンに指定されている場合は出力を行わない。
I2C_SPLread	TimedOctetSeq	I ² C もしくは SPI 通信で受信したデータ。

● Configuration 変数

名称	型	デフォルト値	説明
GPIO_C7_0_IO_select	string	0b11111111	C7～C0 をそれぞれデジタル入力ピンとして使用するか、デジタル出力ピンとして指定するかを指定する。C7 から順に入力ピンとして指定する場所は 1、出力ピンとして使用するところは 0 とし、それを 2 進法で表現された数とみなして整数値を入力する。0b や 0x を値の前につけることで 2 進法や 16 進法で指定することも可能。例えば、C7, C5, C4 を出力ピン、他を入力ピンとする場合は 01001111 となるため、そのまま 2 進法で 0b01001111 としても良いし、16 進法に直して 0x4f、10 進法に直して 79 としても良い。アクティブ状態での変更は無効（反映されない）。
GPIO_D7_4_IO_select	string	0b1111	D7～D4 をそれぞれデジタル入力ピンとして使用するか、デジタル出力ピンとして指定するかを指定する。D7 から順に入力ピンとして指定する場所は 1、出力ピンとして使用するところは 0 とし、それを 2 進法で表現された数とみなして整数値を入力する。0b や 0x を値の前につけることで 2 進法や 16 進法で指定することも可能。例えば、D7,D5,D4 を出力ピン、他を入力ピンとする場合は 0100 となるため、そのまま 2 進法で 0b0100 としても良いし、16 進法に直して 0x4、10 進法に直して 4 としても良い。アクティブ状態での変更は無効（反映されない）。
SPI_baudrate	int	100000	SPI 通信のクロックレート。単位は Hz。アクティブ状態での変更は無効（反映されない）。
SPI_mode	int	0	SPI 通信のクロックの極性 (polarity) と位相 (phase) を決定する番号。0 の時はいずれも 0、1 の時は極性が 0 で位相が 1、2 の時は極性が 1 で位相が 0、3 の時はいずれも 1 となる。SPI を使用しない場合はこの設定は無視される。アクティブ状態での変更は無効（反映されない）。
SPI_cs_pin	string	D4	SPI 通信で CS (chip select) ピンとして利用するピン (Blinka では CS0(D3) ピンを使用しないため、GPIO の 1 つを CS ピンとして使用する必要がある)。D4～D7の中から 1 つを選択する。CS ピンとして使用されるピンは GPIO の設定 (GPIO_D7_4_IO_select) で出力ピンとする必要がある。また、CS ピンとして指定したピンに対する入力ポートからの値の変更は無視される。SPI を使用しない場合はこの設定は無視される。アクティブ状態での変更は無効（反映されない）。
SPI_cs_talking	int	0	SPI 通信での通信時の CS (chip select) ピンの状態。通信時に Low とする場合は 0、High とする場合には 1 となる。SPI を使用しない場合はこの設定は無視される。アクティブ状態での変更は無効（反映されない）。

名称	型	デフォルト値	説明
I2C_device_address	string	0x08	I ² C デバイスの 7 ビットアドレス。0b や 0x を値の前につけることで 2 進法や 16 進法で指定することも可能。I ² C を使用しない場合はこの設定は無視される。アクティブ状態での変更は無効（反映されない）。
SER_select	string	None	I ² C もしくは SPI 通信を使用するか、使用しないかを選択する。I2C か SPI であればそれぞれの通信を使用し、None であれば使用しない。アクティブ状態での変更は無効（反映されない）。
SER_timeout	int	1000	I ² C もしくは SPI 通信においてバスをロックするときの最大待ち時間。単位はミリ秒。この時間を超えてもバスがロックできない場合はエラーとする。SPI や I ² C を使用しない場合はこの設定は無視される。

- サービスポート
なし
- 依存ライブラリ
pyusb（Windows の場合）, libusb（Mac OSX, Linux の場合）
pyftdi
Adafruit Blinka
- 利用ハードウェア
Adafruit FT232H Breakout ボード

2.2 GPIO の利用例で使用するコンポーネント群

2.2.1 入力データ変化カウントコンポーネント (CountChange)

CountChange は、入力値が 1 つ前の入力値と比べ指定の変化をするたびに出力値が増加するコンポーネントです。どのような変化をカウントするかは Configuration の change で「増加したとき」、「減少したとき」、「変化したとき」（増加でも減少でも出力）、「変化しないとき」のいずれから選択できます。また、出力の変化範囲は Configuration の min, max で、増加幅は変化幅は Configuration の step で変更可能です。例えば、min=0, max=4, step=2, change=increase のとき、入力値の変化が 1 → 4 → 7 → 3 → 5 → 2 → 2 → 5 なら入力値が増加した 1 → 4 のタイミングで 0 が、4 → 7 で 2 が、3 → 5 で 4 が、0 → 5 で（max まで達したので初期値に戻り）0 が出力されます。

- InPort

名称	型	説明
data	TimedLong	変化を検知する対象。

- OutPort

名称	型	説明
count	TimedLong	入力値が変化するたびに出力される値。出力される値の範囲や変化のタイミングについては Configuration で指定できる。

- Configuration 変数

名称	型	デフォルト値	説明
min	int	0	出力される値の最小値。 $\text{min} \leq \text{max}$ でなければならない。出力値が $[\text{min}, \text{max}]$ の範囲を超えた場合、step が正ならこの値が出力される。step が非負のとき、最初に入力が指定の条件を満たしたときに出力される値もこの値になる。
max	int	9	出力される値の最大値。 $\text{min} \leq \text{max}$ でなければならない。出力値が $[\text{min}, \text{max}]$ の範囲を超えた場合、step が負ならこの値が出力される。step が負のとき、最初に入力が指定の条件を満たしたときに出力される値もこの値になる。
step	int	1	指定された入力の変化を検知するたびに出力値が変化する幅。例えば、現在の出力値が 0 で step が 2 なら次に出力される値は 2、その次に出力される値は 4 となる。
change	string	increase	出力ポートから値が出力されるタイミング。increase の場合は入力値が 1 つ前の入力値と比べ増加した場合に、decrease の場合は減少した場合に、change の場合は増加もしくは減少した場合に、nochange の場合は変化しなかった場合に出力ポートから値が出力される。

- サービスポート
なし

2.2.2 Lookup Table コンポーネント (ConfigLUT)

ConfigLUT は、1 次元の Lookup Table (LUT) コンポーネントです。Configuration で LUT の値となる整数のリストをカンマ区切りで指定します。要素番号を入力ポートから指定すると、LUT の対応する値が出力ポートから出力されます。要素番号は 0 から始まり、範囲外の要素の番号が指定された場合は何も行いません。負の値を入力した場合には、-1 であれば最終要素の値、-2 であれば最終要素の 1 つ前の値というように逆順に数えた要素の値を出力します。

- InPort

名称	型	説明
index	TimedLong	値を出力する LUT の要素番号。先頭要素の番号は 0 番とする。負の値を入力した場合は、-1 であれば最終要素の値、-2 であれば最終要素の 1 つ前の値というように、逆順に数えた要素の値を出力する。

- OutPort

名称	型	説明
count	TimedLong	LUT の指定された要素の値。

- Configuration 変数

名称	型	デフォルト値	説明
table	string	0, 1, 2, 3	LUT の値。値はカンマ区切りで指定する（カンマ区切りで値を増減することで要素数を変更可能）。各要素は整数値で指定する。

- サービスポート
なし

2.3 シリアル通信の利用例で使用するコンポーネント群

2.3.1 OctetSeq データ出力コンポーネント (SendOctetSeq)

SendOctetSeq は、コンソールから入力された値を RTC::TimedOctetSeq 型のデータとして出力します。データはカンマ区切りで入力します。カンマで区切られた各データは [0, 255] の整数である必要があります。また、各データは 0b や 0x を値の前につけることで 2 進法や 16 進法で指定することも可能です。

- InPort
なし
- OutPort

名称	型	説明
out	TimedOctetSeq	コンソールから入力された値を RTC::TimedOctetSeq 型のデータとしたもの。

- Configuration 変数
なし
- サービスポート
なし

2.3.2 気圧・温湿度センサ計算コンポーネント (BME280Decode)

BME280Decode は、気圧・温湿度センサ BME280[2] の 8 バイトのバイト列 (press_msb, press_lsb, press_xlsb, temp_msb, temp_lsb, temp_xlsb, hum_msb, hum_lsb) とキャリブレーションデータ (calib00~calib41) から気圧、温度、湿度を計算し、各出力ポートから出力します。キャリブレーションデータは Configuration から入力します。

- InPort

名称	型	説明
SensorByteData	TimedOctetSeq	センサから得られるアドレス 0xF7 から 0xFE の 8 バイトのバイト列。

- OutPort

名称	型	説明
Pressure	TimedDouble	計測した圧力 [hPa]。
Temperature	TimedDouble	計測した温度 [°C]。
Humidity	TimedDouble	計測した湿度 [%]。

- Configuration 変数

名称	型	デフォルト値	説明
calib00_25	string	1	アドレス 0x88 から 0xA1 の 26 バイトのキャリブレーションデータ。バイト列を hex 文字列（1 バイトにつき 2 つの 16 進数を含む文字列）もしくは utf-8 でデコードした文字列で指定する。どちらの文字列なのかによって Configuration 変数 calib_data_type も変更すること。アクティブ状態での変更は無効（反映されない）。
calib26_41	string	1	アドレス 0xE1 から 0xF0 の 16 バイトのキャリブレーションデータのバイト列。使用するのは最初の 7 バイトのため、7 バイト以上のデータがあればよい。バイト列を hex 文字列（1 バイトにつき 2 つの 16 進数を含む文字列）もしくは utf-8 でデコードした文字列で指定する。どちらの文字列なのかによって Configuration 変数 calib_data_type も変更すること。アクティブ状態での変更は無効（反映されない）。
calib_data_type	string	hex	キャリブレーションデータ (calib00_25, calib26_41) を hex 文字列（1 バイトにつき 2 つの 16 進数を含む文字列）として入力したのか utf-8 でデコードした文字列で入力したのかを指定する。

- サービスポート
なし

3 Adafruit_FT232H_Breakout の使用方法

3.1 使用前の準備

Adafruit_FT232H_Breakout の使用にあたり、事前にソフトウェアのインストールなどの準備が必要となります。OS ごとの概要は下記のとおりですが、詳細は公式サイトの手順 [3] を参照してください。

- Windows の場合
 1. FT232H のドライバーを更新する
 2. pyftdi と pyusb をインストールする
 3. Adafruit Blinka をインストールする
- Mac OSX の場合
 1. libusb をインストールする
 2. pyftdi と Adafruit Blinka をインストールする
- Linux の場合
 1. libusb をインストールする
 2. udev rules ファイルを作成する
 3. pyftdi と Adafruit Blinka をインストールする

3.2 RTSystemEditor によるシステム構築の手順

RTSystemEditor を用いたシステム構築は通常、以下の手順で行われます。

RTSystemEditor を用いたシステム構築の手順

1. ネームサーバの起動
2. RTSystemEditor の起動
3. ネームサーバへの接続
4. コンポーネントの起動
5. システムの構築と実行

これらの手順はどのようなシステムでも共通ですので、操作方法は OpenRTM-aist のホームページ (<https://www.openrtm.org/openrtm/>) を参照してください。以下の節では手順 5 に関して、Adafruit_FT232H_Breakout の使用手順を説明します。

3.3 Adafruit_FT232H_Breakout の使用手順

Adafruit_FT232H_Breakout の使用手順をまとめると以下のようになります。

Adafruit_FT232H_Breakout の使用手順

0. (コンポーネント実行前の準備)
 - (a) Adafruit FT232H Breakout ボードが新しいタイプ (コネクタが USB Type-C のもの) の場合、I²C 通信を使うならボードの I2C Mode スイッチをオンに、SPI 通信を使うならオフにする
 - (b) 入出力の対象となる回路を作成し、Adafruit FT232H Breakout ボードの各ピンに接続する
 - (c) Adafruit FT232H Breakout ボードと PC を USB で接続する
1. Configuration を設定する
 - (a) シリアル通信 (I²C もしくは SPI 通信) を利用するか、利用する場合はどちらを利用するのかを SER.select で指定する
 - (b) GPIO を利用する場合は、GPIO_C7_0_IO.select と GPIO_D7_4_IO.select で各ピンをそれぞれディジタル入力ピンとして使用するか、ディジタル出力ピンとして使用するかを指定する
SPI 通信を利用する場合は CS (chip select) ピンを SPI_cs_pin で指定し、GPIO_D7_4_IO.select 設定時にそのピンは出力ピンに指定する
 - (c) I²C 通信を利用する場合は I2C_device_address でデバイスの 7 ビットアドレスを、SPI 通信を利用する場合は SPI_baudrate, SPI_mode, SPI_cs_talking で通信パラメータを設定する
2. コンポーネントをアクティブ化する
3. InPort, OutPort を他のコンポーネントに接続し、システムを拡張していく

Adafruit FT232H Breakout ボードを接続せずにコンポーネントを実行すると以下のようなエラーになります。

```
RuntimeError: BLINKA_FT232H environment variable set, but no FT232H device found
```

Configuration の設定はアクティブ化前に行ってください (アクティブ化後は変更しても反映されないため、設定を変更する場合は一度非アクティブ化する必要があります)。また、手順 3 の他のコンポーネントの接続については、手順 1 や手順 2 の前に行っても構いません。

以下の節では、本コンポーネントの機能である GPIO 機能とシリアル通信機能のそれぞれの使い方について説明します。GPIO 機能とシリアル通信機能の両方を同時に使用することも可能ですが、シリアル通信については I²C 通信か SPI 通信のいずれかを選択して使用することになります。

3.3.1 GPIO の利用

(a) I/O ピンの入出力指定（Configuration 変数 GPIO_C7_0_IO_select, GPIO_D7_4_IO_select）

C7～C0 および D7～D4 のピンをそれぞれデジタル出力として使用するか、デジタル入力として使用するかを Configuration 変数で指定します。

C7～C0 の入出力は GPIO_C7_0_IO_select で指定します。まず、C7 から C0 まで順番に、入力ピンとして用いる場合には 1、出力ピンとして用いる場合には 0 として対応する数字を並べていきます。例えば、C5, C3, C0 を出力ピンとして使い、他は入力ピンとして使う場合、C7 は入力ピンなので 1、C6 も入力ピンなので 1、C5 は出力ピンなので 0 というように並べていくと、11010110 という数ができます。この数を 2 進法で表現されている数と考えて GPIO_C7_0_IO_select に指定します。この例では、2 進法での 11010110 は 10 進法では 214 ですので、GPIO_C7_0_IO_select に 214 を入力します。10 進法に変換せず、2 進法であることを示す 0b（ゼロビー）を頭につけて 0b11010110 と入力することも可能です。また、この値は 16 進法で表すと d6 ですので、16 進法であることを示す 0x（ゼロエックス）を頭につけて 0xd6 と入力することもできます。

例) C5, C3, C0 を出力ピン、C7, C6, C4, C2, C1 を入力ピンとして使う場合

ピン	C7	C6	C5	C4	C3	C2	C1	C0
入出力指定	Input	Input	Output	Input	Output	Input	Input	Output
GPIO_C7_0_IO_select に対応する値	1	1	0	1	0	1	1	0

同様に、D7～D4 の入出力は GPIO_D7_4_IO_select で指定します。まず、D7 から D4 まで順番に、入力ピンとして用いる場合には 1、出力ピンとして用いる場合には 0 として対応する数字を並べていきます。例えば、D6 を出力ピンとして使い、他は入力ピンとして使う場合、D7 は入力ピンなので 1、D6 は出力ピンなので 0、D5 は入力ピンなので 0 というように並べていくと、1011 という数ができます。この数を 2 進法で表現されている数と考えて GPIO_D7_4_IO_select に指定します。この例では、2 進法での 1011 は 10 進法では 11 ですので、GPIO_D7_4_IO_select に 11 を入力します。10 進法に変換せず、2 進法であることを示す 0b を頭につけて 0b1011 と入力することも可能です。また、この値は 16 進法で表すと b ですので、16 進法であることを示す 0x を頭につけて 0xb と入力することもできます。

Configuration の設定が終了したら、RTSystemEditor 上でコンポーネントを右クリックし、Activate を選択してアクティブ化します。指定に誤りがある場合は、“value not integer”（整数値が入力されていない）や“out of range”（GPIO.C7_0.IO_select に 2 進法で 9 桁の値が入力されているなど範囲外の値が入力されている）というメッセージがコンソールに表示され、コンポーネントがエラー状態になります。そのときは値に誤りがないかを確認してください。誤りを修正したら、コンポーネントを右クリックして Reset を選択し、エラー状態を解除してからもう一度アクティブ化を行ってください。

(b) I/O ピンからの出力（入力ポート C7_0out, D7out, D6out, D5out, D4out）

出力ピンから High (H) または Low (L) の電圧を出力するには、コンポーネントの対応する入力ポートに値を送信します。ただし、GPIO.C7_0.IO_select や GPIO.D7_4.IO_select で入力ピンに指定されているピンに対しては入力ポートに値を入力しても何も行われません。

C7～C0 については、C7_0out に値を入力します。まず、C7 から C0 まで順番に、High を出力する場合には 1、Low を出力する場合は 0 として対応する数字を並べていきます。例えば、C5, C3 から High を出力し、他は Low を出力する（もしくは入力ピンとしている）場合、C7 は Low なので 0、C6 も Low なので 0、C5 は High なので 1 というように並べていくと、00101000 という数ができます。（GPIO.C7_0.IO_select で入力ピンとして指定するピンに対しては 1, 0 のどちらにしても問題ありませんが、この例では 0 としています。）この数を 2 進法で表現されている数と考え、整数値を C7_0in に入力します。この例では、2 進法での 00101000 は 10 進法では 40 ですので、C7_0in に 40 を入力します。

例) C5, C3 から High (H) を出力、C7, C6, C4, C2, C1, C0 から Low (L) を出力する（もしくは入力ピンとしている）場合

ピン	C7	C6	C5	C4	C3	C2	C1	C0
出力電圧	L	L	H	L	H	L	L	L
C7_0out への入力に対応する値	0	0	1	0	1	0	0	0

D7～D4 については、それぞれのピンに対して別々の入力ポートが用意されています。例えば、D7 ピンからの出力を指定する場合は D7out を使用します。各ポートに 0 が入力されると対応するピンから Low が出力され、0 以外の値が入力されると High が出力されます。

(c) I/O ピンからの入力（出力ポート C7_0in, D7in, D6in, D5in, D4in）

入力ピンに High (H) または Low (L) のどちらの電圧が入力されているのかを知るには、コンポーネントの対応する出力ポートから値を受信します。ただし、GPIO.C7_0.IO_select で全てのピンが出力ピンに指定されている場合や、GPIO.D7_4.IO_select で出力ピンに指定されているピンについては、対応する出力ポートからは値が出力されません。

C7～C0 については、C7_0in から値が出力されます。まず、C7 から C0 まで順番に、High が入力されている場合には 1、Low を出力する場合もしくは出力ピンに指定されている場合は 0 として対応する数字を並べていきます。例えば、C7, C4, C2 に High が入力され、他は Low が入力されているか出力ピンとしている場合、C7 は High なので 1、C6 は Low または出力ピンなので 0、C5 も Low または出力ピンなので 0 というように並べていくと、10010100 という数ができます。この数を 2 進法で表現されている数と考え、整数値を C7_0out から出力します。この例では、2 進法での 10010100 は 10 進法では 148 ですので、C7_0out から 148 が出力されます。

例) C7, C4, C2 に High (H) が入力され、C6, C5, C3, C1, C0 から Low (L) を出力する（もしくは出力ピンとしている）場合

ピン	C7	C6	C5	C4	C3	C2	C1	C0
入力電圧	H	L	L	H	L	H	L	L
C7.0in からの出力に対応する値	1	0	0	1	0	1	0	0

D7～D4 については、それぞれのピンに対して別々の出力ポートが用意されています。例えば、D7 ピンへの入力を知りたい場合は D7in を使用します。各ピンに Low が入力されていると対応するポートから 0 が出力され、High が入力されていると 1 が出力されます。

3.3.2 シリアル通信の利用

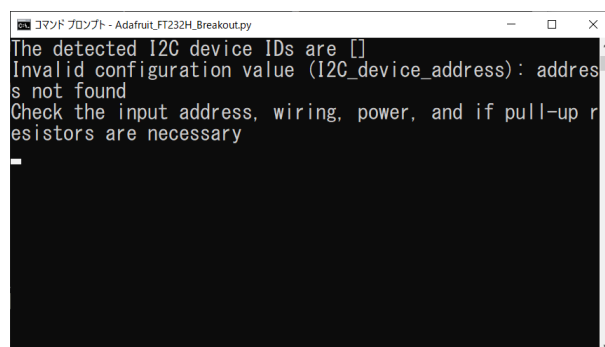
(a) シリアル通信の利用設定 (Configuration 変数 `SER_select`, `I2C_device_address`, `SPI_baudrate`, `SPI_mode`, `SPI_cs_pin`, `SPI_cs_talking`, `GPIO_D7_4_IO_select`)

Configuration 変数でシリアル通信の利用設定を行います。まず、I²C 通信を使うか SPI 通信を使用するかを `SER_select` で指定します。I²C 通信を使う場合は `I2C`、SPI 通信を使用する場合は `SPI` を選択します。

I²C 通信を使う場合は、`I2C_device_address` で通信する I²C デバイスの 7 ビットアドレスを設定します。設定するアドレスは通信するデバイスのデータシートを参照してください。指定する値は例えば 0x76 のように先頭に 0x をつけて 16 進法で入力しても、0b1110110 のように先頭に 0b をつけて 2 進法で入力しても、118 のように 10 進法で入力しても構いません。

SPI 通信を利用する場合は、D7～D4 の中から 1 つを CS (chip select) ピンとして回路を構成し、`SPI_cs_pin` で指定します。また、このピンは `GPIO_D7_4_IO_select` で出力ポートに指定 (3.3.1 節を参照) してください。さらに、`SPI_baudrate`, `SPI_mode`, `SPI_cs_talking` で通信パラメータを設定します。`SPI_baudrate` は SPI 通信のクロックレート、`SPI_mode` はクロックの極性 (polarity) と位相 (phase) を決定する数値、`SPI_cs_talking` は CS ピンの状態が通信時に H となる場合は 1、L となる場合は 0 とします。設定値は通信するデバイスのデータシートを参照してください。

Configuration の設定が終了したら、RTSystemEditor 上でコンポーネントを右クリックし、Activate を選択してアクティブ化します。指定に誤りがある場合は、“value not integer” (整数値が入力されていない) や “out of range” (`GPIO_C7_0_IO_select` に 2 進法で 9 桁の値が入力されているなど範囲外の値が入力されている) というメッセージがコンソールに表示され、コンポーネントがエラー状態になります。そのときは値に誤りがないかを確認してください。I²C 通信を使う場合、`I2C_device_address` で指定したアドレスのデバイスが見つからない場合には address not found と表示されます。この場合、特に検出されたデバイスの一覧 (The detected I2C device IDs are [] の [] 内) に目的のアドレスが表示されない場合は Configuration 変数の値だけでなく、I²C デバイスとボードとの接続がデバイスが正しく行われているか、コネクタが USB Type-C のボードの場合はボードの I²C Mode スイッチが on になっているかを確認してください。



```

コマンド プロンプト - Adafruit_FT232H_Breakout.py
The detected I2C device IDs are []
Invalid configuration value (I2C_device_address): address
s not found
Check the input address, wiring, power, and if pull-up r
esistors are necessary

```

誤りを修正したら、コンポーネントを右クリックして Reset を選択し、エラー状態を解除してからもう一度アクティブ化を行ってください。

(b) デバイスとの通信（入力ポート `I2C_SPIwcommand_rbytes`, 出力ポート `I2C_SPIread`）

デバイスとの通信には入出力ポートを用います。送信するデータは入力ポート `I2C_SPIwcommand_rbytes` に `TimedOctetSeq` データとして入力します。`OctetSeq` データの最終要素の前までにデバイスに送信するバイト列を設定し、最終要素にはデバイスから受信するバイト数を設定します。データの送信のみを行う場合は最終要素の値を 0 にし、データの受信のみを行う場合は 1 要素のみとして受信バイト数のみを指定します。

要素番号	0	1	2	...	N-1	N
データ	送信バイト列（N バイト）					受信バイト数

例を以下に示します。

- 12, 25 という 2 バイトのデータを送信し、その後 10 バイトのデータを受信する

12, 25, 10

- 100, 50, 30 という 3 バイトのデータを送信する（送信のみで受信は行わない）

100, 50, 30, 0

- 7 バイトのデータを受信する（受信のみで送信は行わない）

7

データの受信を行う場合、受信したデータは出力ポートの `I2C_SPIread` から `TimedOctetSeq` 型のデータとして出力されます。

4 Adafruit_FT232H_Breakout の利用例

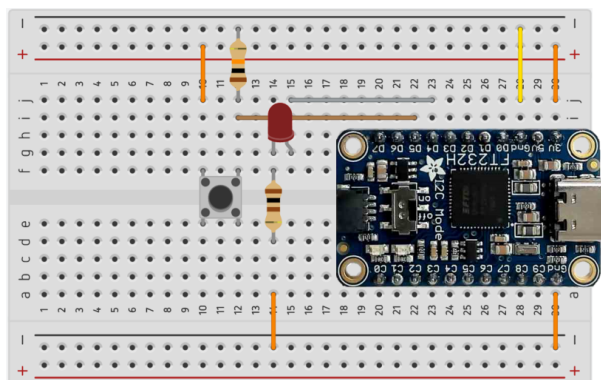
ここでは、提供しているコンポーネント群を用いた `Adafruit_FT232H_Breakout` の機能の利用例を紹介します。

4.1 GPIO の利用例

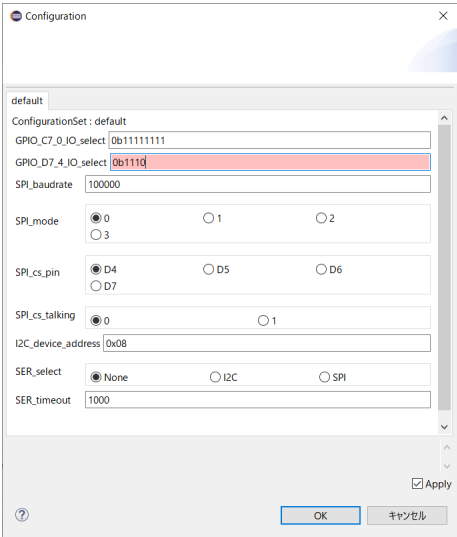
GPIO の利用例として、LED の点消灯の切り替えやスイッチの状態の取得を行います。

4.1.1 ポート D を用いたスイッチ情報の取得と LED の点消灯

まずは回路を準備します。図のように D4 に LED と保護抵抗を直列に接続し GND へ、D5 にタクトスイッチとプルダウン抵抗を並列に接続し、スイッチのもう一方は 3V に接続します。（抵抗の値は使用する LED に合わせて適宜変更を行って下さい。）これで、D4 から High を出力したときに LED が点灯、Low を出力したときに消灯します。また、スイッチを押しているときに D5 に High が入力され、押していないときに Low が入力されます。

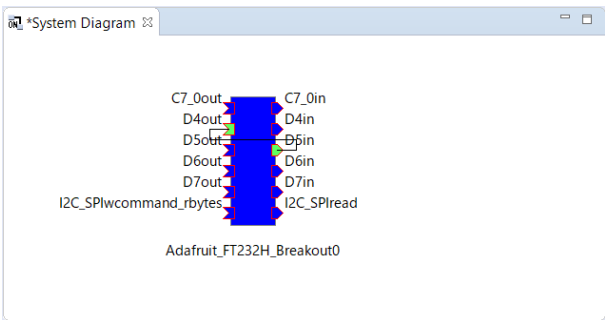


次にコンポーネントを準備します。Adafruit FT232H Breakout ボードがPC に接続されている状態でコンポーネントを起動します。D4 はLED に点消灯のための電圧を出力するため出力ポート、D5 はスイッチが ON か OFF かを知るための電圧を受け取るため入力ポートとなります。そのため、システムエディタで Adafruit_FT232H_Breakout の Configuration 変数 GPIO_D7_4_IO_select の値を 0b1110 とします。(最下位ビットが D4 の設定のため、出力に対応する 0 に、その 1 つ上のビットが D5 の設定のため、入力に対応する 1 にしています。その他は D6, D7 の設定ですので入出力のどちらに設定しても問題ありませんが、ここではどちらも入力としています。)



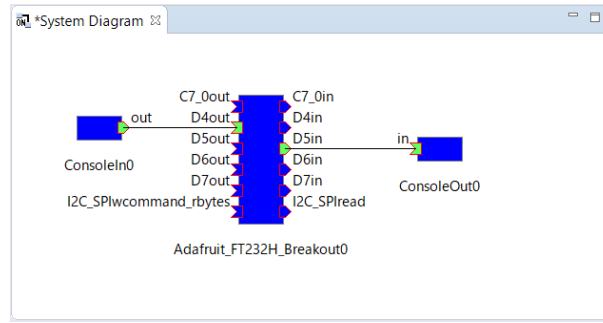
続いて、システムエディタ上で Adafruit_FT232H_Breakout の D4out と D5in を接続します。これで準備は完了です。

Adafruit_FT232H_Breakout	
D5in (OutPort)	– D4out (InPort)



コンポーネントアクティブ化すると、スイッチを押している間だけ LED が点灯することが確認できます。LED 部分とスイッチ部分の個々の動作は OpenRTM-aist に例として用意されている ConsoleIn コンポーネントと ConsoleOut コンポーネントを使って確認することができます。システムエディタ上で D4out と D5in を接続している線をクリックし、Del キーを押して接続を解除します。ConsoleIn, ConsoleOut を起動し、Adafruit_FT232H_Breakout とポートを以下の通りに接続します。

ConsoleIn	Adafruit_FT232H_Breakout	Adafruit_FT232H_Breakout	ConsoleOut
out (OutPort)	–	D5in (OutPort)	– in (InPort)
	D4out (InPort)		

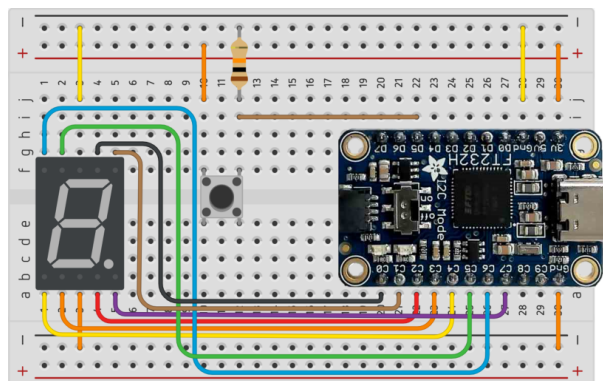
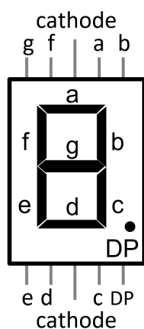


コンポーネントをアクティブ化し、ConsoleIn のコンソールから 1 を入力すると D4 から High が出力されるため LED を点灯させ、0 を入力すると D4 から Low が出力されるため LED を消灯することができます。また、ConsoleOut のコンソールを確認すると、スイッチを押していないときは D5 に Low が入力されているため 0、スイッチを押しているときは D5 に High が入力されているため 1 が表示されます。

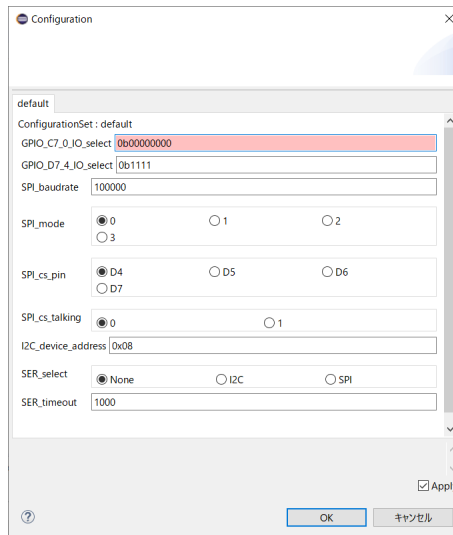
4.1.2 ポート C を用いた LED アレイの点消灯

C0～7 についてはまとめて 1 つの入出力ポートに対応しているため、7 セグメントディスプレイやバー表示 LED のように複数の LED の状態をまとめて変化させることも可能です。ここでは、スイッチを押した回数に合わせ 7 セグメント LED の表示パターンを切り替える例を示します。

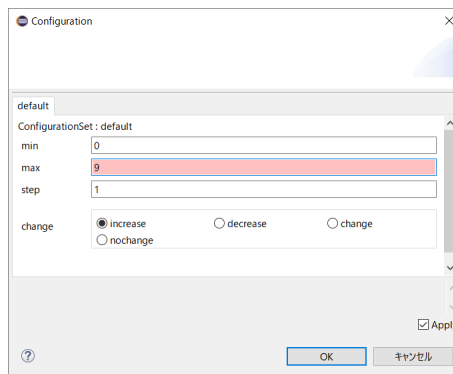
まず、回路を準備します。前節と同様に D5 にタクトスイッチとプルダウン抵抗を並列に接続し、スイッチのもう一方は 3V に接続します。これで、スイッチを押しているときに D5 に High が入力され、押していないときに Low が入力されます。また、C0 から C7 に 7 セグメント LED の各セグメントを接続します。ここでは、図のような共通カソード（カソードコモン）の 7 セグメント LED を使用（例えば [4]）し、C0 にセグメント a（右上の |）、C1 にセグメント b（右下の |）、…、C6 にセグメント g（中央の -、C7 にセグメント DP（小数点）が抵抗を介して接続されているものとしします。（セグメントと端子の対応や抵抗の値は使用する素子に合わせて適宜変更を行って下さい。）これで、C0 から High を出力したときにセグメント a が点灯、Low を出力したときにセグメント a が消灯というように、各ピンから High を出力したときに対応するセグメントが点灯します。



次にコンポーネントを準備します。Adafruit FT232H Breakout ボードが PC に接続されている状態でコンポーネントを起動します。C0 から C7 は 7 セグメントディスプレイの各セグメントが接続されているため出力ポート、D5 はスイッチが ON か OFF かを知るための電圧を受け取るため入力ポートとなります。そのため、システムエディタで Adafruit_FT232H.Breakout の Configuration 変数 GPIO_C7_0.IO_select の値を 0b00000000(0x00 や単に 0 としても同じです) に、GPIO_D7_4.IO_select の値を 0b1111 とします。（C0 から C7 は全て出力のため出力に対応する 0、最下位ビットの 1 つ上のビットが D5 の設定のため、入力に対応する 1 にしています。その他のビットは D4, D6, D7 の設定ですので入出力のどちらに設定しても問題ありませんが、ここではどちらも入力としています。）



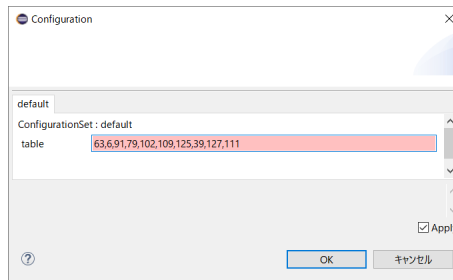
続いて **CountChange** を起動し、**CountChange** の **Configuration** 変数にスイッチを押した回数のおえ方を設定します。今回利用するのは 1 桁の 7 セグメントディスプレイですので、0 から 9 の値をループするようにします。カウントの最小値 **min** を 0、最大値 **max** を 9、刻み幅 **step** を 1 とします。また、スイッチを離している間 0 が、押している間 1 が送られてくるため、**change** を **increase** とします。これでスイッチが押されるたびに発生する 0 → 1 の変化が検出されカウントが 1 増えるようになります。（逆に **change** を **decrease** とすればスイッチを離したタイミングでカウントが 1 増えるようになります。）



最後に、**CounfigLUT** を起動し、**CountChange** の **Configuration** 変数に 7 セグメントディスプレイの点灯パターンを設定します。例えば、0 を点灯するにはセグメント a, b, c, d, e, f の 6 つを点灯する必要があります。つまり、この例の接続方法の場合は C0~C5 が High に、C6,7 が Low になるような値を出力すればよいことになります。（Adafruit FT232H Breakout ボードの各ピンと 7 セグメントディスプレイの各セグメントの対応が例と異なる場合には適宜読み替えてください。）3.3.1 節で説明した通り、これは 2 進法で 00111111、つまり 10 進法では 63 という値になります。同様に 1 を表示するときに出る値は 6、2 を表示するときに出る値は 91、... というように求めた値をカンマ区切りで **table** に入力します。

63,6,91,79,102,109,125,39,127,111

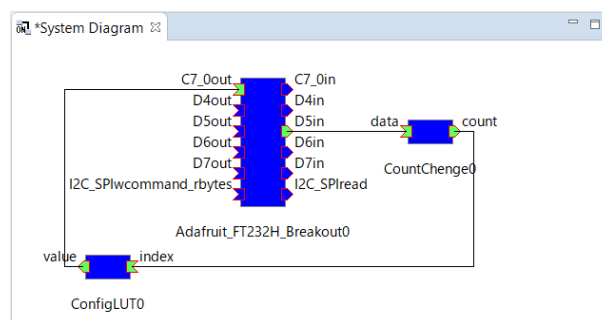
これでこのコンポーネントに 0 が入力されたときには 63、1 が入力されたときには 6、... というように、カンマで区切られた値のうち「入力された値番目」の値が出力されるようになります。



続いて、システムエディタ上で Adafruit_FT232H_Breakout, ConfigLUT, CountChange を以下の通り接続します。
これで準備は完了です。

ConfigLUT		Adafruit_FT232H_Breakout	Adafruit_FT232H_Breakout	CountChange
value (OutPort)	–	C7_0out (InPort)	D5in (OutPort)	– data (InPort)

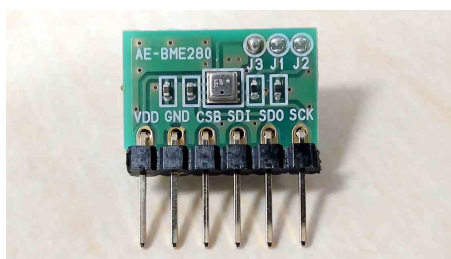
CountChange	ConfigLUT
count (OutPort)	– index (InPort)



コンポーネントをアクティブ化すると、スイッチを押すたびに 7 セグメント LED の表示が $0 \rightarrow 1 \rightarrow 2 \rightarrow \dots \rightarrow 9 \rightarrow 0 \rightarrow \dots$ と順に変化します。

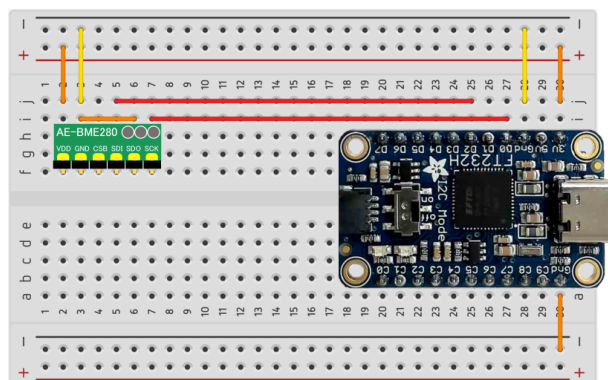
4.2 シリアル通信の利用例

シリアル通信の利用例として、BME280 を使用した温湿度・気圧センサモジュールキット [5] と I²C 通信を行い、温度、湿度、気圧の情報を取得します。

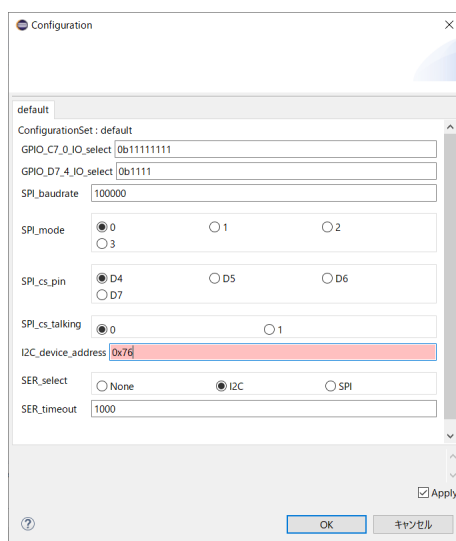


まずは回路を準備します。温湿度・気圧センサモジュールキットの取扱説明書を参照し、J1, J2, J3 をはんだでジャンパします (J1, J2 はジャンパせず、SDI と SDO をそれぞれ 4.7kΩ の抵抗でプルアップしても構いません)。

さらに、Adafruit FT232H Breakout ボードのピン配置 [6] を参照し、センサモジュールの VDD を Breakout ボードの 3V に、GND を GND に、SDO を GND に、SCK を D0 に SDI を D1 または D2 に接続します。また、Adafruit FT232H Breakout ボードが新しいタイプ（コネクタが USB Type-C のもの）の場合はボードの I2C Mode スイッチをオンに、古いタイプの場合は D1 と D2 を短絡します。

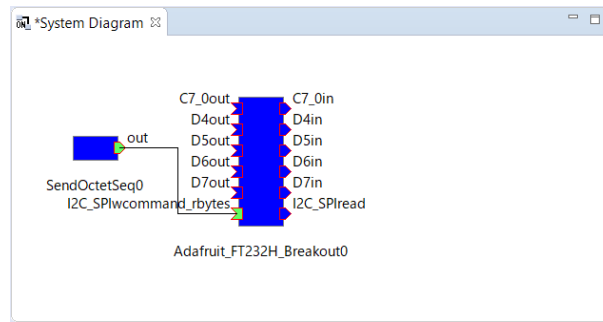


次にコンポーネントを準備します。Adafruit FT232H Breakout ボードが PC に接続されている状態でコンポーネントを起動します。I²C 通信を利用するため、システムエディタで Adafruit_FT232H.Breakout の Configuration 変数 SER_select の値を I2C とします。また、I2C_device_address の値を BME280 の I²C アドレスである 0x76 とします。



計測を行う前に、センサデータから温度、湿度、気圧に変換するためのキャリブレーションデータを準備します。システムエディタ上で Adafruit_FT232H.Breakout, SendOctetSeq を以下の通り接続します。

SendOctetSeq	Adafruit_FT232H.Breakout
out (OutPort)	– I2C_SPIwcommand_rbytes (InPort)



コンポーネントアクティブ化します。BME280 ではキャリブレーションデータはアドレス 0x88～0xA1 と 0xE1～0xF0 に保存されており、このうち計算に必要なのが 0x88～0xA1 の 26 バイト (calib00～calib25) と 0xE1～0xE7 の 7 バイト (calib26～calib32) です。calib00～calib25 に対してはアドレス 0x88 から 26 バイトのデータを読み込むため、SendOctetSeq のコンソールから以下のように入力します。

0x88, 26

Adafruit_FT232H_Breakout のコンソールに表示される “Serial data =” の後の文字列をコピーしておきます。

コマンド プロンプト - SendOctetSeq.py

```

input data: 0x88, 26
input data=[136, 26]
out. data=b'\x88\x1a'
input data:

```

コマンド プロンプト - Adafruit_FT232H_Breakout.py

```

Serial data = b'4o\x03g2\x00\xcc\x8c\x1b\xd6\xd0\x0b\xb1
\x14!\x00\x09\xff\x0c0 \xd1\x88\x13\x00K'

```

同様に、calib26～calib32 に対してはアドレス 0xE1 から 7 バイトのデータを読み込むため、SendOctetSeq のコンソールから以下のように入力します。

0xE1, 7

Adafruit_FT232H_Breakout のコンソールに表示される “Serial data =” の後の文字列をコピーしておきます。

コマンド プロンプト - SendOctetSeq.py

```

input data: 0x88, 26
input data=[136, 26]
out. data=b'\x88\x1a'
input data: 0xE1, 7
input data=[225, 7]
out. data=b'\xe1\x07'
input data:

```

コマンド プロンプト - Adafruit_FT232H_Breakout.py

```

Serial data = b'4o\x03g2\x00\xcc\x8c\x1b\xd6\xd0\x0b\xb1
\x14!\x00\x09\xff\x0c0 \xd1\x88\x13\x00K'
Serial data = b'L\x01\x00\x18.\x03\x1e'

```

次に、得られたデータを hex 文字列に変換します。Python を起動し、bytearray.hex() 関数を用いて calib00～calib25, calib26～calib32 それぞれを変換します。例えば、calib26～calib32 が b'L\x01\x00\x18.\x03\x1e' の場合は以下のようにします。

```

>>> data = b'L\x01\x00\x18.\x03\x1e'
>>> print(data.hex())
4c0100182e031e

```

```

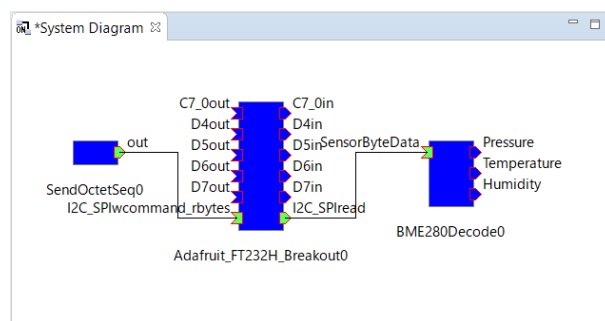
Python 3.7 (64-bit)
Type "help", "copyright", "credits" or "license()" for more
>>> data = b'4o\x03g2\x00\xcc\x8c\x1b\xd6\xd0\x0b\xb1\x14l\x00\xf9\xff\x0c0 \xd1\x88\x13\x00K'
>>> print(data.hex())
346f03673200cc8c1bd6d00bb1146c00f9ff0c3020d18813004b
>>> data = b'L\x01\x00\x18.\x03\xe'
>>> print(data.hex())
4c0100182e031e
>>> _

```

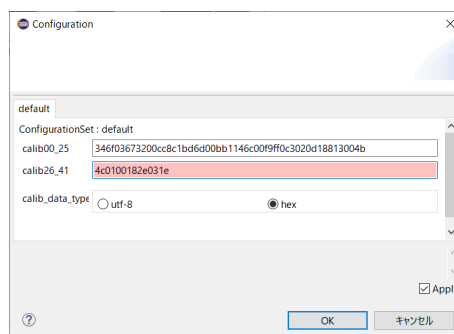
この例の場合、最後の 4c0100182e031e が hex 文字列に変換されたキャリブレーションデータです。キャリブレーションデータはデバイスごとに固定のため、この値を記録しておけば次に計測を行う際はキャリブレーションデータの取得は省略することができます。

このキャリブレーションデータを用いて計測を行います。システムエディタ上で Adafruit_FT232H_Breakout, SendOctetSeq に加えて、BME280Decode のポートを以下の通りに接続します。

Adafruit_FT232H_Breakout	BME280Decode
I2C_SPIread (OutPort)	– SensorByteData (InPort)



BME280Decode の Configuration 変数 calib_data_type を hex とし、calib00_25 と calib26_41 にそれぞれ hex 文字列にしたキャリブレーションデータを入力します。入力したら BME280Decode もアクティブ化します。



SendOctetSeq のコンソールから測定に関するコマンドを入力していきます。ここでは、単発測定を行う手順をします。

- 0xF5, 0x00, 0 (動作設定：フィルタなし)
- 0xF2, 0x01, 0 (湿度測定条件設定：湿度オーバーサンプリング x1)
- 0xF4, 0x25, 0 (測定条件設定：温度・気圧オーバーサンプリング x1、単発測定実行（実行後スリープ）)
- 0xF7, 8 (データ取得：8 バイトのデータ要求)

```
コマンド プロンプト - SendOctetSeq.py
input data: 0xF5, 0x00, 0
input data=[245, 0, 0]
out.data=b' \xf5\x00\x00'
input data: 0xF2, 0x01, 0
input data=[242, 1, 0]
out.data=b' \xf2\x01\x00'
input data: 0xF4, 0x25, 0
input data=[244, 37, 0]
out.data=b' \xf4\x25\x00'
input data: 0xF7, 8
input data=[247, 8]
out.data=b' \xf7\x08'
input data: _
```

最初のコマンドで動作を設定、2つ目と3つ目のコマンドで測定条件の設定と測定、4つ目のコマンドで測定したデータを取得しています。最初の3つはセンサの設定を指定しているだけで受信は行わないので、最後の要素は0（受信バイト数は0）としています。最後のコマンドで BME280Decode の出力ポートから気圧 [hPa]、気温 [°C]、湿度 [%] の値が出力され、コンソールにもその値が表示されます。

```
コマンド プロンプト - BME280Decode.py
Pressure: 1004.57234375 hPa, Temperature: 24.34 deg C,
Humidity: 40.2373046875 %
```

もう一度測定を行う場合は最後の2つのコマンドを再び送信します。

```
0xF4, 0x25, 0
0xF7, 8
```

5 お問い合わせ

本コンポーネントにつきましては、まだ改善の余地があるものと考えております。ご要望、バグ報告、マニュアルの記述の不備等に関しましては、芝浦工業大学デザイン工学部デザイン工学科の佐々木までご連絡ください。

【問合せ先】

〒135-8548

東京都江東区豊洲 3-7-5

芝浦工業大学 本部棟 6 階 06K10 佐々木研究室

Tel:03-5859-8834

sasaki-t at ieec. org

参考文献

- [1] Adafruit, “Overview | Adafruit FT232H Breakout | Adafruit Learning System,” <https://learn.adafruit.com/adafruit-ft232h-breakout>
- [2] Bosch Sensortec, “Humidity Sensor BME280 | Bosch Sensortec,” <https://www.bosch-sensortec.com/products/environmental-sensors/humidity-sensors-bme280/>

- [3] Adafruit, “Setup | CircuitPython Libraries on any Computer with FT232H | Adafruit Learning System,” <https://learn.adafruit.com/circuitpython-on-any-computer-with-ft232h/setup>
- [4] 秋月電子通商, “赤色 7 セグメント L E D 表示器 1 文字カソードコモン ボディ黒 C-551SRD: LED(発光ダイオード) 秋月電子通商-電子部品・ネット通販”, <https://akizukidenshi.com/catalog/g/gI-00640/>
- [5] 秋月電子通商, “BME 280 使用 温湿度・気圧センサモジュールキット: センサー一般 秋月電子通商-電子部品・ネット通販”, <https://akizukidenshi.com/catalog/g/gK-09421/>
- [6] Adafruit, “Pinout | CircuitPython Libraries on any Computer with FT232H | Adafruit Learning System,” <https://learn.adafruit.com/circuitpython-on-any-computer-with-ft232h/pinouts>