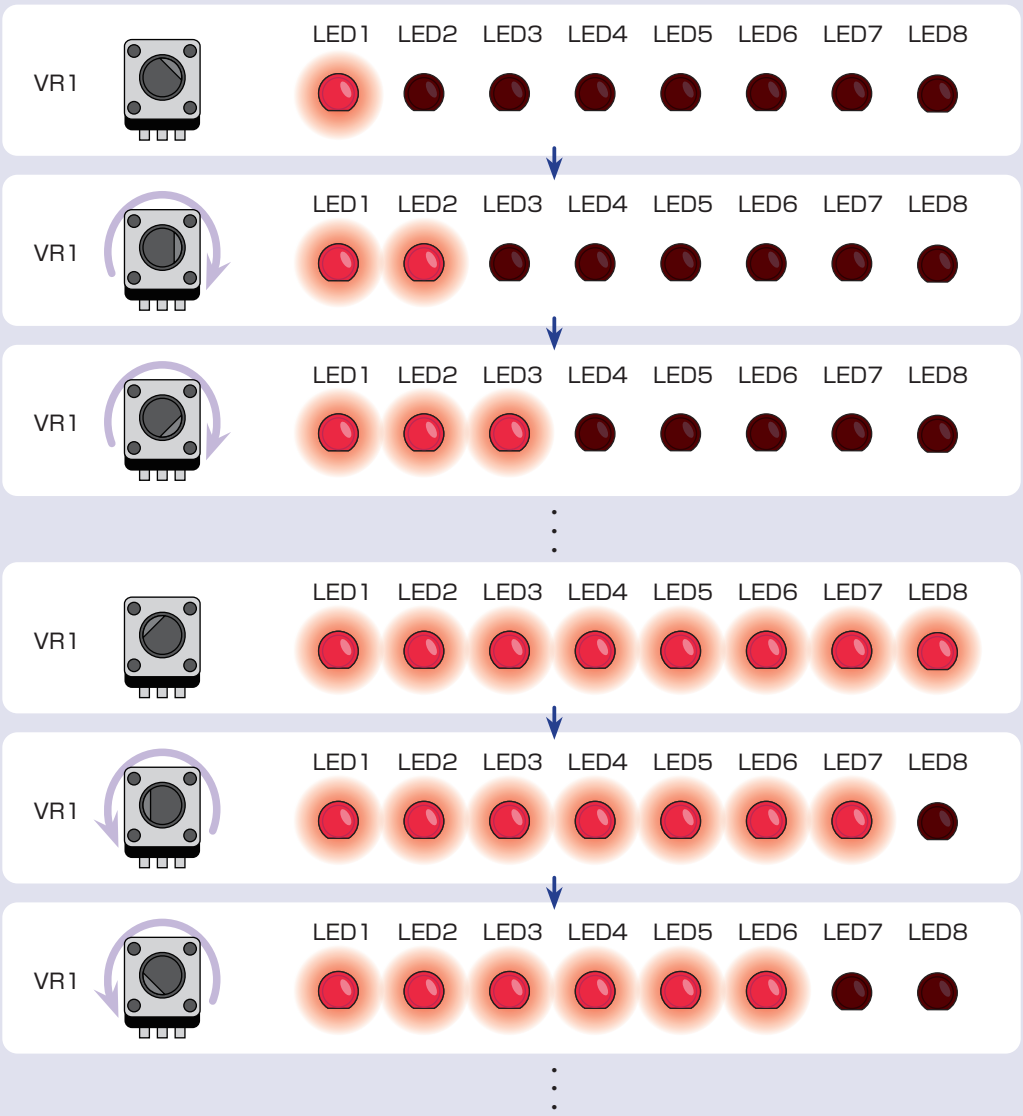
A/D 変換

学習内容

マイコンでセンサなどからのアナログ信号入力を処理するには，ディジタル信号に変換する必要があります。この変換を A/D 変換（アナログ／ディジタル変換）といいます。ARM の A/D 変換を使ってみましょう。

課題 09-1

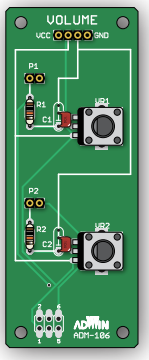
ボリュームの回転角度によって，LED の点灯パターンを変化させる



A/D 変換

配線 09-1

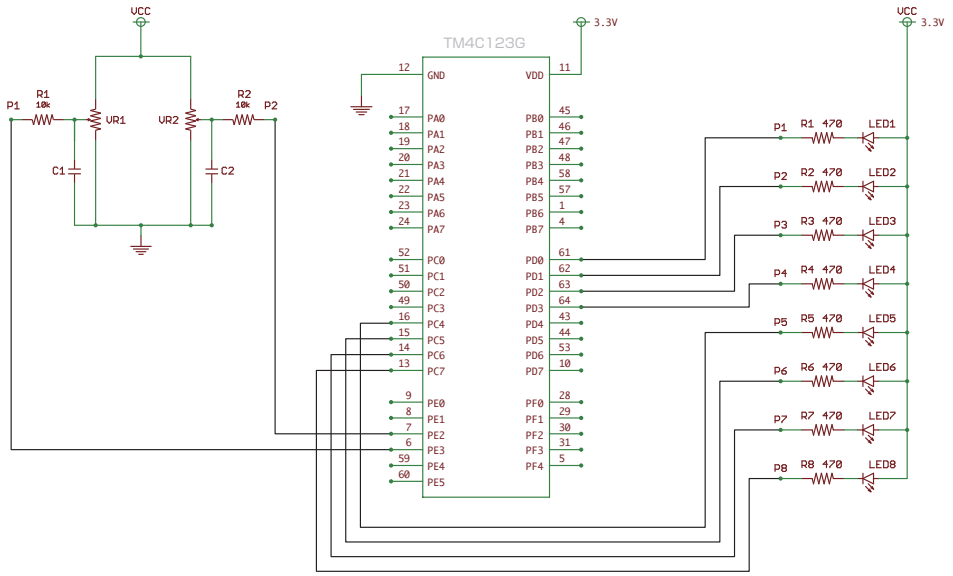
今回はマイコンボードとボリュームボード，LED ボードを使用します。これらのボードはベースボード上で以下のように接続されています。



ボリュームボード		マイコンボード		LED ボード
P1 - VR1	⟷	3.09	PE3（AIN0）	
P2 - VR2	⟷	3.08	PE2（AIN1）	
		3.03	PD0	⟷ P1 - LED1
		3.04	PD1	⟷ P2 - LED2
		3.05	PD2	⟷ P3 - LED3
		3.06	PD3	⟷ P4 - LED4
		4.04	PC4	⟷ P5 - LED5
		4.05	PC5	⟷ P6 - LED6
		4.06	PC6	⟷ P7 - LED7
		4.07	PC7	⟷ P8 - LED8
VCC	⟷	1.01	+3.3V	⟷ VCC
GND	⟷	2.01	GND	

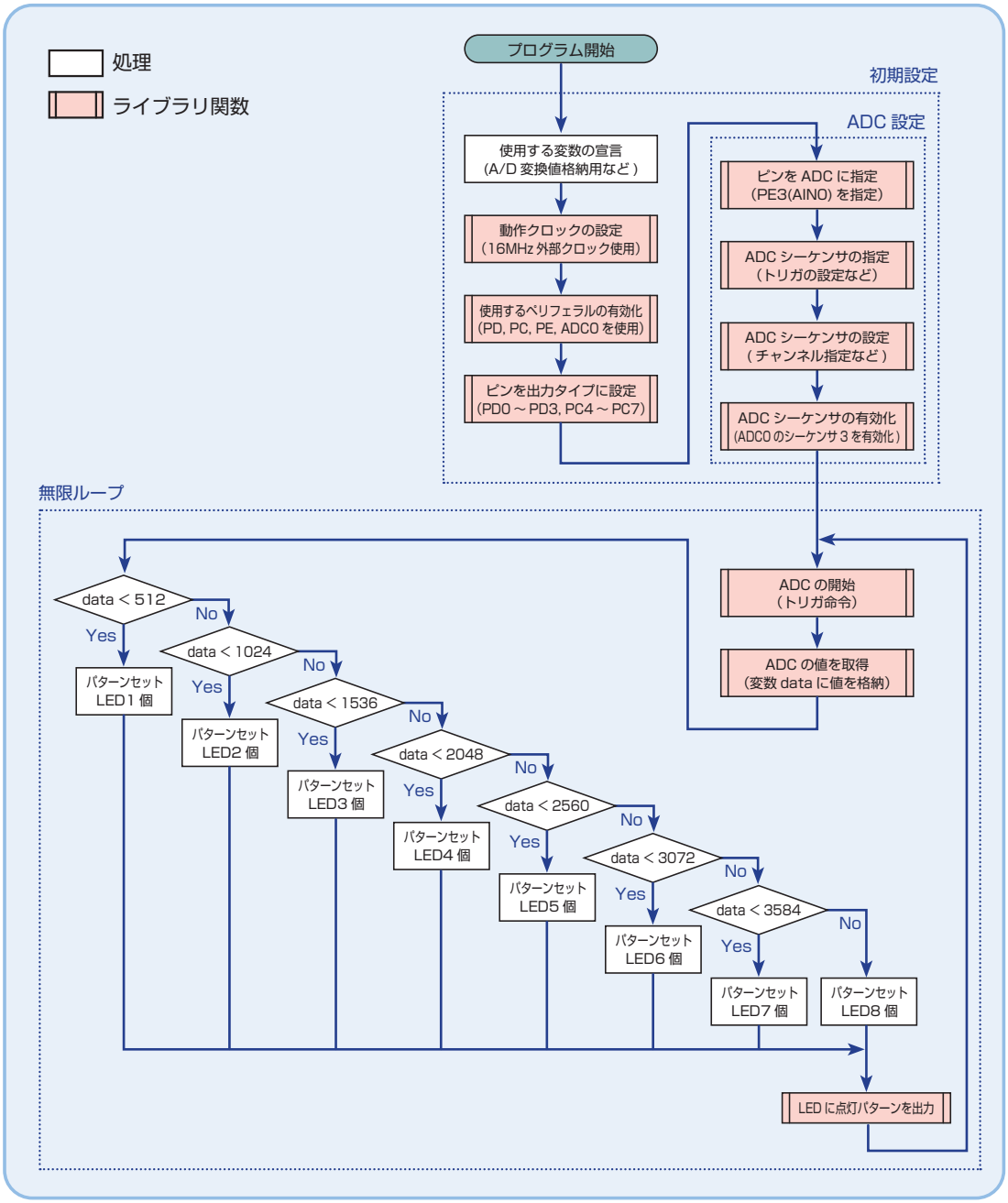
マイコンボードと各ボードをベースボードから取り外し，ジャンパワイヤ（別売）で直接接続することも可能です。時間に余裕のある場合は，STEP 01 のピンアサインに注意しつつ，お好みのピンで課題を実現してみましょう。GND はマイコンボードの 2.01 ピン，3.02 ピンのどちらでもかまいません。

回路図



フローチャート　09-1

フローチャートは以下のようになります。



A/D 変換

TM4C123GH6PM の A/D 変換について

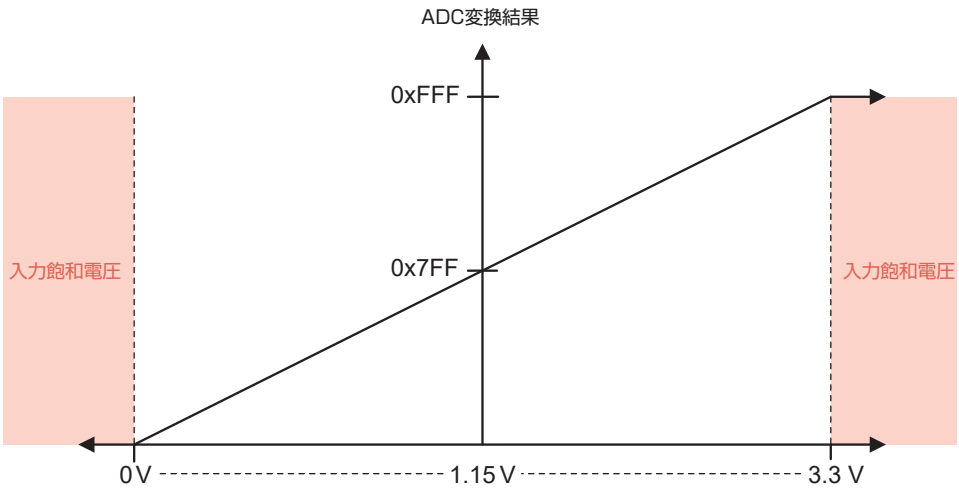
TM4C123GH6PM には 12 ビットの A/D 変換機能があります。この A/D 変換機能は 0V ～ 3.3V までの電圧を測定できます。（リファレンス電圧 VREFN を 0V，VREFP を 3.3V とする標準的な使い方の場合）
12 ビットというのは，0V ～ 3.3V までの電圧を 12 ビットで表現するということです。
12 ビットとは

$$2^{12} = 4096$$

なので，0V ～ 3.3V を 0 ～ 4095 の 4096 個の数値で表現することになります。例えば，

0V ⇒ 000000000000 = 0
1.15V ⇒ 011111111111 = 2047
3.3V ⇒ 111111111111 = 4095

というようになります。

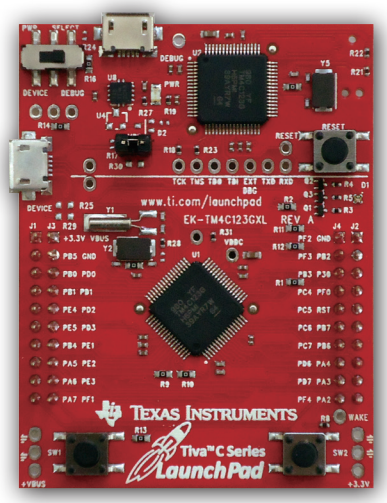


しかし，0V ～ 3.3V まで変化する電圧すべてを数値として表現できるわけではありません。0V ～ 3.3V を 4096 個の数値で表現するため，

$$3.3V \div 4095 \fallingdotseq 0.806 \times 10^{-3} = 0.806mV/bit$$

0.806mV 未満の電圧の変化を測定できないのです。つまり，これが測定誤差となります。取得できた数値には必ず誤差が含まれていることを知っておきましょう。

A/D 変換



ピン番号	GPIO	アナログ	ボード上の機能等
1.02	PB5	AIN11	—
1.05	PE4	AIN9	—
1.06	PE5	AIN8	—
1.07	PB4	AIN10	—
2.03	PE0	AIN3	—
2.06	PB7	—	3.04 に接続
	PD1	AIN6	
2.07	PB6	—	3.03 に接続
	PD0	AIN7	
3.03	PD0	AIN7	2.07 に接続
	PB6	—	
3.04	PD1	AIN6	2.06 に接続
	PB7	—	
3.05	PD2	AIN5	—
3.06	PD3	AIN4	—
3.07	PE1	AIN2	—
3.08	PE2	AIN1	—
3.09	PE3	AIN0	—

TM4C123GH6PM は２つの A/D 変換モジュール ADC0，ADC1 を持っており，12 個のアナログ信号入力チャンネル（AIN0 ～ AIN11）を共有しています。例題では，AIN0 にエレモのボリュームを接続しています。

TM4C123GH6PM の A/D 変換は 1 秒間に最大 1,000,000 回の測定ができます。つまり，最速で 1 マイクロ秒に 1 回測定できることになります。

A/D 変換

インクルードファイル 09-1

STEP 09-1 で使用するインクルードファイルを解説します。

Tiva ヘッダファイル driverlib/adc.h

```
#include "driverlib/adc.h"
```

Tiva のヘッダファイルで，ADC のためのマクロおよび関数が宣言されています。次に示すライブラリ関数の設定例では，以下のマクロおよび関数がここで宣言されています。

- ADC_TRIGGER_PROCESSOR
- ADC_CTL_CH0
- ADC_CTL_IE
- ADC_CTL_END
- ADCSequenceConfigure(ui32Base, ui32SequenceNum, ui32Trigger, ui32Priority)
- ADCSequenceStepConfigure(ui32Base, ui32SequenceNum, ui32Step, ui32Config)
- ADCSequenceEnable(ui32Base, ui32SequenceNum)
- ADCProcessorTrigger(ui32Base, ui32SequenceNum)
- ADCSequenceDataGet(ui32Base, ui32SequenceNum, *pui32Buffer)

ライブラリ関数　09-1

STEP 09 で使用するライブラリ関数を解説します。
関数の引数は ui32 ～が「符号無し 32 ビット整数」, *pui32 ～が「符号無し 32 ビット整数のポインタ」です。
ライブラリ関数は TivaWare で提供されています。

使用するペリフェラルの有効化　SysCtkPeripheralEnable(ui32Peripheral)

使用したいペリフェラル（ここでは A/D 変換）を有効化する。

```
SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
```

設定項目 例

- SYSCTL_PERIPH_ADC0　：　例は ADC(Analog-to-Digital Converter) 0 を使用するときの設定値。

トリガ発生の条件と優先順位の設定
ADCSequenceConfigure(ui32Base, ui32SequenceNum, ui32Trigger, ui32Priority)

トリガ発生条件と優先順位の設定を行う。

```
ADCSequenceConfigure(ADC0_BASE, 3, ADC_TRIGGER_PROCESSOR, 0);
```

設定項目 例

- ADC0_BASE　：使用したい ADC の指定。
  例は ADC0。
- 3　：シーケンサ番号の指定。番号によって，扱えるデータ数が変わってくる。
  3 なら扱えるデータ数が 1 つ，1 と 2 なら 4 つ，0 なら 8 つ扱える。
  今回は 1 つの A/D データしか扱わないため 3 に設定している。
- ADC_TRIGGER_PROCESSOR：A/D 変換を開始する (トリガ) の条件を指定する項目。
  このほかに ADC_TRIGGER_TIMER　タイマの割り込みでトリガ
  ADC_TRIGGER_EXTERNAL　外部信号でトリガ　などがある。
  今回は，プログラム上で任意のタイミングでトリガするために
  ADC_TRIGGER_PROCESSOR としている。
- 0　：優先度の指定する。
  0 ～ 3 まで指定でき，0 が高優先度，3 が低優先度となる。

A/D 変換

ライブラリ関数　09-1

ADC シーケンサのチャンネル指定や諸設定
ADCSequenceStepConfigure(ui32Base, ui32SequenceNum, ui32Step, ui32Config)

ADC のチャンネル指定や諸設定を行う。

```
ADCSequenceStepConfigure(ADC0_BASE, 3, 0, ADC_CTL_CH0 | ADC_CTL_IE | ADC_CTL_END);
```

設定項目 例

- ADC0_BASE　：設定したい ADC の指定。
  ADCSequenceConfigure で設定したものと同じにすること。
- 3　：シーケンサ番号の指定。
  ADCSequenceConfigure で設定したものと同じにすること。
- ADC_CTL_CH1 | ADC_CTL_IE | ADC_CTL_END：データを取得するチャンネルの指定などを行う項目。
  - ADC_CTL_CH0　：A/D 変換でしようするチャンネルを指定する。
    チャンネルは 0 ～ 15 まで指定できる。
    ただし，ボードにより使用できるチャンネルが制限されている場合がある。
  - ADC_CTL_IE　：ADC による割り込みを許可したい場合に設定する。
  - ADC_CTL_END　：ADC の終了を定義する。

ADC シーケンサの有効化　ADCSequenceEnable(ui32Base, ui32SequenceNum)

ADC シーケンサを有効化する。

```
ADCSequenceEnable(ADC0_BASE, 3);
```

設定項目 例

- ADC0_BASE　：開始する ADC の指定。
  ADCSequenceConfigure で設定したものと同じにすること。
- 3　：シーケンサ番号の指定。
  ADCSequenceConfigure で設定したものと同じにすること。

ADC の開始　ADCProcessorTrigger(ui32Base, ui32SequenceNum)

ADC を開始する。

```
ADCProcessorTrigger(ADC0_BASE, 3);
```

設定項目 例

- ADC0_BASE　：開始する ADC の指定。
  ADCSequenceConfigure で設定したものと同じにすること。
- 3　：シーケンサ番号の指定。
  ADCSequenceConfigure で設定したものと同じにすること。

変数に A/D 値を格納
ADCSequenceDataGet(ui32Base, ui32SequenceNum, *pui32Buffer)

指定した変数に A/D 値を格納し，格納したサンプル数を返す。

```
ADCSequenceDataGet(ADC0_BASE, 3, &data);
```

設定項目 例

- ADC0_BASE　：開始する ADC の指定。
  ADCSequenceConfigure で設定したものと同じにすること。
- 3　：シーケンサ番号の指定。
  ADCSequenceConfigure で設定したものと同じにすること。
- &data　：A/D 値を格納する変数のポインタを指定。

A/D 変換

コーディング 09-1

フローチャートを元に，ソースを記述してください。ソースが完成したら，実行して動作を確認しましょう。
以下に解答例ソースを示します。解答例やサンプルソースを参考に，皆さんで工夫してみてください。

step09-1.c

CD-ROM の「サンプルソース」フォルダに、各ステップの c ファイルを収録しています

```
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_types.h"
#include "inc/hw_memmap.h"
#include "driverlib/gpio.h"
#include "driverlib/sysctl.h"
#include "driverlib/adc.h"

// ボリューム入力によるレベルメータ
void main(void) {
        // A/D 変換値格納用
        uint32_t data;
        // LED 点灯パターン
        char State;
        // 動作クロックの設定
        SysCtlClockSet(SYSCTL_SYSDIV_1 | SYSCTL_USE_OSC | SYSCTL_OSC_MAIN | SYSCTL_XTAL_16MHZ);

        // 使用するペリフェラルの有効化
        // : LED 用に I/O ポート C，D を，アナログ入力用にポート E を使用
        SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);
        SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
        SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
        // : A/D 変換モジュール 0 を使用
        SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);

        // PD0~PD3,PC4 ～ PC7 を出力タイプに設定
        GPIOPinTypeGPIOOutput(GPIO_PORTD_BASE, 0x0F);
        GPIOPinTypeGPIOOutput(GPIO_PORTC_BASE, 0xF0);

        // ADC の設定
        // : PE3(AIN0) を ADC に設定
        GPIOPinTypeADC(GPIO_PORTE_BASE, GPIO_PIN_3);
        // : 使用する ADC シーケンサの指定
        ADCSequenceConfigure(ADC0_BASE, 3, ADC_TRIGGER_PROCESSOR, 0);
        // : ADC シーケンサの設定
        ADCSequenceStepConfigure(ADC0_BASE, 3, 0, ADC_CTL_CH0 | ADC_CTL_IE | ADC_CTL_END);
        // : ADC シーケンサを有効にする
        ADCSequenceEnable(ADC0_BASE, 3);
```

```
39
40          while (1) {
41                  // A/D 変換開始
42                  ADCProcessorTrigger(ADC0_BASE, 3);
43                  // A/D 変換値の取得
44                  ADCSequenceDataGet(ADC0_BASE, 3, &data);
45                  // A/D 値によって点灯パターンを設定
46                  if (data < 512) {
47                          // LED を 1 個点灯
48                          State = 0x01;
49                  } else if (data < 1024) {
50                          // LED を 2 個点灯
51                          State = 0x03;
52                  } else if (data < 1536) {
53                          // LED を 3 個点灯
54                          State = 0x07;
55                  } else if (data < 2048) {
56                          // LED を 4 個点灯
57                          State = 0x0F;
58                  } else if (data < 2560) {
59                          // LED を 5 個点灯
60                          State = 0x1F;
61                  } else if (data < 3072) {
62                          // LED を 6 個点灯
63                          State = 0x3F;
64                  } else if (data < 3584) {
65                          // LED を 7 個点灯
66                          State = 0x7F;
67                  } else {
68                          // LED を 8 個点灯
69                          State = 0xFF;
70                  }
71                  // LED に点灯パターンを出力
72                  GPIOPinWrite(GPIO_PORTD_BASE, 0x0F, ~State);
73                  GPIOPinWrite(GPIO_PORTC_BASE, 0xF0, ~State);
74          }
75  }
```

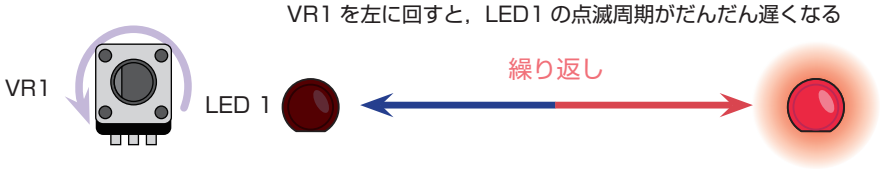
A/D 変換

課題 09-2

以下のように，LED の点滅周期をボリュームで制御してみましょう。

VR1 を右に回すと，LED1 の点滅周期がだんだん速くなる

VR1 を左に回すと，LED1 の点滅周期がだんだん遅くなる



解答は，巻末の解答例集参照